Appendix A – Cover

Solent University

FACULTY OF BUSINESS LAW AND DIGITAL TECHNOLOGIES

M.Sc. In Applied AI & Data Science

2022

Ankush Babbar

“A Smart House Price Prediction System using Machine Learning Models”

Supervisor : **Dr Olufemi Isiaq**
Date of submission : **September 2022**
Student ID : **Q15807339**

SOLENT UNIVERSITY
FACULTY OF BUSINESS LAW AND DIGITAL TECHNOLOGIES

MSc Applied AI and Data Science
Academic Year 2021-2022

Ankush Babbar

A Smart House Price Prediction System using Machine Learning Models

Supervisor: Dr Olufemi Isiaq

September 2022

This report is submitted in partial fulfilment of the requirements of Solent University for the degree of MSc Artificial Intelligence and Data Science

ACKNOWLEDGEMENTS

I would really like to thank everyone who assisted and encouraged me in completing this research project.

Firstly, I would like to thank my supervisor Dr Femi, for supporting me through my project by guiding me from where to start and how to end. His consistent assistance helped me in achieving the research.

Secondly, I would like to thank my family for supporting me morally and emotionally in due course of my dissertation.

Thirdly, I would like to thank the university for giving me an opportunity to learn and research on a vast topic in helping me completing my masters.

Lastly, I am very thankful to God for providing me with such supporting people.

ABSTRACT

Machine Learning has a significant and quicker growth in this decade. Every day, new Machine Learning applications and algorithms are being developed. It plays a significant role in spam detection, recommendations, medical treatments as well. The present machine learning approach helps us improve public safety, security alerts, and medical interventions. However, not much is being worked in domain of real estate sector, house prices continue to rise year after year and the most difficult issue is managing or forecasting real estate values. As a result, stakeholder, purchasers, and builders want to know which factors influence housing prices. We address the prediction of house prices generated by machine learning models in this research paper. We would also investigate the elements that influence housing market such as location, territory, stock market, and ageing and ultimately, develop a system to assist clients in acquiring a property that meets their needs. This research intends to analyse, evaluate, and construct an intelligent system deploying artificial intelligence on the real estate sector using various Machine Learning models such as K-Means Clustering, Random Forest Regressor, XGBoost, Neural Networks with an addition of areas of interest that people might look for most visited venues. Moreover, this research will explore the differences between many advanced models using both classic and advanced machine learning methodologies. In addition, it will extensively examine a variety of ways in model implementation and deliver an optimistic outcome for housing price prediction.

Keywords: House Price Prediction, Artificial Intelligence, Machine Learning, K-Means Clustering, Random Forest Regression, Sequential Neural Network Regression, XGBoos

CONTENTS

ACKNOWLEDGEMENTS i

ABSTRACT ii

FIGURES iii

TABLES iv

ACRONYMS v

CHAPTER 1: INTRODUCTION & BACKGROUND 1

1.1 Why House Price Prediction? 2

1.2 Why United Kingdom? 2

1.3 Why Artificial Intelligence? 4

1.4 Aim(s) 5

1.5 Objectives 5

1.6 Research Questions 6

1.7 Project Overview 6

CHAPTER 2: LITERATURE REVIEW 7

CHAPTER 3: METHODOLOGY 15

3.1 Phase I: Investigation of determinants 15

3.2 Phase II: Data Collection 16

3.3 Phase III: Data Analysis 17

3.4 Phase IV: Model Development and Deployment 18

3.5 Phase V: Evaluation 18

CHAPTER 4: IMPLEMENTATION 19

4.1 Phase I: Hypothesis Testing 19

4.1.1 Hypothesis I: Income of Householders vs House Price 21

4.1.2 Hypothesis II: Inflation vs House Price 23

4.1.3Hypothesis III: Economy (GDP) vs House Price Index 26

4.1.4 Hypothesis IV: Interest Rate (Bank Rate) vs House Price Index 29

4.1.5 Hypothesis V: Outstanding Debt (Balance) vs House Prices 31

4.1.6 Hypothesis VI: Stock Market vs House Price Index 33

4.2 Phase II: Data Collection 35

4.3 Phase III: Data Analysis 38

4.3.1 EDA on Land Registry 39

4.3.2 EDA on data acquired through API 41

4.3.3 Comparison between different indices 44

4.4 Phase IV: Model development and deployment 46

4.4.1 Unsupervised Learning Model (K-Means Clustering) 47

4.4.2 Prediction of House Price Index with Sequential Neural Networks 50

4.4.3 Machine Learning models for House Price Prediction 52

4.4.4 Graphical User Interface 56

4.5 Phase V: Evaluation 61

4.5.1 Correlation Coefficient (R2) 61

4.5.2Mean Absolute Error (MAE): 62

4.5.3 Mean Squared Error (MSE): 62

4.5.4 Root Mean Squared Error (RMSE) 62

CHAPTER 5: RESULTS 63

CHAPTER 6: DISCUSSION: 65

6.1 Hypothesis I: Income of householder's vs House Price 65

6.2 Hypothesis II: Inflation vs House Price 66

6.3 Hypothesis III: Economy (GDP) vs House Price 66

6.4 Hypothesis IV: Interest Rate (Bank Rate) vs House Price Index 66

6.5 Hypothesis V: Outstanding Debt (Balance) vs House Prices 66

6.6 Hypothesis VI: Stock Market vs House Price Index 67

6.7 Different Indexes 67

6.8 Modelling 67

CHAPTER 7: CONCLUSIONS 68

CHAPTER 8: LIMITATIONS AND FUTURE SCOPE 69

CHAPTER 9: REFERENCES 70

9.1 Text Reference 70

9.2 Figure References: 77

9.3 Code References 77

CHAPTER 10: APPENDIX 78

10.1 Ethics Form 83

10.2 Code 83

FIGURES

Figure 1: Average House Price in United Kingdom 3
Figure 2:Annual house price change in United Kingdom 4
Figure 3: Project Workflow 7
Figure 4: Steps to Hypothesis testing 21
Figure 5: Correlation coefficients 23
Figure 6: UK inflation data with HPI (1988-2022) 25
Figure 7: UK HPI vs GDP Data(1960-2022) 28
Figure 8: UK bank Rate (1975-2022) 31
Figure 9: EDA in machine learning process 39
Figure 10: EDA of Land Registry data price 39
Figure 11: Percentage change in House price index EDA 40
Figure 12: Average sales volume country wise EDA 40
Figure 13: Geographical mapping of prices country wise 41
Figure 14: geographical maps with prices of regions of England 42
Figure 15: Region wise bar plots of average house prices 43
Figure 16: Average growth in 1 year region wise 44
Figure 17: Plotting of different indexes 45
Figure 18: Comparison of change in price by different indexes 46
Figure 19: Bar chart of prices by different indexes 46
Figure 20: Machine learning model development cycle 47
Figure 21: Code for finding co-ordinates 48
Figure 22: Dataframe of neighbourhood venues 48
Figure 23: Most common venues dataframe 49
Figure 24: Elbow Method and distortion score to find K 49
Figure 25: Folium map with clustering 50
Figure 26: UK HPI line chart 51
Figure 27: Architecture of Neural Network 52
Figure 28: Architecture of GUI 57
Figure 29: Frame 1 of GUI 58
Figure 30: GUI Frame 1 know more button display 59
Figure 31: Frame 2 part 1 of GUI 60
Figure 32: Frame 2 part 2 of GUI 60
Figure 33: Frame 3 of GUI 61
Figure 34: Formula for R2 score 62
Figure 35: Formula for MAE 62
Figure 36: Formula for MSE 62
Figure 37: Formula for RMSE 62
Figure 38: Plotting of predications over Train and test data 64

Figure 39: Dataset from Kaggle(Income vs House Price) 78
Figure 40: Dataset from gov.uk (income vs House price) 78
Figure 41: Lag plots for inflation and HPI........ 79
Figure 42: UK GDP Data(1960-2022) 79
Figure 43: EDA on land registry data HPI........ 80
Figure 44: Comparison of house price by different indexes........ 80
Figure 45: Change in House Prices by different indexes 81
Figure 46: Epochs vs loss for Neural Network 81
Figure 47: Histogram of prices of price paid transactions........ 82
Figure 48: Box plot for House Prices vs property types........ 82
Figure 49: Box plot for top 4 cities in transactions........ 83
Figure 50: crosstab for number of records with property types........ 83

TABLES

Table 1 : Brief of Literature Review........ 10
Table 2: Data Description of Income 22
Table 3: Data Description of Inflation data 24
Table 4: Data description for GDP data........ 27
Table 5: Data Description for Bank rate History 29
Table 6: Data Description for Outstanding Debt........ 32
Table 7: Data Description of Stock Market 33
Table 8: Input parameters for API to get bedroom prices........ 36
Table 9: Input parameters for API to get crime data 36
Table 10: Input parameters of development calculator API 37
Table 11 : Colour coding of the regions on Geographical map 42
Table 12: Data Description of different indexes 44
Table 13: Hyper parameters for RandomForest 54
Table 14: Hyper parameters for XGBoost........ 55
Table 15: Hyperparameters for Sequential Neural Network........ 55
Table 16: Results of Hypothesis testing........ 63
Table 17: Performance Grid for Different Models 64

ACRONYMS

AI	Artificial Intelligence
ANOVA	Analysis of Variance
API	Application Programming Interface
CATBOOST	Categorical Boosting
CDA	Confirmatory Data Analysis
CPI	Consumer Price Index
EDA	Exploratory Data Analysis
GARCH	Generalized Autoregressive Conditional Heteroskedasticity
GDP	Gross Domestic Product
GUI	Graphical User Interface
HPI	House Price Index
IDE	Integrated Development Environment
LIGHTGBM	Light Gradient Boosting Machine
MAE	Mean Absolute Error
MAX	Maximum
MIN	Minimum
ML	Machine Learning
MSE	Mean Square Error
R2	coefficient of determination
RMSE	Root Mean Square Error
RPI	Retail Price Index
UK	United Kingdom
URL	Uniform Resource Location
XGBOOST	Extreme Gradient Boosting
YFINANCE	Yahoo Finance

CHAPTER 1: INTRODUCTION & BACKGROUND

Have you ever considered buying a property? You can browse a variety of homes online and get ideas for properties that interests you. Following which, you might consider the looks and the price. However, you might not know what you are paying for and what all is covered in the price. Are the houses valuated simply as others? Maybe the price is higher due to nearby shopping centre or a good school. What are you paying for, and how much does the features of the property affects the price? When it comes to pricing of a property, there are many unanswered/unresolved questions.

The significance of the real estate market cannot be understated. It is a key portion of an individual's wealth, and for professional investors, it is a different asset class that may be utilized to further diversify their investments. As a result, developers, appraisers, homebuyers, and other real estate market participants, such as lenders rely on accurate house price forecasts [1].

The Behaviour of the real estate market has macroeconomic implications. This sector alone accounts for a significant share of the GDP (Gross Domestic Product). Moreover, House price fluctuations also have a minimal impact on other markets, not only to banking sector but also to business investments or personal consumption [2,3]. Even the real estate market has impact at the micro level. From a personal perspective, a housing decision is one of the important financial decisions in most people's lives. Thus, understanding rental decisions and real estate market determinants is focused by macro and micro-economics. For instance, it is generally established that housing markets exhibit long-run mean reversion and short run momentum [4].

1.1 Why House Price Prediction?

Housing, a sub-division of real estate industry is a crucial component of a sustainable economy. Owning a property is seen as having a good social standing in various nations and is a goal of young individuals entering the employment market. Although the reasons are different, the parties who chooses to own a house can enter the market and can take advantage of the increase in values. The most important issue in stabilizing the real estate market is economic feasibility, it is only sustainable if it is affordable and economical. The cost-effectiveness of a home also affects the stability of its use as an investment tool [5,6].

The financial crisis of 2008 expanded from United States of America to other countries, every nation developed a growing interest in the housing market (Asian crisis in 1997, as well as the asset price surge before to the 1929 and 2008 US crises, are instances of big financial crises that were created by speculative home price bubbles). Prolonged price increase, combined with disproportionate investment, leads to speculative price fluctuations known as bubbles, which have a detrimental impact on long-term economic growth [7]. Since house prices directly or indirectly affects economic steadiness, it is essential to control the real estate market and the price of housing.[8]

1.2 Why United Kingdom?

We choose United Kingdom for a variety of reasons. First and foremost, the housing industry in the country is vital for both resident and non-resident investors. The Rapid growth of real estate business over the past 15 years has caught the interest of overseas real estate investors. The sector's pivotal role, particularly during the economic downturn has sparked intense interest in the behaviour of the real estate markets. The rate of owner occupation in the United Kingdom is significantly greater than in other European countries. Furthermore, "new figures based on HM Land Registry records reveal that foreign ownership has increased around 180 percent in the last 11 years".[9] "The Value of inward

Foreign Direct Investment position climbed in 2020 by £288.7 billion (17.6%), while the outward increased by £13.5 billion (0.8%)". The value of inbound profits in professional and support sectors was roughly five times greater in 2020 than in 2019 [10]. Currently, currency rates are another reason to invest in property in the country due to European Union referendum the value of the pound has fallen. The uncertainty around Brexit, followed by the COVID-19 epidemic has kept the value of the pound low which gave a chance for foreign investors to take advantage to expand their property investment portfolio. Average house price has increased from £150k to £276k in past 15 years (see figure 1,2). As per Bank of England, the house prices increased from around £10k in 1977 to £200k in 40 years (see figure 2).

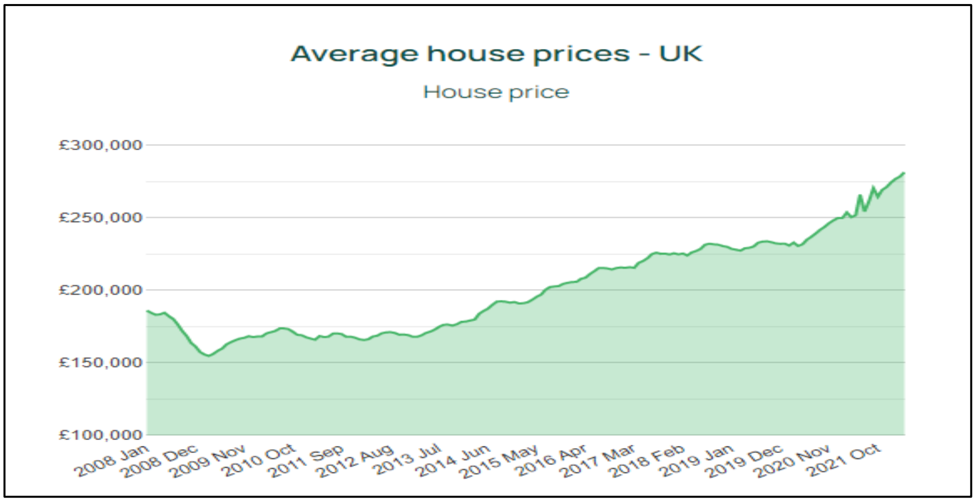


Figure 1: Average House Price in United Kingdom

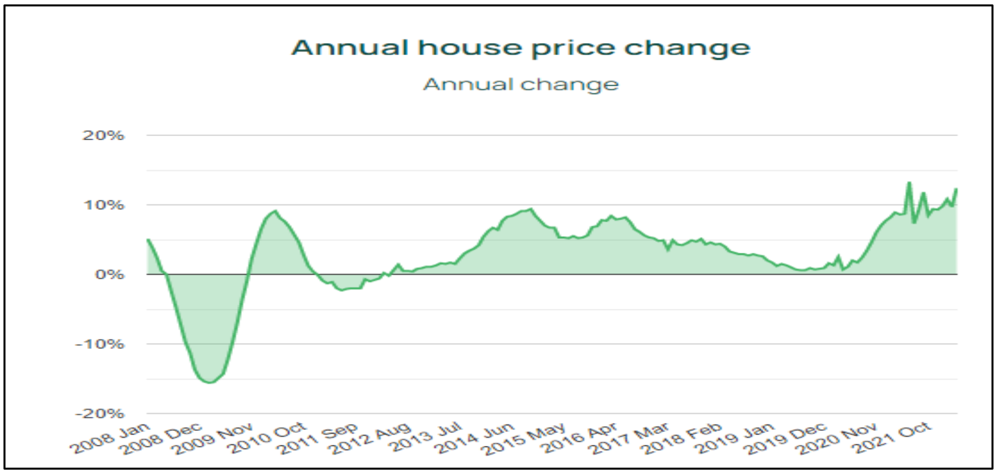


Figure 2:Annual house price change in United Kingdom

Thus, this research will look at the importance of various characteristics for homes in United Kingdom. We analyse numerous factors that influence pricing, such as exteriors, surroundings, year built and others which contribute towards price. Moreover, in the analysis we will also include external factors such as the stock market, interest rates, unemployment rates, house price index and environmental factors will utilize statistical techniques to analyse how each of many factors affects the pricing. Around 80 percent of the United Kingdom's people reside in cities, where the need for accommodation is greatest. The number of houses falls well short of needs, and at an annual level of replacement of 0.5 percent [39]. England's real estate market is one the least adaptive in the civilized economies due to large number of owner-occupiers, poor renting segment and an absence of long-term financing solutions.

1.3 Why Artificial Intelligence?

Since, there is so much data on real estate markets and transactions, artificial intelligence and machine learning may be of great assistance in processing and evaluating existing data. Although, real estate may seem to be the only thing that is still tangible in a world that is becoming more and more virtual, artificial intelligence is also taking over this industry. It is rapidly being used by real estate businesses in all aspects of house purchasing, selling, and financing. Some of the industry's top names like Compass [11], Zillow [12] and LoanSnap [13] are now using Artificial Intelligence to assist buyers locate the appropriate mortgage and the perfect property.

Zillow has updated its popular house price "Zestimate" [12] which uses neural networks to predict house prices. Most real estate information, including land records, title documents, purchase price and even mortgage liens is public data. The issue was that gathering all the information required visiting local offices, which took a lot of time. Nowadays, algorithms can quickly process documents to find information about property values, home improvements, debt levels.

1.4 Aim(s)

- To create a smart system for the users to show the areas of interest with the estimate prices of houses and to analyse the dynamics of the housing market in United Kingdom's economy.

1.5 Objectives

- To investigate housing market in United Kingdom and corresponding variables in relation to literature review through statistical analysis.
- To deploy machine learning model through Interactive Graphical User Interface (a complete system with intuitive interface) which would display key venues and analysis of the real estate market of United Kingdom.
- To assist real estate agents, stakeholders, builders, purchasers in identification of key areas(venues) in location by using unsupervised

machine learning model and determine which type of house would be best as per user’s budget using artificial intelligence.

1.6 Research Questions

- Can Artificial Intelligence be used to deploy model for dynamic housing market?
- To examine the impact of variables/factors related to housing market using statistical analysis (hypothesis testing).
- How to apply Machine learning to predict the house prices in United Kingdom?
- How an Artificial Intelligence system can reduce the effort and time of purchasers in finding desired houses.
- What is the significance of using Machine learning models instead of using classical approach for predicting housing prices?

1.7 Project Overview

The research approach for carrying out this study is how a smart System will be built and evaluated to determine the house prices. The study technique includes a suggested intelligent system capable of predicting prices and analysis of investment using artificial intelligence and neural networks (deep learning). This part is divided into mainly five sub-sections ,namely, investigation of the variables that impact the outcome variable which would be the house price ;dynamic data collection for all the identified important variables through various Application programming interface(APIs) ;carrying out analysis of the data which contains cleaning and pre-processing , finding out variable interaction on the effect of House Price, combining the likewise variables ,and how variables are related together; after which a model would be implemented that would be deployed on an interactive Graphical User Interface; and finally to evaluate the effectiveness of the system that meets commercial deployment (see figure 3)

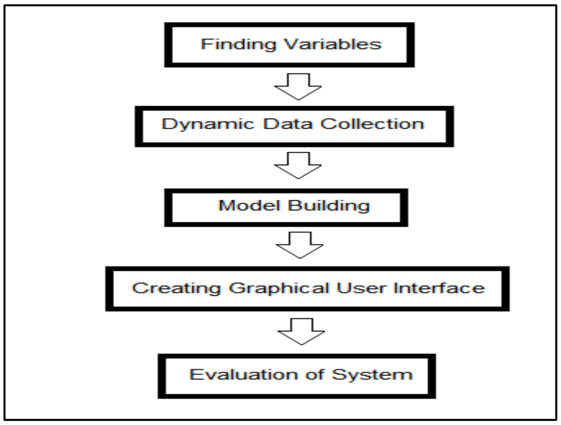


Figure 3: Project Workflow

CHAPTER 2: LITERATURE REVIEW

(Takats, E. (2012).) explores the impact of ageing on housing values. The report concludes that ageing will have a considerable negative impact on house values. The paper's key contribution is to utilize a large panel of international house prices to quantify the effects of continuing demographic shifts on present and future price trends. The article shows convincing evidence that demographic considerations, and hence ageing have an impact on property values. The study begins by developing a model to formalize the role of demographic trends in determining property prices. The model aids in identifying the old age dependence ratio and total population as significant demographic effect factors. The empirical research makes use of international panel of house price data from bank for international settlements, the data covers 22 advanced economics from 1970 to 2009. While controlling for macroeconomic factors, the research deploys a panel regression model in differences relating the old age dependence ratio and total population to actual housing prices.

(Sivitanides, Petros. (2018).) Validated and quantified the impact of important macroeconomic variables on housing prices in London. The Halifax and Nationwide House price indices were used with the date ranging from 1983 to 2016. The author estimated alternate error-correction and partial-adjustment models. A classical demand-supply paradigm was used to explain the intertemporal behaviour of London housing prices, which states that prices at each point in time are controlled by the interaction of supply and demand. The research confirmed and quantified that the long-term impacts of Gross Domestic Product, London population and cumulative construction completions on London's house values.

(Jha & Radu (2020).) created machine learning models for predicting the price of the house. This research covered XGBoost, CatBoost, RandomForest, Lasso, Voting Regressor and more, which are being employed to predict prices on a residential dataset of 62,723 records from January 2015 to November 2019 and was obtained from the Florida State Real Estate Assessor website. Various cleaning and pre-processing techniques were used with feature selection and feature engineering. Empirical results showed that XGBoost algorithm outperforms other models based on performance, coefficient of determination (R2), Mean Square Error, Absolute Mean Error, and computation time to predict the price of houses.

(Gohl, Niklas & Haan (2022)), investigated housing market price expectations in short, medium, and long term. About six hypotheses concerning the occurrence, development, importance of pricing expectations is developed and tested. Data from personalized household survey, previous sales and rental offerings, satellites was used. Additionally, data from the Empirica housing data bank for the years 2012 to 2016 to generate postal code specific house price trends with satellite data. The data bank contained around 484,604 observations for rental and non-rental properties. The research stated that in long run, individual house price forecasts exhibit patterns of mean reversion. Furthermore, it demonstrated that most tenure selections are not significantly connected to future price predictions.

(Bissoondeeal, R.K. and Tsiaras, L. (2021).) studied the non-linear relationship between the housing and stock markets. The research was carried out at both national and regional levels. It also investigated the relation of London property values with rest of the country. Copula theory is used to simulate the conditional joint distribution of the stock and housing market to evaluate their dependent structure. To estimate the distribution, Generalized Autoregressive Conditional Heteroscedasticity (GARCH) model was used. Quarterly data from 1973 to 2016 was used which was taken from the Nationwide website, All Share Price Index reflected stock market activity, three-month treasury bill rate was utilized as interest rate. Moreover, how closely other parts of the country were tied to those in London. The research concluded that both markets have positive relationship and may display left tail dependency.

(Quang Truong, Minh Nguyen, 2019) used both classic and advanced machine learning methodologies to evaluate the differences between numerous advanced models and presented an optimistic outcome for housing prediction. The dataset contained about 300,000 data points with 26 variables for pricing in Beijing between 2009 and 2018. Before implementing the model, several data pre-processing approaches were used. GridSearchCV was used to refine models such as RandomForest, XGBoost, LightGBM and hybrid regression. RandomForest produced the lowest error on training set, whereas Stacked Generalization regression produced acceptable results.

(Chandler, D. (2014)) researched on the housing market's economics and explained by home prices are likely to be more volatile than those in other markets. It was also discussed topics of the property market in the United Kingdom. It was discovered that the value of household home wealth is likely to be variable in the near run and then assessed the impact of this volatility on families.

(Adam, Roland, 2009) investigated that construction costs and new built prices have a significant impact on house values. Building expenses include the cost of construction materials as well as manning. The greater the cost of development reduces construction and housing stock, and lower level of housing space generates higher rent and house prices. Furthermore, changes in prices impacts the building activities in construction sector, basically on profitability of development projects. Thus, building costs have favourable effect on values of houses.

(Lu Decee Xu and Tang, B. (2012)) research investigated on the drivers of United Kingdom home prices based on quarterly data from 1971 to 2012. Co-integration technique and associated error correction model was used. According to the test, GDP, unemployment, credit, interest rate has a positive effect on prices, while income and supply of money have a negative effect. The Model suggested that increase of prices is controlled by building cost, income, interest rate and credit. The research recommended investors to pay close attention to fluctuations in interest rate.

(Visit, Gan, C. and Lee, M. (2004)) experimentally evaluated the predictive capacity of the hedonic model with artificial neural network model. Data of about 200 sample properties was collected from Christchurch, New Zealand. Geographical location as well as other characteristics of property such as type, number of bathrooms, bedrooms, garages, home size are used. Three layered Neural Network was used in the modelling phase. Weighted Least Square technique was used to calculate accuracy instead of ordinary least square technique. The research found that hedonic pricing models do not outperform neural network prices models.

Table 1 : Brief of Literature Review

Year	Name	Publisher/ Journal	Title	Model	Data	Description/conclusion

2004	Visit Limsombunchai, Gan, C. and Lee, M.	Research Gate	House Price Prediction: Hedonic Price Model vs. Artificial Neural Network	3 Layered Artificial Neural Network	200 sample properties in New Zealand	Artificial neural network outperforms from hedonic model
2009	Adams, Zeno & Füss, Roland	Journal of Housing Economics	Macroeconomic determinants of international housing markets	Panel Co-integration Analysis	Data from 15 countries over a period of 30 years	Empirical results indicate Prices to increase by 0.6% in response to 1% increase in economic activity
2012	Takáts, E.	Journal of Housing Economics	Aging and house prices	Regression Model	22 advanced economics from 1970-2009	Ageing has negative impact on House Values

2012	Lu Decee Xu and Tang, B.	Semantics Scholar	ON THE DETERMINANTS OF UK HOUSE PRICES.	Co-integration technology and associated error correction model	Quarterly Data from 1971-2012	GDP, credit, unemployment has positive impact on prices, while income and supply have negative
2014	Daniel Chandler and Richard Disney	FISCAL STUDIES,	The Housing Market in the United Kingdom: Effects of House Price Volatility on Households	Demand - Supply Model	Disposable income from 1983-2013	Housing market is volatile than other markets
2018	Sivitanides, Petros.	Journal of Property Investment and Finance	Macroeconomic drivers of London	Classical demand-supply paradigm	Halifax and Nationwide Indices	Long-term impacts of GDP, population on house prices

			house prices.		1983-2016	
2019	Truong, Q., Nguyen, M., Dang, H. and Mei, B.	Procedia Computer Science,	Housing Price Prediction via Improved Machine Learning Techniques.	RandomForest, XGBoost, LightGBM	300,000 data points with 26 variables	Stacked Generalization regression produced acceptable results
2020	Jha, Shashi Bhushan & Babiceanu, Radu & Pandey, Vijay & Jha, Rajesh Kumar.		Housing Market Prediction Problem using Different Machine Learning Algorithms: A Case Study	XGBoost, CatBoost,RandomForest, Lasso, Voting Regressor	62,7234 records	XGBoost outperforms in prediction

2021	Bissoondeeal, R.K. and Tsiaras, L.	Journal of Real esate finance and economics	Investigating the Links between UK House Prices and Share Prices with Copulas.	GARCH Model	Quarterly data from 1973-2016	Stock Market and House Price have positive relationship
2022	Gohl, Niklas & Haan, Peter & Michelsen, Claus & Weinhardt, Felix.	SSRN Electronic Journal	House Price Expectations.	Six Hypothesis tested	484,604 properties	Long run price exhibits pattern of mean reversion

CHAPTER 3: METHODOLOGY

Data Science applications seek to gather data and perspective from collected data. The originality and applicability of the discovered insights are highlighted. However, the influence of a project might be significantly lessened if the outcomes are not effectively conveyed. [24]

The research strategy for carrying out this study is to build and assess a smart system to predict housing values thus following a quantitative research methodology [25]. The research method involves a proposed intelligent system capable of forecasting pricing and investment analysis utilizing artificial intelligence (neural networks) and machine learning models. This section is divided into five sub-sections: investigation of the variables that impact the outcome variable, which is the house price; dynamic data collection for all the identified important variables through various application programming interfaces(APIs); data analysis, which includes cleaning and pre-processing; variable interaction on the effect of house price, combining the likewise variables, and how variables are related to each other; after which a model would be developed and deployed on an interactive Graphical User Interface(GUI), and lastly the efficiency of the system that satisfies commercial deployment would be evaluated.

3.1 Phase I: Investigation of determinants

The First phase of the research was the investigation of various factors or determinants for house price, which would be done through literature reviews, researching on journals and articles published by the various government agencies. As per Takatas[14] the demography affects the price of houses and suggests that major shift is taking place. Aging may lower property values since young professionals tend to buy properties, but old age people lead to sales, as Krainer [26] documents. As a result, if the proportion of the elderly compared to persons of working age grows, price of the property my decrease, while United Nations Projections [27] too confirms it would happen, in developed countries, the proportion of old age population will nearly quadruple in the coming forty years. Several Studies [28,29] indicate strong indications of cyclical downturn impacts through labour market and income on housing demand. Some of these [29] point to the impact of rising costs or the mortgage crisis. As per official projections, demographic trends are often viewed as the primary basic driver of property prices in the long run, although changes fluctuations in financial conditions, reflected across both lending rates and availability of credit, effect housing market in the near term.[20]. We have taken six hypotheses in the first phase of the research, namely:

- Does the house price and income of households are related?
- Does inflation affect House Prices?
- Does economy effects house price?
- Does interest rate of the banks have an influence on the house prices?
- Does mortgage lending and bankruptcies influence house prices?
- Do the stock markets affect house prices?

3.2 Phase II: Data Collection

The practice of obtaining and quantifying information on variables of interest in a structured framework for answering well-defined research questions, experiment hypotheses and assess outcomes is known as data collection [30]. Since, there would be multiple elements from various domains, the data was collected through different APIs and saved at a local data storage (computer’s hardware). The data was then integrated, and different data processing techniques (data pre-processing, cleaning) were out. Because “garbage in, garbage out” [31] the data needs to be cleansed to obtain reliable findings. Data was gathered from the following sources:

- Office for National Statistics
- Gov.uk
- Ukfinance.org.uk
- Land Registry Data
- Yahoo Finance
- Kaggle
- Propertydata.co.uk
- opendatacommunities.org
- Foursquare API
- Nationwide House Price Index
- Halifax House Price Index
- Land Surveyors House Price Index
- Transactional History from Land Registry

3.3 Phase III: Data Analysis

Data analysis is the way of forming, expressing, unfolding, estimating, and understanding data by means of statistical methods. Data analysis in applications can be classified as, Exploratory Data Analysis (EDA) which focus on uncovering new data characteristics, while Confirmatory Data Analysis (CDA) aids in the confirmation or falsification of existing hypotheses [32]. Pre-processing techniques such as data cleaning, removing null values, data manipulation was carried to feed the machine learning model. Moreover, visualization techniques such as plotting graphs were carried out to get information from the data.

3.4 Phase IV: Model Development and Deployment

Machine learning's fundamental purpose is to develop a model that can make predictions [33]. This phase asses different machine learning algorithms such as K-Means clustering, RandomForest Regressor, XGBoost, Sequential Neural Networks and select the best performing algorithm that suits the best. After obtaining the relevant variables, a regression model was built to estimate property prices, and a classification model to group most visited venues in the selected location (city). Both models were placed on GUI, which provided the user with the required results based on their search criteria.

The interface is classified into three categories: Firstly, if the user wants to know where he/she should seek the property based on their interests which also provides the estimate of the prices of that location. Secondly, if the user wants to know the estimate of the house prices as per their specifications (such as year built, energy rating, location, square footage), moreover it contains additional criteria for predicting the prices of house from demolition or refurbishment which shows profit percentage, this part would be beneficial for purchasers as well as for investors as it would shows the areas to maximize their profits. While the last section shows the graphs of house price index prediction in the next years, prices of house country wise in United Kingdom.

3.5 Phase V: Evaluation

The purpose of evaluating a model is to offer a valid assessment of its predictive ability, performance measurements on the other hand must account for the specificity of numerical forecasts, the differences between true and prediction function values are the most often used performance indicators [34]. The system was assessed based on its efficacy in user engagement, price projections and whether it is suitable for commercial implementation. Various accuracy metrics were used to evaluate performance on train and test datasets such as Mean Absolute Error (MAE), R-squared (R2) Score, Mean Squared Error (MSE). The primary goal of this research was to create a smart system that uses the architecture and outperforms previous designs.

CHAPTER 4: IMPLEMENTATION

Implementing a machine learning algorithm provides a thorough understanding of the way the algorithm works. When implementing a machine learning algorithm, several micro-decisions must be made, and these decisions are frequently omitted from theoretical algorithm definitions. Because few individuals take the effort to design several the more complicated algorithms as a learning exercise, learning and parameterizing these decisions may swiftly push anyone to intermediate and advanced levels of comprehensions of a certain approach. To create Machine learning applications, one must first choose a platform, an integrated development environment (IDE), and a programming language.

For the implementation part, Python is the programming language that has been used as python is a popular language for data science, numerical computation, machine learning which enhances the efficiency and effectiveness by its extensive libraries [35].

4.1 Phase I: Hypothesis Testing

A hypothesis is either a preliminary claim or a formal explanation of concept (provable or disprovable) that demonstrates how two or more factors are predicted to interact [36]. It might be simple propositions of an expected outcome or assertions of the presence of a correlation. Statistical hypotheses are classified into two types: The Null hypothesis (Ho) which states that samples are not affected that is no change or no difference, while the alterative hypothesis (H1 or Ha) is the notion that sample observations are affected by some non-random cause [37].

Hypothesis testing is an approach to evaluate whether to reject or accept the null hypothesis. This procedure consists of the following steps [38]:

- Specify the hypothesis: The hypotheses must be stated.
- Create an analytical strategy: The analytical plan explains how to test the hypothesis using the dataset
- Statistical testing: Determine the test statistic provided such as t statistic, z-score, percentage.
- Interpret the findings: Use the selected statistical method outlined in the analysis plan.

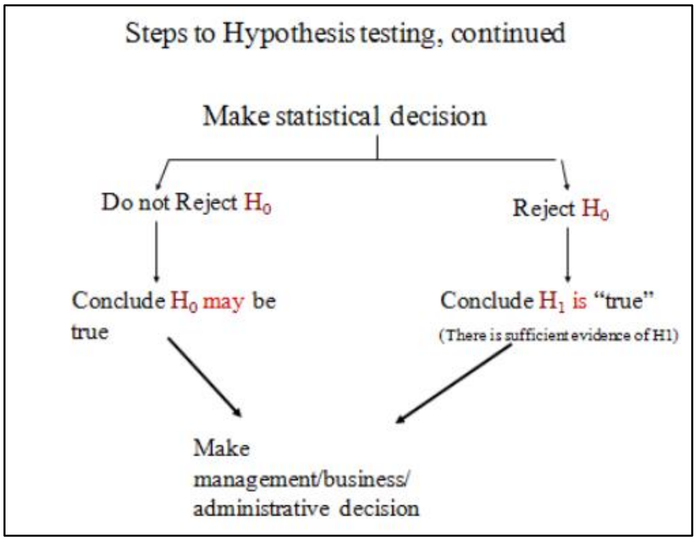

Steps to Hypothesis testing, continued
Make statistical decision
Do not Reject H_0
Reject H_0
Conclude H_0 may be true
Conclude H_1 is "true"
(There is sufficient evidence of H1)
Make management/business/ administrative decision

Figure 4: Steps to Hypothesis testing

We have taken six hypotheses in the first phase of the research.

4.1.1 Hypothesis I: Income of Householders vs House Price

In 2012, Lu Decee and Tang [22] conducted a study on the many factors influencing house prices and discovered that income and money supply had a negative impact on house prices. Ming-chi-chen [40] researched on the link with housing prices and income. While Atalay [41] investigated the influence of housing price volatility on observed economic health using longitudinal data on financial behaviour satisfaction. Thus, it seems that income has a relationship with house prices.

Data Description:

- Dataset downloaded from Kaggle
    - The data was retrieved from Kaggle which contained the average house price in the United Kingdom between 1975-2020 and the average income between 1999-2020.
    - Rows contained the yearly data, while the columns were namely, year, average house prices, and median salary.
- Dataset downloaded from gov.uk
    - The data was retrieved from gov.uk while contained the yearly average house prices and average income for different dwellings from quarter 2 of 1992 to quarter 4 of 2021.
    - Rows contained quarterly data, while columns were year, quarter, average prices and income for new dwellings, other dwellings, all dwellings, first time home buyers and former buyers.

Table 2: Data Description of Income

Index	Data Source	Description	#Rows	#Columns
Income	Kaggle	Monthly data from 1999-2020	46	3
Income	Gov.uk	Quarterly data from 1992-2021	119	17

Python Libraries Used:

- Pandas (used for importing datasets files, data manipulation and data analysis).
- Seaborn (used for plotting advanced statistical plots and graphical representation).
- Matplotlib (used for making basic graphs)
- SciPy. Stats (used for obtaining probability distributions, generate descriptive statistical values).

Hypothesis:

- **Null Hypothesis**: There is a no relation between Income and house price
- **Alternate Hypothesis**: There is a relation between income and house price.

Statistical Testing Used:

- Correlation statistical tests.
- Regression Lines.

Since, we need to find out the relationship between house price and the income of the householders and both variables are continuous, finding the correlation and linear regression is most used technique [42] used for exploring the relationship.

Correlation evaluates the magnitude of a linear connection between two features, whereas regression defines the relation statistically.

Correlation coefficients evaluate the strength of relationship, a positive association is indicated by positive coefficient, while a negative association is depicted by a negative coefficient whereas value near zero indicates no association.

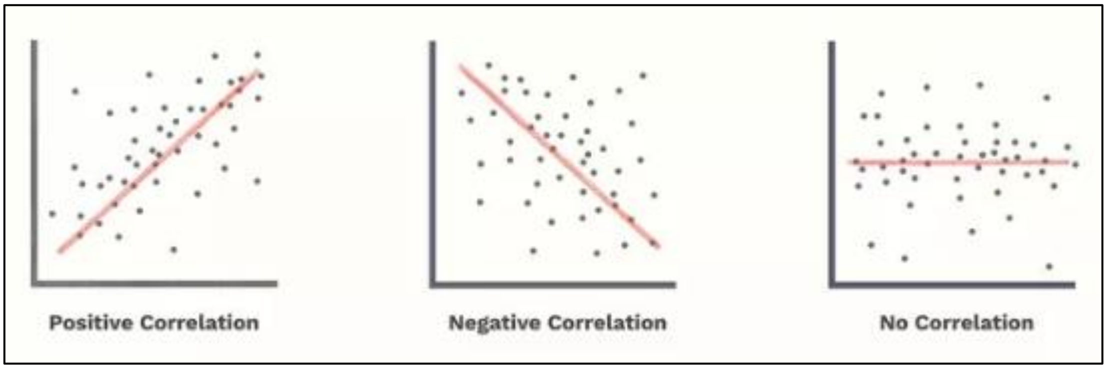


Figure 5: Correlation coefficients

4.1.2 Hypothesis II: Inflation vs House Price

Inflation is defined as a consistent increase in the overall price of all commodities and services produced in each industry. There are several theories as to what the cause is. According to the Keynesian school of economics, inflation is caused by market processes such as supply and demand, which cause price shifts. Increased supply and service costs lead to higher product prices, resulting in “cost-push” inflation [43]. As the price of bricks increases, the cost of constructing rises.

Data description:

Inflation is basically measured through two indexes, Retail Price Index (RPI) and Consumer Price Index (CPI). RPI is produced by Office of National Statistics, and includes mortgage interest payments, whereas CPI includes other goods and services except housing costs.

Table 3: Data Description of Inflation data

Index	Data Source	Description	#Rows	#Columns
Retail Price Index (RPI)	Office of National Statistics	Monthly data from 1897-2022	426	2 (Date, Index)
Consumer Price Index (CPI)	Online Scrapping from rateinflation.com	Monthly data from 1988-2022	414	2 (Date, Index)
House Price Index (HPI)	Office of National Statistics	Monthly data from 1968-2022	650	2 (Date, Index)

Python Libraries Used:

- Pandas
- Matplotlib
- Numpy (used for manipulation of arrays)
- Seaborn
- Statsmodel.tsa.stattools (used for time series analysis)
- Pandas.plotting (Lag_plot used to look for patterns)
- Statsmodel.tsa.api (used vector autoregression for time series analysis)
- Scipy.stats.stats (used kendalltau to measure correlation)

Statistical Testing Used:

- Visualization technique
- Lag plots
- KendallTau

- Granger Causality Test

Hypothesis:

- **Null Hypothesis:** Time series Inflation does not Granger-cause time series house price index
- **Alternate Hypothesis:** Time series Inflation does not Granger-cause time series house price index

Various visualization techniques were used using seaborn library such as line plots for both the RPI and CPI inflation indexes with UK HPI (see fig 6). A lag plot is a sort of scatter plot in which two parameters(x,y) are lagged. A lag is a specified length of time in a time series data. Lag plots are important in checking the outliers, finding patterns, seasonality, and correlation [44] (see appendix fig 41).

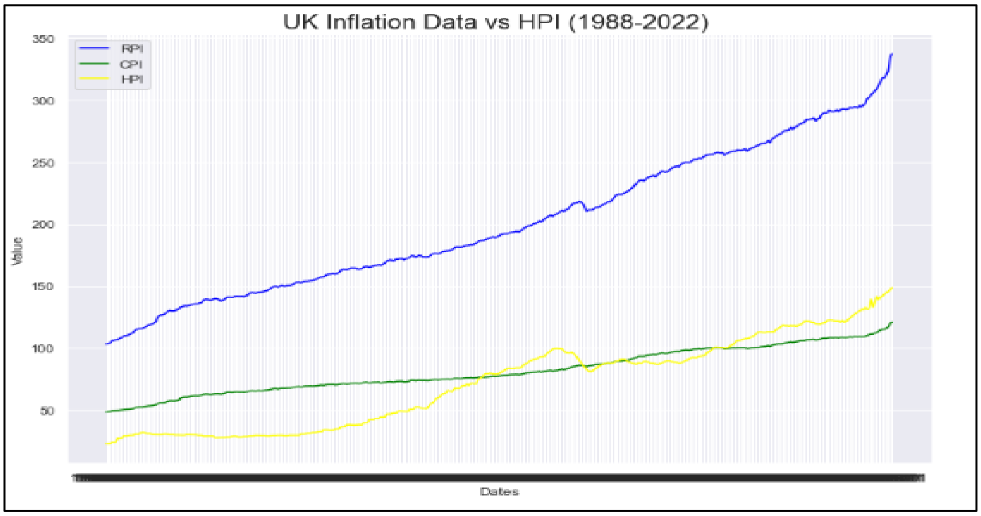


Figure 6: UK inflation data with HPI (1988-2022)

After which, Kendall’s Tau statistical test was used. Kendall’s Tau is a measure of the strength of an association between variables. The analysis of Kendall’s Tau yields two things namely, a correlation coefficient and a p-value. The coefficient varies between -1 and 1, while negative represents the variables are negatively connected while a positive one represents the positive connection [45]. The p-value shows the likelihood of finding results have a real association, moreover, p-value helps in concluding the reject or accept the null hypothesis.

After making the time series stationary through differencing and testing through Dicky fuller test, lag order was calculated using Vector autoregressive model to as to input the max lag into the statistical test named granger causality test, which is used to examine if one time series can foretell another. The value of time series at a particular lag is beneficial for forecasting the value of another series at a later period, is called “Granger-causes” [46]. This test provides with two values: F-test statistics and p-value, if the p-value is less than 0.5, null hypothesis can be rejected, and vice versa.

4.1.3Hypothesis III: Economy (GDP) vs House Price Index

Lu Xu [22] investigated if the Gross Domestic Product (GDP) has a favourable influence on the housing market and found out that GDP has a positive impact on the real estate market. While Aizenman [47] utilized a quarterly dataset spanning four decades to investigate the link between economic growth and housing market. The main aim of this hypothesis was to check the correlation and causation of the two indices.

Dataset Description:

- For economy data, we have used GDP as it is main factor which tells about the state of the economy.

- From the dataset, maximum GDP is 7.44 whereas the minimum is -9.27 with a mean of 2.24 and standard deviation of 2.73
- For House Price Index data, we have used data published by Office of National Statistics.

Table 4: Data description for GDP data

Index	Data Source	Description	#Rows	#Columns
Gross Domestic Product (GDP	World Bank	Yearly data from 1968-2021	54	2 (year, GDP_growth%)
House Price Index (HPI)	Office of National Statistics	Monthly data from 1968-2022	650	2 (Date, Index)

Python Libraries Used:

- Pandas
- Matplotlib
- Numpy
- Seaborn
- Statsmodel.tsa.stattools
- Pandas.plotting
- Statsmodel.tsa.api

Statistical Testing Used:

- Visualization technique
- Lag plots
- Granger Causality Test

Hypothesis:

- **Null Hypothesis:** Time series GDP does not Granger-cause time series HPI
- **Alternate Hypothesis:** Time series GDP Granger-causes time series HPI

Herein, plotting of graphs of the times series of GDP and HPI was done to visualize the relationship of both the series (See fig 7).

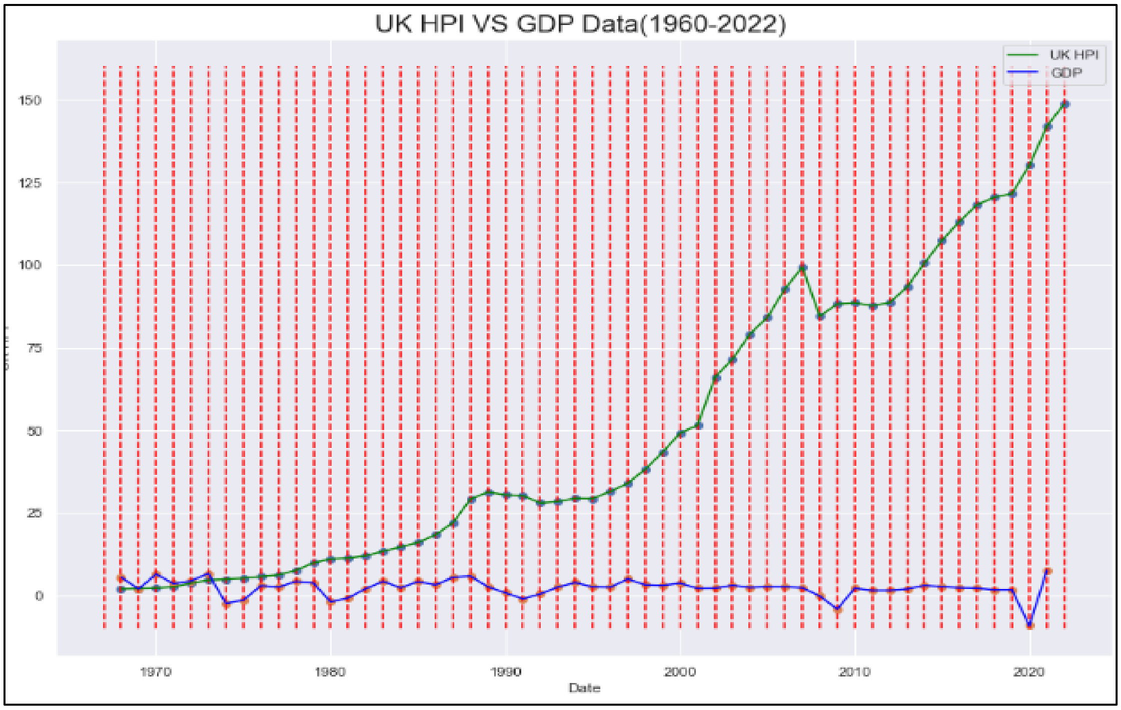


Figure 7: UK HPI vs GDP Data(1960-2022)

Since, GDP data was yearly data and HPI data was monthly, thus, HPI data was resampled from monthly to yearly data. After which Granger Causality test was performed to check the correlation and causation of the both the time series.

4.1.4 Hypothesis IV: Interest Rate (Bank Rate) vs House Price Index

Different interest rate concepts, when combined, described, or offer variables that affect interest rates. “These approaches are different due to disagreements over whether rates of interest are a financial or a real phenomenon, such concepts are the classical theory of interest, the Keynesian liquidity preference theory of interest, the loanable funds theory of interest, the Friedman monetarist framework” [43].

Xu [48] found that interest rates in the real estate market have varied effects on supply and demand. Moreover, found that interest rates and property prices have a complicated and interdependent connection. According to the research [43], interest rates have a negligible link with property values, however the association is associated favourably. The hypothesis which I took is too related to interest rate, whether it affects the real estate market prices or not.

Dataset Description

- Bank rate history data was fetched from Bank of England’s database.
- Bank rate data had maximum value as 17 whereas the minimum as 0.10 with a mean of 9.208 and standard deviation of 3.70

Table 5: Data Description for Bank rate History

Index	Data Source	Description	#Rows	#Columns
Bank Rate History	Bank of England	data from 1975-2022	244	2 (Date, Rate)

House Price Index (HPI)	Office of National Statistics	Monthly data from 1968-2022	650	2 (Date, Index)

Python Libraries Used:

- Pandas
- Matplotlib
- Numpy
- Seaborn
- Statsmodel.tsa.stattools
- Pandas.plotting
- Statsmodel.tsa.api

Statistical Testing Used:

- Visualization technique
- Lag plots
- Granger Causality Test

Hypothesis:

- **Null Hypothesis:** Time series Bank Rate does not Granger-cause time series HP
- **Alternate Hypothesis:** Time series Bank Rate Granger-causes time series HP

Data from the time series was plotted to visual the correlation or trend (see fig 8). Since, the bank rate was not monthly, thus data pre-processing technique was used to convert it into monthly to compare it with house price time series.

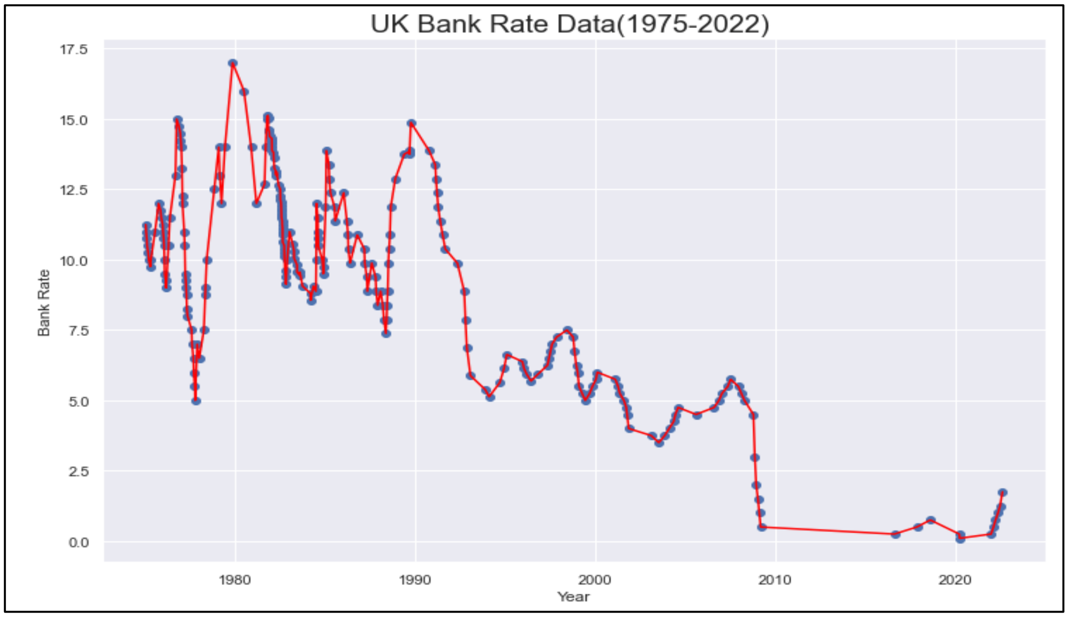


Figure 8: UK bank Rate (1975-2022)

After cleaning the dataset as per the requirements, stationarity, seasonality, and trend was checked, the time series was converted to stationary data with the help of adfuller test using statsmodels library, as the basic requirement for granger causality test is that the data should be stationary. Lag_plots were plotted and applied vector autoregressive model to get the max lag for the granger test. After, which granger test was applied to get the f-test score and p-value.

4.1.5 Hypothesis V: Outstanding Debt (Balance) vs House Prices

Gimeno, R. [49] researched to examine the interaction between mortgage loans and housing prices in Spain and found out that both are interdependent. Whereas Richard Disney [50] researched the relationship between household indebtedness and property prices and discovered that house price has little impact on the indebtedness. We take the hypothesis to check whether the outstanding debt(balance) has an impact on the house prices or not, whether increase in debt causes house prices to increase.

Data Description:

- Data for outstanding loans were fetched from fca.org

Table 6: Data Description for Outstanding Debt

Index	Data Source	Description	#Rows	#Columns
Outstanding Debt	FCA.org	data from 2007-2022	61	6
House Price Index (HPI)	Office of National Statistics	Monthly data from 1968-2022	650	2 (Date, price)

Python Libraries Used:

- Pandas
- Matplotlib
- Numpy
- Seaborn
- Statsmodel.tsa.stattools
- Pandas.plotting
- Statsmodel.tsa.api

Statistical Testing Used:

- Visualization technique
- Lag plots

- Granger Causality Test

Hypothesis:

- **Null Hypothesis:** Time series Loan data does not Granger-cause time series HP
- **Alternate Hypothesis:** Time series Loan data Granger-causes time series HP

Visualization techniques were followed to plot the time series such as line plots, histplots and some data cleaning techniques were followed to match the time series together, moreover, before passing the data into granger test, the series were converted to stationary using differencing and adfuller test. At last, the data was fed to the test to get the f-test and p-values.

4.1.6 Hypothesis VI: Stock Market vs House Price Index

Bissoondeeal [18] investigated the link between the housing market and the stock market and discovered that the two markets are favourably associated. Moreover, Kim Liow [51] examined the long- and short-term relationship between the real estate sector and stock markets. To examine the researchers, we took the hypothesis to check whether the stock markets impact the housing market or not.

Data Description:

- FTSE All-share Index was collected from yahoo finance from 2005-2021 with the interval of 1 month with the help of yfinance library in python.

Table 7: Data Description of Stock Market

Index	Data Source	Description	#Rows	#Columns
FTSE- All Share	Yahoo finance	data from 2005-2021	206	6

House Price Index (HPI)	Land Registry	Monthly data from 1968-2021	645	2

Python Libraries Used:

- Pandas
- yfinance
- Matplotlib
- Statsmodel.tsa.api
- Scipy
- seaborn

Statistical Testing Used:

- Visualization technique
- Linear Regression
- Cross Correlation Function

Data manipulation techniques were followed to match the starting and ending dates of the time series, also percentage change was calculated to check the changes in values over time. Graphs were plotted for both percentage change and original time series. After which, correlation was calculated between the percentage changes, Linear Regression lines were plotted, and cross correlation function was applied on the time series to check similarity between the series. Cross Correlation Function is measure of degree of similarity; it helps in telling whether one time series is helpful in forecasting another one.

4.2 Phase II: Data Collection

One of the most critical steps in doing research is data collection. Data gathering is a challenging activity that demands meticulous preparation, tenacity, and other qualities to be completed properly [30]. Data collection begins with defining what type of data is wanted, then selecting a sample from certain demographic.

API is an abbreviation for Application Programming Interface. APIs are methods that allow two components to interact with another by defining and enforcing rules and protocols. API architecture describes as Client and server, the program that sends request is Client, whereas the application that sends the answer is server [52].

For the property information and prices, we have used propertydata.com API. To use an API, we need to have an API Key and the URL (Uniform Resource Locator) to get data using the python library requests. Following is the API Key which was used to extract data from propertydata.co.uk

API KEY: ‘L1SGXGR1LQ’

To get the prices of houses which specific number of bedrooms, we need to have specific URL, which returns the mean of house prices from a given radius for a specified full UK postcode and selectable filters. Following is the input as listed in the documentation of API [53].

URL:

https://api.propertydata.co.uk/prices?key={API_KEY}
&postcode={POSTCODE}&bedrooms= {0-6}

Table 8: Input parameters for API to get bedroom prices

Key	API KEY	Required Field
Postcode	Full UK Postcode	Required Field
Bedrooms	Filter by number (0-5)	
Type	Filter by type	
Points	Number of points to analyze	Default 20, minimum 15, maximum 100

Using the Python Library, requests.get() a response(reply) from the server is got which contains all the required data as per input parameters.

To get the crime data of the location, we need to have specific URL, which returns the analytics on the local crime from a given radius for a specified full UK postcode [53]. Following is the input as listed in the documentation of API

URL :

https://api.propertydata.co.uk/crime?key={API_KEY}&postcode={POSTCODE}

Table 9: Input parameters for API to get crime data

KEY	API KEY
Postcode	Full UK Postcode

The response which we get from the request, contains crime ratings such as Very High crime, high crime, average crime, low crime, very low crime. The crime analytics contains the labels such as violence, shoplifting, drugs, public orders, burglary, thefts, robbery of the postcode that was supplied in the URL [53].

We have also used a development calculate in the interface which calculates the profit and profit percentage as per the input criteria. This development calculator combines the cost of construction with market data on prices per square footage and lets you knows the estimate of profit one can make.

URL:

https://api.propertydata.co.uk/development-
calculator?key={API_KEY}&postcode={POST_CODE}
&purchase_price={PURCHASE_PRICE}&sqft_pre_development={PRE_SQUARE
FOOTAGE}&sqft_post_development={POST_SQUARE_FOOTAGE}&project_type={P
ROJECT_TYPE}&finish_quality={FINISH_QUALITY}

Following are the input parameters :

Table 10: Input parameters of development calculator API

Key	API KEY
Postcode	Full UK postcode
Purchase_price	Initial purchase price of property
sqft_pre_development	Square footage of property before development
sqft_post_development	Square footage of property after development
project_type	Type of development (refurbish / demolition)
finish_quality	Finish quality (premium / medium / basic)

Another API is used in classification of the interest of the people, or to find the most visited venues by the people in the location specified. For which, I have used FOURSQUARE API. This API too requires an API key and URL to get the response from the server. To access the data, additional information is required for authentication such as client ID, client secret and version [54].

API KEY: 'fsq3lREtx31InF1Fo6N72vvhvlJsyukU2DtVPxkju8HMVGU='

CLIENT_ID: WXER4KXQTW0IHEGA0HLO2EMDOOWEZKAX4YFQLRDMZ2TSBRVM

CLIENT_SECRET: 4NMKXNOKJBU4FR312SHLECHR1WSY0ONMQP2JXUYOZ0TWKTJG

VERSION: 20210118

The URL for get response from the server is listed below:

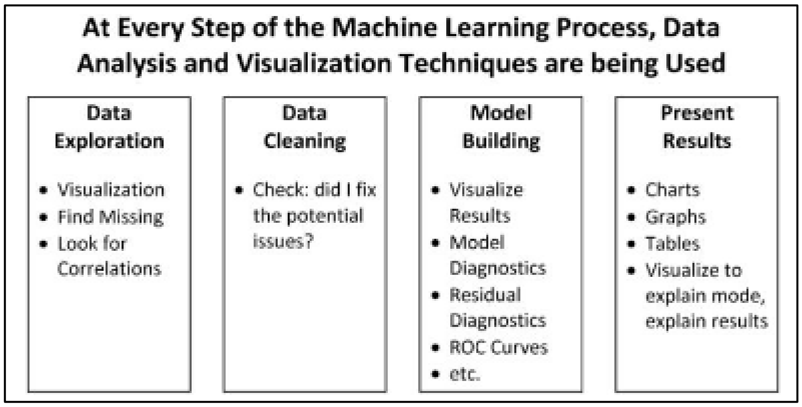
URL:
'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'

In the above listed URL, we need to provide the Client ID, Client Secret, Version, the coordinates of the location for which the search is done, radius of the location, and the limit of the results required. For the research purposes, I have set the limit to 1000 and radius as 500.

The rest of the data for the analysis and machine learning models were basically collected from Office of National Statistics, Land Registry.

4.3 Phase III: Data Analysis

Data must be examined to create a satisfactory outcome. Python, with simple instructions and syntax, provides a powerful open-source replacement to established methodologies and application. Exploratory Data Analysis (EDA) is a way of summarizing data by recognizing its key properties and visualizing it by means of proper illustrations [55]. EDA emphasizes on validating hypotheses necessary for prototype building and hypothesis validation, as well as addressing incorrect values and transforming inputs as required. Data patterns are shown using EDA tools like bar graphs, scatterplots, compute, and display associations among parameters via a heat map.



At Every Step of the Machine Learning Process, Data Analysis and Visualization Techniques are being Used

Data Exploration
- Visualization
- Find Missing
- Look for Correlations

Data Cleaning
- Check: did I fix the potential issues?

Model Building
- Visualize Results
- Model Diagnostics
- Residual Diagnostics
- ROC Curves
- etc.

Present Results
- Charts
- Graphs
- Tables
- Visualize to explain mode, explain results

Figure 9: EDA in machine learning process

4.3.1 EDA on Land Registry

EDA was carried out on the land registry data, which contained all the data of the cities of the United Kingdom with average house price, index, percentage change of the prices and indexes, sales volumes. The data was from 2004 to 2022 with 136934 rows and 54 columns. First, separate data frames were created bifurcating based on the regions, and with the help of seaborn library lineplots were created (see figure 10) for average house prices and average house price indexes.

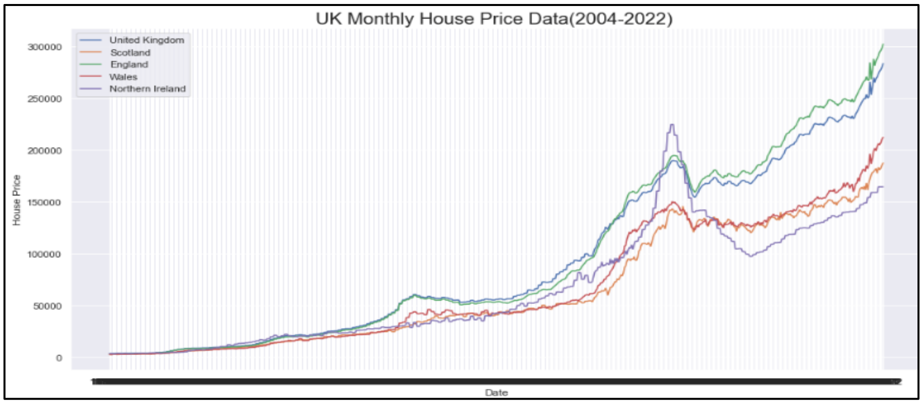


Figure 10: EDA of Land Registry data price

Moreover, the percentage change in house price index was plotted as per the region areas using the seaborn library lineplots.

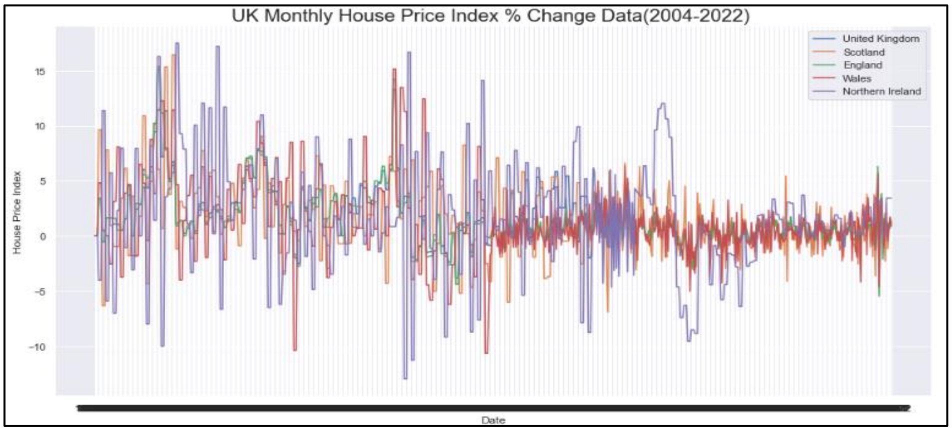


Figure 11: Percentage change in House price index EDA

The dataset contained sales volumes, thus average sales volume was plotted country wise with the help of seaborn library bar () (see fig 12)

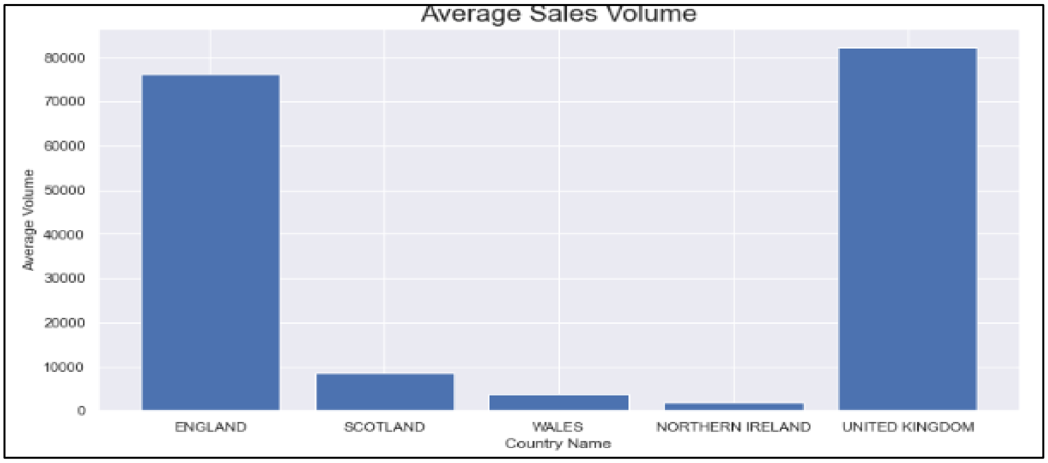


Figure 12: Average sales volume country wise EDA

Furthermore, geographical charting of home prices was done nation wise, for which geocoder library was utilized and prices were plotted with co-ordinates on the map of the United Kingdom using Folium library of Python (see figure 13) where green represented Scotland, Yellow represented North Ireland, Red represented England, and blue shows Wales.

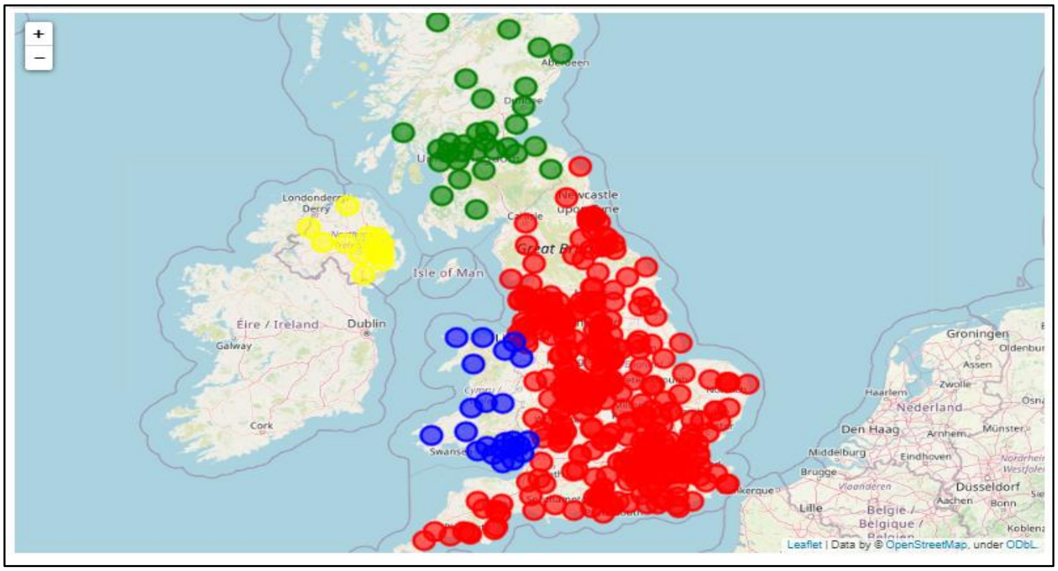

+
−
Aberdeen
Dundee
Londonderry
Derry
Newcastle upon Tyne
Isle of Man
Éire / Ireland
Dublin
Galway
Cork
Swansea
Groningen
Oldenbur
Assen
Zwolle
Haarlem
Nederland
Den Haag
Arnhem
Münster
Middelburg
Brugge
Eindhoven
Vlaanderen
Düsseldorf
Aachen
Bonn
Lille
België / Belgique / Belgien
Koblenz
Leaflet | Data by © OpenStreetMap, under ODbL.

Figure 13: Geographical mapping of prices country wise

4.3.2 EDA on data acquired through API

Data acquired through API was also subjected to EDA. Propertydata.co.uk API was used to obtain data for all areas in the United Kingdom, including the northeast, northwest, east midlands, and others. A For Loop was used to pass the values of the region list in the URL, and the data was recorded in a dictionary of data frames. Visualization was performed on the data such as bar charts and geographical plots.

Region List created:

[north_east','north_west','east_midlands','west_midlands','east_of_england','greater_london','south_east', 'south_west','wales','scotland','northern_ireland']

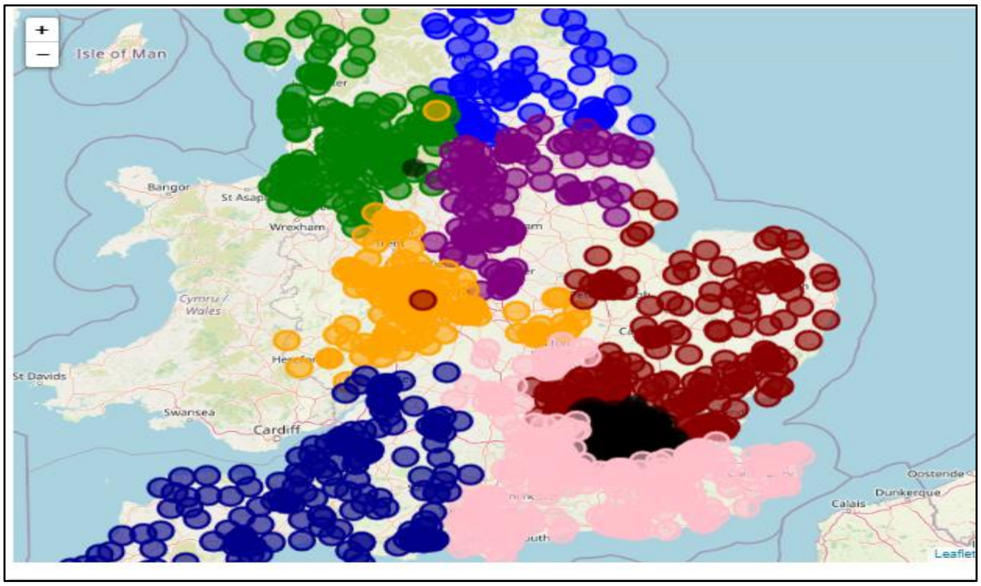

+
–
Isle of Man
Bangor
St Asaph
Wrexham
Cymru / Wales
St Davids
Swansea
Cardiff
Oostende
Dunkerque
Calais
Leaflet

Figure 14: geographical maps with prices of regions of England

Figure 14 represented the all the regions of England with the prices of the cities, below is the legend of the colours represented by regions in above mentioned figure.

Table 11 : Colour coding of the regions on Geographical map

Color	Region Name
Blue	Northeast
Green	Northwest
Purple	East Midlands
Orange	West Midlands
Dark Red	East of England
Light Red	Greater London
Pink	Southeast
Dark Blue	Southwest

Same as the above figure, for Wales, Scotland and Northern Ireland cities with prices were plotted to have a quick view of the locations of the cities and prices on a geographical map.

Furthermore, bar plots were visualized for average house prices region wise (see fig)

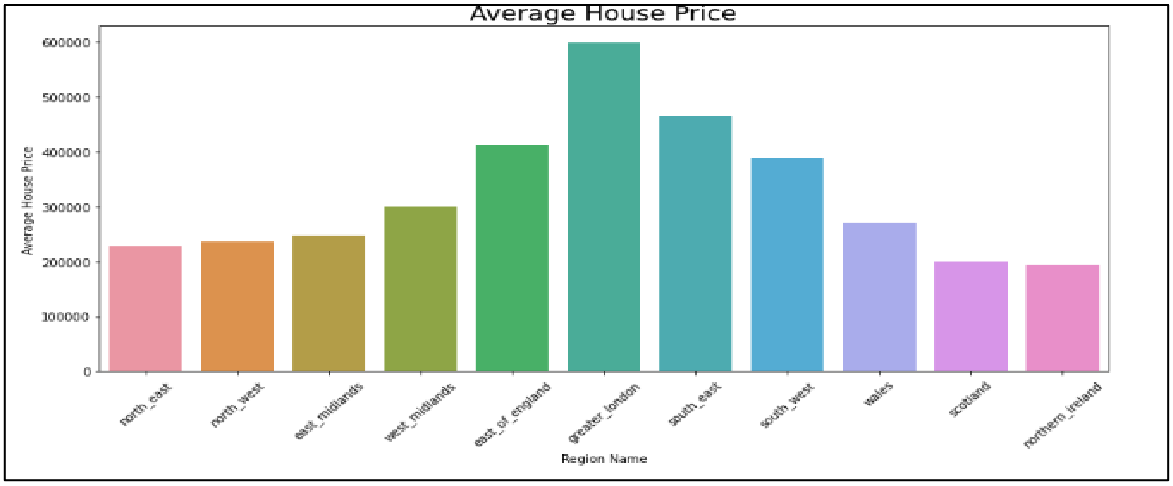


Figure 15: Region wise bar plots of average house prices

Average growth in the previous year was plotted and insights were found (see fig 16)

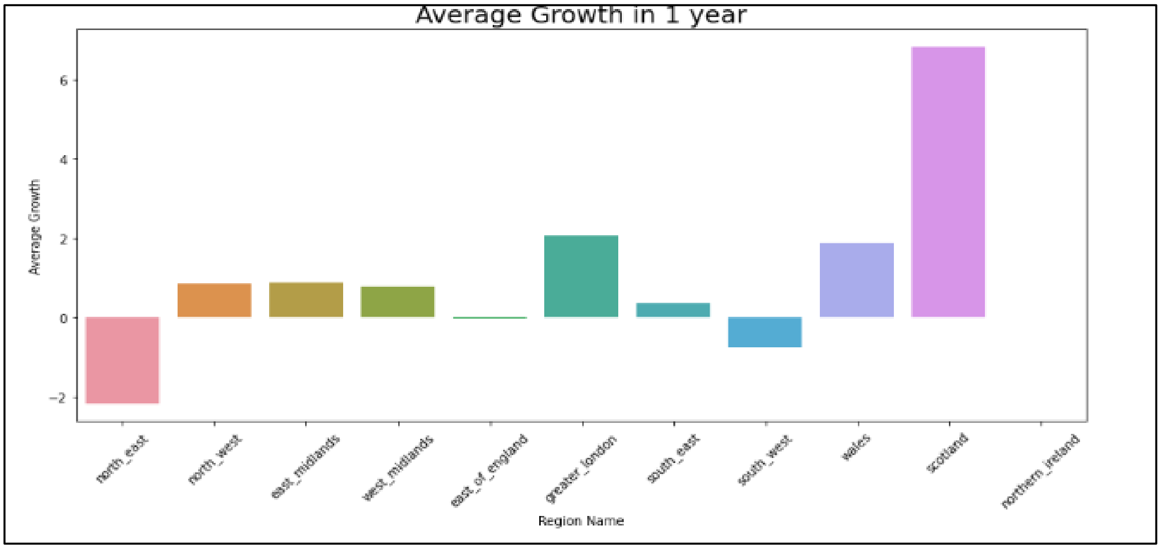


Figure 16: Average growth in 1 year region wise

4.3.3 Comparison between different indices

Since, the research is based on predicting property prices, house price indices are among the most carefully monitored indicators. In United Kingdom, there are many significant house price indices, these indexes vary in what is quantified as well as the data. We concentrate on the four indicators [56] since they all strive to gauge house prices utilizing real market data and seem to be reasonably long-lasting, allowing to assess variances in movements in the trend over a prolonged duration:

- Halifax Indices
- Nationwide Building society indices
- An index by LSL property services/ Acadata (LSL Acad) based on land surveyors
- Land Registry price index.

Dataset Description:

Table 12: Data Description of different indexes

Index	Source	Data Description	# Rows	# Columns
Halifax HPI	Halifax	Monthly data from 2019-2022	42	3
Nationwide HPI	Nationwide Building Society	Monthly data from 1991-2022	379	7
Land Surveyors	LSL Acad E&W	Monthly data from 2005-2022	208	3
UK HPI	Land Registry UK	Monthly data from 1968-2022	650	3

- Halifax HPI Index Max is 508.4 and Min is 397.8 with a standard deviation of 32.052 and mean as 434.53

- Nationwide HPI Index Max is 541.83 and Min is 98.95 with a standard deviation of 127.83 and mean as 277.50
- Land Surveyors HPI Index Max is 356.24 and Min is 194.62 with a standard deviation of 40.07 and mean as 255.69
- Land Registry HPI Index Max is 1.88 and Min is 148.68 with a standard deviation of 41.73 and mean as 48.61

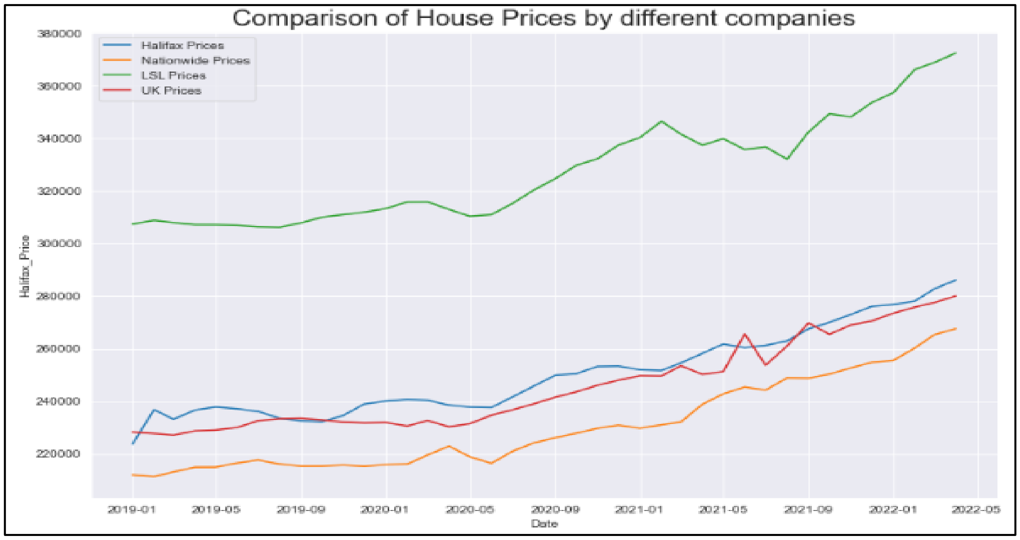


Figure 17: Plotting of different indexes

Following data cleaning and pre-processing techniques such as equalizing the length of time series for comparison, removing null values, and extracting a subset of the dataset based on region name, visualization techniques such as plotting of line charts (see figure 17) and bar charts (see figure 18) were used to compare the different indexes. Moreover, percentage change was calculated and plotted (see figure 19).

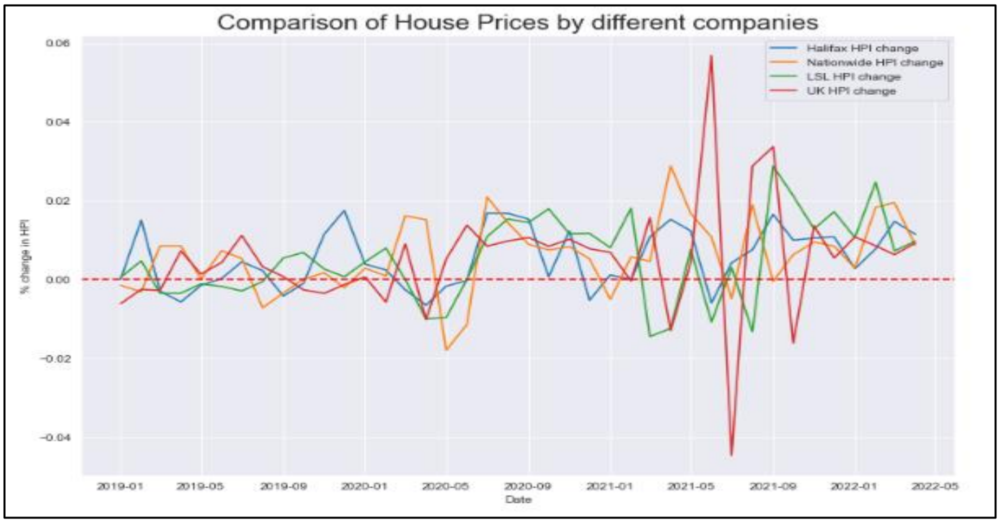


Figure 18: Comparison of change in price by different indexes

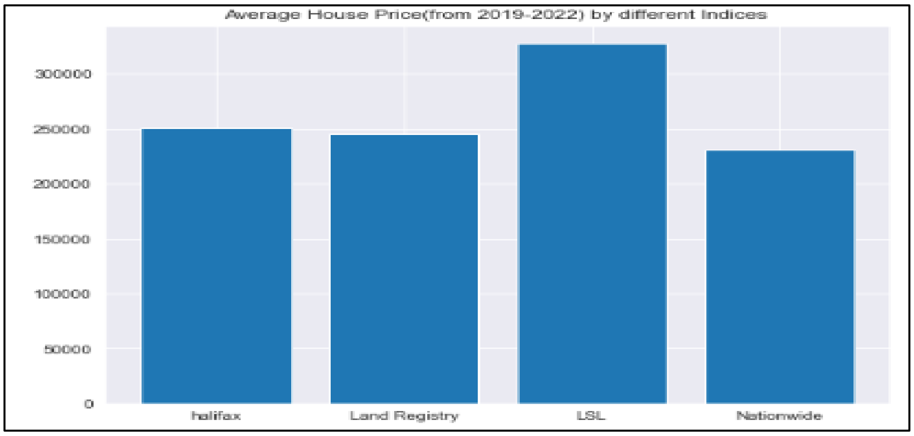


Figure 19: Bar chart of prices by different indexes

4.4 Phase IV: Model development and deployment

Machine Learning development phase consists of various processes, such as identifying source data, creating models, deploying them and maintenance. The activities are divided into two major parts: Model development and Model operations [57]. The first part consists of steps including building of model, hyper-tunning and choosing the model for deployment (see fig 20)

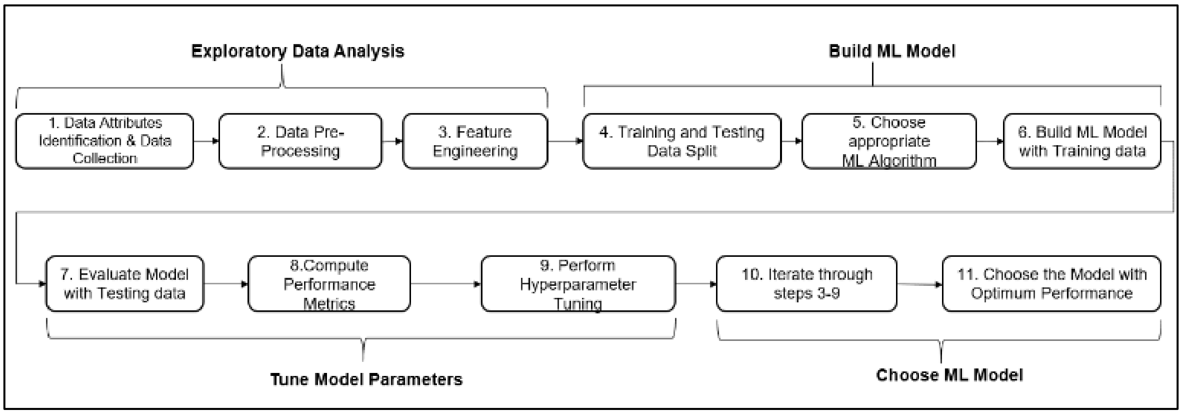

Exploratory Data Analysis
Build ML Model
1. Data Attributes Identification & Data Collection
2. Data Pre-Processing
3. Feature Engineering
4. Training and Testing Data Split
5. Choose appropriate ML Algorithm
6. Build ML Model with Training data
7. Evaluate Model with Testing data
8.Compute Performance Metrics
9. Perform Hyperparameter Tuning
10. Iterate through steps 3-9
11. Choose the Model with Optimum Performance
Tune Model Parameters
Choose ML Model

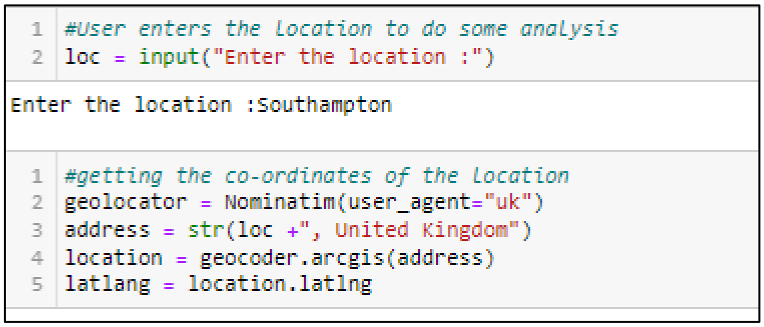
Figure 20: Machine learning model development cycle

We primarily used three models for research purpose to be deployed on the GUI. Firstly, an unsupervised learning model (KMeans Clustering) for clustering the most common venues for the location, a sequential neural network model for predicting price index, and one of the supervised learning model random forest or XGBoost for prediction of house prices.

4.4.1 Unsupervised Learning Model (K-Means Clustering)

J.B. MacQueen suggested the KMeans technique as a type of cluster algorithm. This approach is commonly used in pattern detection, this method attempts to identify K divisions that fulfil a specific requirement to achieve the best possible result [58]. The KMeans method has the advantages of being concise, efficient, and fast.

To cluster the most common venues in a location, we need to first have a location and find its co-ordinates using geocoder library of python. (See figure 21)



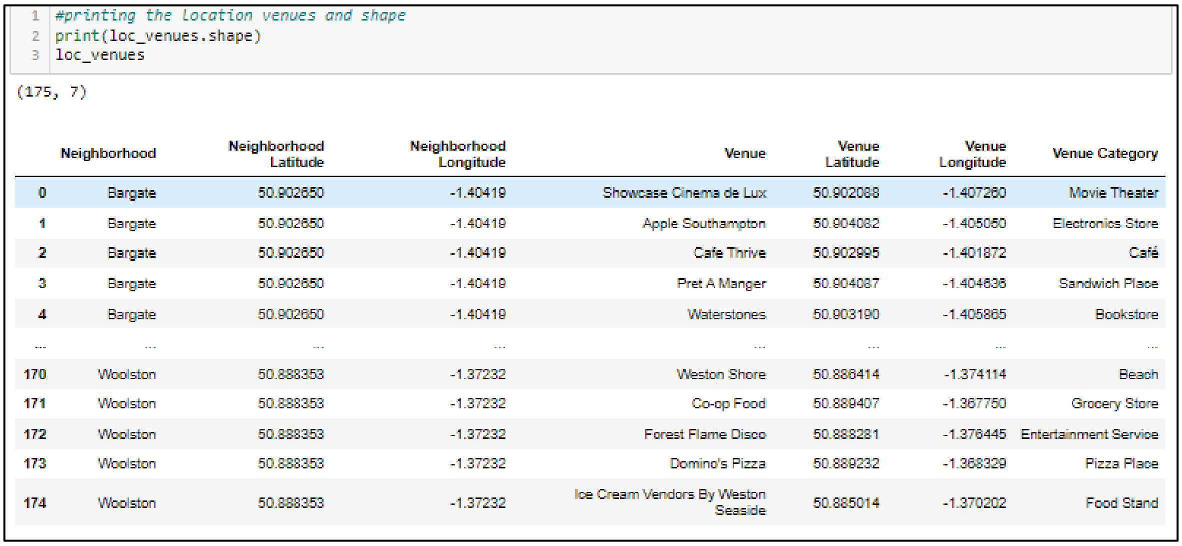
```
#User enters the location to do some analysis
loc = input("Enter the location :")
```

```
Enter the location :Southampton
```

```
#getting the co-ordinates of the location
geolocator = Nominatim(user_agent="uk")
address = str(loc +", United Kingdom")
location = geocoder.arcgis(address)
latlang = location.latlng
```

Figure 21: Code for finding co-ordinates

The neighbourhoods of the specified location are then matched, and the same method of obtaining co-ordinates is followed. Using the Foursquare API, as explained previously in the data collection step, we then send the neighbourhoods co-ordinates to receive the most frequently visited venues in that specific region which is stored in a dataframe for further processing.(see figure 22).



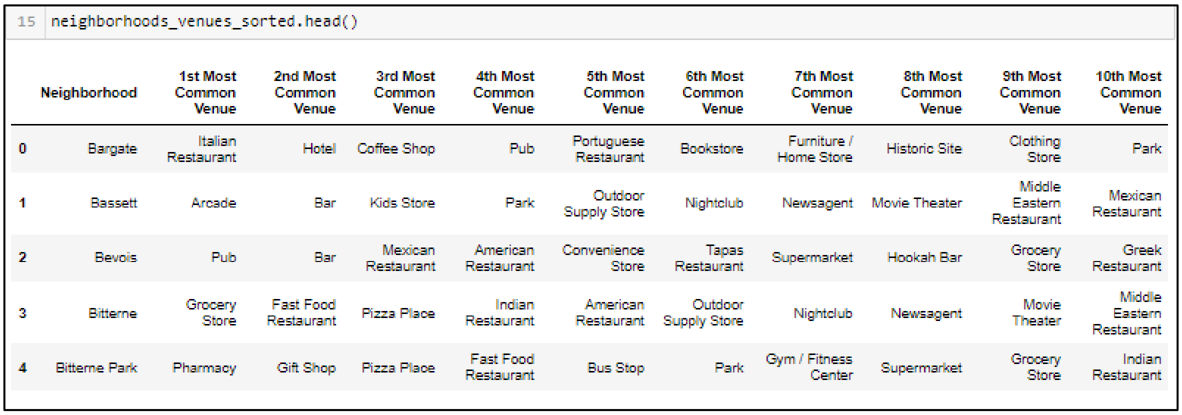
```
#printing the location venues and shape
print(loc_venues.shape)
loc_venues
```

```
(175, 7)
```

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Bargate	50.902650	-1.40419	Showcase Cinema de Lux	50.902088	-1.407260	Movie Theater
1	Bargate	50.902650	-1.40419	Apple Southampton	50.904082	-1.405050	Electronics Store
2	Bargate	50.902650	-1.40419	Cafe Thrive	50.902995	-1.401872	Café
3	Bargate	50.902650	-1.40419	Pret A Manger	50.904087	-1.404636	Sandwich Place
4	Bargate	50.902650	-1.40419	Waterstones	50.903190	-1.405865	Bookstore
...	...	...	...	...	...	...	...
170	Woolston	50.888353	-1.37232	Weston Shore	50.886414	-1.374114	Beach
171	Woolston	50.888353	-1.37232	Co-op Food	50.889407	-1.367750	Grocery Store
172	Woolston	50.888353	-1.37232	Forest Flame Disco	50.888281	-1.376445	Entertainment Service
173	Woolston	50.888353	-1.37232	Domino's Pizza	50.889232	-1.368329	Pizza Place
174	Woolston	50.888353	-1.37232	Ice Cream Vendors By Weston Seaside	50.885014	-1.370202	Food Stand

Figure 22: Dataframe of neighbourhood venues

The dataframe is then grouped and sorted to get all the venues as per the neighbourhood(see fig 23)



```
neighborhoods_venues_sorted.head()
```

	Neighborhood	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th Most Common Venue	10th Most Common Venue
0	Bargate	Italian Restaurant	Hotel	Coffee Shop	Pub	Portuguese Restaurant	Bookstore	Furniture / Home Store	Historic Site	Clothing Store	Park
1	Bassett	Arcade	Bar	Kids Store	Park	Outdoor Supply Store	Nightclub	Newsagent	Movie Theater	Middle Eastern Restaurant	Mexican Restaurant
2	Bevois	Pub	Bar	Mexican Restaurant	American Restaurant	Convenience Store	Tapas Restaurant	Supermarket	Hookah Bar	Grocery Store	Greek Restaurant
3	Bitterne	Grocery Store	Fast Food Restaurant	Pizza Place	Indian Restaurant	American Restaurant	Outdoor Supply Store	Nightclub	Newsagent	Movie Theater	Middle Eastern Restaurant
4	Bitterne Park	Pharmacy	Gift Shop	Pizza Place	Fast Food Restaurant	Bus Stop	Park	Gym / Fitness Center	Supermarket	Grocery Store	Indian Restaurant

Figure 23: Most common venues dataframe

Since the data required for any machine learning algorithm must be numerical, the dataframe is converted into numerical data using the technique called one hot encoding. Furthermore, for KMeans, the basic requirement is that the number of clusters must be supplied. The Elbow Method is one of the most prominent approaches for determining the ideal value of K, where K is the cluster number. We have used Elbow method to find value of K (see figure 24)

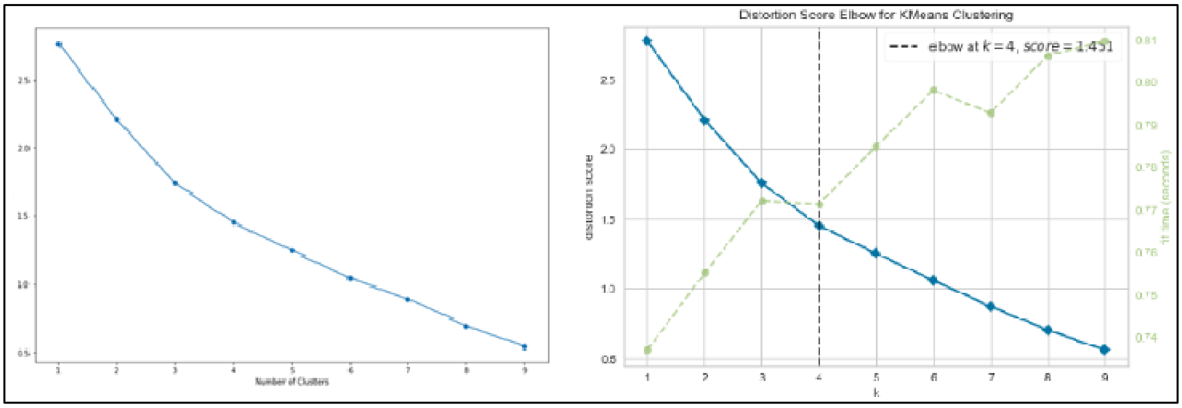


Figure 24: Elbow Method and distortion score to find K

We tried the Elbow method for values of K ranging from 1 to 10, and we could plainly see that the distortion score was 4, thus we used value of K as 4 for the KMeans method. KMeans clustering was used on the dummy dataset created using one hot encoding, and the cluster labels were joined with neighbourhood venues that had the most common venues list. A geographical map was then plotted with the clusters using folium map. (See figure 25)

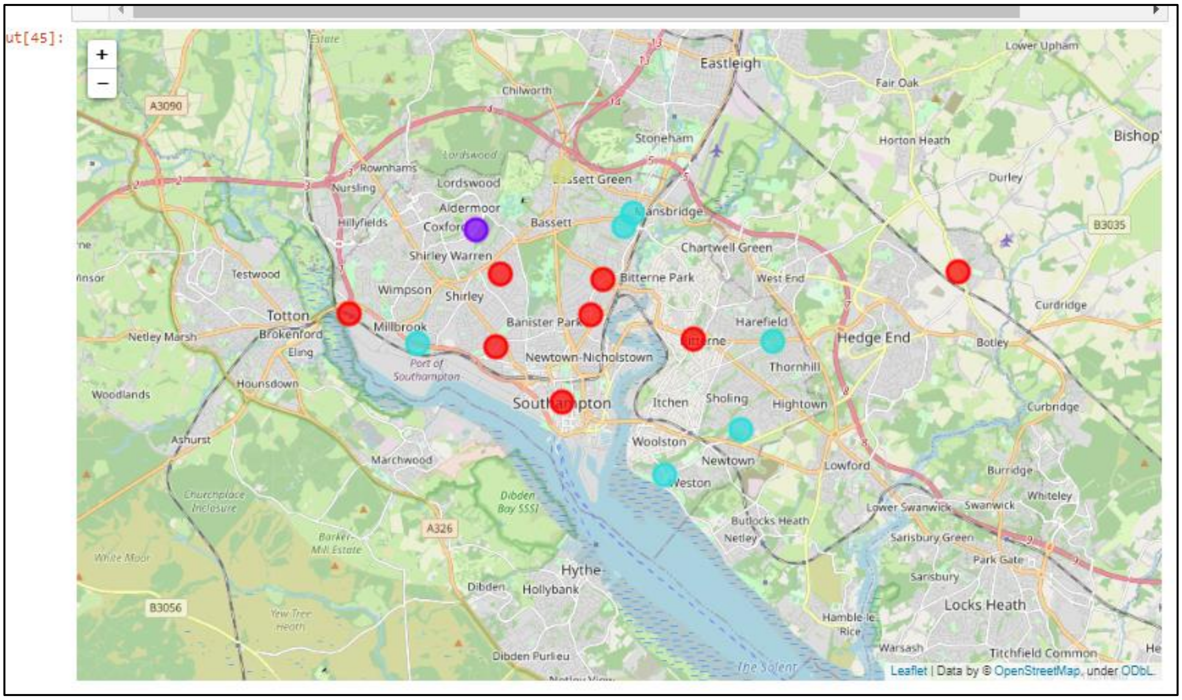

ut[45]:
+
−
Eastleigh
Chilworth
Stoneham
Fair Oak
Lower Upham
Horton Heath
Bishop
Durley
A3090
Rownhams
Nursling
Lordswood
Aldermoor
Coxford
Bassett
Bassett Green
Swaythling
Mansbridge
Hillyfields
Shirley Warren
Chartwell Green
Bitterne Park
West End
B3035
Testwood
Wimpson
Shirley
Totton
Millbrook
Banister Park
Harefield
Hedge End
Curdridge
Botley
Netley Marsh
Brokenford
Eling
Port of Southampton
Newtown-Nicholstown
Bitterne
Thornhill
Woodlands
Hounsdown
Southampton
Itchen
Sholing
Hightown
Curbridge
Ashurst
Woolston
Newtown
Lowford
Marchwood
Weston
Burridge
Whiteley
Churchplace Inclosure
Dibden Bay SSSI
Bursledon
Lower Swanwick
Swanwick
A326
Bullocks Heath
Netley
Sarisbury Green
Barker Mill Estate
Park Gate
White Moor
Hythe
Sarisbury
Dibden
Hollybank
Locks Heath
B3056
Yew Tree Heath
Hamble-le-Rice
Dibden Purlieu
Warsash
Titchfield Common
The Solent
Leaflet | Data by © OpenStreetMap, under ODbL.

Figure 25: Folium map with clustering

4.4.2 Prediction of House Price Index with Sequential Neural Networks

The purpose of this phase of deployment was to forecast the house price index for the next months. Several researchers have utilized neural networks to anticipate housing values, including Wang in 2021 [59] who employed a back propagation neural network to predict prices, and Varma in 2018 [60].

The dataset which was used for the modelling was published by the Office of National Statistics which included monthly indices from 1986 to 2022, after which EDA was conducted such as plotting a line chart using Seaborn Library (see figure 26)

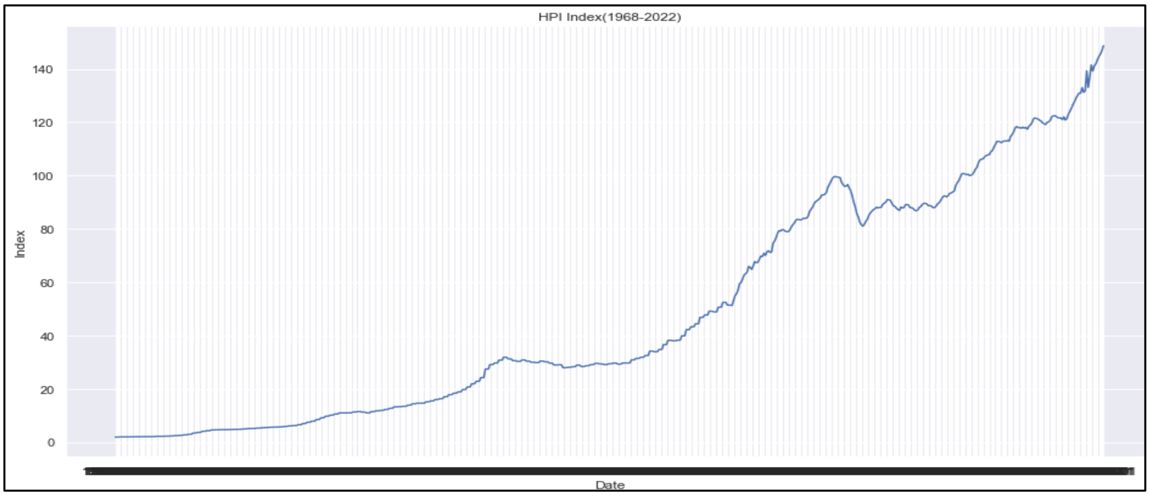


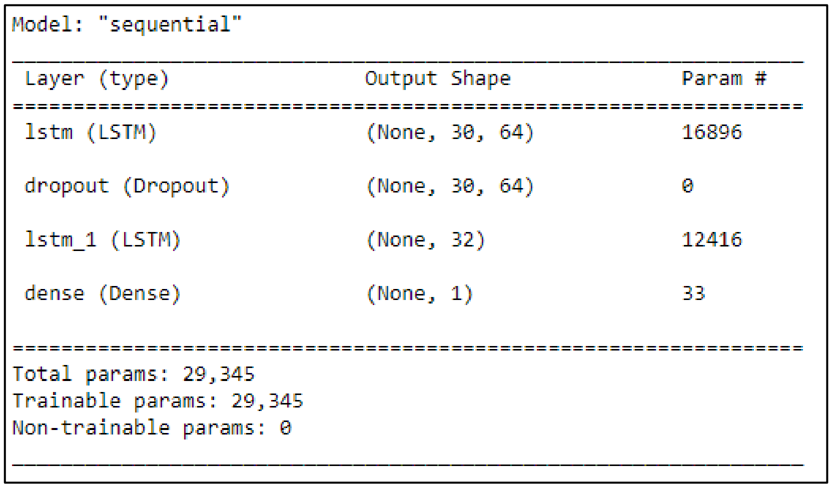
Figure 26: UK HPI line chart

Furthermore, the dataset was standardized as part of the feature scaling approach since data scaling is a suggested pre-processing step when dealing with algorithms. The MinMaxScalar library was used to scale down the data values. MinMaxScalar scales the value between 0 and 1. Following that, the dataset was scaled using the fit_transform() function,.

Before feeding the model with data, the dataset was split into 65:35 ratio using train and test split. Dividing the datasets is required for an impartial assessment of prediction accuracy. The train test split is used to evaluate the effectiveness of algorithms for machine learning suitable for forecast based models. This approach is a quick and simple method that enables us to evaluate our own machine learning output to machine outputs.

A data matrix was built from the dataset with data append using a time step of 30 since the past 30 observations would be used to forecast the future observation. As a result, the previous 30 observations become independent data, while the 31st observation becomes dependent data, this is how the entire dataset was appended to a data matrix which would be fed to the neural network.

The data was then reshaped for model using reshape function. A sequential neural network model was built with four layers (see figure 27),



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            16896

 dropout (Dropout)           (None, 30, 64)            0

 lstm_1 (LSTM)               (None, 32)                12416

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 29,345
Trainable params: 29,345
Non-trainable params: 0
_________________________________________________________________
```

Figure 27: Architecture of Neural Network

The first layer is a dense layer with 64 neurons, while the second layer is a dropout layer that randomly changes the input to 0, preventing overfitting. The Third layer is another dense layer with 32 neurons, followed by a dropout layer. The data is fed to model with 100 epochs and the validation data parameter was set to test data, while the batch size is set to 64. After which the data is predicted using model.predict() and the next 9 months of the values were predicted.

4.4.3 Machine Learning models for House Price Prediction

For predictions, Limsobum Chai in 2004 [23] utilized layered neural networks, Takats in 2021 [14] used regression models, while Truong in 2019 [19] and Jha in 2020 [16] used diverse models such as RandomForest and XGBoost. For housing price projections, I have also used machine learning techniques like RandomForest, XGBoost and sequential Neural Network.

The dataset used in this phase was obtained from the gov.uk website and includes the price paid transactions for 2022 for England and Wales. The dataset had around 231624 rows and 18 columns. Web scraping was performed from opendatacommunities.org to obtain the energy ratings, construction age bands, total floor area, and energy tariff of the properties listed in the above-mentioned dataset via API calls using a for loop was saved in a dataframe that was joined with the original dataset.

The new dataset underwent data cleaning because it had many null values, for which duplicate values were eliminated, nan values were replaced with unknown, and outliers were visualized with the help of box plots and thus removed. EDA was conducted on the cleaned dataset using the seaborn library, which included plots of histograms, box plots, joint plots. In addition, a new column of city population rankings was added to the dataset.

To examine feature correlation, heatmaps were plotted for numerical columns, whereas point biserial relationships were used for categorical variables. The connection between a binary and continuous variable is measured by point biserial, which gives two outputs correlation factor and p-value [61]. -1 indicates negative correlation while 1 indicates positive and zero shows no correlation. Furthermore, ANOVA tests were used to determine the relationship between categorical and continuous variables. It is a statistical test used to determine how much one variable influences the other.

Before feeding the data into the model, the categorical variables need to be transformed into numerical values, which was accomplished using dummy variables. The data was then scaled using the MinMaxScaler library and split using an 80:20 split. After which machine learning models were used.

RandomForest Regressor

Truong in 2019 [19] and Jha in 2020 [16] used RandomForest and XGBoost models for prediction of house prices. RandomForest is an ensemble approach that can handle both regression and classification problems by combing many decisions tress using a method called as bootstrap and aggregation. The core concept is to use numerous decision tress to arrive at an outcome rather than depending on independent trees.

The RandomForest Regressor was used, and its accuracy was tested, hyper parameter tunning operation was carried out with the assistance of HalvingRandomSearchCV. This technique searches for given parameters by halving them one by one. The parameters that were passed to obtain the optimal parameters are as follows:

Table 13: Hyper parameters for RandomForest

Parameter	Values
Bootstrap	True
Max_depth	110,100,90,80
Max_features	3,2
Min_samples_leaf	5,4,3
Min_samples_split	12,10,8
N_estimators	1000,300,200,100

XGBoost Regressor

XGBoost is a prevalent and effective open-source gradient boosted trees algorithm. It is supervised learning method which combines the estimations of a group of smaller, weaker model to attempt to forecast target [64]. The weak learners are regression tress and each tree link to another which carries continuous score.

Many studies utilise the algorithm to estimate house prices, in this instance, we employed the algorithm to determine accuracy after which hypertunning of parameters was carried out using same Cross validation approach, HalvingRandomSearchCV. The parameters that were input to the technique are as follows:

Table 14: Hyper parameters for XGBoost

Parameters	Values
N_estimators	1500, 1100, 900, 500, 100
Max_depth	15,10,5,3,2
Learning_rate	0.20,0.15,0.1,0.05
Min_child_weight	4,3,2,1

Sequential Neural Network

After running random forest and XGBoost on the data, it was time to run sequential neural network on it. However, to determine the parameters and number of layers, the Keras tuner library was used in conjunction with the RandomSearch Function, which provided the summary of the findings. The following are the best parameters from the results summary:

Table 15: Hyperparameters for Sequential Neural Network

Number of Layers	Number of Units
Layer 1	224
Layer 2	160
Layer 3	64
Layer 4	64
Layer 5	96
Layer 6	128

Layer 7	128
Layer 8	192
Layer 9	32
Learning Rate	0.01

The loss was then computed and plotted after the model was built and data was fed with number of epochs set to 100 and the validation data parameter set to testing data.

4.4.4 Graphical User Interface

A Graphical User Interface (GUI) is an interactive graphic component system for computer applications. A GUI displays elements which connects information and specify movements that a user does, When the person interacts with things, they differ in hue, magnitude, or visibility [62]. A GUI contains buttons, icons, texts.

For GUI we have used Tkinter, there are several GUI frameworks in python, however Tkinter is the only one that is incorporated into python standard library. It is cross-platform and easy to code [63].

The GUI framework is made up of three distinct frames. First, Interest City search, then know the price frame, and last research (see figure 28)

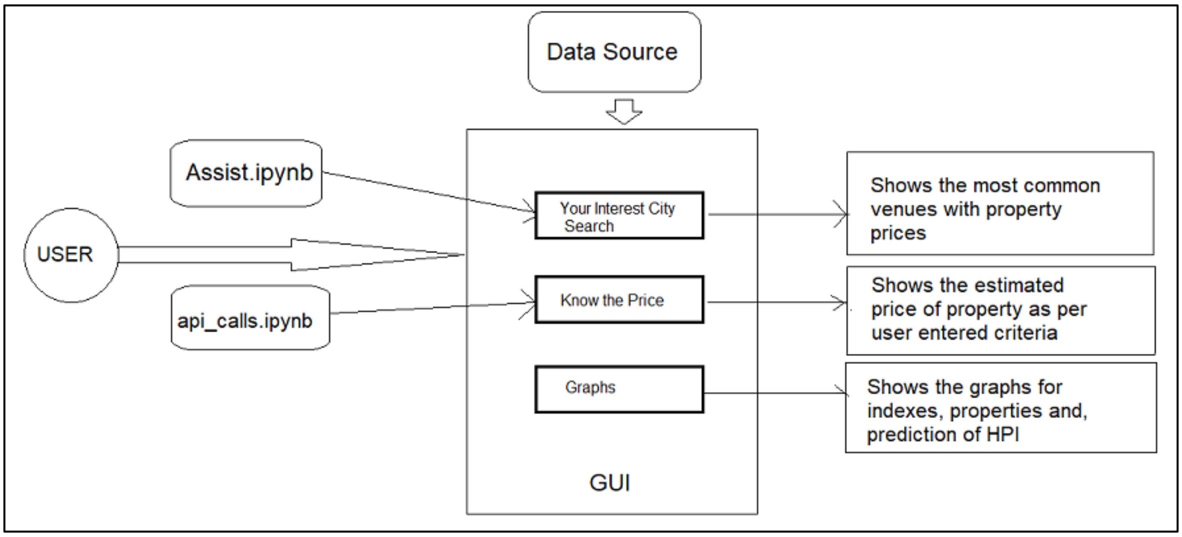
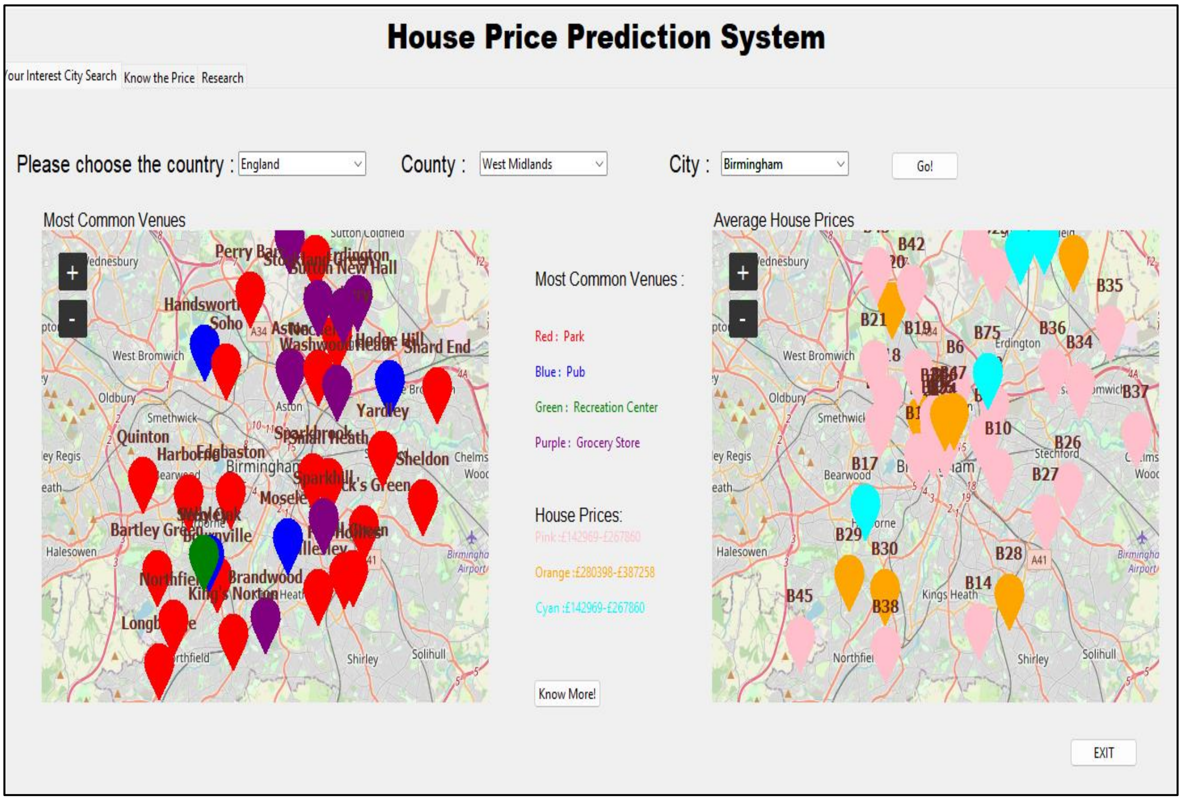

Data Source
Assist.ipynb
USER
api_calls.ipynb
Your Interest City Search
Know the Price
Graphs
GUI
Shows the most common venues with property prices
Shows the estimated price of property as per user entered criteria
Shows the graphs for indexes, properties and, prediction of HPI

Figure 28: Architecture of GUI

Frame I: Know your City Search

The First frame is all about the city that the user is interested in; the user just picks the appropriate country, county, and city. The city can only be selected after selecting country and county from the dropdown option available. After clicking on the go button, the GUI presents two graphs. The First graph shows the most prevalent venues clustered by colours, while the second graphs show the pricing of properties in that selected city clustered by price as high, medium, and low. (See figure 29). When a user presses the go button, the GUI calls the assist.ipynb file, which performs all the operations of the KMeans algorithm in the background and provides a dataframe with the clustered venues, the GUI then shows the venues in the form of a map using the Tkintermapview function, while the second graph is obtained by the dataframe containing the property prices filtered by the chosen city name.


House Price Prediction System

Please choose the country : England

County : West Midlands

City : Birmingham

Most Common Venues :

Red : Park

Blue : Pub

Green : Recreation Center

Purple : Grocery Store

House Prices:

Pink :£142969-£267860

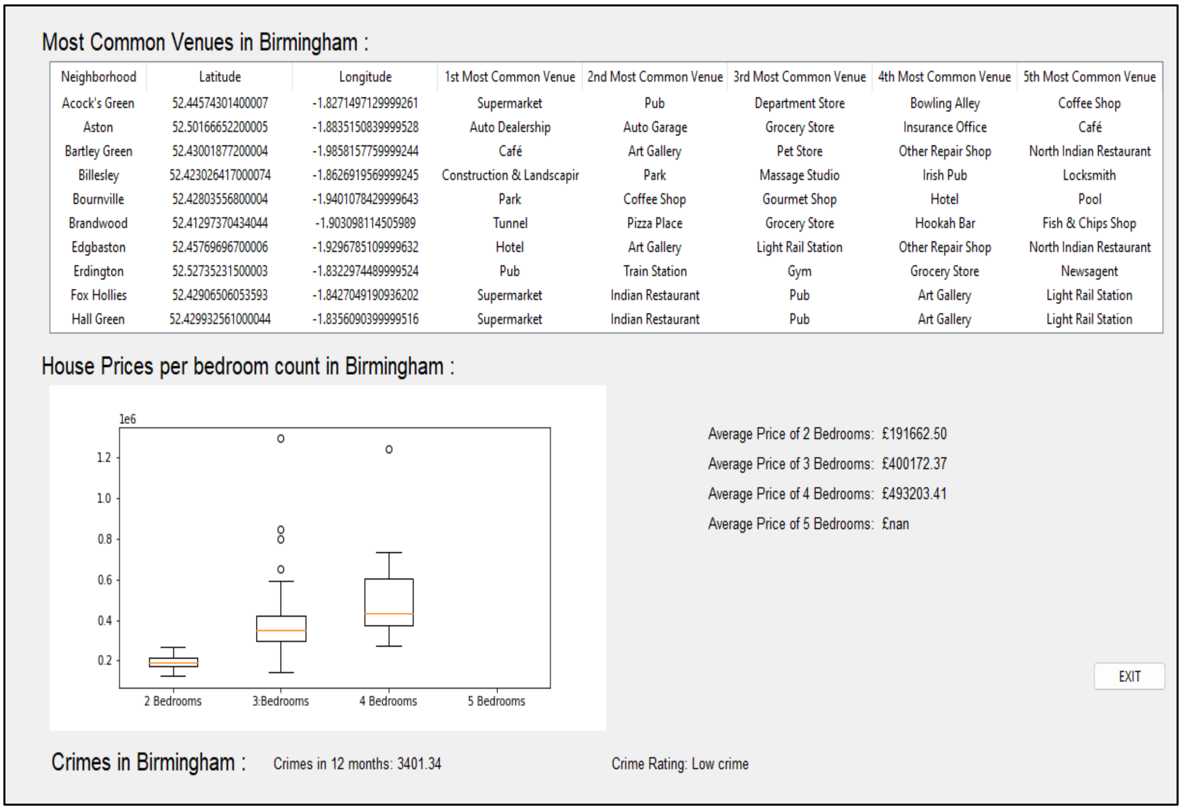
Orange :£280398-£387258

Cyan :£142969-£267860

'our Interest City Search
Know the Price
Research
Go!
Most Common Venues
Average House Prices
Know More!
EXIT

Figure 29: Frame 1 of GUI

Furthermore, when the user clicks on the know more button, all the top 10 venues in the selected city's neighbourhoods are displayed, as well as a box plot of property prices with bedroom count. Furthermore, the city's crime data is displayed as the number of crimes in the last 12 months and the city's crime rating (see figure 30). The working behind this frame is that a new window is displayed, the dataframe that is fetched from assist.ipynb is displayed , a box plot is created with the filtered dataframe that contained house prices and crime data analytics is fetched by an API call from propertydata.co.uk


Most Common Venues in Birmingham :

Neighborhood	Latitude	Longitude	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
Acock's Green	52.44574301400007	-1.8271497129999261	Supermarket	Pub	Department Store	Bowling Alley	Coffee Shop
Aston	52.50166652200005	-1.8835150839999528	Auto Dealership	Auto Garage	Grocery Store	Insurance Office	Café
Bartley Green	52.43001877200004	-1.9858157759999244	Café	Art Gallery	Pet Store	Other Repair Shop	North Indian Restaurant
Billesley	52.423026417000074	-1.8626919569999245	Construction & Landscapir	Park	Massage Studio	Irish Pub	Locksmith
Bournville	52.42803556800004	-1.9401078429999643	Park	Coffee Shop	Gourmet Shop	Hotel	Pool
Brandwood	52.41297370434044	-1.903098114505989	Tunnel	Pizza Place	Grocery Store	Hookah Bar	Fish & Chips Shop
Edgbaston	52.45769696700006	-1.9296785109999632	Hotel	Art Gallery	Light Rail Station	Other Repair Shop	North Indian Restaurant
Erdington	52.52735231500003	-1.8322974489999524	Pub	Train Station	Gym	Grocery Store	Newsagent
Fox Hollies	52.42906506053593	-1.8427049190936202	Supermarket	Indian Restaurant	Pub	Art Gallery	Light Rail Station
Hall Green	52.429932561000044	-1.8356090399999516	Supermarket	Indian Restaurant	Pub	Art Gallery	Light Rail Station

House Prices per bedroom count in Birmingham :

Average Price of 2 Bedrooms: £191662.50
Average Price of 3 Bedrooms: £400172.37
Average Price of 4 Bedrooms: £493203.41
Average Price of 5 Bedrooms: £nan

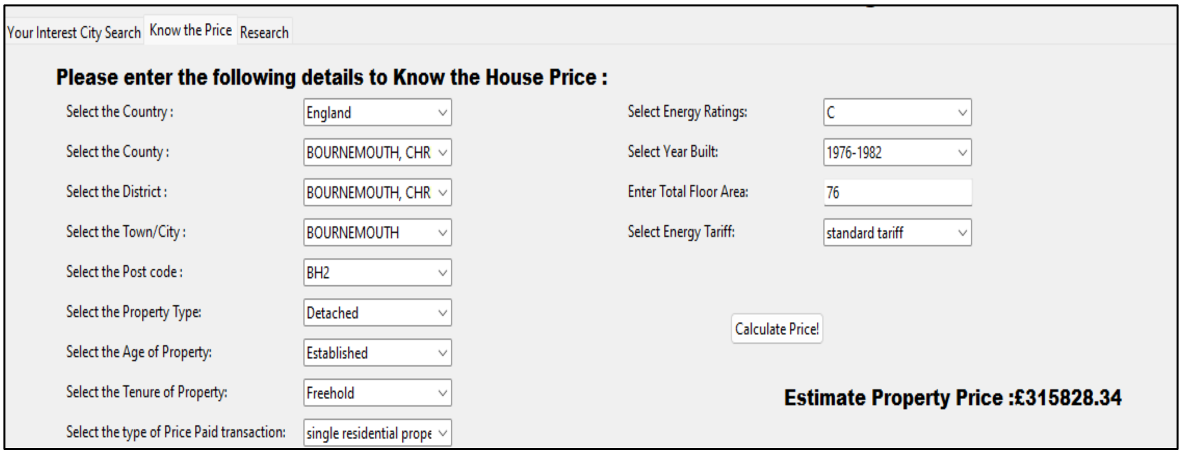
Crimes in Birmingham : Crimes in 12 months: 3401.34

Crime Rating: Low crime

Figure 30: GUI Frame 1 know more button display

Frame II: Know the Price

The First part of the frame talks about the estimated property prices based on the user selected criteria (see figure 31), while the second part of the frame is a development calculator that shows real time market prices combined with construction cost and shows the profit an investment could make (see figure 32). The working behind this frame is that when the user enters all the information, the machine learning model that is being loaded in the GUI predicts the prices, while the second portion of the frame accesses an API and displays the expected profit percentage.

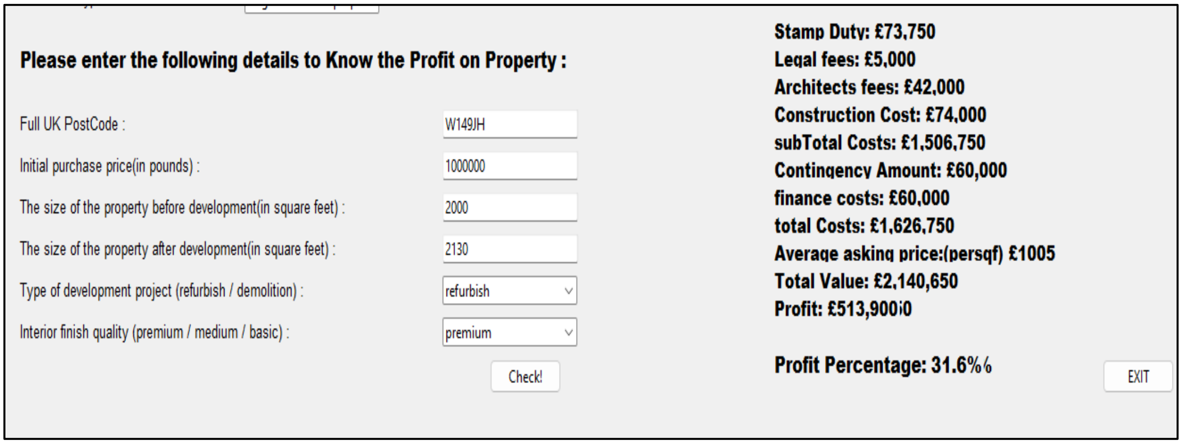


Please enter the following details to Know the House Price :

Select the Country : England
Select the County : BOURNEMOUTH, CHR
Select the District : BOURNEMOUTH, CHR
Select the Town/City : BOURNEMOUTH
Select the Post code : BH2
Select the Property Type: Detached
Select the Age of Property: Established
Select the Tenure of Property: Freehold
Select the type of Price Paid transaction: single residential prope
Select Energy Ratings: C
Select Year Built: 1976-1982
Enter Total Floor Area: 76
Select Energy Tariff: standard tariff

Estimate Property Price :£315828.34

Figure 31: Frame 2 part 1 of GUI



Please enter the following details to Know the Profit on Property :

Full UK PostCode : W149JH
Initial purchase price(in pounds) : 1000000
The size of the property before development(in square feet) : 2000
The size of the property after development(in square feet) : 2130
Type of development project (refurbish / demolition) : refurbish
Interior finish quality (premium / medium / basic) : premium

Stamp Duty: £73,750
Legal fees: £5,000
Architects fees: £42,000
Construction Cost: £74,000
subTotal Costs: £1,506,750
Contingency Amount: £60,000
finance costs: £60,000
total Costs: £1,626,750
Average asking price:(persqf) £1005
Total Value: £2,140,650
Profit: £513,900¡0

Profit Percentage: 31.6%%

Figure 32: Frame 2 part 2 of GUI

Frame III: Research

The Third Frame displays all the graphs that the user may view and analyse; it displays four graphs, the first about future house price index prediction, the second displays line chart about the average price across the country, the third shows average house prices by property type and finally the average sales volume of house. The logic behind this frame is that the first graph is predicted by the sequential model that is loaded in the GUI, while additional graphs are presented by the dataframe that are obtained from the data source (see figure 33).

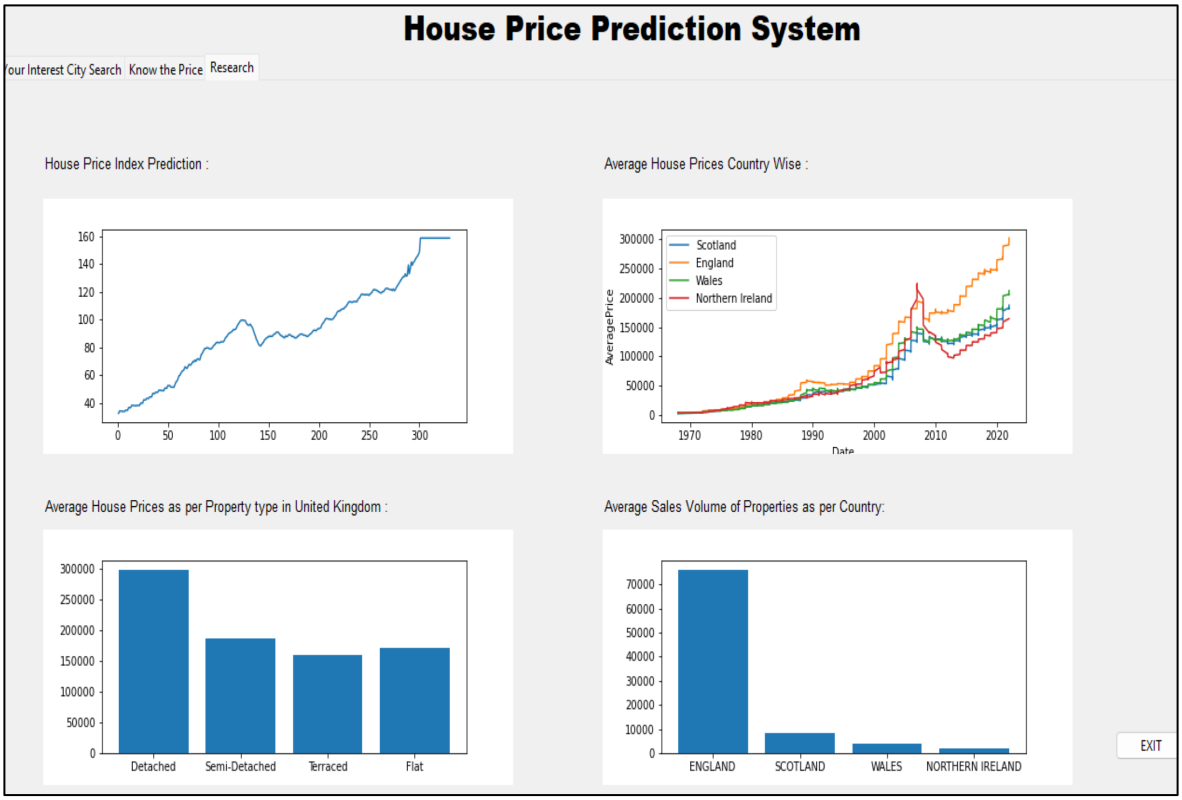

House Price Prediction System
'our Interest City Search
Know the Price
Research
House Price Index Prediction :
160
140
120
100
80
60
40
0
50
100
150
200
250
300
Average House Prices Country Wise :
Scotland
England
Wales
Northern Ireland
AveragePrice
300000
250000
200000
150000
100000
50000
0
1970
1980
1990
2000
2010
2020
Date
Average House Prices as per Property type in United Kingdom :
300000
250000
200000
150000
100000
50000
0
Detached
Semi-Detached
Terraced
Flat
Average Sales Volume of Properties as per Country:
70000
60000
50000
40000
30000
20000
10000
0
ENGLAND
SCOTLAND
WALES
NORTHERN IRELAND
EXIT

Figure 33: Frame 3 of GUI

4.5 Phase V: Evaluation

When addressing the stability of a predictive model, the effectiveness of a model must be estimated, utilizing a wide variety of performance criteria to obtain a highly accurate model. Following are the evaluation metrices that were used to check the performance of the models used.

4.5.1 Correlation Coefficient (R2)

R2 is a metric that quantifies the amount of variance explained by independent variables for a dependent variable in a regression model. The formula for R2 score is described as:

$$R^2 = \frac{\sum_{i=1}^{n}\left(Xi - \overline{X}\right)\left(Yi - \overline{Y}\right)}{\sqrt{\sum_{i=1}^{n}\left(Xi - \overline{X}\right)^2\left(Yi - \overline{Y}\right)}^2}$$

Figure 34: Formula for R2 score

4.5.2Mean Absolute Error (MAE):

The MAE quantifies the average difference between expected and observed values. The formula is described as:

$$\mathbf{MAE} = \frac{1}{n}\sum_{i=1}^{n} | x_i - x |$$

Figure 35: Formula for MAE

4.5.3 Mean Squared Error (MSE):

The degree of inaccuracy is measured by MSE, the average of squared difference between the expected and the observed values. The formula is described as:

$$MSE = \frac{\Sigma(y_i - \hat{y}_i)^2}{n}$$

Figure 36: Formula for MSE

4.5.4 Root Mean Squared Error (RMSE)

RMSE is the measure of how spread the residuals are, it is the standard deviation of the errors. The formula is defined as:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Figure 37: Formula for RMSE

The above-mentioned techniques were used to find the correctness and performance of the machine learning models, and the model with highest performance was deployed in the GUI.

CHAPTER 5: RESULTS

This section follows the previous chapter in presenting the results. This section summaries the key things of the study which was based in the implementation section of the research. Firstly, we would focus on the hypothesis phase

Table 16: Results of Hypothesis testing

Hypothesis	Test carried	Results	Test carried	Result
Hypothesis I: Income vs HP	Linear regression	3920.94, 150 / 9872, 1715	Correlation coefficient	0.877 /0.99
Hypothesis II: Inflation vs HPI	Granger causality F-test	4.4852 /3.070	p-value	0.0 /0.0006
Hypothesis III: GDP vs HPI	Granger causality F-test	4.4719	p-value	0.0168
Hypothesis IV: Interest rate vs HPI	Granger causality F-test	1.6911 / 1.1592	p-value	0.0795 / 0.3160
Hypothesis V: outstanding debt vs HP	Granger causality F-test	1.7096 / 1.3518	p-value	0.1269 / 0.2532
Hypothesis VI: Stock Market vs HPI	Linear Regression	27.80	Cross-correlation	Number of lags increases

				correlation decreases.

Sequential Neural network performed in projecting future house price indices (see figure16) as the MSE for training data was 26.935 and 112.374 for testing data.

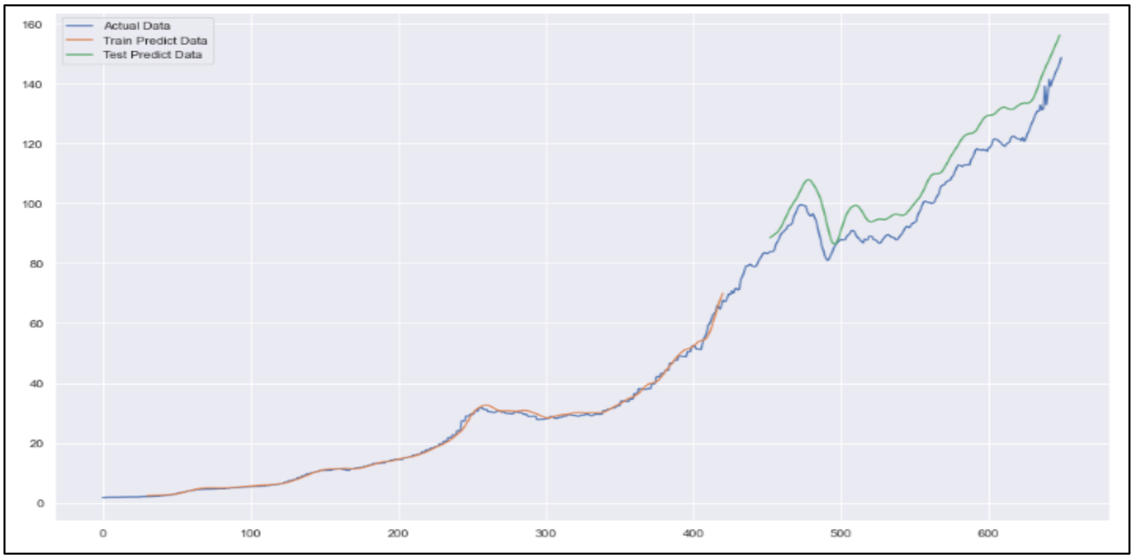


Figure 38: Plotting of predications over Train and test data

Evaluation techniques were applied to the different models used in the implementation part; the results are as follows:

Table 17: Performance Grid for Different Models

Model	Tunning	MAE	MSE	RMSE	R2
Random Forest	Scaling using MinMaxScalar	0.062	0.009	0.075	0.72
Random Forest	Hyperparameter tunning	0.126	0.027	0.1667	0.212
XGBoost	Scaling using MinMaxScalar	0.064	0.0093	0.0968	0.733

XGBoost	Hyperparameter tunning	0.058	0.008	0.090	0.766
XGBoost	Reduced features	58347	691578	83161	0.7825
XGBoost	Reduced features with hyperparameters tunning	69493	901493	94947	0.71
Sequential Neural Networks	Reduced features with MinMaxScaling	0.067	0.0108	0.104	0.68

CHAPTER 6: DISCUSSION:

6.1 Hypothesis I: Income of householder's vs House Price

Once applying the linear regression line using the stats library linregress function in python to the dataset acquired from Kaggle, we saw that the slope of the regression line was £ 3920.94, indicating the property prices are growing on average about 3900 each year, while the average UK pay is falling by 150 per year. The correlation coefficient was calculated to be 0.877

Another dataset obtained from gov.uk revealed that the average house price and average income from former home buyers are the highest since 1992. According to regression lines, house prices increased by 9872 while income increased by 1715 per year and gave a correlation value of 0.99.

Thus, the income and average prices of all dwellings are significantly connected.

6.2 Hypothesis II: Inflation vs House Price

After plotting the RPI and CPI line charts, the RPI values and variations were substantially higher than the CPI values. The granger causality test ran on RPI and CPI data with respect to HPI data, the F-test statistics were 4.4852 and 3.070 with p-values of 0.0 and 0.0006, respectively. Since, p-values were smaller than 0.05, we may reject the null hypothesis of the test and infer that the inflation does affect house prices.

6.3 Hypothesis III: Economy (GDP) vs House Price

When the UK GDP data was plotted, there was a dip around 2008 due to the financial crisis, and another in 2019 due to COVID-19.

The data was subjected to the granger causality test, which yielded a F-test statistics of 4.4719 and a p-value of 0.0168, we may reject the null hypothesis since the p-value is smaller than 0.05 and infer that knowing the GDP is important for forecasting future HPI.

6.4 Hypothesis IV: Interest Rate (Bank Rate) vs House Price Index

At lag=10, the granger causality test yielded a f-test statistics of 1.6911 and p-value of 0.0795. While the reverse, gave an f-test statistics as 1.1592 and p-value of 0.3160. We may accept the null hypothesis and conclude that interest rates of the bank do not affect the house prices or vice versa. However, according to the research, the influence of an increase in rates will be determined by the amount and timings of increase.

6.5 Hypothesis V: Outstanding Debt (Balance) vs House Prices

Plotting the data of loan arrears and outstanding balances each year, the insights were discovered that 2009 had the highest. Moreover, in prior years, the arrears loan count and outstanding balances have dropped with quarter fourth the lowest.

The granger causality test yielded an f-test score of 1.7096 and a p-value of 0.1269, whereas the reverse yielded an f-test score of 1.3518 and a p-value of 0.2532. Because p-value is less than 0.05, we reject the null hypothesis and conclude outstanding loans are not useful for predicting future house prices and vice versa.

6.6 Hypothesis VI: Stock Market vs House Price Index

The scatter plot of the HPI vs Stock market revealed a positive association, with the slope of regression line being 27.80. The cross-correlation statistics value shows that as the number of lags increases, the correlation between the two get less positive, this indicates that the stock market price during a particular month is highly predictive of the housing price index.

6.7 Different Indexes

When we compared different house price indices, we noticed that nationwide has the highest index values, followed by Halifax, whereas the UK HPI has the lowest. Furthermore, land surveyors report the highest property prices, while nationwide claim the lowest. When the variations in indices were visualized, the UK HPI varied the most.

6.8 Modelling

We learned during the EDA phase that England has the highest average property costs, while Northern Ireland had the lowest, however, Northern Ireland had the highest house price index with highest fluctuations in index as compared to others. Based on Sales volumes, England led the charts. On the grounds of property types, detached homes have the highest prices while terraced had the lowest.

Furthermore, according to deeper insights into regions, greater London has the highest pricing as well as weekly rents. However, when compared to other countries, Scotland experienced the largest average growth in a year, but Wales topped the charts in terms of 3 and 5 years

According to the findings of ANOVA tests done in implementation phase, on price paid transactions data, current energy ratings, property type, construction age bands, and energy tariff are substantially associated to house prices.

While, choosing the machine learning model to be deployed on the GUI, the model with highest R2 score was chosen, Thus, XGBoost with highest r2 score of 0.7825 was deployed on the GUI for prediction of house prices.

CHAPTER 7: CONCLUSIONS

This research explored various factors that influenced property prices in United Kingdom. After analysing several hypotheses, we determined that household income, inflation, GDP, and stock market had a positive impact on prices, whereas interest rates and outstanding debts (bankruptcies) had no effect. Even previous studies validate this.

Although mortgage companies' natural inclination for adopting their preferred index, there is not one optimal house price index in practice. This article studied machine learning models for forecasting and concluded that XGBoost surpasses the others on predictions.

We looked at how advances in AI helped us comprehend the open, complicated, and dynamic real estate market. As a property valuation help tool, algorithms provide a comparative advantage in housing market. The proposed would help a variety of groups, including homeowners by informing them of the anticipated price and regions to seek to buy a property on their preferences. Investors would gain since it would show them where to invest to optimize profits. It may help real estate brokers and developers in determining which factors influence the most. Prediction of house prices is critical for enabling budget allocation, finance solutions for buyers and investors.

CHAPTER 8: LIMITATIONS AND FUTURE SCOPE

The research did, however, have several shortcomings. Firstly, the property price might be influenced by other economic factors such as governmental decisions, sanctions, currency exchanges, which were not factored into the calculation. Secondly, due to lack of property data basically in the areas of Northern Ireland and Scotland, not much analysis was conducted in that areas. Lastly, the model's low performance might be due to lack of some environmental factors and an insufficient sample data. Moreover, due to low system resources, data took much time to train.

However, there is always room for improvement and learning, and there is lot of future potential for this research as the scope was limited, such as more research testing the influence of new independent variables, testing more external factors, gathering more data from other locations of UK which lacked in this project. Training the data on high-end machines could benefit in better learning by the algorithms which might result in better accuracy. Furthermore, more complicated neural networks could be used to enhance performance.

CHAPTER 9: REFERENCES

9.1 Text Reference

[1] Frew J., and G. D. Jud, 2003, “Estimating the Value of Apartment Buildings”, The Journal of Real Estate Research, 25(1): 77 - 86.

[2] Iacoviello, M. and S. Neri (2010): “Housing Market Spillovers: Evidence from an Estimated

DSGE Model,” American Economic Journal: Macroeconomics, 2, 125–64.

[3] Duca, J. V., J. Muellbauer, and A. Murphy (2021): “What Drives House Price Cycles?

International Experience and Policy Issues,” Journal of Economic Literature, 59, 773–864

[4] Case, K. E. and R. J. Shiller (1989): “The Efficiency of the Market for Single-Family

Homes,” The American Economic Review, 79, 125–137.

[5] Nuuter, Tiina, Irene Lill, and Laura Tupenaite. 2014. Ranking of housing market sustainability in selected European Countries. WSEAS Transactions on Business and Economics 11: 778–86.

[6] Wang, Zhimin, Jung Hoon, and Benson Lim. 2012. The impacts of housing affordability on social and economic sustainability in Beijing. Australasian Journal of Construction Economics and Building-Conference Series 1: 47–55

[7] Beltratti, Andrea, and Claudio Morana. 2010. International house prices and macroeconomic fluctuations. Journal of Banking & Finance 34: 533–45

[8] Uyar, Kangallı, Sinem Güler, and Nihal Yayla. 2016. Konut Fiyatlarının Hedonik Fiyatlama Yaklaşımına Göre Mekânsal Ekonometrik Modeller ile Tahmini: İstanbul

Konut Piyasası Örneği. Social Sciences (NWSASOS) 3C0153 11: 326–42. [Google Scholar]

[9] www.buyassociation.co.uk. (2021). Overseas investors snapping up UK property investments | BuyAssociation. [online]

[10] www.ons.gov.uk. (n.d.). Foreign direct investment involving UK companies - Office for National Statistics. [online]

[11] investors.compass.com. (n.d.). Compass Launches AI-Powered Video Studio. [online] Available at: https://investors.compass.com/news/news-details/2021/Compass-Launches-AI-Powered-Video-Studio/default.aspx#:~:text=Using%20AI%2Dpowered%20phrase%20recognition

[12] VentureBeat. (2021). Zillow utilizes explainer AI, data to revolutionize how people sell houses. [online] Available at: https://venturebeat.com/ai/zillow-utilizes-ai-data-to-revolutionize-how-people-sell-houses/

[13] Robertson, C. (2020). LoanSnap Review: Using Artificial Intelligence to Make Smart Home Loans. [online] The Truth About Mortgage. Available at: https://www.thetruthaboutmortgage.com/loansnap-review/#:~:text=In%20LoanSnap

[14] Takáts, E. (2012). Aging and house prices. Journal of Housing Economics, 21(2), pp.131–141. doi: 10.1016/j.jhe.2012.04.001.

[15] Sivitanides, Petros. (2018). Macroeconomic drivers of London house prices. Journal of Property Investment and Finance. 36. 10.1108/JPIF-02-2018-0012.

[16] Jha, Shashi Bhushan & Babiceanu, Radu & Pandey, Vijay & Jha, Rajesh Kumar. (2020). Housing Market Prediction Problem using Different Machine Learning Algorithms: A Case Study.

[17] Gohl, Niklas & Haan, Peter & Michelsen, Claus & Weinhardt, Felix. (2022). House Price Expectations. SSRN Electronic Journal. 10.2139/ssrn.4020245.

[18] Bissoondeeal, R.K. and Tsiaras, L. (2021). Investigating the Links between UK House Prices and Share Prices with Copulas. The Journal of Real Estate Finance and Economics. doi:10.1007/s11146-021-09854-0.

[19] Truong, Q., Nguyen, M., Dang, H. and Mei, B. (2020). Housing Price Prediction via Improved Machine Learning Techniques. Procedia Computer Science, 174, pp.433–442. doi:10.1016/j.procs.2020.06.111.

[20] Chandler, D. and Disney, R. (2014). The Housing Market in the United Kingdom: Effects of House Price Volatility on Households. Fiscal Studies, [online] 35(3), pp.371–394. Available at: https://www.jstor.org/stable/24440364 [Accessed 29 Aug. 2022].

[21] Adams, Zeno & Füss, Roland, 2010. "Macroeconomic determinants of international housing markets," Journal of Housing Economics, Elsevier, vol. 19(1), pages 38-50, March.

[22] Lu Decee Xu and Tang, B. (2012). ON THE DETERMINANTS OF UK HOUSE PRICES. [online] Available at: https://www.semanticscholar.org/paper/ON-THE-DETERMINANTS-OF-UK-HOUSE-PRICES-Xu-Tang/84fe44e24c8c9d18eabfc164b2dedd83324d29cd.

[23] Visit Limsombunchai, Gan, C. and Lee, M. (2004). House Price Prediction: Hedonic Price Model vs. Artificial Neural Network. [online] ResearchGate.

[24] Angelov, P.P., Gu, X. and Príncipe, J.C. (2018). A Generalized Methodology for Data Analysis. IEEE Transactions on Cybernetics, [online] 48(10), pp.2981–2993. doi:10.1109/TCYB.2017.2753880.

[25] Bhandari, P. (2022). An introduction to quantitative research. [online] Scribbr. Available at: https://www.scribbr.co.uk/research-methods/introduction-to-quantitative-research/.

[26] Krainer, J., 2005. Housing markets and demographics, FRBSF Economic

Letter, No. 2005-21, Federal Reserve Bank of San Francisco.

[27] UN, 2011. World Population Prospects: the 2010 revision population database. <http://esa.un.org/unpp/>.

[28] Lee,I.O., Painter,G(2013),'What happens to household formation in a recession?',Journal of Urban Economics, 76-93-109

[29] Paciorek, A.D.(2013) The Long and the Short of Household Formation. Finance and Economics Discussion Series. Divisions of Research & Statistics and Monetary Affairs. Federal Reserve Board, Washington DC.

[30] Kabir, S.M.S. (2016). Methods Of Data Collection. [online] ResearchGate.

[31] Kilkenny, Monique & Robinson, Kerin. (2018). Data quality: "Garbage in – garbage out". Health Information Management Journal. 47. 183335831877435. 10.1177/1833358318774357.

[32] Ghosh, Debamita. (2017). ANALYSIS OF DATA.ResearchGate

[33] Lee, Wei-Meng. (2019). Deploying Machine Learning Models. 10.1002/9781119557500.ch12.

[34] Cichosz, Pawel. (2015). Regression model evaluation. 10.1002/9781118950951.ch10.

[35] Raschka, Sebastian & Patterson, Joshua & Nolet, Corey. (2020). Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. Information. 11. 193. 10.3390/info11040193.

[36] Søren Johansen (1991). Estimation and Hypothesis Testing of Cointegration Vectors inGaussian Vector Autoregressive Models. Econometrica Vol. 59, No. 6 (Nov., 1991), pp. 1551-1580

[37] Kolawole, Ayotunde & Sekumade, Adelomo. (2017). Hypotheses and Hypothesis Testing. 10.13140/RG.2.2.28299.39202.

[38] tattrek.com. (n.d.). AP Statistics Tutorial. [online] Available at: https://stattrek.com/tutorials/ap-statistics-tutorial

[39] Schmickler, Arno & Park, Kenneth. (2014). UK Social Housing and Housing Market in England: A Statistical Review and Trends. LHI Journal of Land, Housing, and Urban Affairs. 5. 193-201. 10.5804/LHIJ.2014.5.3.193.

[40] Chen, Ming-Chi & Tsai, I-Chun & Chang, Chin-Oh. (2007). House prices and household income: Do they move apart? Evidence from Taiwan. Habitat International. 31. 243-256. 10.1016/j.habitatint.2007.02.005.

[41] Atalay, Kadir & Edwards, Rebecca. (2022). House Prices, Housing Wealth and Financial Well-being. Journal of Urban Economics. 129. 103438. 10.1016/j.jue.2022.103438.

[42] Bewick, V., Cheek, L. and Ball, J. (2003). Statistics review 7: Correlation and regression. Critical Care, [online] 7(6), pp.451–459.

[43] UKEssays.com. (2009). Inflation And Interest Rates And Uk Housing Market Economics Essay. [online] Available at: https://www.ukessays.com/essays/economics/inflation-and-interest-rates-and-uk-housing-market-economics-essay.php.

[44] Stephanie (2016). Lag Plot: Definition, Examples. [online] Statistics How To. Available at: https://www.statisticshowto.com/lag-plot/.

[45] StatsTest.com. (n.d.). Kendall’s Tau. [online] Available at: https://www.statstest.com/kendalls-tau/.

[46] Zach (2021). How to Perform a Granger-Causality Test in Python. [online] Statology. Available at: https://www.statology.org/granger-causality-test-in-python/

[47] Aizenman, Joshua & Jinjarak, Yothin & Zheng, Huanhuan. (2019). Housing Bubbles, Economic Growth, and Institutions. Open Economies Review. 30. 10.1007/s11079-019-09535-9.

[48] Xu, T. (2017). The Relationship between Interest Rates, Income, GDP Growth and House Prices. Research in Economics and Management, 2(1), p.30. doi:10.22158/rem.v2n1p30.

[49] Gimeno, R. and Martinez-Carrascal, C. (2006). The Interaction between House Prices and Loans for House Purchase: The Spanish Case. SSRN Electronic Journal. doi:10.2139/ssrn.901145.

[50] Disney, R., Bridges, S., & Gathergood, J. (2010). House Price Shocks and Household Indebtedness in the United Kingdom. Economica, 77(307), 472–496. http://www.jstor.org/stable/40865113

[51] Liow, Kim. (2006). Dynamic relationship between stock and property markets. Applied Financial Economics. 16. 371-376. 10.1080/09603100500390885.

[52] Amazon Web Services (n.d.). What is an API? - API Beginner's Guide - AWS. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/what-is/api/.

[53] propertydata.co.uk. (n.d.). Property Analytics API Documentation - PropertyData. [online] Available at: https://propertydata.co.uk/api/documentation

[54] Foursquare Developer Documentation. (n.d.). Overview. [online] Available at: https://developer.foursquare.com/reference/api-explorer-overview

[55] Pramanik, Jitendra & Samal, Abhaya Kumar & Sahoo, Kabita & Pani, Dr. Subhendu. (2019). Exploratory Data Analysis using Python. International Journal of Innovative Technology and Exploring Engineering. 8. 4727-4735.

[56] Chandler, D. and Disney, R. (n.d.). Measuring house prices: a comparison of different indices IFS Briefing Note BN146. [online] Available at: https://ifs.org.uk/bns/bn146.pdf.

[57] KDnuggets. (n.d.). Machine Learning Model Development and Model Operations: Principles and Practices. [online] Available at: https://www.kdnuggets.com/2021/10/machine-learning-model-development-operations-principles-practice.html.

[58] Li, Youguo & Wu, Haiyan. (2012). A Clustering Method Based on K-Means Algorithm. Physics Procedia. 25. 1104-1109. 10.1016/j.phpro.2012.03.206.

[59] Wang, chusheng. (2021). House Price Prediction Model Based on Neural Network. 10.21203/rs.3.rs-805003/v1.

[60] Varma, Ayush & Sarma, Abhijit & Doshi, Sagar & Nair, Rohini. (2018). House Price Prediction Using Machine Learning and Neural Networks. 1936-1939. 10.1109/ICICCT.2018.8473231.

[61] Zach (2020). How to Calculate Point-Biserial Correlation in Python. [online] Statology. Available at: https://statology.org/point-biserial-correlation-python/

[62] Computer Hope (2019). What is a GUI (Graphical User Interface)? [online] Computerhope.com. Available at: https://www.computerhope.com/jargon/g/gui.htm.

[63] Python, R. (2022). Python GUI Programming With Tkinter – Real Python. [online] realpython.com. Available at: https://realpython.com/python-gui-tkinter/.

[64] docs.aws.amazon.com. (n.d.). How XGBoost Works - Amazon SageMaker. [online] Available at: https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html.

9.2 Figure References:

[1] figure 1,2 are referenced from https://propertydata.co.uk/charts/house-prices

[2] figure 4 is referenced from Kolawole, Ayotunde & Sekumade, Adelomo. (2017). Hypotheses and Hypothesis Testing. 10.13140/RG.2.2.28299.39202.

[3] Figure 5 is referenced from Investopedia.com

[9] Figure 9 referenced from Pramanik, Jitendra & Samal, Abhaya Kumar & Sahoo, Kabita & Pani, Dr. Subhendu. (2019). Exploratory Data Analysis using Python. International Journal of Innovative Technology and Exploring Engineering. 8. 4727-4735.

[10] Figure 20 referenced from KDnuggets. (n.d.). Machine Learning Model Development and Model Operations: Principles and Practices

[11] figure 34 reference from https://www.frontiersin.org/articles/10.3389/feart.2021.761990/full

[12] figure 35 reference from https://www.statisticshowto.com/absolute-error/

[13] figure 36 referenced from https://statisticsbyjim.com/regression/mean-squared-error-mse/

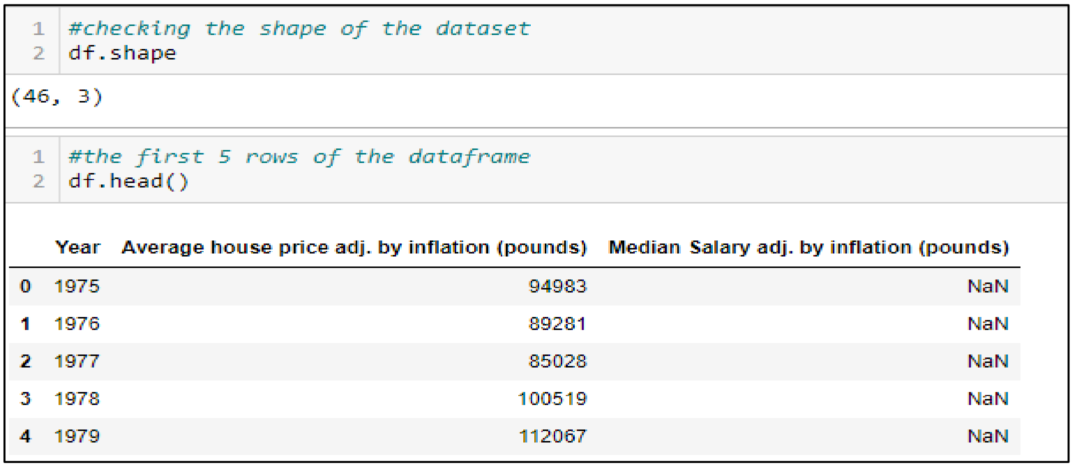
[14] figure 37 referenced from https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e

9.3 Code References

[1] https://github.com/ankushbabbar1992/Coursera_Capstone

[2] https://github.com/krishnaik06

CHAPTER 10: APPENDIX



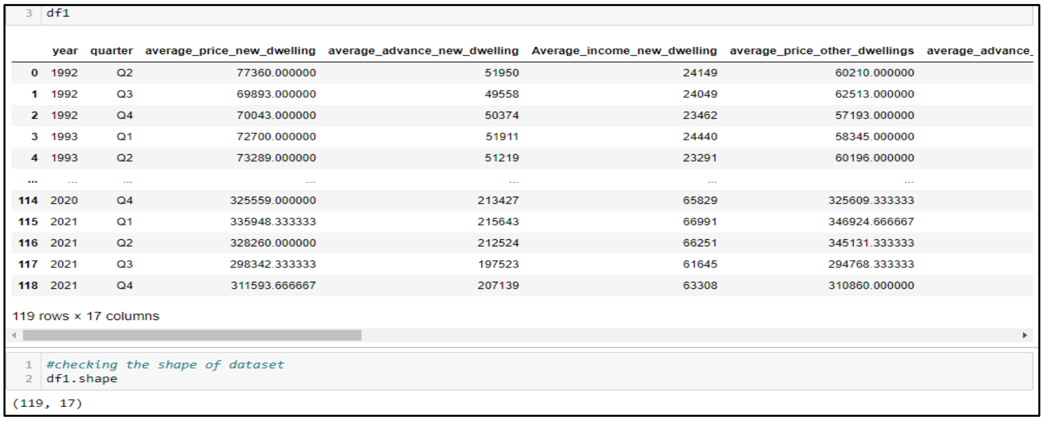
```
#checking the shape of the dataset
df.shape
```

```
(46, 3)
```

```
#the first 5 rows of the dataframe
df.head()
```

	Year	Average house price adj. by inflation (pounds)	Median Salary adj. by inflation (pounds)
0	1975	94983	NaN
1	1976	89281	NaN
2	1977	85028	NaN
3	1978	100519	NaN
4	1979	112067	NaN

Figure 39: Dataset from Kaggle(Income vs House Price)



```
df1
```

	year	quarter	average_price_new_dwelling	average_advance_new_dwelling	Average_income_new_dwelling	average_price_other_dwellings	average_advance_
0	1992	Q2	77360.000000	51950	24149	60210.000000	
1	1992	Q3	69893.000000	49558	24049	62513.000000	
2	1992	Q4	70043.000000	50374	23462	57193.000000	
3	1993	Q1	72700.000000	51911	24440	58345.000000	
4	1993	Q2	73289.000000	51219	23291	60196.000000	
...	...	...	...	...	...	...	
114	2020	Q4	325559.000000	213427	65829	325609.333333	
115	2021	Q1	335948.333333	215643	66991	346924.666667	
116	2021	Q2	328260.000000	212524	66251	345131.333333	
117	2021	Q3	298342.333333	197523	61645	294768.333333	
118	2021	Q4	311593.666667	207139	63308	310860.000000	

119 rows × 17 columns

```
#checking the shape of dataset
df1.shape
```

```
(119, 17)
```

Figure 40: Dataset from gov.uk (income vs House price)

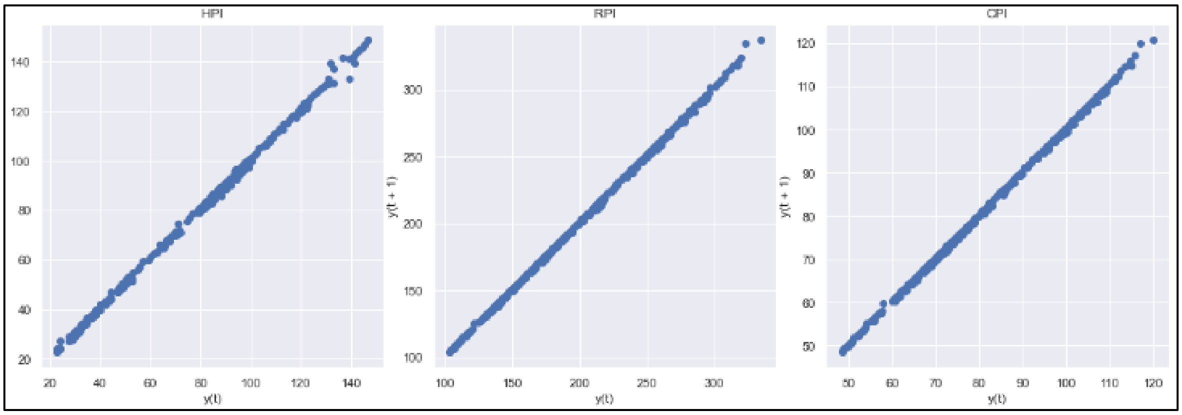


Figure 41: Lag plots for inflation and HPI

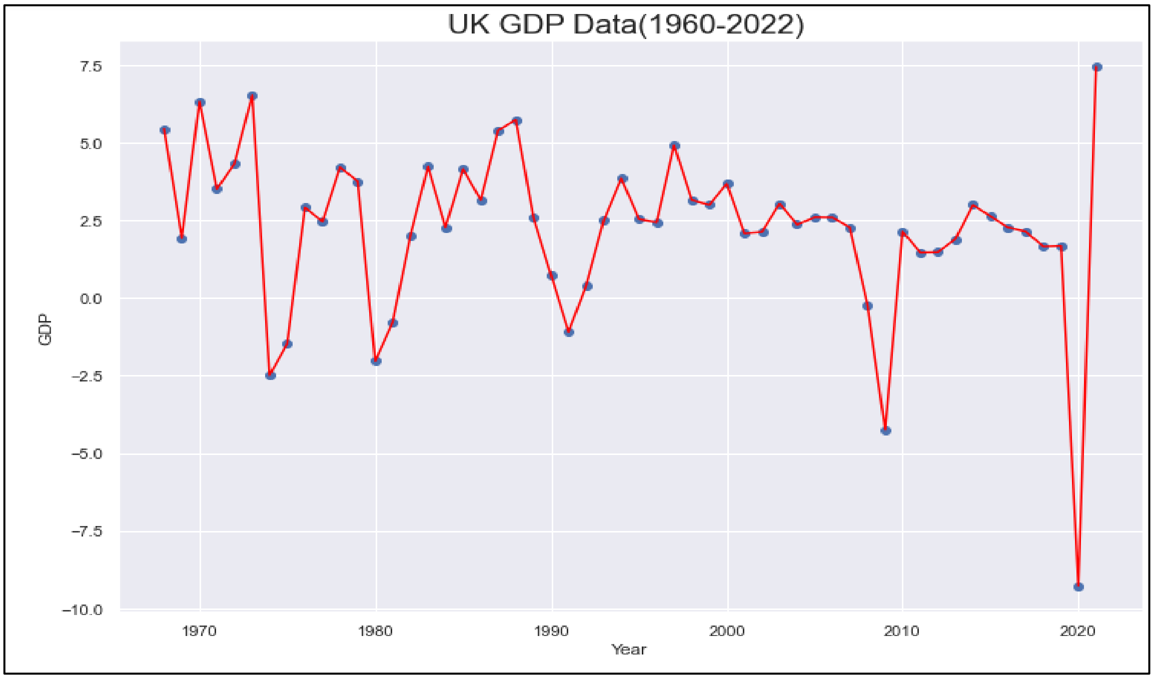


Figure 42: UK GDP Data(1960-2022)

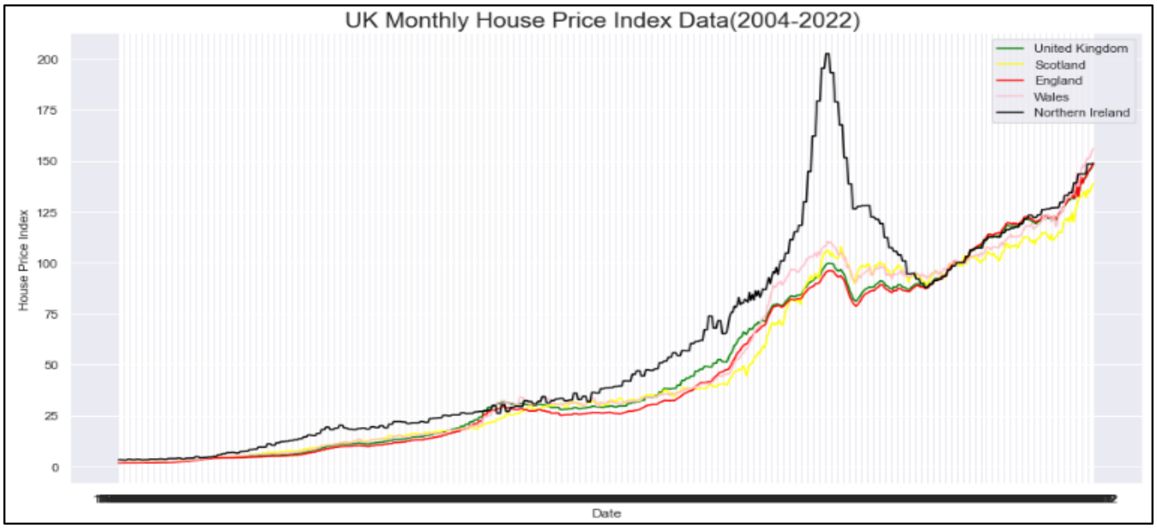


Figure 43: EDA on land registry data HPI

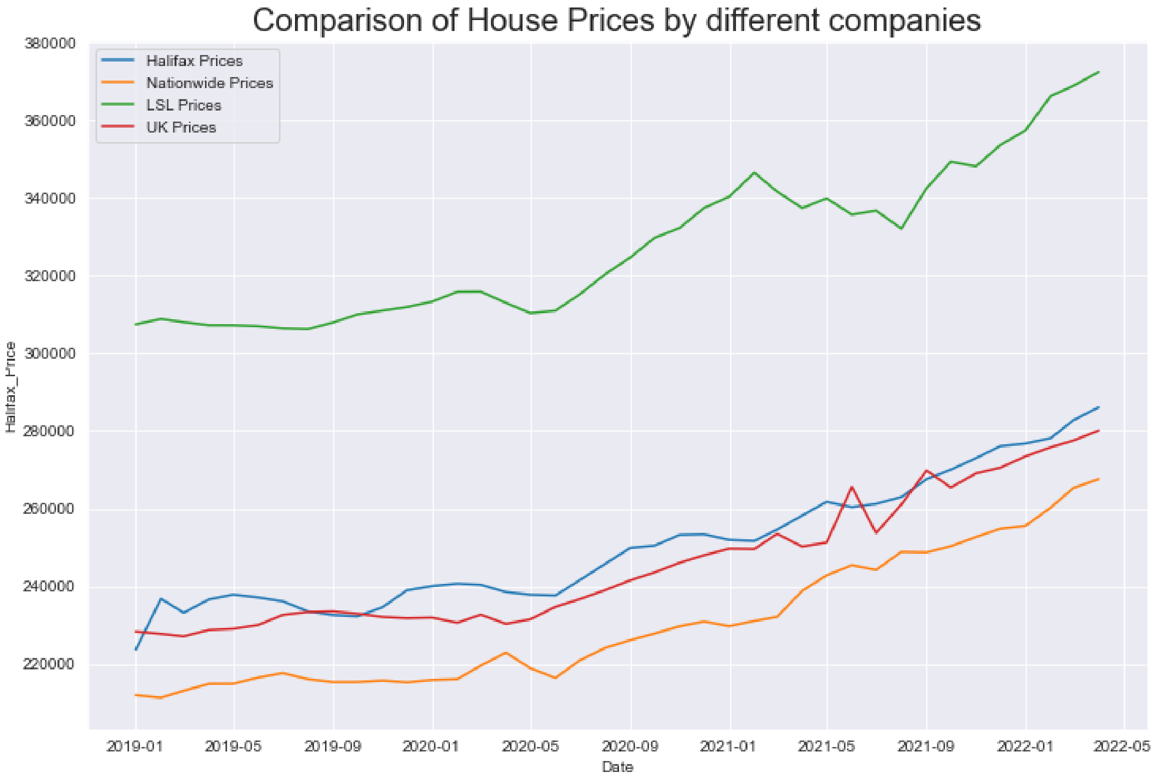


Figure 44: Comparison of house price by different indexes

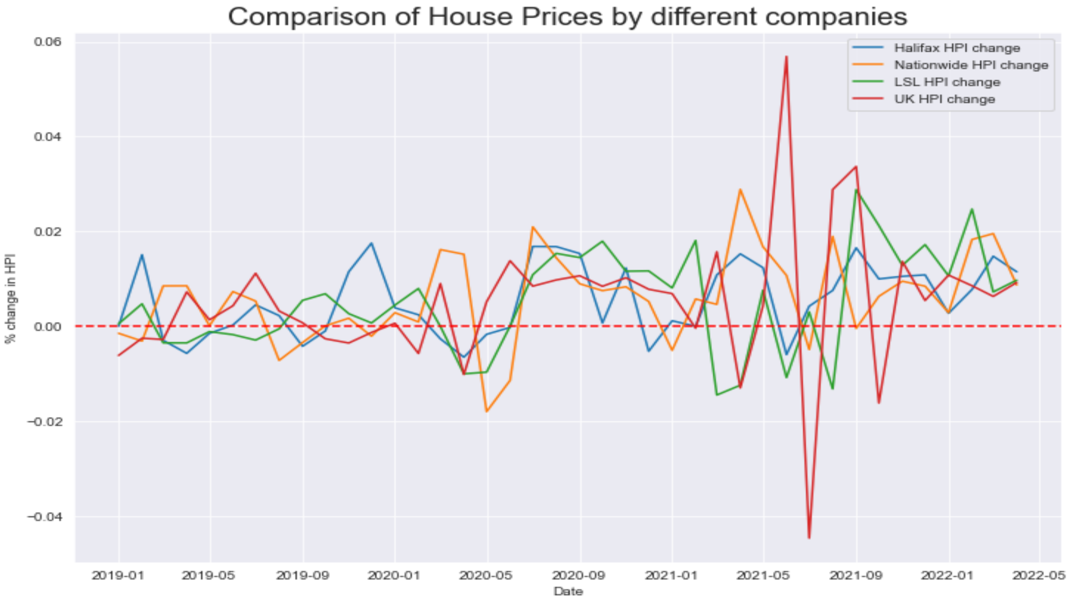


Figure 45: Change in House Prices by different indexes

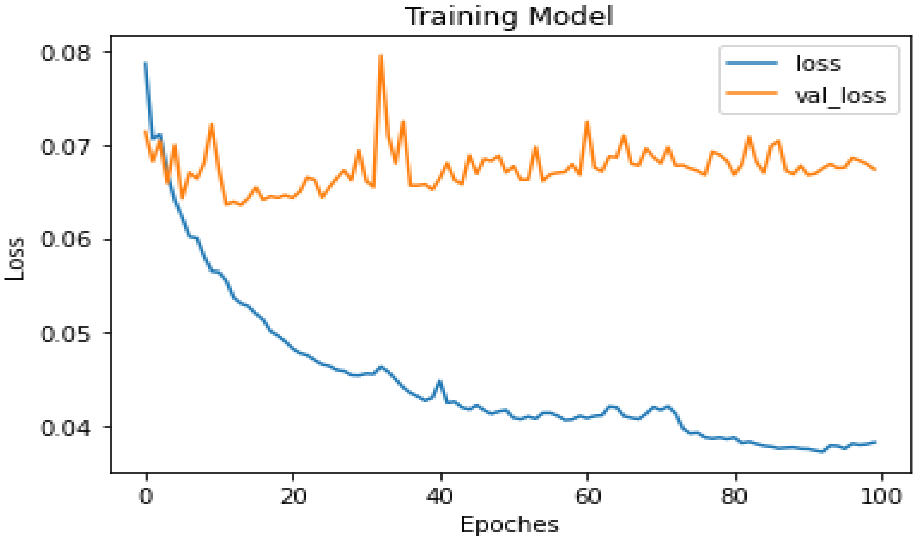


Figure 46: Epochs vs loss for Neural Network

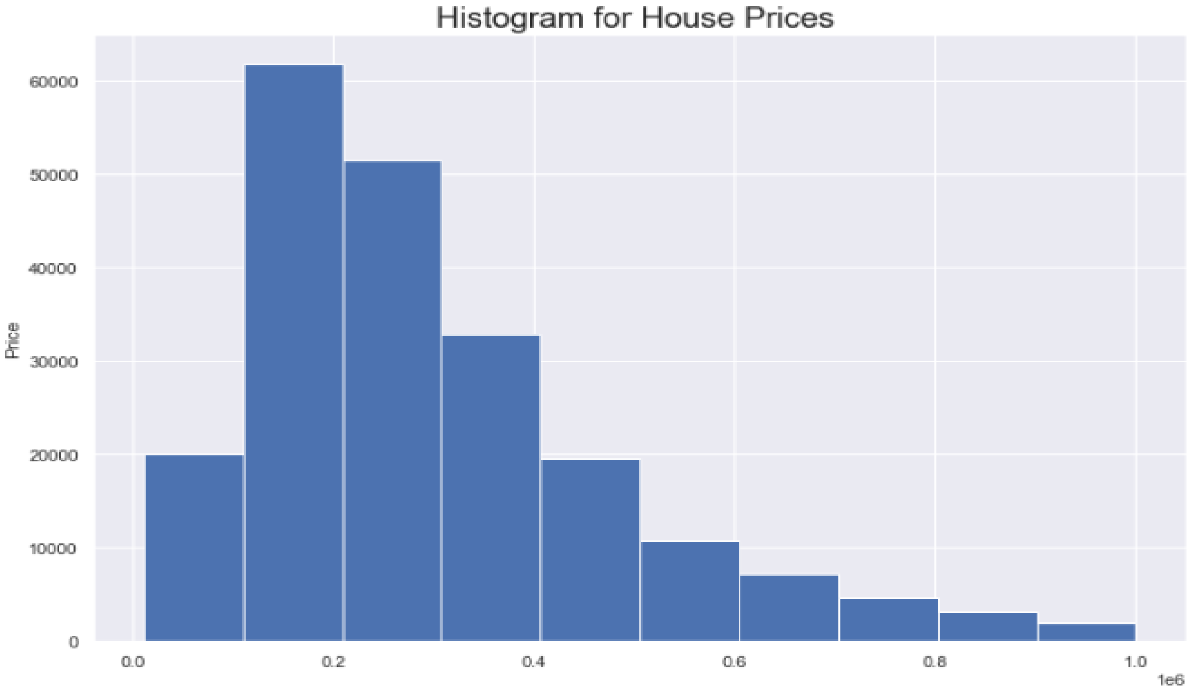


Figure 47: Histogram of prices of price paid transactions

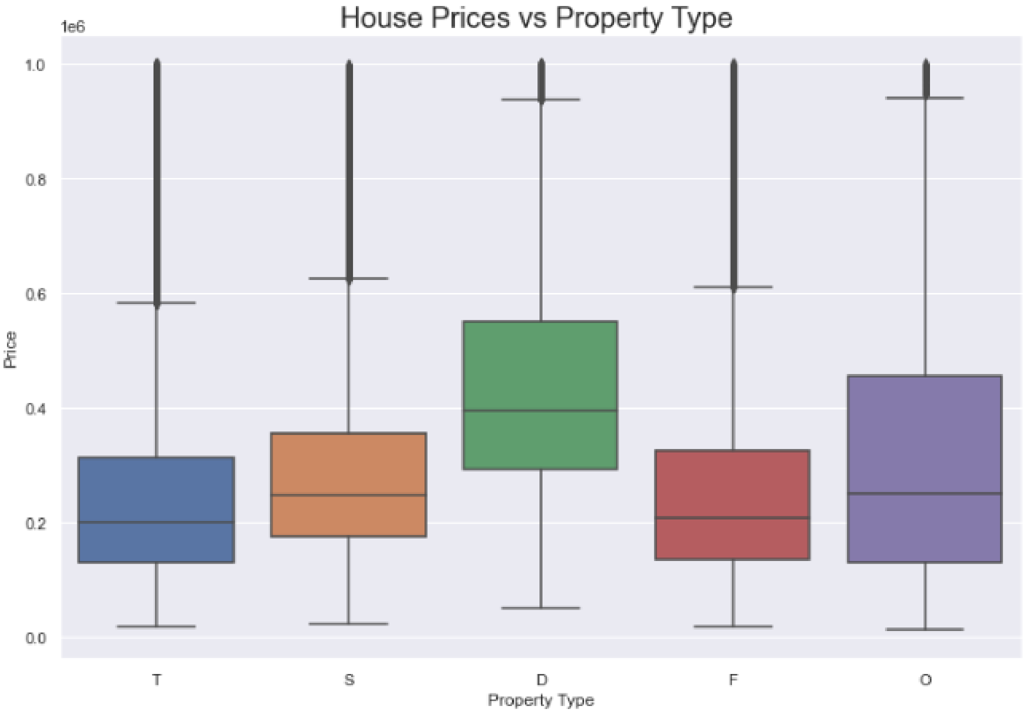


Figure 48: Box plot for House Prices vs property types

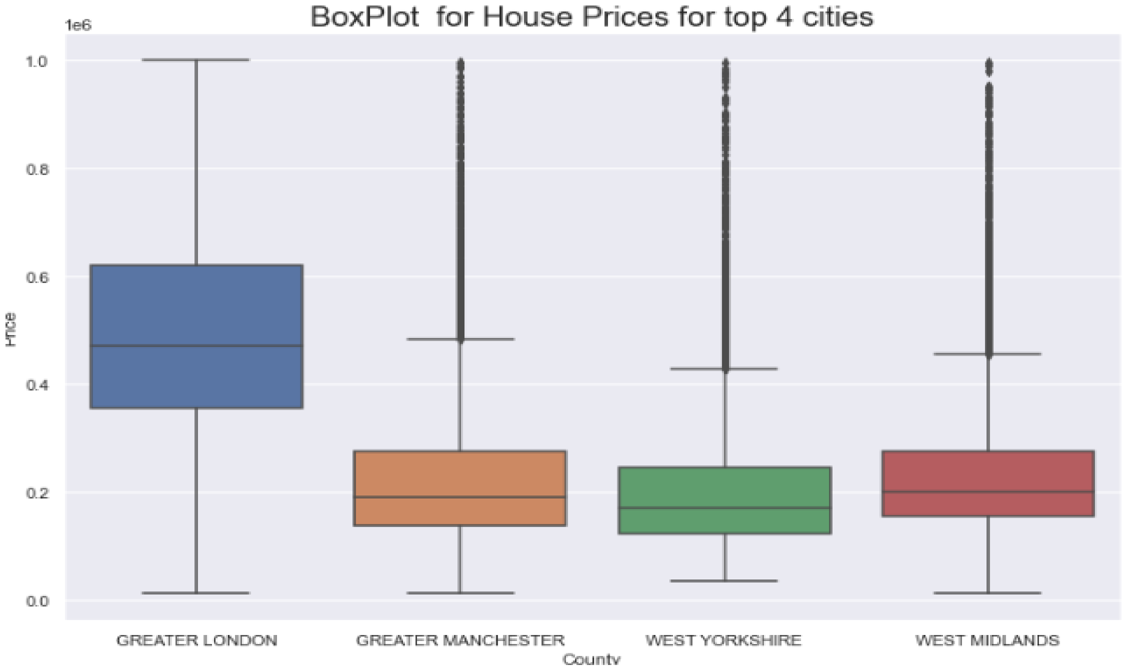


Figure 49: Box plot for top 4 cities in transactions

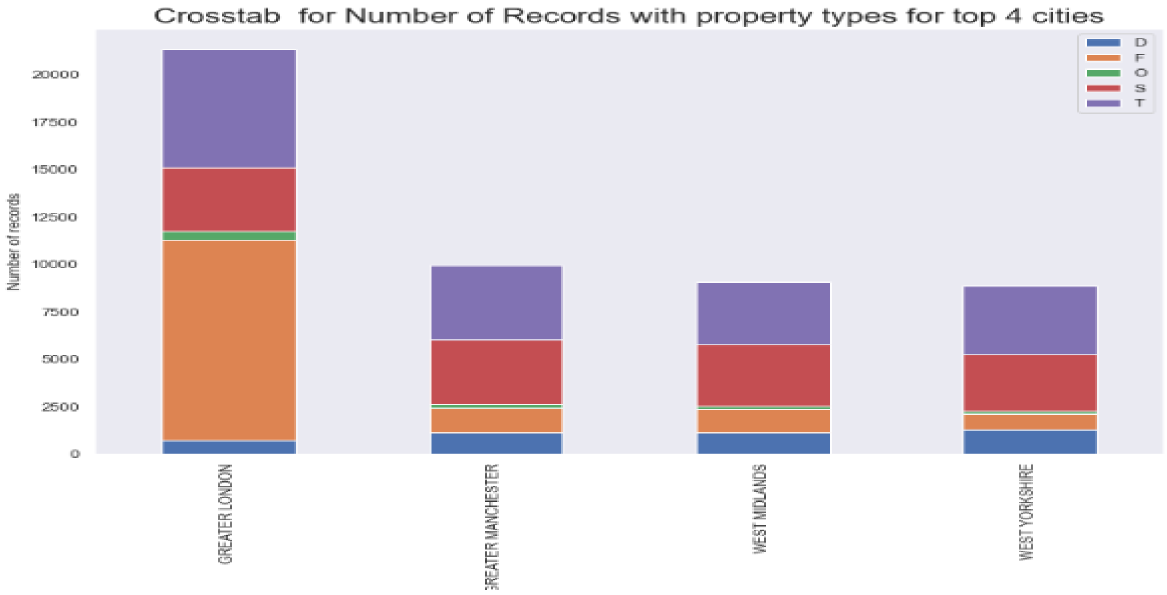


Figure 50: crosstab for number of records with property types

10.1 Ethics Form

10.2 Code

Ethical clearance for research and innovation projects

Project status

Status

● ● ● Awaiting supervisor review

Actions

Date	Who	Action	Comments
15:34:00 27 August 2022	Ankush Babbar	Principal investigator submitted	

Get Help

Ethics release checklist (ERC)

Project details

Project name: House Price Prediction

Principal investigator: Ankush Babbar

Faculty: Faculty of Business, Law and Digital Technologies ▾

Level: Postgraduate ▾

Course: Msc In Applied AI & Business

Unit code: COM726

Supervisor name: Femi Isiaq

Other investigators:

Checklist

Question	Yes	No
Q1. Will the project involve human participants other than the investigator(s)?	○	●
Q1a. Will the project involve **vulnerable participants** such as children, young people, disabled people, the elderly, people with declared mental health issues, prisoners, people in health or social care settings, addicts, or those with learning difficulties or cognitive impairment either contacted directly or via a **gatekeeper** (for example a professional who runs an organisation through which participants are accessed; a service provider; a care-giver; a relative or a guardian)?	○	●
Q1b. Will the project involve the use of **control groups** or the **use of deception**?	○	●
Q1c. Will the project involve any **risk to the participants' health** (e.g. intrusive intervention such as the administration of drugs or other substances, or vigorous physical exercise), or involve psychological stress, anxiety, humiliation, physical pain or discomfort to the investigator(s) and/or the participants?	○	●
Q1d. Will the project involve **financial inducement** offered to participants other than reasonable expenses and compensation for time?	○	●
Q1e. Will the project be carried out by individuals unconnected with the University but who wish to use staff and/or students of the University as participants?	○	●
Q2. Will the project involve sensitive materials or topics that might be considered offensive, distressing, politically or socially sensitive, deeply personal or in breach of the law (for example criminal activities, sexual behaviour, ethnic status, personal appearance, experience of violence, addiction, religion, or financial circumstances)?	○	●
Q3. Will the project have detrimental impact on the environment, habitat or species?	○	●
Q4. Will the project involve living animal subjects?	○	●
Q5. Will the project involve the development for export of 'controlled' goods regulated by the Export Control Organisation (ECO)? (This specifically means military goods, so called dual-use goods (which are civilian goods but with a potential military use or application), products used for torture and repression, radioactive sources.) Further information from the Export Control Organisation	○	●
Q6. Does your research involve: the storage of records on a computer, electronic transmissions, or visits to websites, which are associated with terrorist or extreme groups or other security sensitive material? Further information from the Information Commissioners Office	○	●

Declarations

I/we, the investigator(s), confirm that:

☑ The information contained in this checklist is correct.

☑ I/we have assessed the ethical considerations in relation to the project in line with the University Ethics Policy.

☑ I/we understand that the ethical considerations of the project will need to be re-assessed if there are any changes to it.

☑ I/we will endeavor to preserve the reputation of the University and protect the health and safety of all those involved when conducting this research/enterprise project.

☑ If personal data is to be collected as part of my project, I confirm that my project and I, as Principal Investigator, will adhere to the General Data Protection Regulation (GDPR) and the Data Protection Act 2018. I also confirm that I will seek advice on the DPA, as necessary, by referring to the Information Commissioner's Office further guidance on DPA and/or by contacting information.rights@solent.ac.uk. By Personal data, I understand any data that I will collect as part of my project that can identify an individual, whether in personal or family life, business or profession.

☑ I/we have read the prevent agenda.

Hypothesis I : Income Vs House Prices

Dataset taken from Kaggle :https://www.kaggle.com/datasets/samuelcortinhas/uk-median-house-prices-and-salary-19752020

In [1]:
```
#importing various libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
```

In [2]:
```
#importing the dataset
df = pd.read_csv(r"Datasets/income_HP_kaggle.csv")
```

In [3]:
```
#checking the shape of the dataset
df.shape
```

Out[3]:
```
(46, 3)
```

In [26]:
```
#the first 5 rows of the dataframe
df.head()
```

Out[26]:

	Year	Average house price adj. by inflation (pounds)	Median Salary adj. by inflation (pounds)
0	1975	94983	NaN
1	1976	89281	NaN
2	1977	85028	NaN
3	1978	100519	NaN
4	1979	112067	NaN

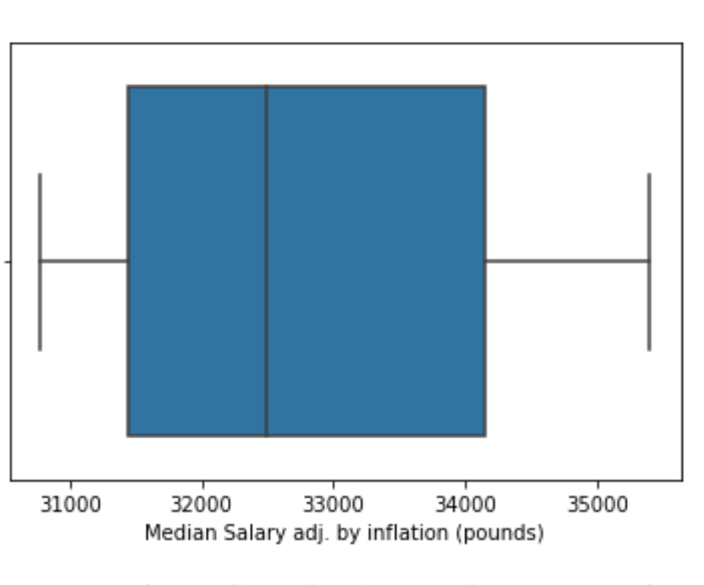
In [4]:
```
#checking the statistics of the dataset
df.describe()
```

Out[4]:

	Year	Average house price adj. by inflation (pounds)	Median Salary adj. by inflation (pounds)
count	46.000000	46.000000	22.000000
mean	1997.500000	164014.717391	32791.958182
std	13.422618	60004.318931	1548.767865
min	1975.000000	85028.000000	30772.440000
25%	1986.250000	110149.250000	31436.870000
50%	1997.500000	141578.500000	32484.225000
75%	2008.750000	225347.750000	34145.020000
max	2020.000000	269297.000000	35399.830000

In [5]:
```
#visualizing the salary
sns.boxplot(x='Median Salary adj. by inflation (pounds)',data=df)
```

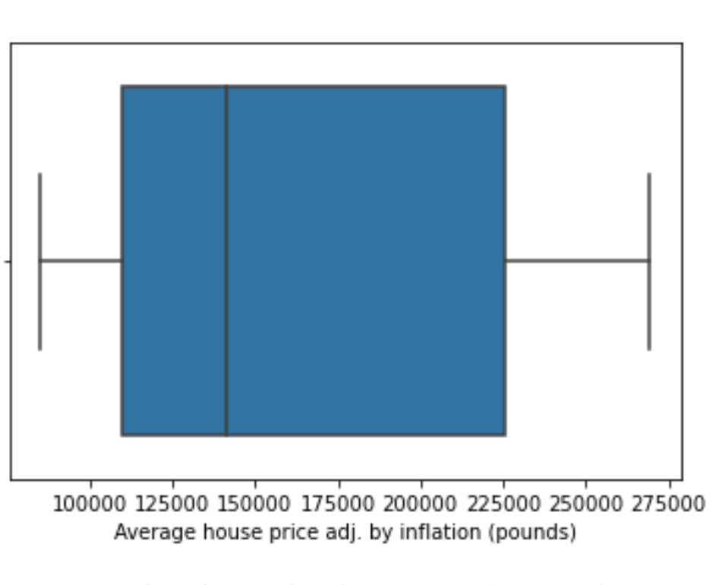
Out[5]:
```
<AxesSubplot:xlabel='Median Salary adj. by inflation (pounds)'>
```



We can see the median salary is around 32,500 pounds.

In [6]:
```
#visualizing the house prices
sns.boxplot(x='Average house price adj. by inflation (pounds)',data=df)
```

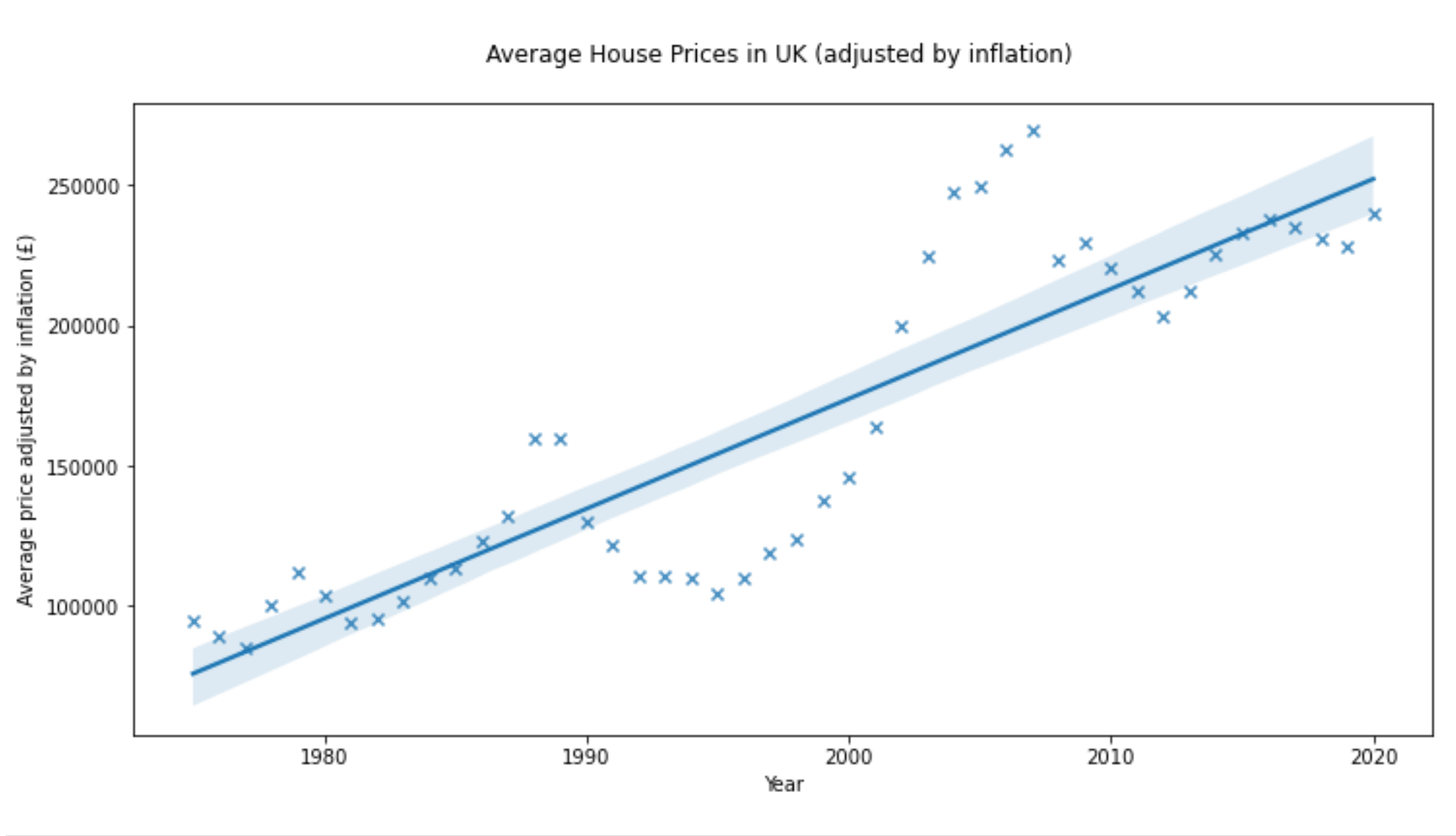
Out[6]:
```
<AxesSubplot:xlabel='Average house price adj. by inflation (pounds)'>
```



We can see that the median house price is around 142000 pounds.

In [7]:
```
# Plotting average house price over the years
plt.figure(figsize=(12,6))
sns.regplot(x='Year', y='Average house price adj. by inflation (pounds)', data=df,marker='x')
plt.title('Average House Prices in UK (adjusted by inflation) \n')
plt.xlabel('Year')
plt.ylabel('Average price adjusted by inflation (£)')
```

Out[7]:
```
Text(0, 0.5, 'Average price adjusted by inflation (£)')
```



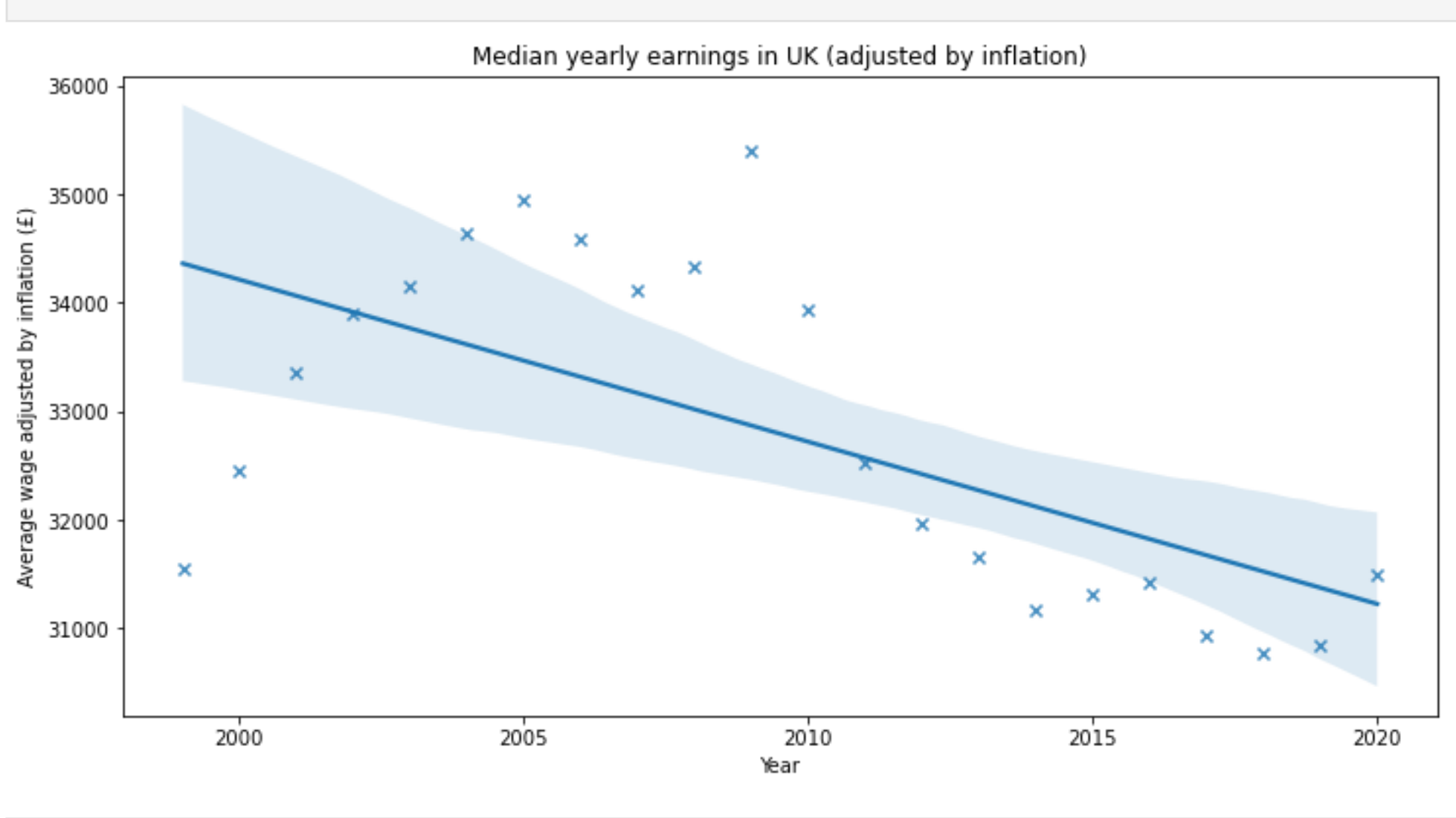
In [8]:
```
# Slope of regression line
slope, intercept, r_value, p_value, std_err = stats.linregress(df['Year'],
                                                               df['Average house price adj. by inflation (pound

print('The slope of the regression line is: £','{:.2f}'.format(slope))
```

```
The slope of the regression line is: £ 3920.94
```

Observations:

The regression line tells us that houses prices are increasing each year on average after inflation has been taken into account. By nearly £3900 a year!
We also see that houses prices took a serious drop in 2008/2009 due to the housing market crash at that time.

In [9]:
```
# Plotting the average salary over time
plt.figure(figsize=(12,6))
sns.regplot(x='Year', y='Median Salary adj. by inflation (pounds)', data=df,marker='x')
plt.title('Median yearly earnings in UK (adjusted by inflation)')
plt.xlabel('Year')
plt.ylabel('Average wage adjusted by inflation (£)')
plt.show()
```



In [10]:
```
# Slope of regression line
slope, intercept, r_value, p_value, std_err = stats.linregress(df.iloc[24:,0],df.iloc[24:,2])

print('The slope of the regression line is: £','{:.2f}'.format(slope))
```

```
The slope of the regression line is: £ -149.39
```

Observations:

The average salary in the UK is actually decreasing (by £150) each year on average after inflation has been taken into account.
It seems they took the biggest hit just after the housing market crash of 2008-2009.
Whilst house prices bounced back after 2010, wages have not.

In [11]:
```
df.corr()
```

Out[11]:

	Year	Average house price adj. by inflation (pounds)	Median Salary adj. by inflation (pounds)
Year	1.000000	0.877091	-0.626351
Average house price adj. by inflation (pounds)	0.877091	1.000000	0.227437
Median Salary adj. by inflation (pounds)	-0.626351	0.227437	1.000000

Conclusion:

Have house prices increased overtime? Yes! Average house price adj. by inflation have gone up by: £3920.94
Have salaries increased? No Median Salary adj. by inflation have gone down by: £-149.39
Is there a correlation between house prices and salaries? Yes a negative correlation as the house prices are increasing while the income is decreasing

Another Dataset from GOV.UK

Data source : https://www.gov.uk/government/collections/uk-house-price-index-reports

In [12]:
```
#importing the dataset
df1 = pd.read_excel(r'Datasets/income_HP_govuk.xlsx')
df1
```

Out[12]:

	year	quarter	average_price_new_dwelling	average_advance_new_dwelling	Average_income_new_dwelling	average_price_other_dwellings
0	1992	Q2	77360.000000	51950	24149	60210.000000
1	1992	Q3	69893.000000	49558	24049	62513.000000
2	1992	Q4	70043.000000	50374	23462	57193.000000
3	1993	Q1	72700.000000	51911	24440	58345.000000
4	1993	Q2	73289.000000	51219	23291	60196.000000
...	...	...	...	...	...	...
114	2020	Q4	325559.000000	213427	65829	325609.333333
115	2021	Q1	335948.333333	215643	66991	346924.666667
116	2021	Q2	328260.000000	212524	66251	345131.333333
117	2021	Q3	298342.333333	197523	61645	294768.333333
118	2021	Q4	311593.666667	207139	63308	310860.000000

119 rows × 17 columns

In [27]:
```
#checking the shape of dataset
df1.shape
```

Out[27]:
```
(119, 17)
```

In [13]:
```
#checking the statistics of the dataset
df1.describe()
```

Out[13]:

	year	average_price_new_dwelling	average_advance_new_dwelling	Average_income_new_dwelling	average_price_other_dwellings
count	119.000000	119.000000	119.000000	119.000000	119.000000
mean	2006.621849	196444.111731	128968.865546	47887.764706	186690.782069
std	8.624946	82279.878773	51671.723409	13887.346746	88209.771830
min	1992.000000	69893.000000	49558.000000	23291.000000	57193.000000
25%	1999.000000	114365.000000	79796.000000	35735.500000	94230.500000
50%	2007.000000	212554.000000	133585.000000	51121.000000	217128.000000
75%	2014.000000	267693.666667	175222.000000	61512.500000	264488.368511
max	2021.000000	335948.333333	215643.000000	68068.000000	346924.666667

In [14]:
```
#checking the null values
df1.isna().sum()
```

Out[14]:
```
year                              0
quarter                           0
average_price_new_dwelling        0
average_advance_new_dwelling      0
Average_income_new_dwelling       0
average_price_other_dwellings     0
average_advance_other_dwellings   0
average_income_other_dwellings    0
average_price_all_dwellings       0
average_advance_all_dwellings     0
average_income_all_dwellings      0
average_price_first_time          0
average_advance_first_time        0
average_income_first_time         0
average_price_former_owner        0
average_advance_former_owner      0
average_income_former_owner       0
dtype: int64
```

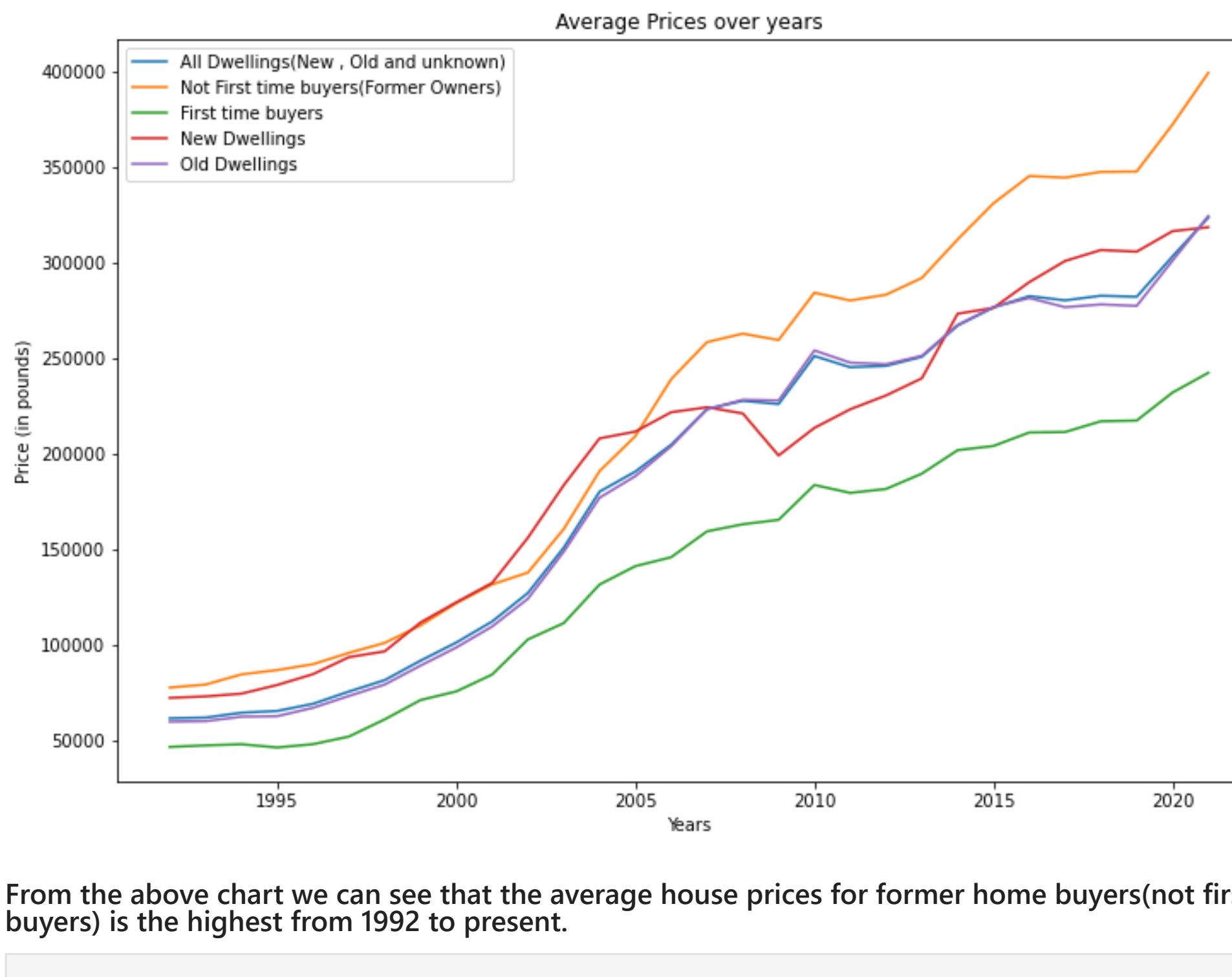
In [15]:
```
# group by the quarters to check the average yearly values
df_group_year = df1.groupby('year').mean()
df_group_year
```

Out[15]:

	average_price_new_dwelling	average_advance_new_dwelling	Average_income_new_dwelling	average_price_other_dwellings
year				
1992	72432.000000	50627.333333	23886.666667	59972.000000
1993	73183.500000	51480.250000	23836.000000	60236.250000
1994	74618.750000	53205.250000	24408.500000	62566.750000
1995	79163.250000	57516.750000	26896.250000	62844.750000
1996	84825.500000	61111.750000	28559.750000	67241.500000
1997	93710.750000	65775.250000	30713.250000	73403.500000
1998	96732.250000	69693.250000	31937.750000	79375.500000
1999	111827.250000	77298.250000	35101.750000	89282.500000
2000	122412.500000	82857.000000	36693.750000	98770.500000
2001	132537.000000	91195.000000	39351.750000	109722.000000
2002	156188.500000	102949.000000	43650.500000	124255.250000
2003	183620.000000	119877.000000	47879.500000	148822.750000
2004	208103.250000	135609.500000	51000.000000	177060.250000
2005	211665.750000	138199.250000	52055.250000	188373.000000
2006	221794.250000	147043.000000	54452.500000	203977.250000
2007	224379.986524	147986.500000	53673.750000	223313.381396
2008	221166.255466	140263.500000	52195.250000	228308.748390
2009	199165.897522	123730.250000	49049.250000	227945.887409
2010	213603.613406	132140.750000	51461.000000	254100.700129
2011	223334.216218	139415.250000	52094.000000	247725.439438
2012	230558.833638	147308.500000	53298.250000	246899.035498
2013	239527.482275	156372.250000	57400.750000	251278.652386
2014	273304.769338	177542.500000	66388.000000	267422.628695
2015	276358.498333	179956.500000	63568.750000	276834.750000
2016	289810.607264	189215.500000	63461.000000	281516.143863
2017	300893.667423	194603.500000	62616.750000	276733.157050
2018	306612.649845	199401.250000	62598.750000	278188.803596
2019	305771.013404	200659.000000	62738.750000	277406.438693
2020	316482.250000	208240.250000	65116.500000	301045.166667
2021	318536.083333	208207.250000	64548.750000	324421.083333

In [16]:
```
#plotting the average prices amoung dwellings over years
plt.figure(figsize=(12, 8))
plt.plot(df_group_year.average_price_all_dwellings, label='All Dwellings(New , Old and unknown)')
plt.plot(df_group_year.average_price_former_owner, label ="Not First time buyers(Former Owners)")
plt.plot(df_group_year.average_price_first_time, label ="First time buyers")
plt.plot(df_group_year.average_price_new_dwelling, label = 'New Dwellings')
plt.plot(df_group_year.average_price_other_dwellings, label ='Old Dwellings')
plt.title('Average Prices over years')
plt.legend()
plt.xlabel('Years')
plt.ylabel('Price (in pounds)')
```

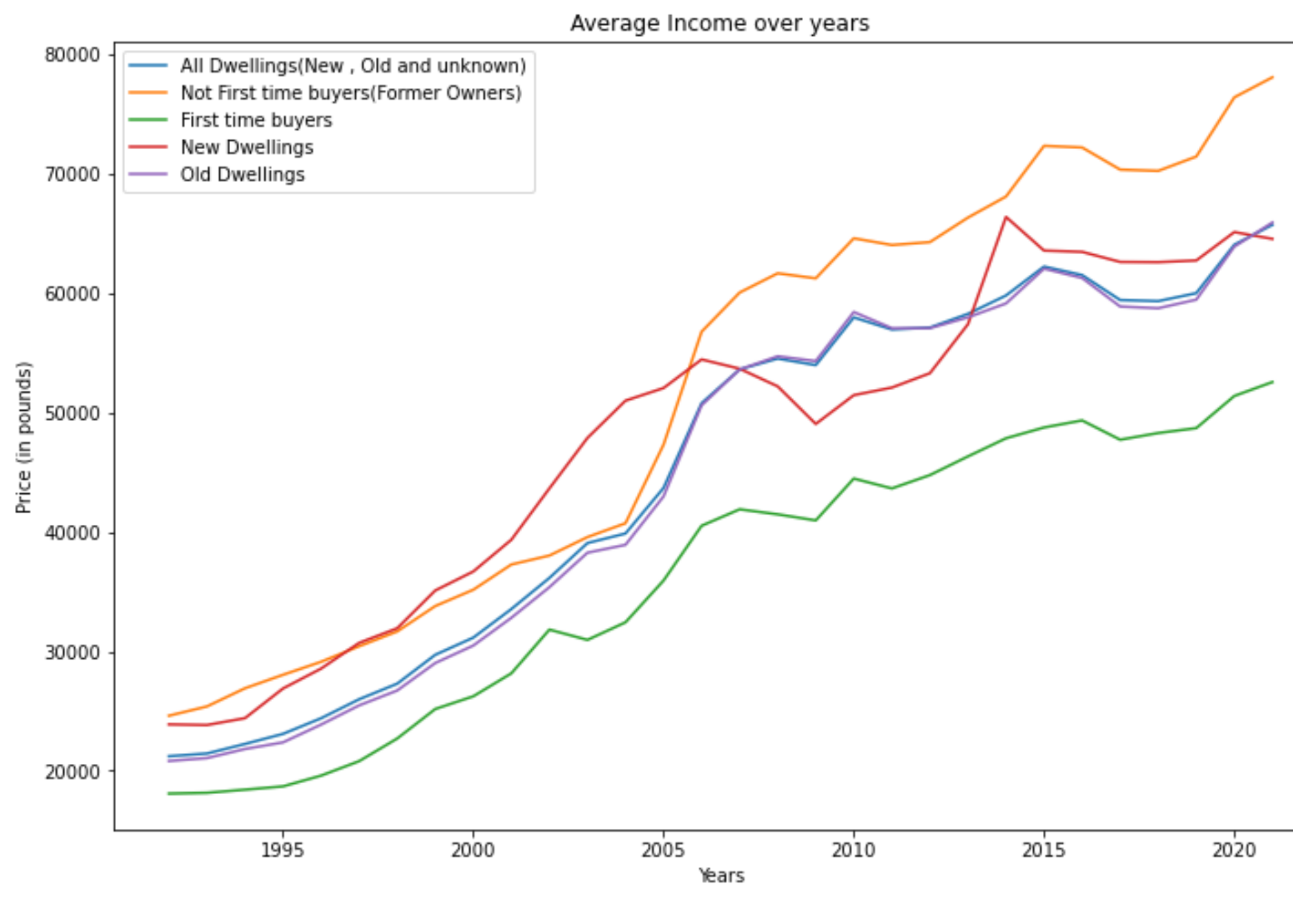
Out[16]:
```
Text(0, 0.5, 'Price (in pounds)')
```



From the above chart we can see that the average house prices for former home buyers(not first time home buyers) is the highest from 1992 to present.

In [17]:
```
#plotting the average prices amoung dwellings over years
plt.figure(figsize=(12, 8))
plt.plot(df_group_year.average_income_all_dwellings, label='All Dwellings(New , Old and unknown)')
plt.plot(df_group_year.average_income_former_owner, label ="Not First time buyers(Former Owners)")
plt.plot(df_group_year.average_income_first_time, label ="First time buyers")
plt.plot(df_group_year.Average_income_new_dwelling, label = 'New Dwellings')
plt.plot(df_group_year.average_income_other_dwellings, label ='Old Dwellings')
plt.title('Average Income over years')
plt.legend()
plt.xlabel('Years')
plt.ylabel('Price (in pounds)')
```

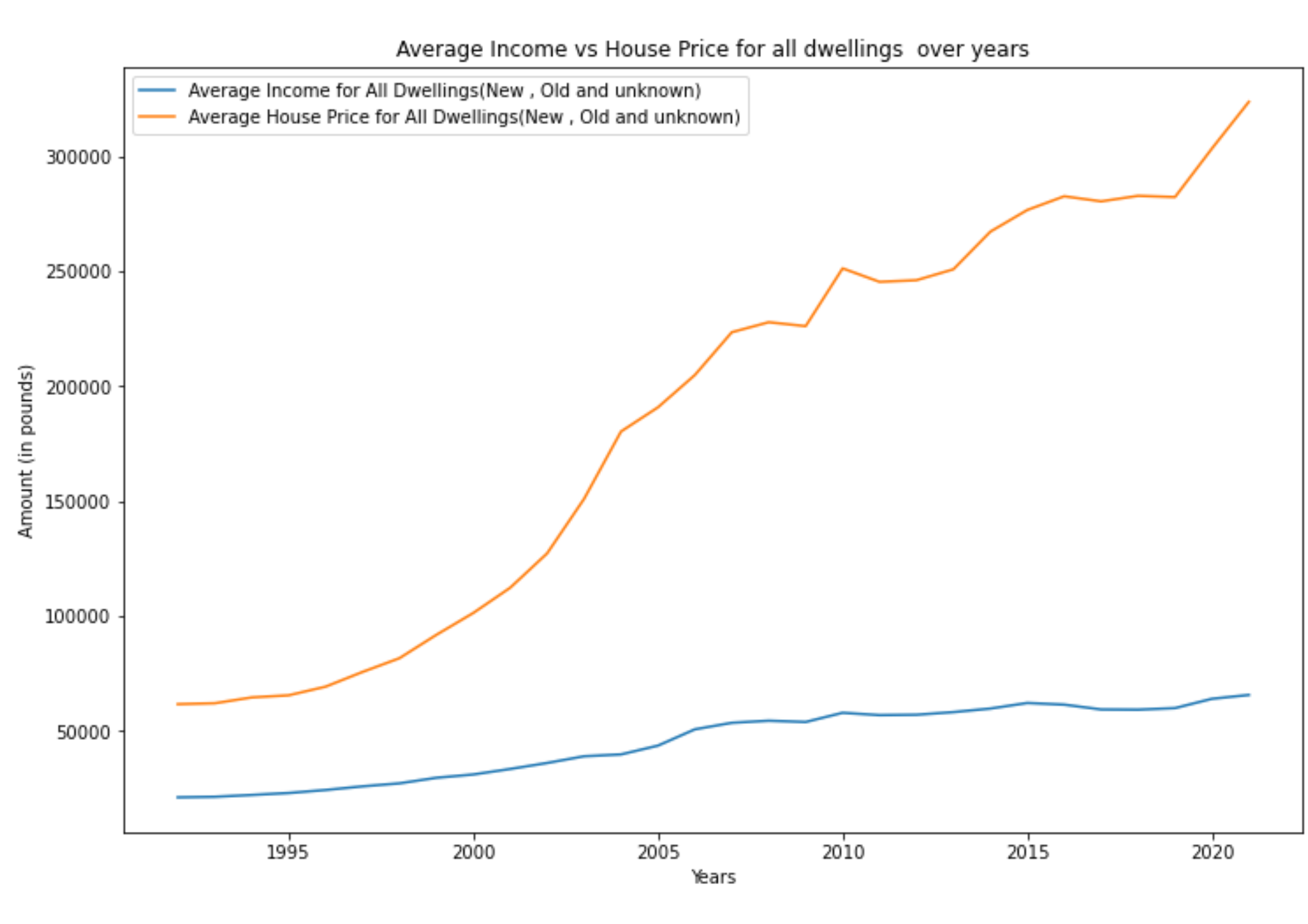
Out[17]:
```
Text(0, 0.5, 'Price (in pounds)')
```



From the above chart we can see that the average income for former home buyers(not first time home buyers) is the highest while the first time home buyers has the lowest.

In [18]:
```
#plotting the average income and house prices for all dwellings
plt.figure(figsize=(12, 8))
plt.plot(df_group_year.average_income_all_dwellings, label='Average Income for All Dwellings(New , Old and unkn
plt.plot(df_group_year.average_price_all_dwellings, label ='Average House Price for All Dwellings(New , Old an
plt.title('Average Income vs House Price for all dwellings  over years')
plt.legend()
plt.xlabel('Years')
plt.ylabel('Amount (in pounds)')
```

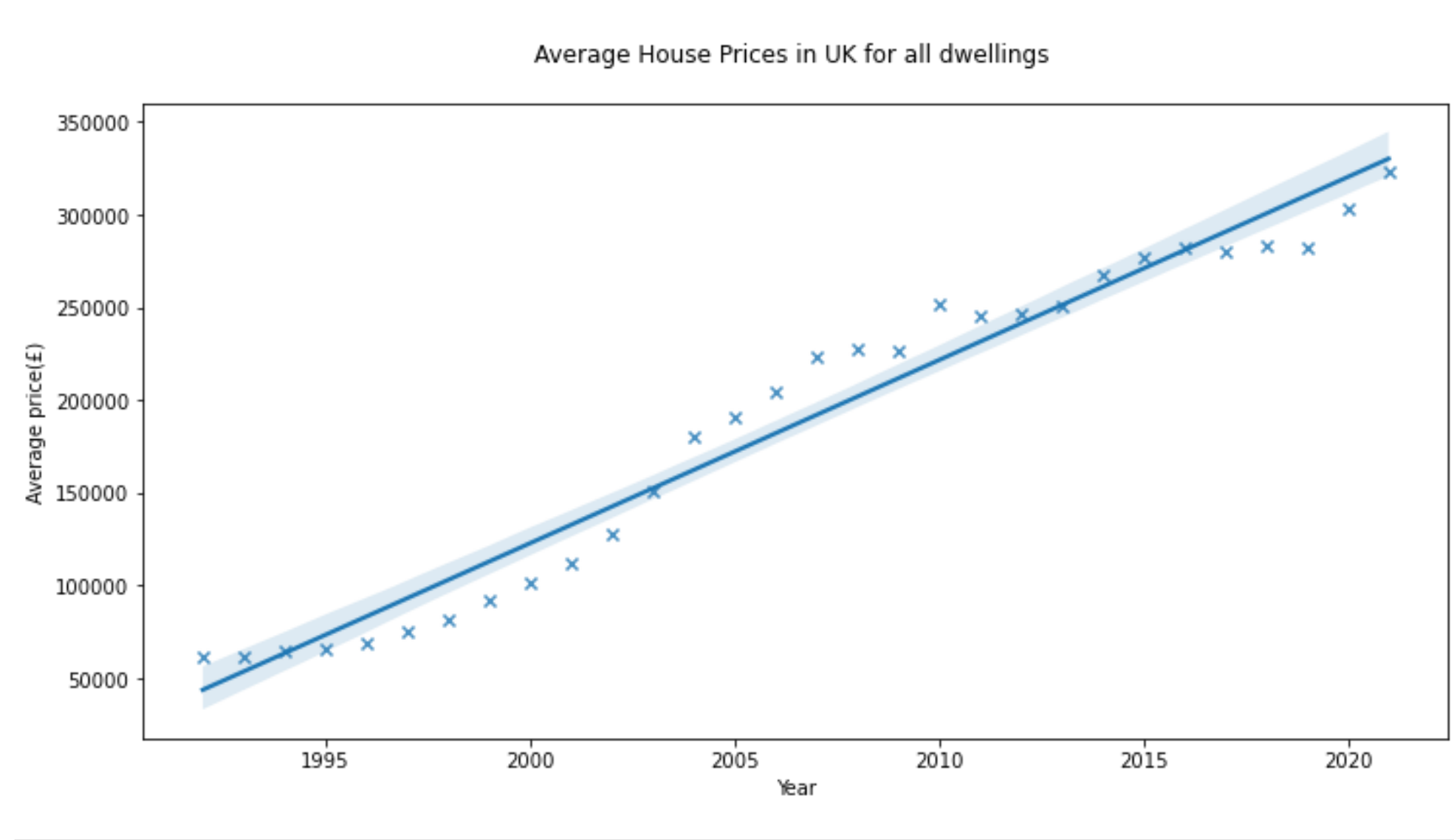
Out[18]:
```
Text(0, 0.5, 'Amount (in pounds)')
```



From the above chart we can see that over the years , income has increased a lot while the house price is not much increased.

In [19]:
```
# Plotting average house price for all dwellings over the years
df_group_year.reset_index(inplace=True)
plt.figure(figsize=(12,6))
sns.regplot(x='year', y='average_price_all_dwellings', data=df_group_year,marker='x')
plt.title('Average House Prices in UK for all dwellings \n')
plt.xlabel('Year')
plt.ylabel('Average price(£)')
```

Out[19]:
```
Text(0, 0.5, 'Average price(£)')
```



In [20]:
```
# Slope of regression line
slope, intercept, r_value, p_value, std_err = stats.linregress(df_group_year['year'],
                                                               df_group_year['average_price_all_dwellings'])

print('The slope of the regression line is: £','{:.2f}'.format(slope))
```

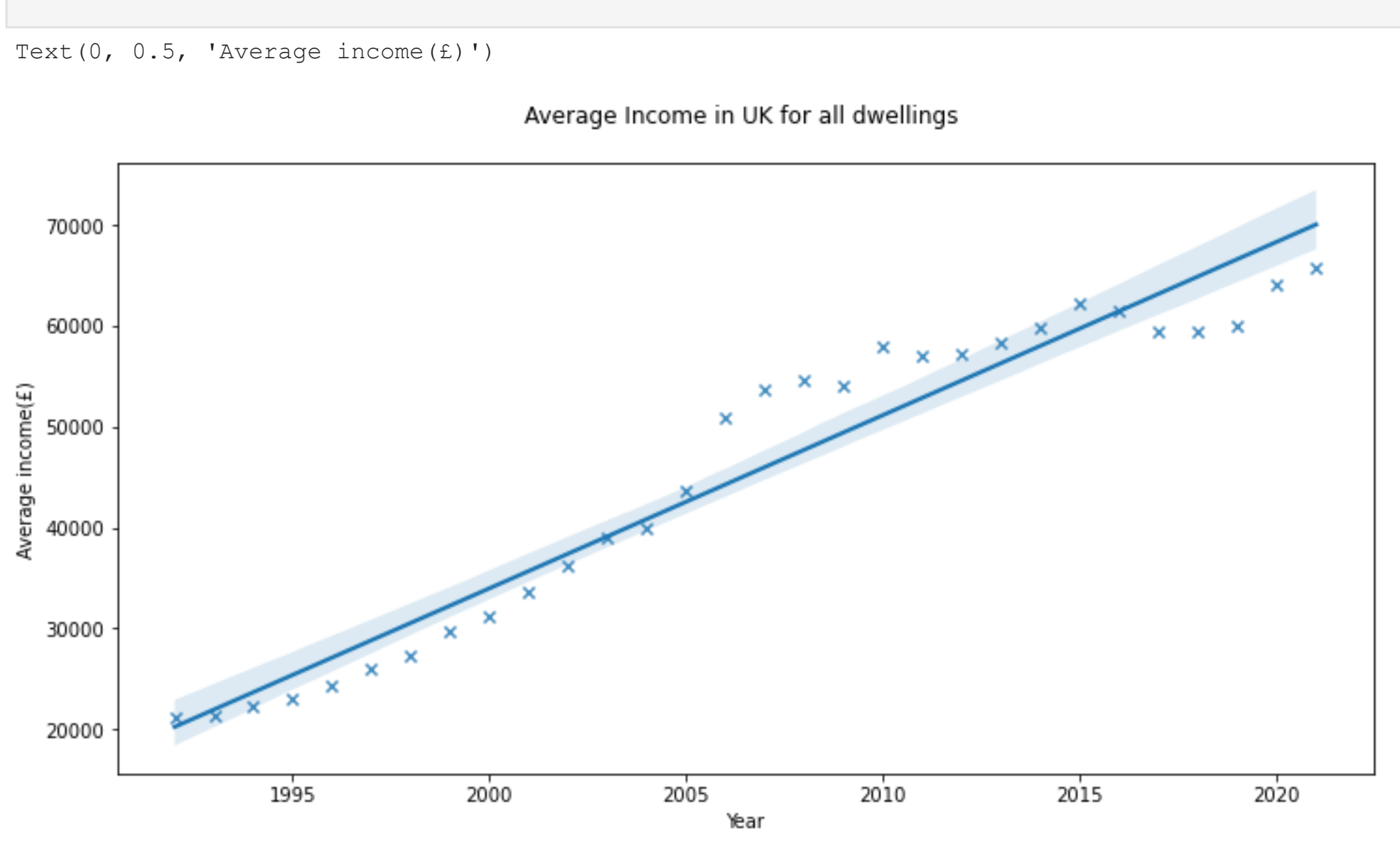
```
The slope of the regression line is: £ 9872.06
```

Observations:

The regression line tells us that houses prices are increasing each year on average by nearly £9872 a year!

In [21]:
```
# Plotting average income for all dwellings over the years

plt.figure(figsize=(12,6))
sns.regplot(x='year', y='average_income_all_dwellings', data=df_group_year,marker='x')
plt.title('Average Income in UK for all dwellings \n')
plt.xlabel('Year')
plt.ylabel('Average income(£)')
```

Out[21]:
```
Text(0, 0.5, 'Average income(£)')
```



In [22]:
```
# Slope of regression line
slope, intercept, r_value, p_value, std_err = stats.linregress(df_group_year['year'],
                                                               df_group_year['average_income_all_dwellings'])

print('The slope of the regression line is: £','{:.2f}'.format(slope))
```

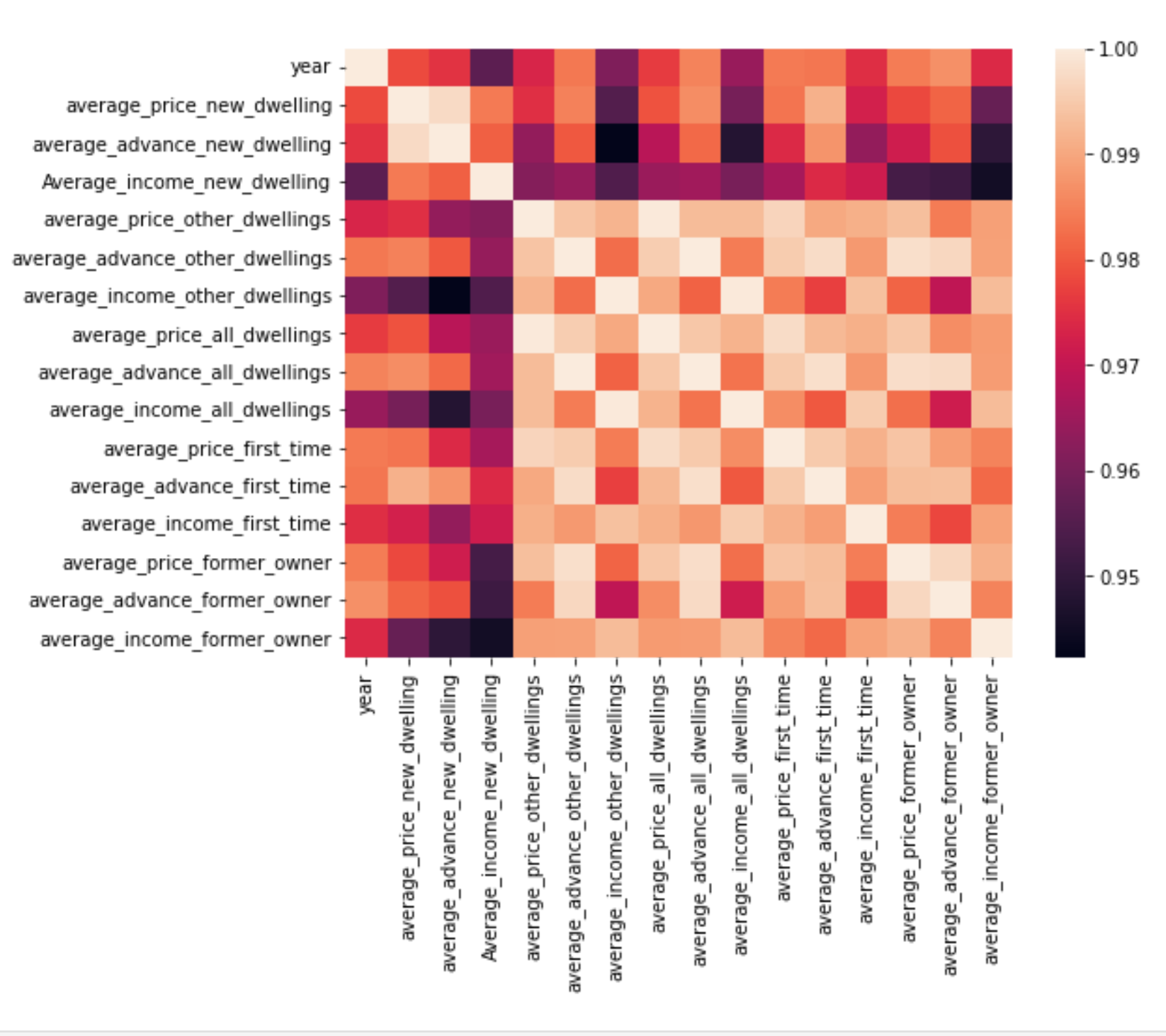
```
The slope of the regression line is: £ 1715.54
```

Observations:

The regression line tells us that income is increasing each year on average by nearly £1715 a year!

In [23]:
```
# checking the correlation between the columns
plt.subplots(figsize=(8, 6))
sns.heatmap(df1.corr())
```

Out[23]:
```
<AxesSubplot:>
```



In [24]:
```
# checking the correlation
df1.corr()
```

Out[24]:

	year	average_price_new_dwelling	average_advance_new_dwelling	Average_income_new_dwelling
year	1.000000	0.978211	0.975330	0.956173
average_price_new_dwelling	0.978211	1.000000	0.997508	0.983804
average_advance_new_dwelling	0.975330	0.997508	1.000000	0.980826
Average_income_new_dwelling	0.956173	0.983804	0.980826	1.000000
average_price_other_dwellings	0.973362	0.974947	0.963733	0.961594
average_advance_other_dwellings	0.983579	0.984847	0.979743	0.963957
average_income_other_dwellings	0.960810	0.954942	0.942338	0.954308
average_price_all_dwellings	0.976465	0.979181	0.968680	0.964550
average_advance_all_dwellings	0.984940	0.986395	0.981837	0.965433
average_income_all_dwellings	0.964286	0.959879	0.947827	0.959939
average_price_first_time	0.983872	0.983196	0.973977	0.966119
average_advance_first_time	0.983470	0.991283	0.987360	0.974082
average_income_first_time	0.974754	0.972419	0.963521	0.971504
average_price_former_owner	0.984120	0.978043	0.971696	0.952886
average_advance_former_owner	0.986818	0.981154	0.979029	0.951763
average_income_former_owner	0.973905	0.957639	0.949218	0.945100

In [25]:
```
#checking the correlation between income of all dwellings and average price all dwellings
df1.average_income_all_dwellings.corr(df1.average_price_all_dwellings)
```

Out[25]:
```
0.991511556641327
```

We can see from the above numbers that the correlation between income of all dwellings and average price all dwellings is highly correlated to each other.

Hypothesis II: Inflation vs House Price Index

Types of Inflation measure :

```
- RPI (Retail Price Index)
- CPI (Consumer Price Index)
```

Retail Price Index (RPI)

The Retail Price Index (RPI) is one of the two main measures of consumer inflation produced by the United Kingdom's Office for National Statistics (ONS). It is not considered an official statistic by the U.K., but it is used for certain types of cost escalation. The RPI was introduced in the U.K. in 1947, and it was made official in 1956.

In [120…

```
#importing various libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from statsmodels.tsa.stattools import grangercausalitytests, adfuller
from pandas.plotting import lag_plot
from statsmodels.tsa.api import VAR
from scipy.stats.stats import kendalltau
```

In [5]:

```
#importing the RPI data
header = ['Date','RPI']
df_RPI = pd.read_excel(r'Datasets/RPI.xlsx',names=header,header=7)
df_RPI
```

Out[5]:

	Date	RPI
0	1987-01-01	100.0
1	1987-02-01	100.4
2	1987-03-01	100.6
3	1987-04-01	101.8
4	1987-05-01	101.9
...	...	...
421	2022-02-01	320.2
422	2022-03-01	323.5
423	2022-04-01	334.6
424	2022-05-01	337.1
425	2022-06-01	340.0

426 rows × 2 columns

In [6]:

```
# checking the statistics of the dataset loaded
df_RPI.describe()
```

Out[6]:

	RPI
count	426.000000
mean	198.366432
std	58.890449
min	100.000000
25%	150.300000
50%	188.350000
75%	250.750000
max	340.000000

We can see from the above table , that maximum RPI is 340 whereas the minium is 100 with a mean of 198.36 and standard deviation of 58.89

In [7]:

```
#checking the shape of dataset
df_RPI.shape
```

Out[7]:

```
(426, 2)
```

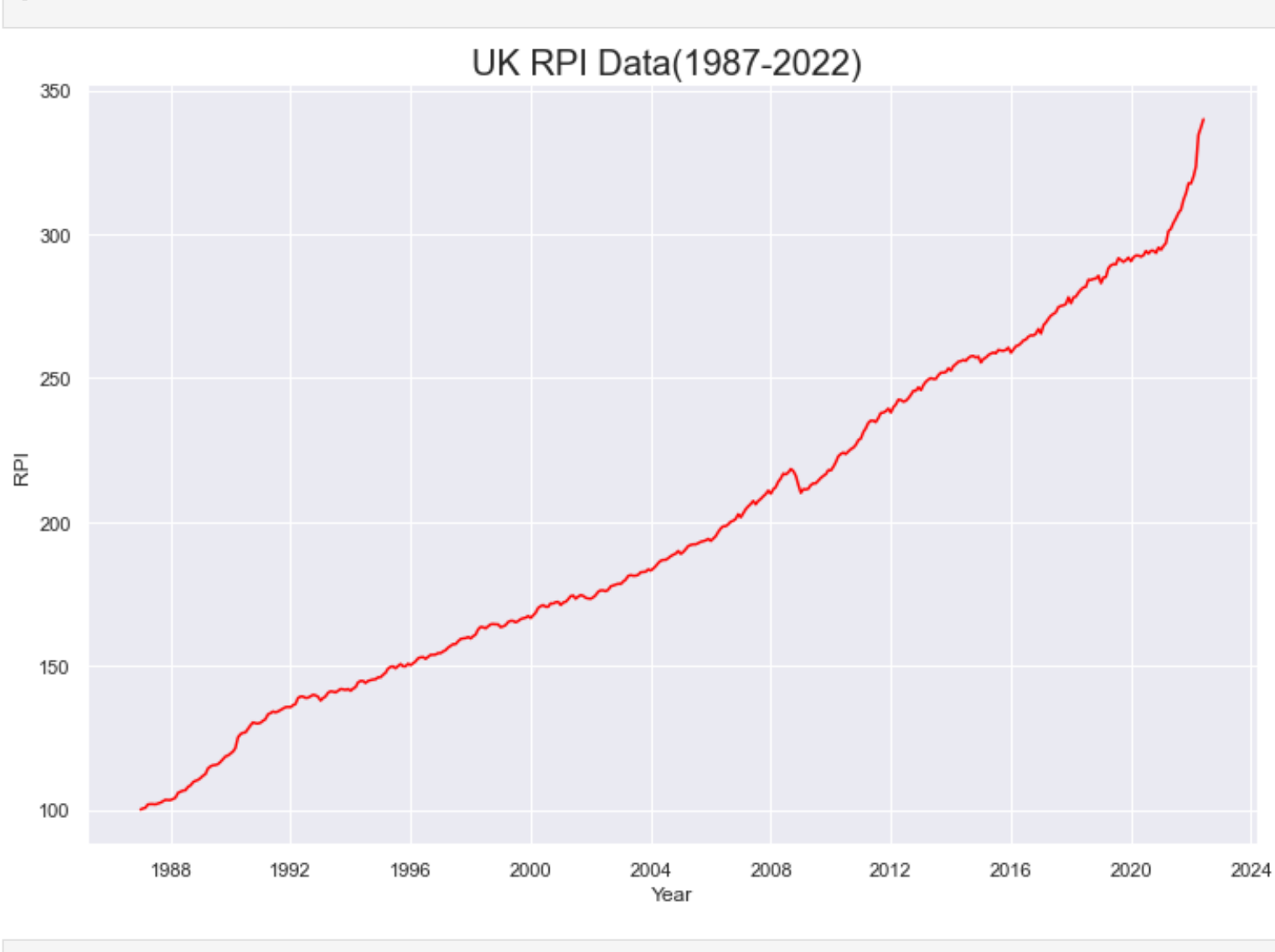
In [8]:

```
#checking the dataset's to and from dates
print("Starting date of RPI data :",df_RPI.Date.min())
print("Ending date of RPI data :",df_RPI.Date.max())
```

```
Starting date of RPI data : 1987-01-01 00:00:00
Ending date of RPI data : 2022-06-01 00:00:00
```

In [10]:

```
#plotting the RPI data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_RPI, x='Date',y='RPI',err_style='bars',color='red')
plt.title("UK RPI Data(1987-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('RPI')
plt.show()
```



In [12]:

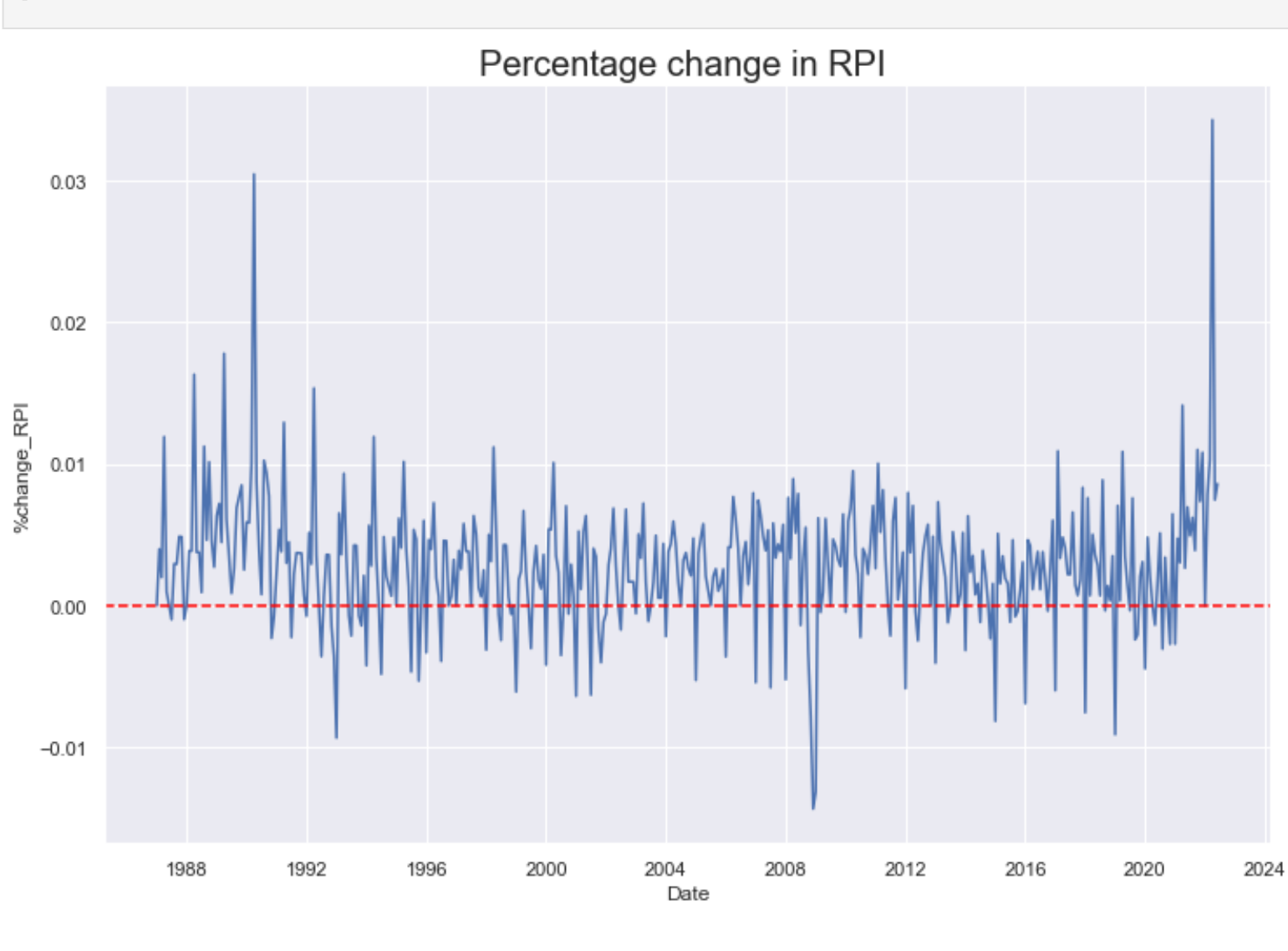
```
#calculating the percentage change in RPI
df_RPI['%change_RPI'] =df_RPI['RPI'].pct_change()
df_RPI.fillna(0,inplace=True)
df_RPI.head()
```

Out[12]:

	Date	RPI	%change_RPI
0	1987-01-01	100.0	0.000000
1	1987-02-01	100.4	0.004000
2	1987-03-01	100.6	0.001992
3	1987-04-01	101.8	0.011928
4	1987-05-01	101.9	0.000982

In [13]:

```
# plotting the percentage change in RPI
plt.figure(figsize = (12,8))
sns.lineplot(data=df_RPI,x='Date',y='%change_RPI')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in RPI ",fontdict={'fontsize': 20})
plt.show()
```



Consumer Price Index (CPI)

Data source : https://www.rateinflation.com/consumer-price-index/uk-historical-cpi/

In [14]:

```
# importing the dataset
df_CPI = pd.read_excel(r'Datasets/CPI.xlsx')
df_CPI.head()
```

Out[14]:

	Date	CPI
0	1988-01-01	48.4
1	1988-02-01	48.5
2	1988-03-01	48.7
3	1988-04-01	49.3
4	1988-05-01	49.5

In [15]:

```
# checking the statistics of the dataset loaded
df_CPI.describe()
```

Out[15]:

	CPI
count	414.000000
mean	81.673188
std	17.697839
min	48.400000
25%	69.050000
50%	77.650000
75%	99.100000
max	121.800000

We can see from the above table , that maximum RPI is 121.80 whereas the minium is 48.40 with a mean of 81.67 and standard deviation of 17.69

In [16]:

```
#checking the shape of dataset
df_CPI.shape
```

Out[16]:

```
(414, 2)
```

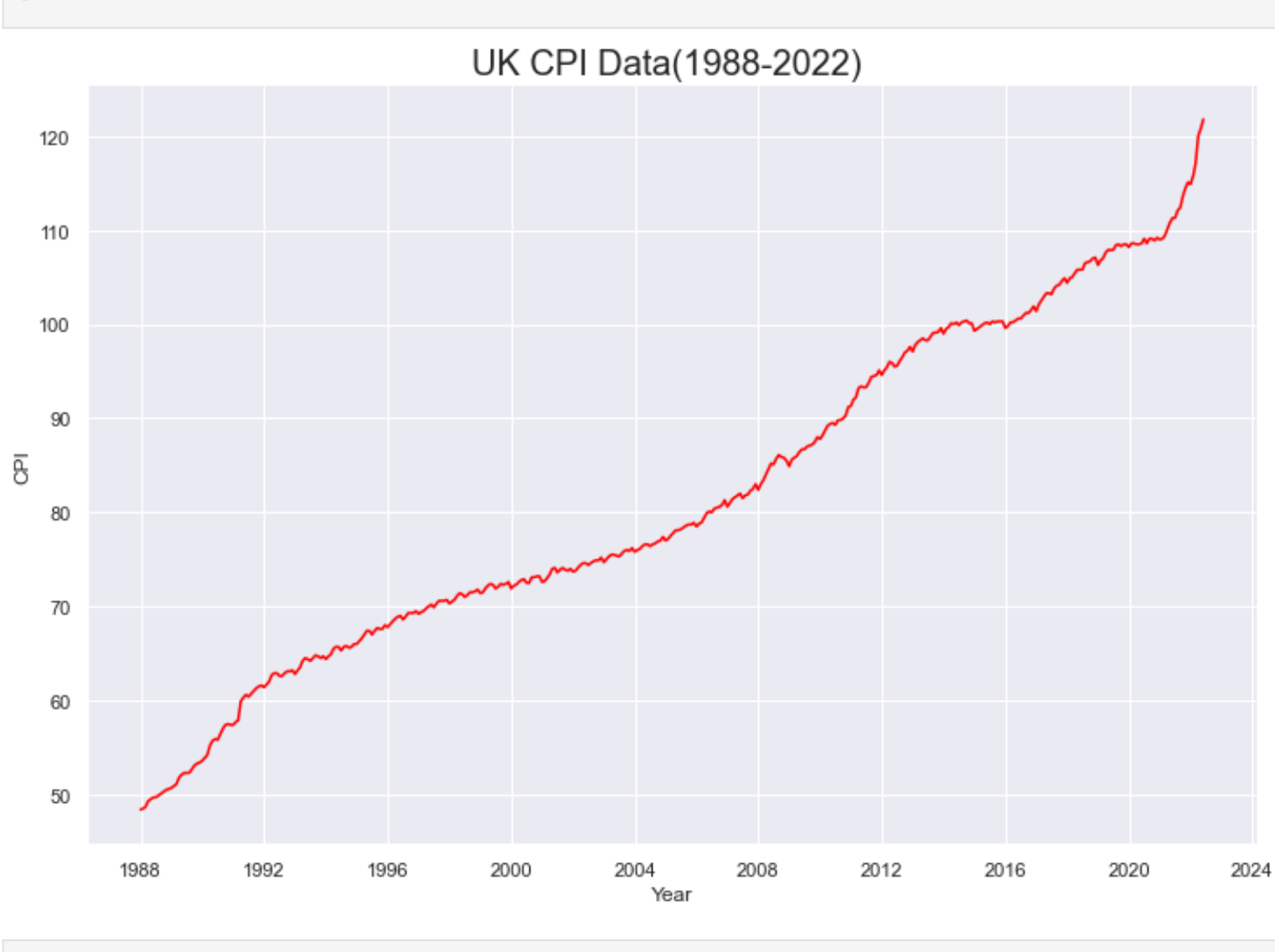
In [17]:

```
#checking the datasets to and from dates
print("Starting date of CPI data :",df_CPI.Date.min())
print("Ending date of CPI data :",df_CPI.Date.max())
```

```
Starting date of CPI data : 1988-01-01 00:00:00
Ending date of CPI data : 2022-06-01 00:00:00
```

In [18]:

```
#plotting the RPI data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_CPI, x='Date',y='CPI',err_style='bars',color='red')
plt.title("UK CPI Data(1988-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('CPI')
plt.show()
```



In [19]:

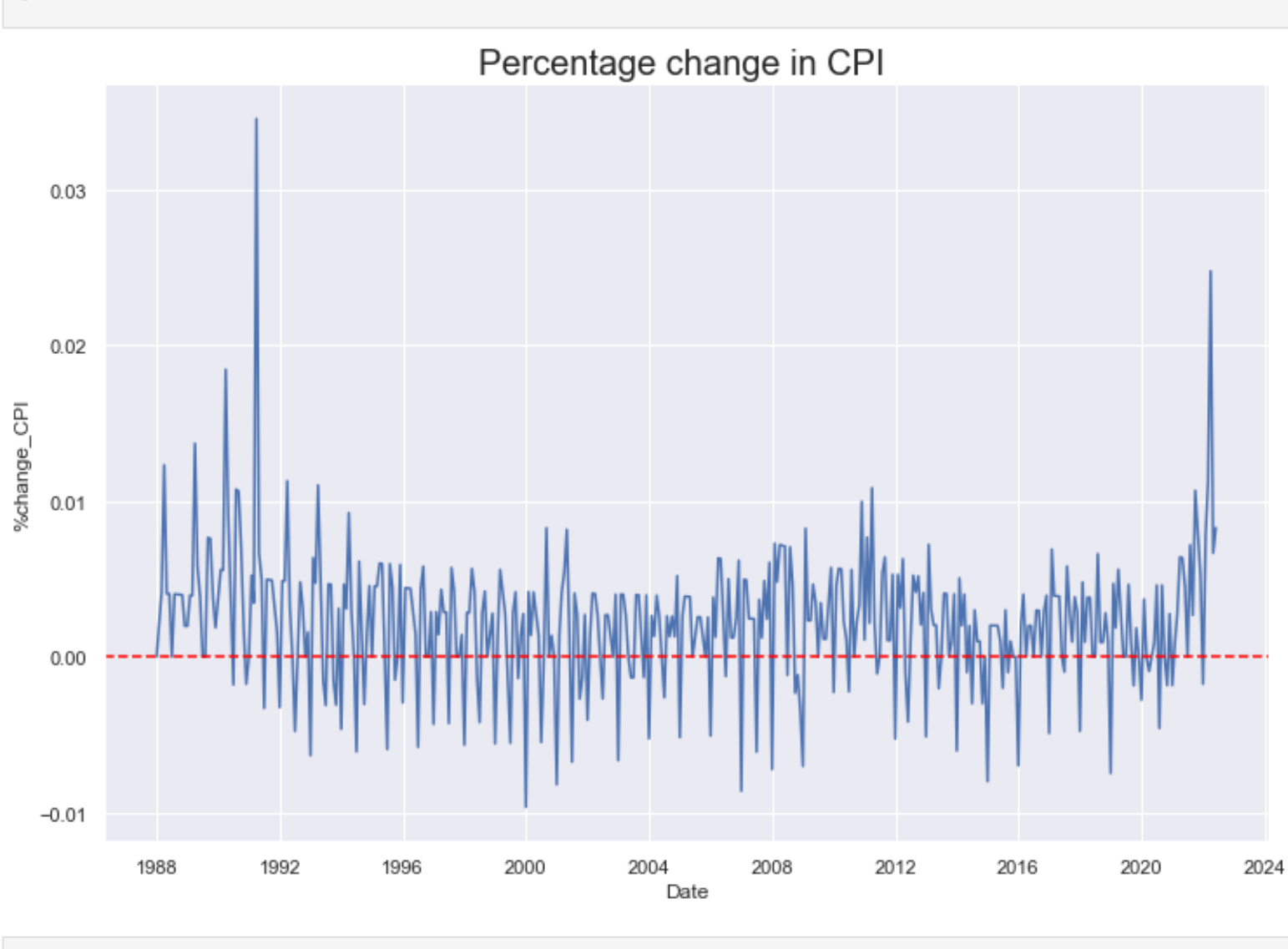
```
#calculating the percentage change in RPI
df_CPI['%change_CPI'] =df_CPI['CPI'].pct_change()
df_CPI.fillna(0,inplace=True)
df_CPI.head()
```

Out[19]:

	Date	CPI	%change_CPI
0	1988-01-01	48.4	0.000000
1	1988-02-01	48.5	0.002066
2	1988-03-01	48.7	0.004124
3	1988-04-01	49.3	0.012320
4	1988-05-01	49.5	0.004057

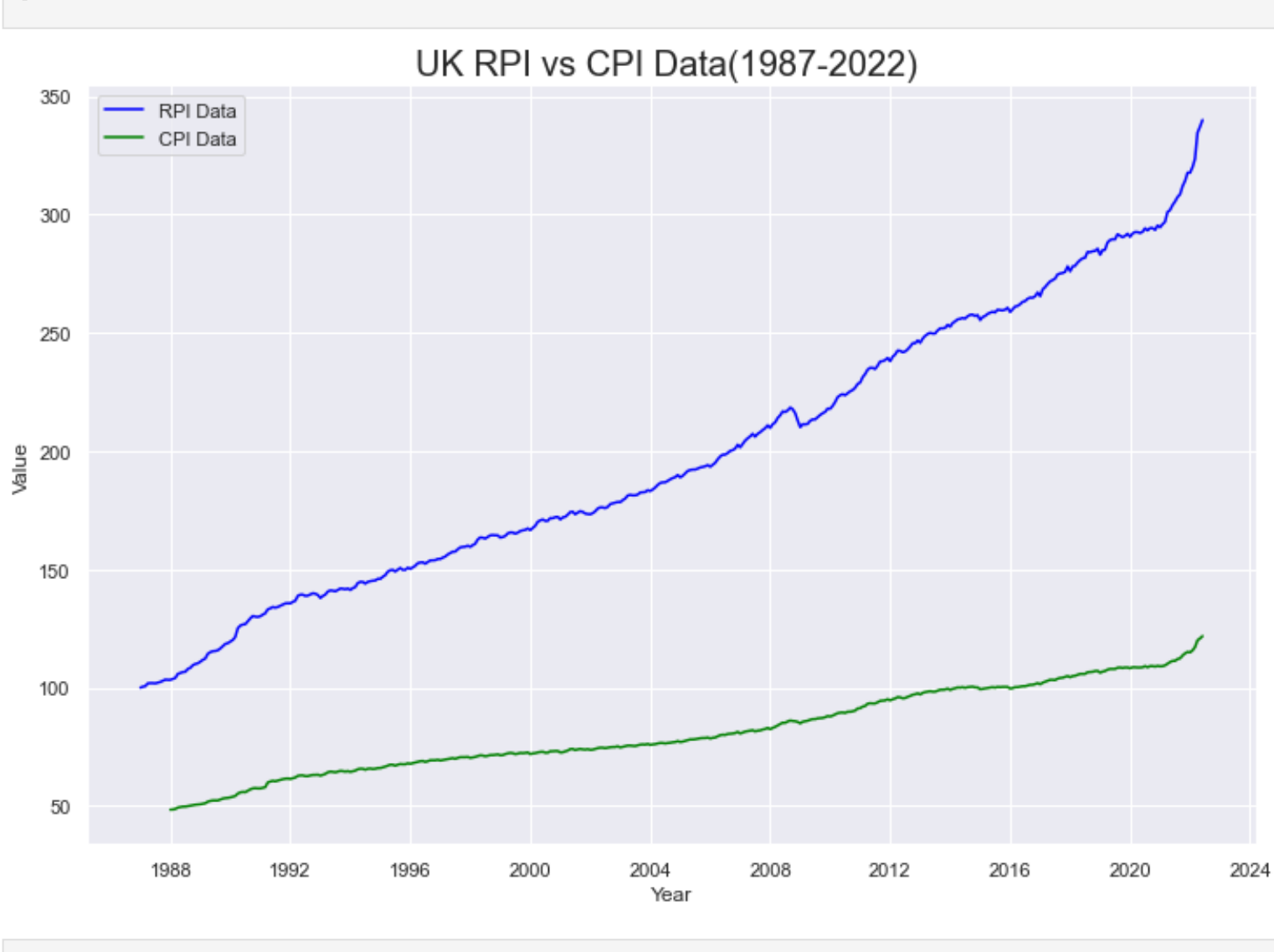
In [21]:

```
# plotting the percentage change in RPI
plt.figure(figsize = (12,8))
sns.lineplot(data=df_CPI,x='Date',y='%change_CPI')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in CPI ",fontdict={'fontsize': 20})
plt.show()
```



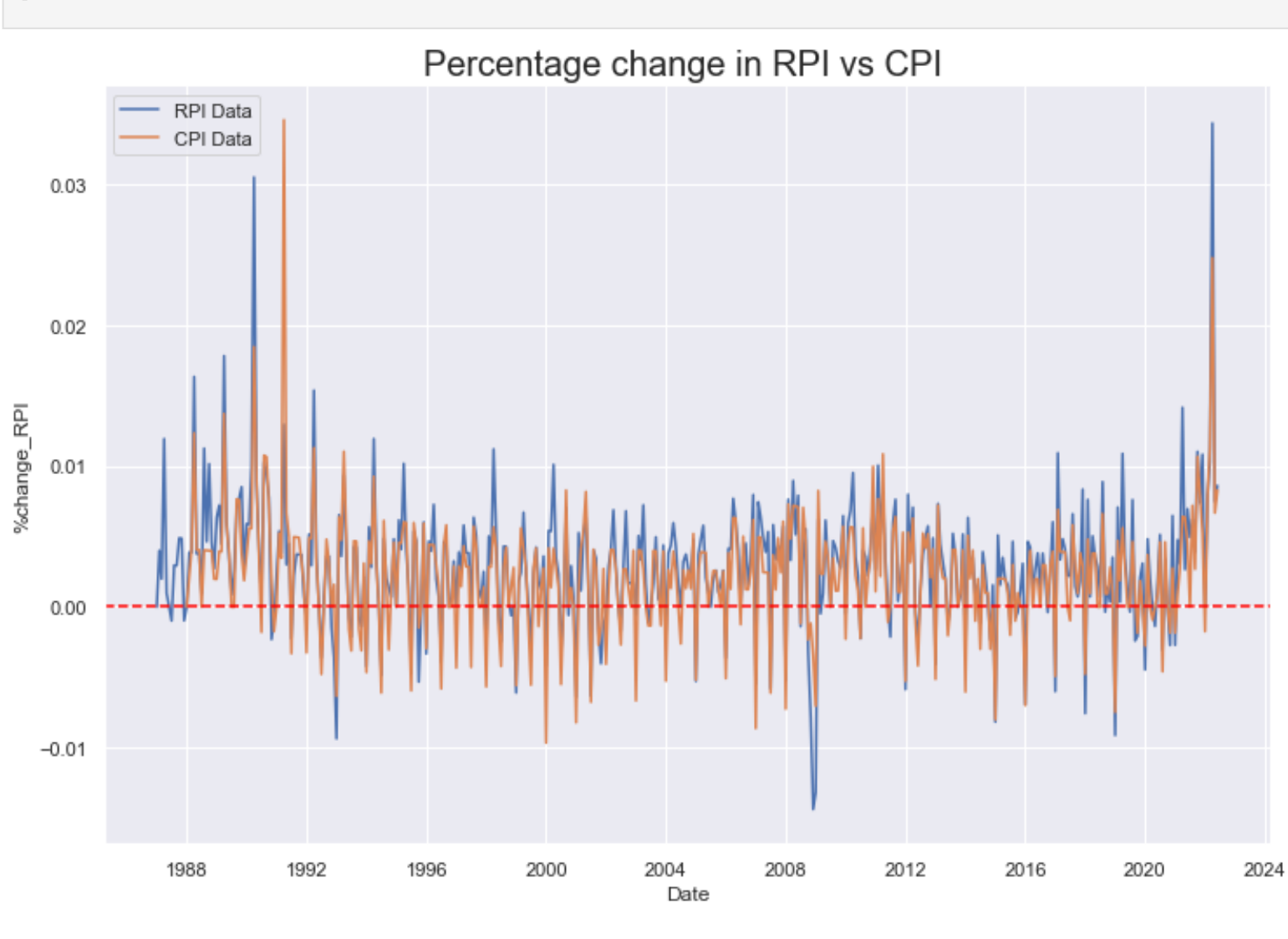
In [35]:

```
#plotting the RPI and CPI data together
plt.figure(figsize=(12,8))
sns.lineplot(data = df_RPI, x='Date',y='RPI',err_style='bars',color='blue',label='RPI Data')
sns.lineplot(data = df_CPI, x='Date',y='CPI',err_style='bars',color='green',label='CPI Data')
plt.title("UK RPI vs CPI Data(1987-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('Value')
plt.show()
```



In [34]:

```
# plotting the percentage change in RPI and CPI together
plt.figure(figsize = (12,8))
sns.lineplot(data=df_RPI,x='Date',y='%change_RPI',label='RPI Data')
sns.lineplot(data=df_CPI,x='Date',y='%change_CPI',label='CPI Data')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in RPI vs CPI ",fontdict={'fontsize': 20})
plt.show()
```



From the above chart , we can see that the percentage change in RPI is more than CPI.

UK HPI Data

In [71]:

```
#importing the dataset for HPI
df_HPI = pd.read_csv(r'indices_dataset/Indices-2022-05_ONS.csv')
# Since , we want for the United Kingdom , we would require to filter the region_name to united kingdom
df_HPI = df_HPI[df_HPI.Region_Name == 'United Kingdom']
# we require only date and index
df_HPI = df_HPI[['Date','Index']]
#Since the dataset of CPI and RPI is from 1988, thus, slicing the HPI dataset from 1988
df_HPI = df_HPI[df_HPI.Date>= '1988-01-01']
df_HPI.head()
```

Out[71]:

	Date	Index
2379	1988-01-01	22.805855
2389	1988-02-01	22.805855
2399	1988-03-01	22.805855
2409	1988-04-01	24.183390
2419	1988-05-01	24.183390

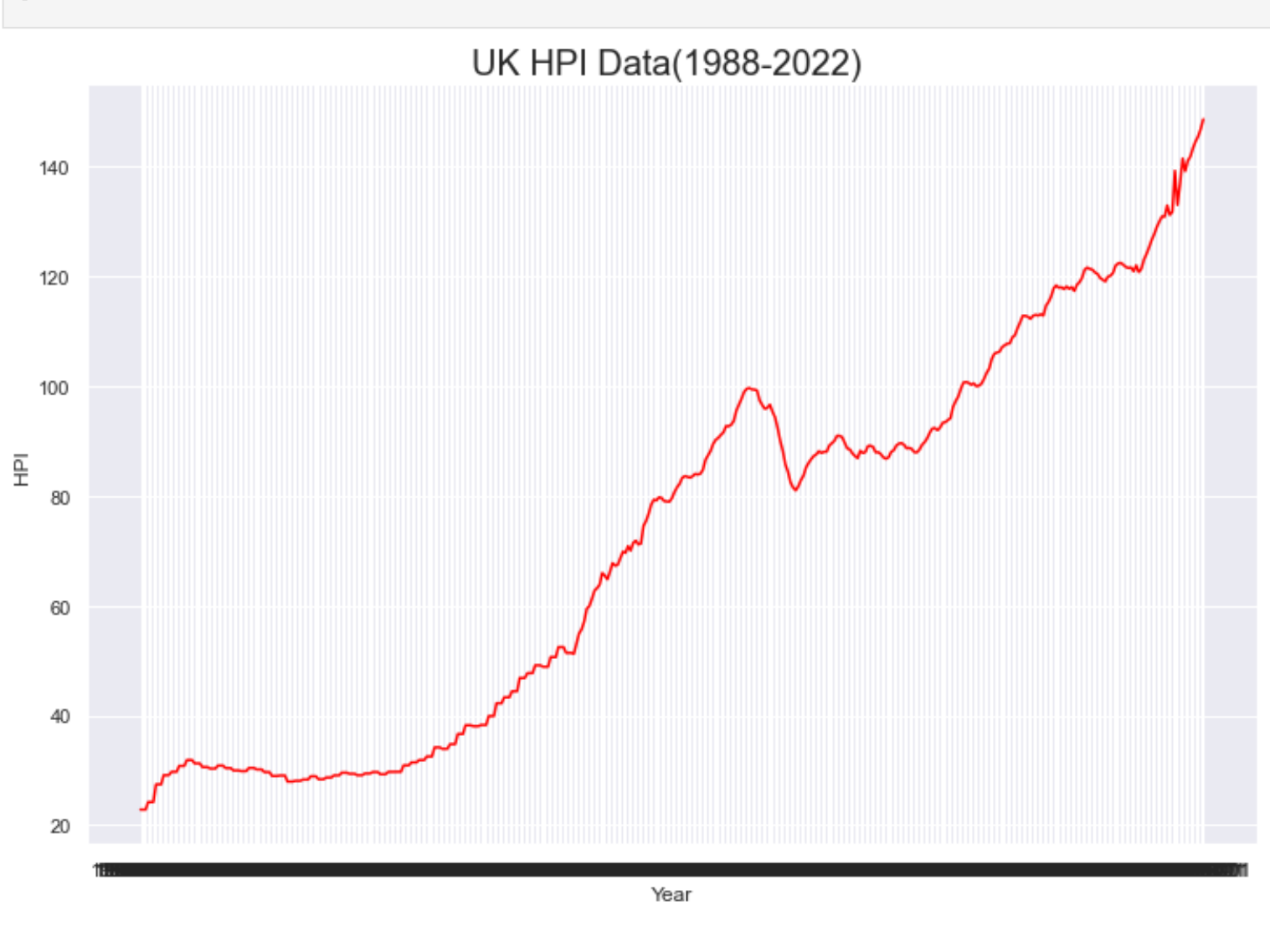
In [72]:

```
#checking the dataset's to and from dates
print("Starting date of HPI data :",df_HPI.Date.min())
print("Ending date of HPI data :",df_HPI.Date.max())
```

```
Starting date of HPI data : 1988-01-01
Ending date of HPI data : 2022-05-01
```

In [57]:

```
#plotting the RPI data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_HPI, x='Date',y='Index',err_style='bars',color='red')
plt.title("UK HPI Data(1988-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('HPI')
plt.show()
```



Checking the correlation between Inflation and HPI

In [73]:

```
# combing the datasets together
df_all = pd.DataFrame()
df_all[['Date','HPI']] = df_HPI[['Date','Index']]
df_all['CPI'] = df_CPI['CPI'][:-1].values
df_all['RPI']=df_RPI['RPI'][12:-1].values
df_all
```

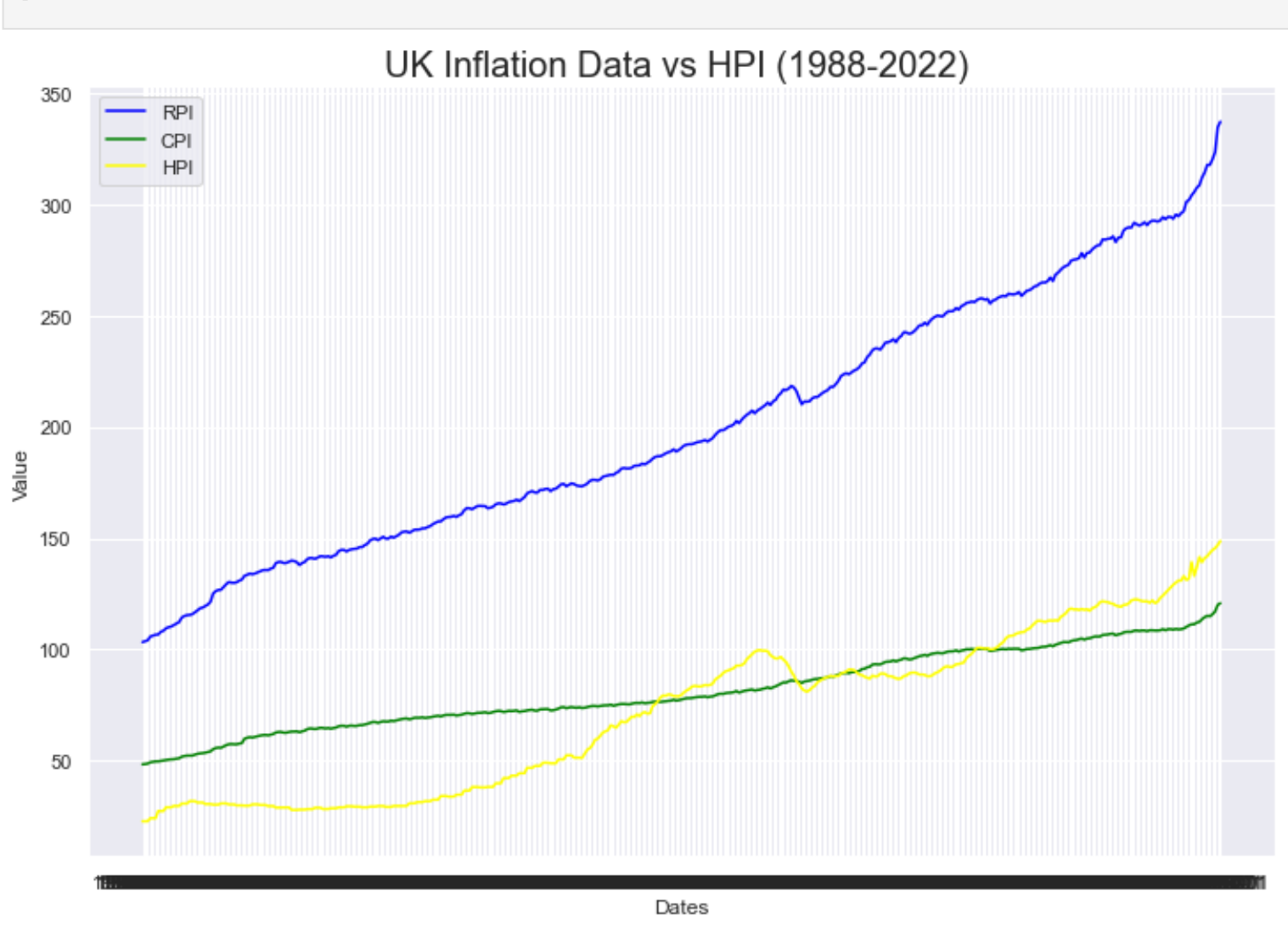
Out[73]:

	Date	HPI	CPI	RPI
2379	1988-01-01	22.805855	48.4	103.3
2389	1988-02-01	22.805855	48.5	103.7
2399	1988-03-01	22.805855	48.7	104.1
2409	1988-04-01	24.183390	49.3	105.8
2419	1988-05-01	24.183390	49.5	106.2
...	...	...	...	...
135249	2022-01-01	143.422582	114.9	317.7
135670	2022-02-01	144.646684	115.8	320.2
136091	2022-03-01	145.556596	117.1	323.5
136512	2022-04-01	146.898652	120.0	334.6
136933	2022-05-01	148.688246	120.8	337.1

413 rows × 4 columns

In [74]:

```
#plotting the RPI and CPI data together with HPI Data
plt.figure(figsize=(12,8))
sns.lineplot(data = df_all, x='Date',y='RPI',color='blue',label='RPI')
sns.lineplot(data = df_all, x='Date',y='CPI',color='green',label='CPI')
sns.lineplot(data = df_all, x='Date',y='HPI',color='yellow',label='HPI')
plt.title("UK Inflation Data vs HPI (1988-2022) ",fontdict={'fontsize':20})
plt.xlabel("Dates")
plt.ylabel('Value')
plt.show()
```

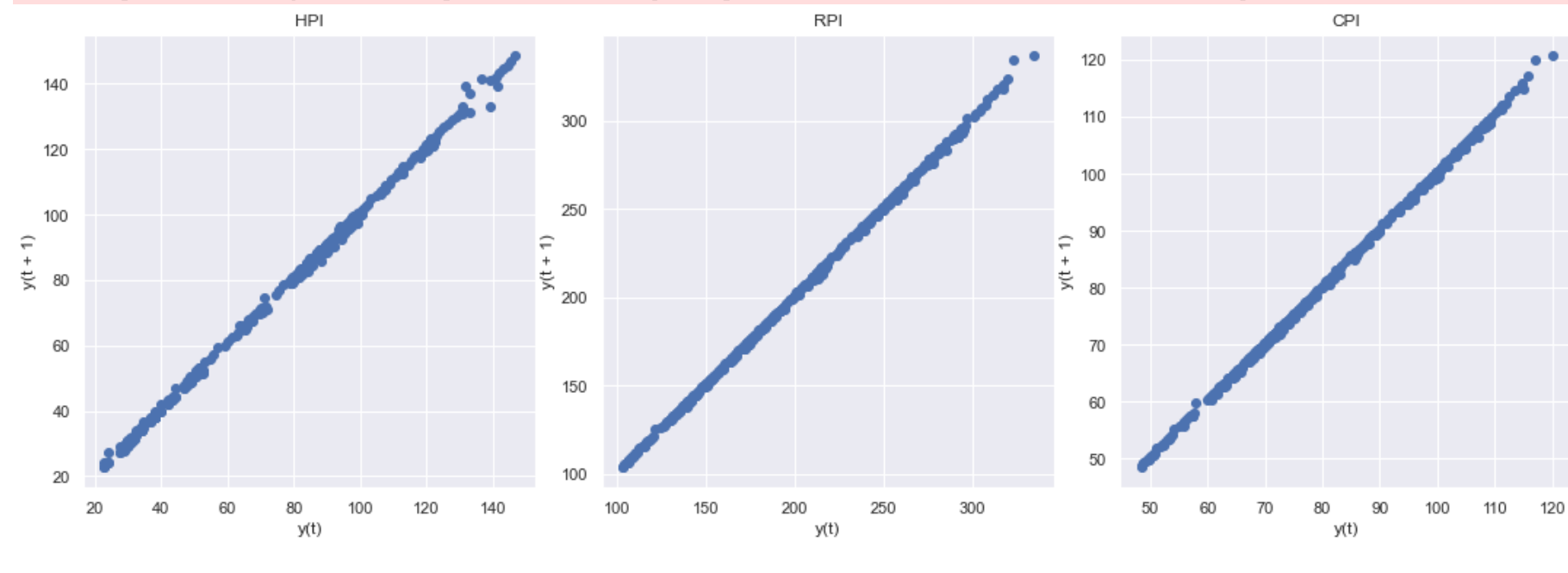


Step 1: Test each of the time-series to determine their order of integration. Ideally, this should involve using a test (such as the ADF test) for which the null hypothesis is non-stationarity.

In [59]:

```
#plotting the lag_plots for GDP and HPI
fig,axes = plt.subplots(1, 3, figsize=(15, 5))
fig.tight_layout()
lag_plot(df_all['HPI'], ax=axes[0])
axes[0].set_title('HPI')
lag_plot(df_all['RPI'], ax=axes[1])
axes[1].set_title('RPI')
lag_plot(df_all['CPI'], ax=axes[2])
axes[2].set_title('CPI')

plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



The above charts shows that it has high autocorelation.

In [75]:

```
# differencing the data
df1 = df_all.copy()
df_all['RPI'] = df_all['RPI']- df_all['RPI'].shift(1)
df_all['CPI'] = df_all['CPI']- df_all['CPI'].shift(1)
df_all['HPI'] = df_all['HPI']- df_all['HPI'].shift(1)
df_all = df_all.dropna()
df_all
```

Out[75]:

	Date	HPI	CPI	RPI
2389	1988-02-01	0.000000	0.1	0.4
2399	1988-03-01	0.000000	0.2	0.4
2409	1988-04-01	1.377535	0.6	1.7
2419	1988-05-01	0.000000	0.2	0.4
2429	1988-06-01	0.000000	0.2	0.4
...	...	...	...	...
135249	2022-01-01	1.530073	-0.2	0.0
135670	2022-02-01	1.224102	0.9	2.5
136091	2022-03-01	0.909912	1.3	3.3
136512	2022-04-01	1.342056	2.9	11.1
136933	2022-05-01	1.789594	0.8	2.5

412 rows × 4 columns

In [121…

```
#checking for co-relation on the data.
df1.corr(method='spearman')
```

Out[121…

	HPI	CPI	RPI
HPI	1.000000	0.966223	0.967465
CPI	0.966223	1.000000	0.999689
RPI	0.967465	0.999689	1.000000

In [88]:

```
#using kendalltau function to check the correlation
kendalltau(df1['HPI'],df1['CPI'])
```

Out[88]:

```
KendalltauResult(correlation=0.8659741606319639, pvalue=1.0577117502818923e-151)
```

In [124…

```
#using kendalltau function to check the correlation
kendalltau(df1['HPI'],df1['RPI'])
```

Out[124…

```
KendalltauResult(correlation=0.8718916363983841, pvalue=5.857463404643057e-154)
```

The above methods show that the series are correlated.

In [62]:

```
#defining a function for dicky fuller test
def adfuller_test(columns):
    result=adfuller(columns)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit ro
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-station
```

In [76]:

```
#testing for HPI
adfuller_test(df_all['HPI'])
```

```
ADF Test Statistic : -3.920995269948388
p-value : 0.0018856532414154793
#Lags Used : 16
Number of Observations Used : 395
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stati
onary
```

In [77]:

```
#testing for CPI
adfuller_test(df_all['CPI'])
```

```
ADF Test Statistic : -1.1690923610077042
p-value : 0.6867481275792435
#Lags Used : 16
Number of Observations Used : 395
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

In [78]:

```
#testing for RPI
adfuller_test(df_all['RPI'])
```

```
ADF Test Statistic : -0.46213541727518187
p-value : 0.8991961332547193
#Lags Used : 14
Number of Observations Used : 397
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

Since , the RPI and CPI is non-stationary , so differencing again and testing with ADF

In [79]:

```
df_all['CPI'] = df_all['CPI']- df_all['CPI'].shift(1)
df_all['RPI'] = df_all['RPI']- df_all['RPI'].shift(1)
df_all = df_all.dropna()
df_all
```

```
C:\Users\hp\AppData\Local\Temp/ipykernel_23040/1750697369.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df_all['CPI'] = df_all['CPI']- df_all['CPI'].shift(1)
C:\Users\hp\AppData\Local\Temp/ipykernel_23040/1750697369.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df_all['RPI'] = df_all['RPI']- df_all['RPI'].shift(1)
```

Out[79]:

	Date	HPI	CPI	RPI
2399	1988-03-01	0.000000	0.1	-1.421085e-14
2409	1988-04-01	1.377535	0.4	1.300000e+00
2419	1988-05-01	0.000000	-0.4	-1.300000e+00
2429	1988-06-01	0.000000	0.0	-1.421085e-14
2439	1988-07-01	3.214248	-0.2	-3.000000e-01
...	...	...	...	...
135249	2022-01-01	1.530073	-0.8	-3.400000e+00
135670	2022-02-01	1.224102	1.1	2.500000e+00
136091	2022-03-01	0.909912	0.4	8.000000e-01
136512	2022-04-01	1.342056	1.6	7.800000e+00
136933	2022-05-01	1.789594	-2.1	-8.600000e+00

411 rows × 4 columns

In [80]:

```
#testing for CPI
adfuller_test(df_all['CPI'])
```

```
ADF Test Statistic : -7.124151186354299
p-value : 3.657055882516593e-10
#Lags Used : 12
Number of Observations Used : 398
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stati
onary
```

In [81]:

```
#testing for RPI
adfuller_test(df_all['RPI'])
```

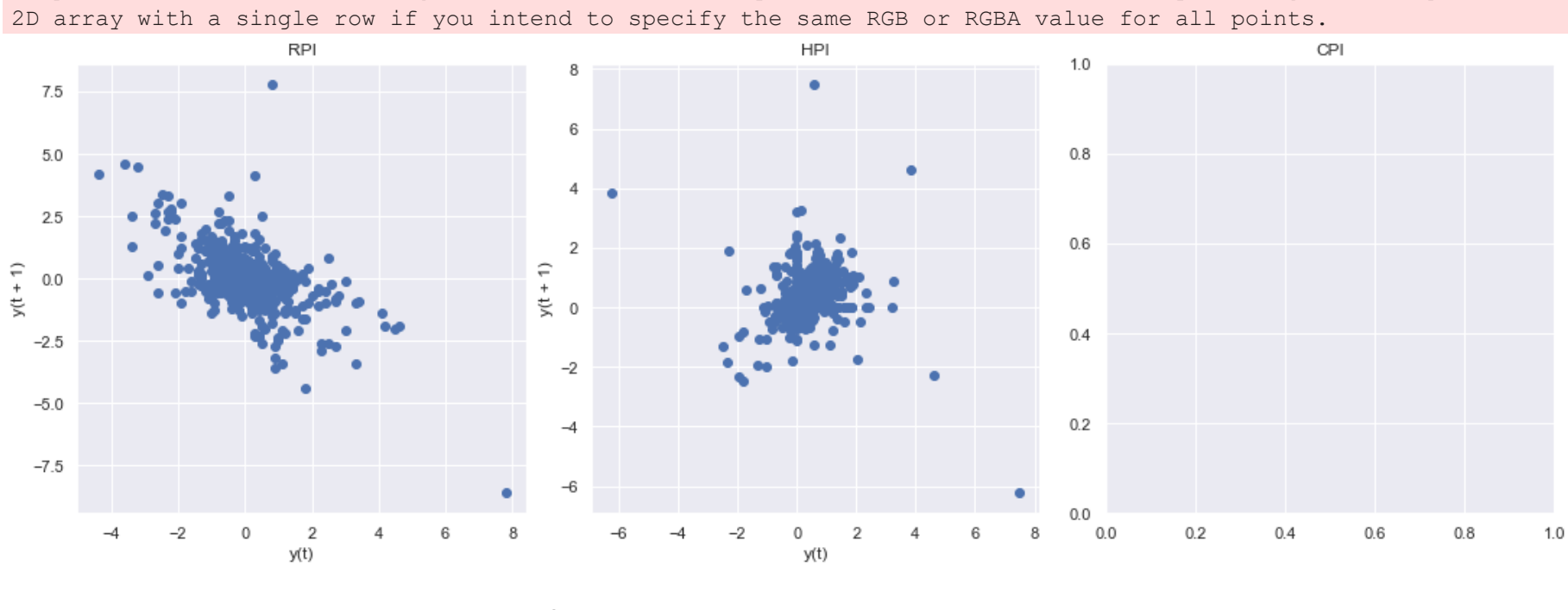
```
ADF Test Statistic : -5.548638500588681
p-value : 1.638916550894879e-06
#Lags Used : 13
Number of Observations Used : 397
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stati
onary
```

Now , the CPI and RPI becomes stationary after 2 differencings.

In [90]:

```
#plotting the lag_plots for GDP and HPI
fig,axes = plt.subplots(1, 3, figsize=(15, 5))
fig.tight_layout()
lag_plot(df_all['RPI'], ax=axes[0])
axes[0].set_title('RPI')
lag_plot(df_all['CPI'], ax=axes[0])
axes[2].set_title('CPI')
lag_plot(df_all['HPI'], ax=axes[1])
axes[1].set_title('HPI')
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



Moreover , the above lag plots give confirmation that the data is stationary

In [95]:

```
# applying VAR model and finding the AIC, BIC
df1 = df1.dropna()
model = VAR(df1[['HPI','RPI']]) #recall that rawData is w/o difference operation
for i in range(1,13):
    result = model.fit(i)
    try:
        print('Lag Order =', i)
        print('AIC : ', result.aic)
        print('BIC : ', result.bic)

    except:
        continue
```

```
Lag Order = 1
AIC :  -0.09146360399969276
BIC :  -0.03290501153343753
Lag Order = 2
AIC :  -0.09738303903962886
BIC :  0.0003929515807174766
Lag Order = 3
AIC :  -0.1451178475012397
BIC :  -0.00798077375544226
Lag Order = 4
AIC :  -0.3691969418157956
BIC :  -0.1925542210119559
Lag Order = 5
AIC :  -0.3702038502234252
BIC :  -0.15391003199574982
Lag Order = 6
AIC :  -0.3692115620718762
BIC :  -0.11312030206817242
Lag Order = 7
AIC :  -0.5557043715813703
BIC :  -0.25966842382754773
Lag Order = 8
AIC :  -0.5659446923288405
BIC :  -0.22981590151001996
Lag Order = 9
AIC :  -0.5697280288300186
BIC :  -0.193357322487138
Lag Order = 10
AIC :  -0.6007268391656906
BIC :  -0.18396421980648292
Lag Order = 11
AIC :  -0.6466030138764322
BIC :  -0.18929755099962875
Lag Order = 12
AIC :  -0.636068263922136
BIC :  -0.13806808595373588
```

```
C:\Users\hp\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:578: ValueWarning: An unsupported ind
ex was provided and will be ignored when e.g. forecasting.
  warnings.warn('An unsupported index was provided and will be'
```

Result: The lag-order p = 11 on the basis of Akaike Information Criterion (AIC)

This test uses the following null and alternative hypotheses:

Null Hypothesis (H0): Time series RPI does not Granger-cause time series HPI

Alternative Hypothesis (HA): Time series RPI Granger-causes time series HPI

In [119…

```python
#perform Granger-Causality test
grangercausalitytests(df1[['HPI', 'RPI']], maxlag=[11])
```

```
Granger Causality
number of lags (no zero) 11
ssr based F test:         F=4.2181  , p=0.0000  , df_denom=379, df_num=11
ssr based chi2 test:   chi2=49.2154 , p=0.0000  , df=11
likelihood ratio test: chi2=46.4281 , p=0.0000  , df=11
parameter F test:         F=4.2181  , p=0.0000  , df_denom=379, df_num=11
```

Out[119…

```
{11: ({'ssr_ftest': (4.218142417434826, 6.4782039148170815e-06, 379.0, 11),
   'ssr_chi2test': (49.215371424529806, 8.657047619775627e-07, 11),
   'lrtest': (46.42805690093439, 2.71605718808654e-06, 11),
   'params_ftest': (4.218142417432906, 6.478203914866771e-06, 379.0, 11.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x23459e33a30>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x23459eda490>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 0., 1., 0.]])])}
```

The F test statistic turns out to be 4.4852 and the corresponding p-value is 0.0.

Since the p-value is less than .05,we can reject the null hypothesis of the test and conclude that knowing the RPI is useful for predicting the future HPI.

Following the same test for CPI data

In [103…

```python
# applying VAR model and finding the AIC, BIC
df1 = df1.dropna()
model = VAR(df1[['HPI','CPI']]) #recall that rawData is w/o difference operation
for i in range(1,13):
    result = model.fit(i)
    try:
        print('Lag Order =', i)
        print('AIC : ', result.aic)
        print('BIC : ', result.bic)

    except:
        continue
```

```
Lag Order = 1
AIC :  -2.222315917954743
BIC :  -2.1637573254884876
Lag Order = 2
AIC :  -2.2089858877892308
BIC :  -2.111209897168884
Lag Order = 3
AIC :  -2.2020087240751423
BIC :  -2.0648716503293447
Lag Order = 4
AIC :  -2.448355956734974
BIC :  -2.2717132359311347
Lag Order = 5
AIC :  -2.4285780951922886
BIC :  -2.2122842769646134
Lag Order = 6
AIC :  -2.423504864146351
BIC :  -2.167413604142647
Lag Order = 7
AIC :  -2.7273374018354035
BIC :  -2.4313014540815807
Lag Order = 8
AIC :  -2.7285743195490118
BIC :  -2.392445528730191
Lag Order = 9
AIC :  -2.727781598137657
BIC :  -2.3514108917947767
Lag Order = 10
AIC :  -2.748946020322022
BIC :  -2.3321834009628146
Lag Order = 11
AIC :  -2.7716360384582344
BIC :  -2.314330575581431
Lag Order = 12
AIC :  -2.755586479693822
BIC :  -2.257586301725422
```

```
C:\Users\hp\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:578: ValueWarning: An unsupported ind
ex was provided and will be ignored when e.g. forecasting.
  warnings.warn('An unsupported index was provided and will be'
```

Result: The lag-order p = 11 on the basis of Akaike Information Criterion (AIC)

This test uses the following null and alternative hypotheses:

Null Hypothesis (H0): Time series CPI does not Granger-cause time series HPI

Alternative Hypothesis (HA): Time series CPI Granger-causes time series HPI

In [117…

```python
#perform Granger-Causality test
grangercausalitytests(df1[['HPI', 'CPI']], maxlag=[11])
```

```
Granger Causality
number of lags (no zero) 11
ssr based F test:         F=3.0708  , p=0.0006  , df_denom=379, df_num=11
ssr based chi2 test:   chi2=35.8282 , p=0.0002  , df=11
likelihood ratio test: chi2=34.3206 , p=0.0003  , df=11
parameter F test:         F=3.0708  , p=0.0006  , df_denom=379, df_num=11
```

Out[117…

```
{11: ({'ssr_ftest': (3.0707602140911643, 0.0005802648757443746, 379.0, 11),
   'ssr_chi2test': (35.82823658762831, 0.000180731185949054, 11),
   'lrtest': (34.32058555604999, 0.0003205095227655369, 11),
   'params_ftest': (3.070760214086964, 0.0005802648757536473, 379.0, 11.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2345cb5f6a0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2345cb0f280>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 0., 1., 0.]])])}
```

The F test statistic turns out to be 3.0708 and the corresponding p-value is 0.0006. for Lag =1

Since the p-value is less than .05,we can reject the null hypothesis of the test and conclude that knowing the CPI is useful for predicting the future HPI.

Thus , we came to know that inflation in the country affects the House Prices.

In [ ]:

Hypothesis III: Economy (GDP: Gross Domestic Product) vs House Price Index(HPI)

GDP Data Source : https://data.worldbank.org/indicator/NY.GDP.MKTP.KD.ZG?locations=GB

In [114…

```
#importing various libraries
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import grangercausalitytests, adfuller
from pandas.plotting import lag_plot
from statsmodels.tsa.api import VAR
```

In [85]:

```
# reading the excel file which contains GDP data
df_gdp = pd.read_excel(r'Datasets/GDP.xls',sheet_name="Sheet1")
df_gdp.head()
```

Out[85]:

	GDP_growth_annual%	Year
0	5.441083	1968
1	1.924097	1969
2	6.317907	1970
3	3.504717	1971
4	4.321668	1972

In [86]:

```
# checking the statistics
df_gdp.describe()
```

Out[86]:

	GDP_growth_annual%	Year
count	54.000000	54.000000
mean	2.242827	1994.500000
std	2.725545	15.732133
min	-9.270411	1968.000000
25%	1.726463	1981.250000
50%	2.473791	1994.500000
75%	3.630456	2007.750000
max	7.441273	2021.000000

We can see from the above table , that maximum GDP is 7.44 whereas the minium is -9.27 with a mean of 2.24 and standard deviation of 2.73

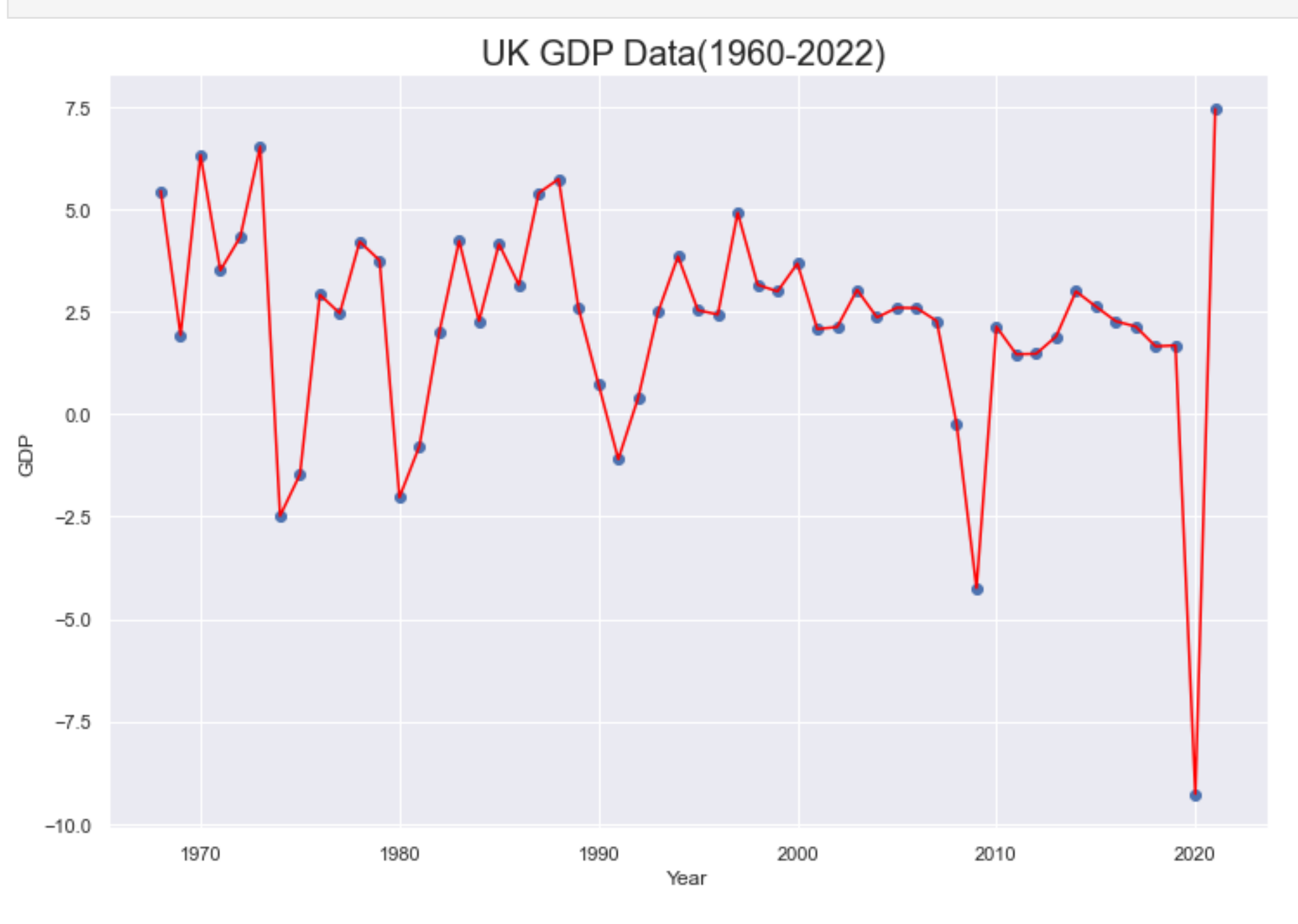
In [87]:

```
#checking the dataset year
print("Starting year of GDP data :",df_gdp.Year.min())
print("Ending year of GDP data :",df_gdp.Year.max())
```

```
Starting year of GDP data : 1968
Ending year of GDP data : 2021
```

In [88]:

```
#plotting the GDP data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_gdp, x='Year',y='GDP_growth_annual%',err_style='bars',color='red')
plt.scatter(df_gdp['Year'],df_gdp['GDP_growth_annual%'])
plt.title("UK GDP Data(1960-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('GDP')
plt.show()
```



We can see that there was a dip around 2008 due to financial crisis and in 2020 due to COVID-19.

UK HPI Data

In [92]:

```
#importing the dataset for HPI
df_HPI = pd.read_csv(r'indices_dataset/Indices-2022-05_ONS.csv')
df_HPI.head()
```

Out[92]:

	Date	Region_Name	Area_Code	Index
0	1968-04-01	England	E92000001	1.680067
1	1968-04-01	Scotland	S92000003	2.108087
2	1968-04-01	Northern Ireland	N92000001	3.300420
3	1968-04-01	Wales	W92000004	2.119327
4	1968-04-01	West Midlands Region	E12000005	2.097808

In [93]:

```
# Since , we want for the United Kingdom , we would require to filter the region_name to united kingdom
df_HPI = df_HPI[df_HPI.Region_Name == 'United Kingdom']
df_HPI.head()
```

Out[93]:

	Date	Region_Name	Area_Code	Index
9	1968-04-01	United Kingdom	K02000001	1.885299
19	1968-05-01	United Kingdom	K02000001	1.885299
29	1968-06-01	United Kingdom	K02000001	1.885299
39	1968-07-01	United Kingdom	K02000001	1.949207
49	1968-08-01	United Kingdom	K02000001	1.949207

In [94]:

```
# we require only date and index
df_HPI = df_HPI[['Date','Index']]
df_HPI
```

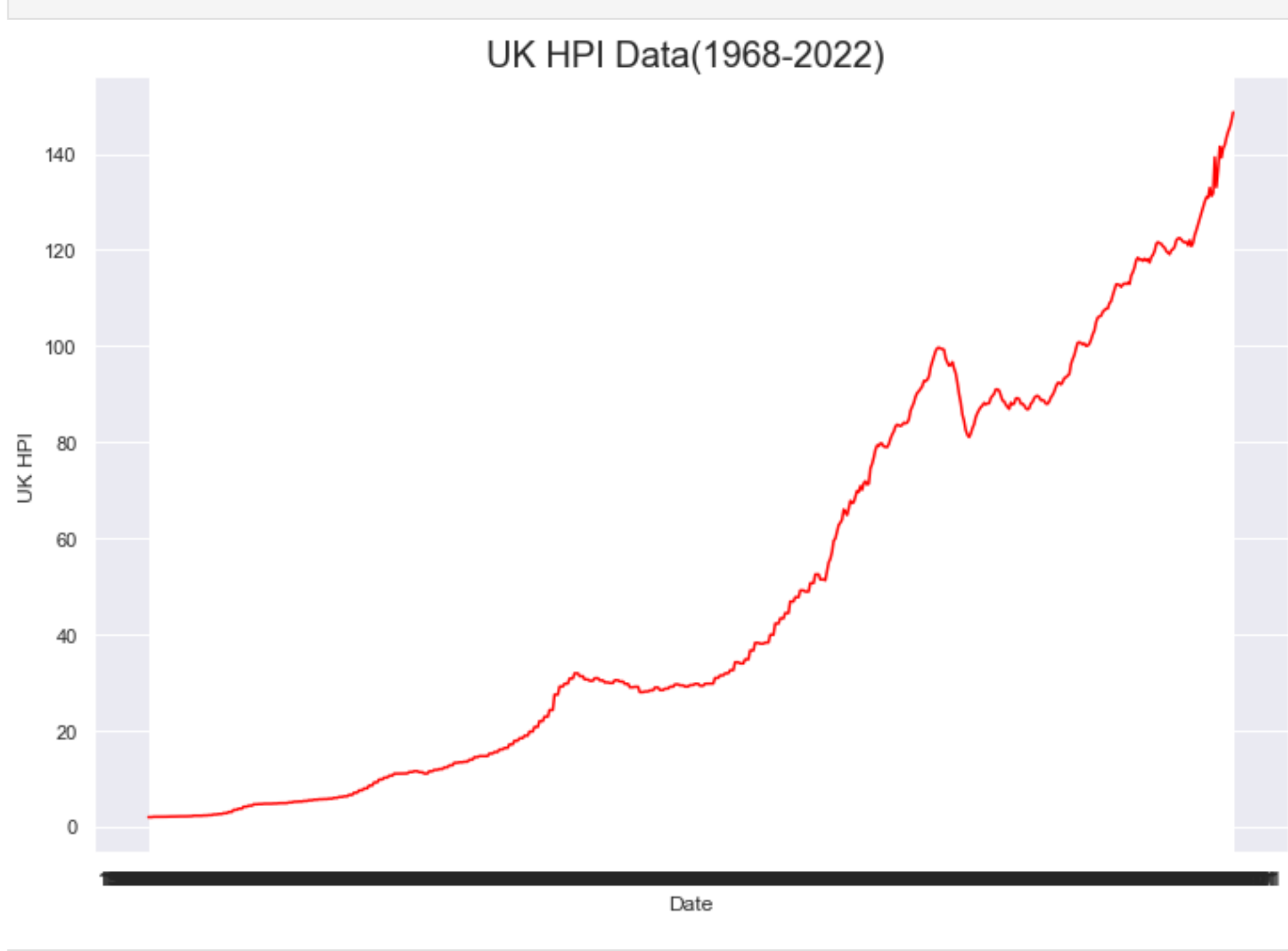
Out[94]:

	Date	Index
9	1968-04-01	1.885299
19	1968-05-01	1.885299
29	1968-06-01	1.885299
39	1968-07-01	1.949207
49	1968-08-01	1.949207
...	...	...
135249	2022-01-01	143.422582
135670	2022-02-01	144.646684
136091	2022-03-01	145.556596
136512	2022-04-01	146.898652
136933	2022-05-01	148.688246

650 rows × 2 columns

In [95]:

```
#plotting the UK HPI data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_HPI, x='Date',y='Index',err_style='bars',color='red')
plt.title("UK HPI Data(1968-2022) ",fontdict={'fontsize':20})
plt.xlabel("Date")
plt.ylabel('UK HPI')
plt.show()
```



In [96]:

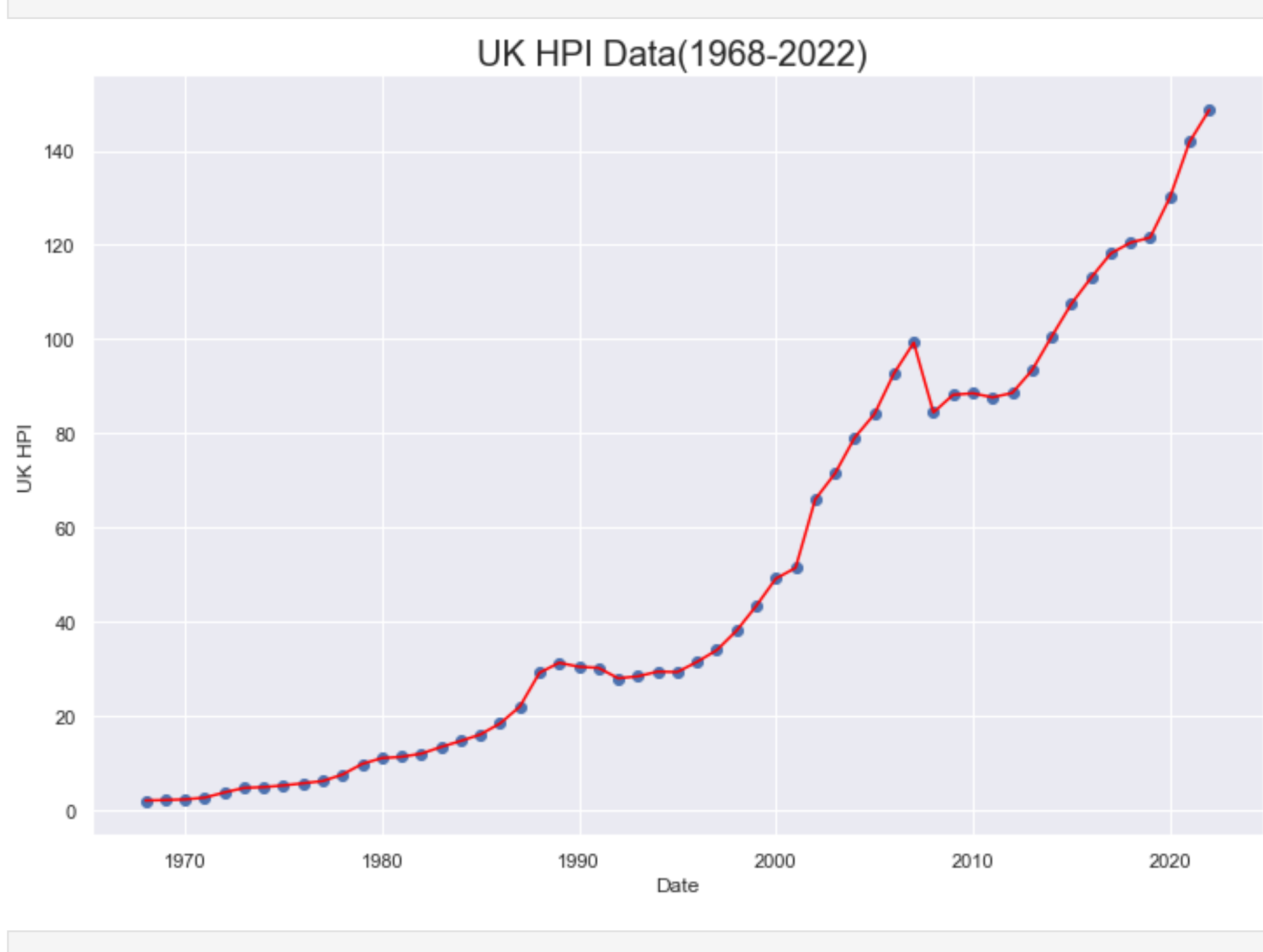
```
#the GDP data is yearly thus , making the HPI data to yearly data
df_HPI['Date'] = pd.to_datetime(df_HPI['Date'])
df_HPI.set_index('Date',inplace=True)
df_HPI =df_HPI.resample('A').last()
df_HPI.reset_index(inplace=True)
df_HPI['year'] = df_HPI['Date'].dt.year
df_HPI.drop('Date',axis=1,inplace=True)
df_HPI.head()
```

Out[96]:

	Index	year
0	1.938556	1968
1	2.045070	1969
2	2.183538	1970
3	2.545686	1971
4	3.653432	1972

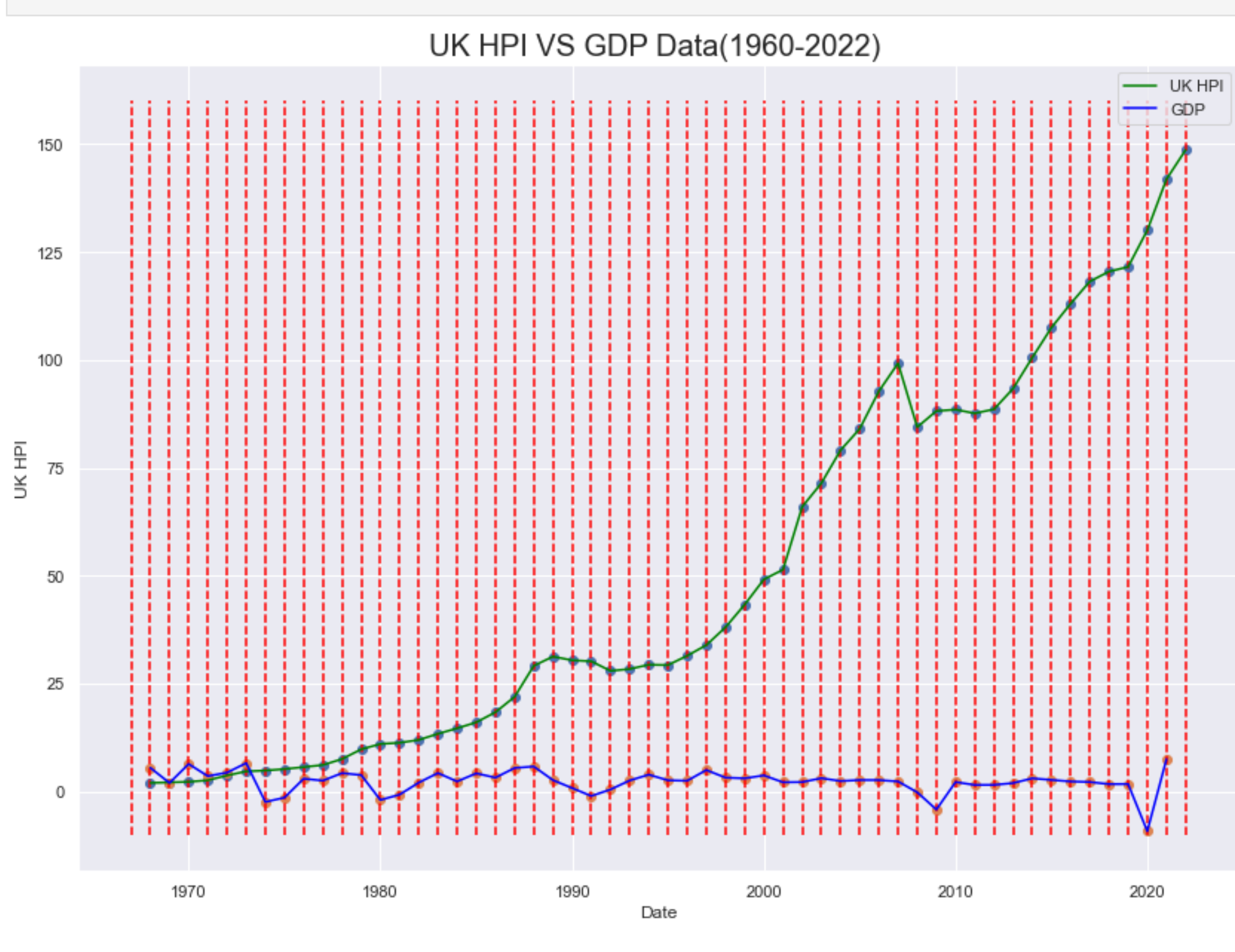
In [97]:

```
#plotting the UK HPI data yearly
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_HPI, x='year',y='Index',err_style='bars',color='red')
plt.scatter(df_HPI['year'],df_HPI['Index'])
plt.title("UK HPI Data(1968-2022) ",fontdict={'fontsize':20})
plt.xlabel("Date")
plt.ylabel('UK HPI')
plt.show()
```



In [99]:

```
#plotting the the time series together

year_lst = range(1967,2023)
fig, ax = plt.subplots(figsize=(14, 10))
ax.vlines([year_lst], -10, 160, linestyles='dashed', colors='red')
sns.lineplot(data = df_HPI, x='year',y='Index',err_style='bars',color='green',ax=ax,label='UK HPI')
plt.scatter(df_HPI['year'],df_HPI['Index'])
sns.lineplot(data = df_gdp, x='Year',y='GDP_growth_annual%',err_style='bars',color='blue',ax=ax, label='GDP')
plt.scatter(df_gdp['Year'],df_gdp['GDP_growth_annual%'])
plt.title("UK HPI VS GDP Data(1960-2022) ",fontdict={'fontsize':20})
plt.xlabel("Date")
plt.ylabel('UK HPI')
plt.legend()
plt.show()
```



Performing Granger-Causality Test

Source :https://rishi-a.github.io/2020/05/25/granger-causality.html

The Granger Causality test is used to determine whether or not one time series is useful for forecasting another.

This test uses the following null and alternative hypotheses:

Null Hypothesis (H0): Time series GDP does not Granger-cause time series HPI

Alternative Hypothesis (HA): Time series GDP Granger-causes time series HPI

The term "Granger-causes" means that knowing the value of time series x at a certain lag is useful for predicting the value of time series y at a later time period.
This test produces an F test statistic with a corresponding p-value. If the p-value is less than a certain significance level (i.e. α = .05), then we can reject the null hypothesis and conclude that we have sufficient evidence to say that time series x Granger-causes time series y.

In [101…

```
df_all = pd.DataFrame()
df_all[['Year','GDP']] = df_gdp[['Year','GDP_growth_annual%']]
df_all['HPI']=df_HPI[['Index']]
df_all.set_index('Year',inplace=True)
df_all.head()
```

Out[101…

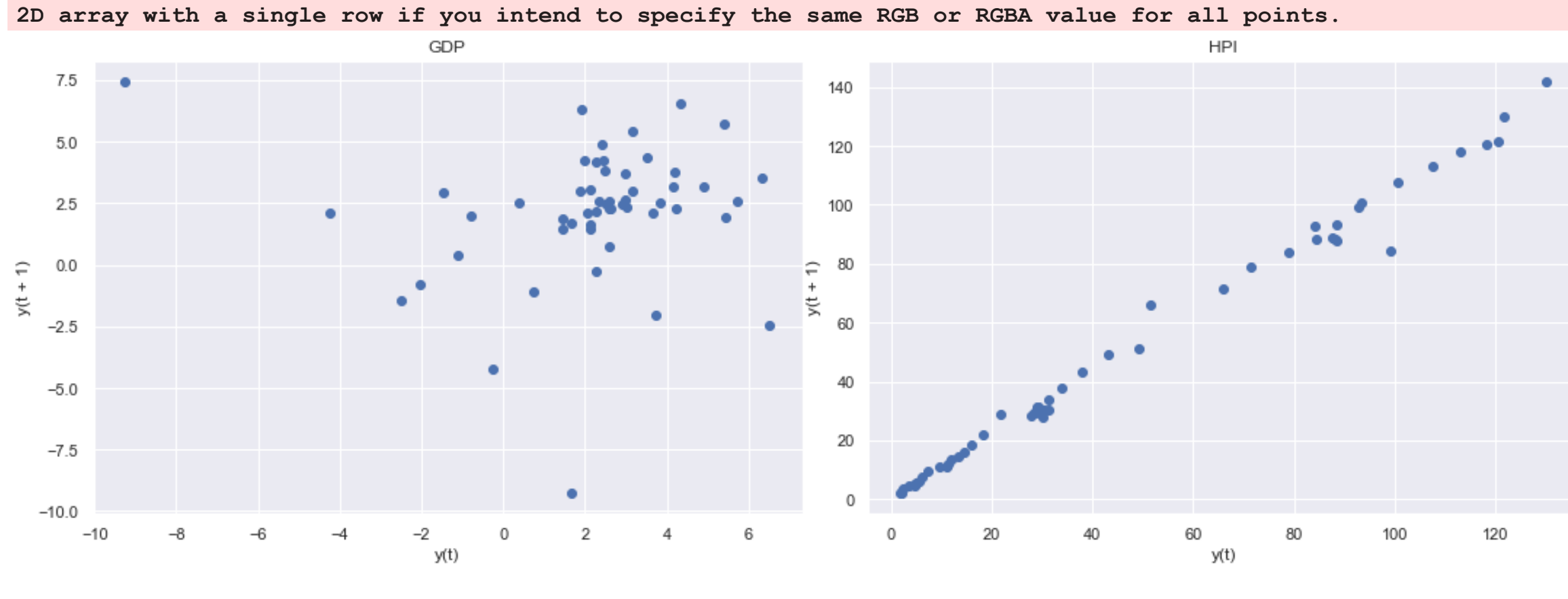
	GDP	HPI
Year		
1968	5.441083	1.938556
1969	1.924097	2.045070
1970	6.317907	2.183538
1971	3.504717	2.545686
1972	4.321668	3.653432

Step 1: Test each of the time-series to determine their order of integration. Ideally, this should involve using a test (such as the ADF test) for which the null hypothesis is non-stationarity.

In [103…

```
#plotting the lag_plots for GDP and HPI
fig,axes = plt.subplots(1, 2, figsize=(15, 5))
fig.tight_layout()
lag_plot(df_all['GDP'], ax=axes[0])
axes[0].set_title('GDP')
lag_plot(df_all['HPI'], ax=axes[1])
axes[1].set_title('HPI')
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



HPI chart shows high autocorrelation while GDP data shows low autocorrelation.

In [104…

```
# differencing the data
df1 = df_all.copy()
df_all['GDP'] = df_all['GDP']- df_all['GDP'].shift(1)
df_all['HPI'] = df_all['HPI']- df_all['HPI'].shift(1)
df_all = df_all.dropna()
df_all
```

Out[104…

	GDP	HPI
Year		
1969	-3.516987	0.106514
1970	4.393810	0.138468
1971	-2.813189	0.362148
1972	0.816950	1.107746
1973	2.202181	0.958627
1974	-9.008252	0.181074
1975	1.010755	0.351496
1976	4.383916	0.458010
1977	-0.452516	0.479313
1978	1.746510	1.384683
1979	-0.455244	2.258098
1980	-5.780384	1.235563
1981	1.243624	0.276937
1982	2.782635	0.660387
1983	2.226965	1.416637
1984	-1.952751	1.278169
1985	1.878310	1.384683
1986	-0.997074	2.343309
1987	2.242398	3.567081
1988	0.339675	7.193793
1989	-3.154811	2.142832
1990	-1.843847	-0.856400
1991	-1.836877	-0.244163
1992	1.504204	-2.214479
1993	2.088749	0.426744
1994	1.356178	0.995737
1995	-1.314339	-0.085349
1996	-0.103127	2.162171
1997	2.480483	2.475117
1998	-1.755052	4.125196
1999	-0.163862	5.263181
2000	0.682258	5.832173
2001	-1.599713	2.275970
2002	0.050566	14.571755
2003	0.906667	5.465031
2004	-0.674260	7.640431
2005	0.237650	4.969209
2006	-0.009175	8.711651
2007	-0.314618	6.490158
2008	-2.509125	-14.810938
2009	-4.007718	3.738172
2010	6.378794	0.325761
2011	-0.673875	-0.867989
2012	0.012324	0.941480
2013	0.420131	4.787345
2014	1.101146	7.184687
2015	-0.368568	6.949851
2016	-0.359133	5.548783
2017	-0.129010	5.155718
2018	-0.483528	2.307336
2019	0.021019	1.082021
2020	-10.942355	8.491921
2021	16.711684	11.830103

In [109…

```
#defining a function for dicky fuller test
def adfuller_test(columns):
    result=adfuller(columns)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit ro
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-statior
```

In [110…

```
adfuller_test(df_all['GDP'])
```

```
ADF Test Statistic : -4.51236990063351
p-value : 0.0001867636432092519
#Lags Used : 9
Number of Observations Used : 43
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stati
onary
```

In [111…

```
adfuller_test(df_all['HPI'])
```

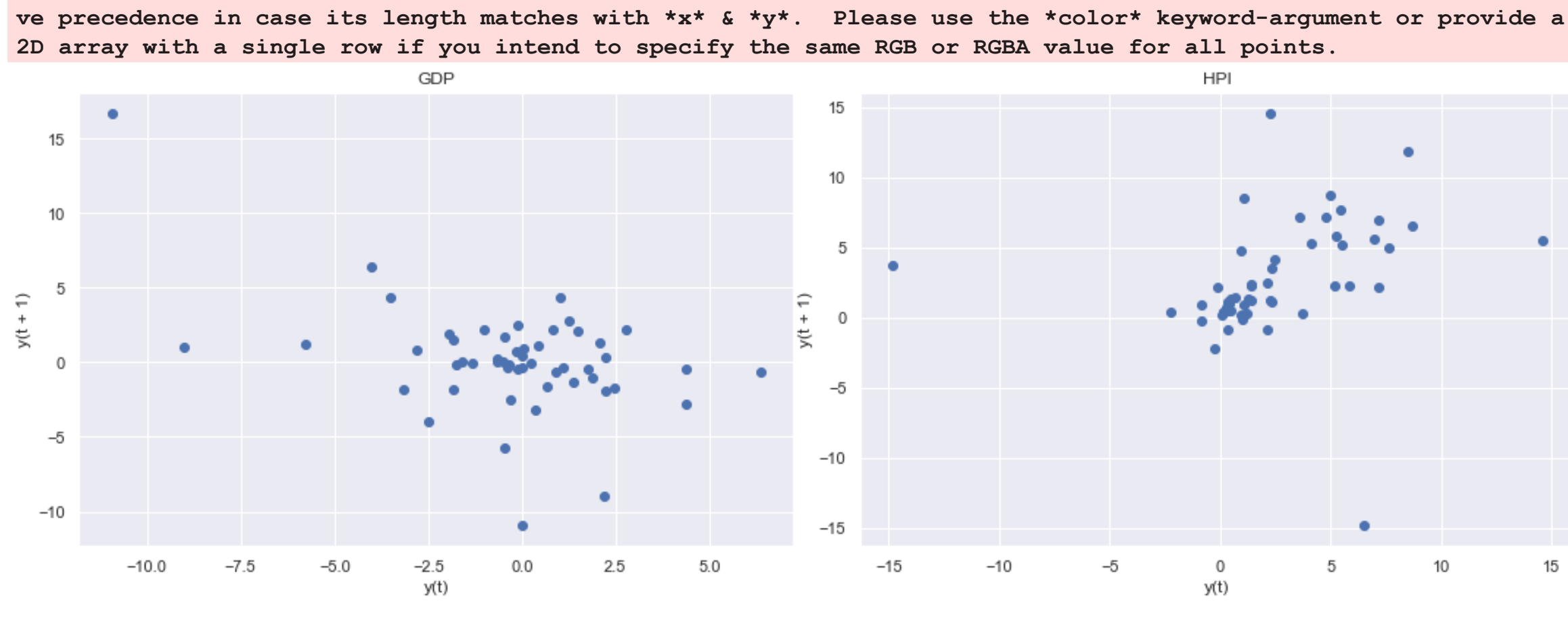
```
ADF Test Statistic : -5.084360078035213
p-value : 1.5070484798456902e-05
#Lags Used : 0
Number of Observations Used : 52
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stati
onary
```

We can see from above tests that ADF Null Hypothesis is rejected: thus data is stationary.

In [112…

```
#plotting the lag_plots for GDP and HPI
fig,axes = plt.subplots(1, 2, figsize=(15, 5))
fig.tight_layout()
lag_plot(df_all['GDP'], ax=axes[0])
axes[0].set_title('GDP')
lag_plot(df_all['HPI'], ax=axes[1])
axes[1].set_title('HPI')
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



Moreover , the above lag plots give confirmation that the data is stationary

Step 2: Let the maximum order of integration for the group of time-series be m. So, if there are two time-series and one is found to be I(1) and the other is I(2), then m = 2. If one is I(0) and the other is I(1), then m = 1, etc. Result: We saw above that for our case m = 1

Step 3: Set up a VAR model in the levels of the data, regardless of the orders of integration of the various time-series. Most importantly, you must not difference the data, no matter what you found at Step 1.//

Step 4: Determine the appropriate maximum lag length for the variables in the VAR, say p, using the usual methods. Specifically, base the choice of p on the usual information criteria, such as AIC, SIC.

In [118…

```
# applying VAR model and finding the AIC, BIC
df1 = df1.dropna()
model = VAR(df1) #recall that rawData is w/o difference operation
for i in range(1,13):
    result = model.fit(i)
    try:
        print('Lag Order =', i)
        print('AIC : ', result.aic)
        print('BIC : ', result.bic)

    except:
        continue
```

```
Lag Order = 1
AIC :  4.825197886745501
BIC :  5.048249801487251
Lag Order = 2
AIC :  4.760587960142102
BIC :  5.1358271367923765
Lag Order = 3
AIC :  4.853492649203838
BIC :  5.383797724853654
Lag Order = 4
AIC :  4.965558476541122
BIC :  5.653886758495254
Lag Order = 5
AIC :  5.112877293516745
BIC :  5.962265998790904
Lag Order = 6
AIC :  5.211353535917136
BIC :  6.224920750158911
Lag Order = 7
AIC :  5.174388548712
BIC :  6.355333826399271
Lag Order = 8
AIC :  5.312771034518857
BIC :  6.664375544967318
Lag Order = 9
AIC :  5.444010399509372
BIC :  6.969636501982086
Lag Order = 10
AIC :  5.606647424386283
BIC :  7.309737529490078
Lag Order = 11
AIC :  5.453425727343577
BIC :  7.337500269713435
Lag Order = 12
AIC :  5.537390410971765
BIC :  7.606044718451965
```

```
C:\Users\hp\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:578: ValueWarning: An unsupported ind
ex was provided and will be ignored when e.g. forecasting.
  warnings.warn('An unsupported index was provided and will be'
```

Result: The lag-order p = 2 on the basis of Akaike Information Criterion (AIC)

In [122…

```
#perform Granger-Causality test
grangercausalitytests(df1[['GDP', 'HPI']], maxlag=[2])
```

```
Granger Causality
number of lags (no zero) 2
ssr based F test:         F=4.4719  , p=0.0168  , df_denom=46, df_num=2
ssr based chi2 test:   chi2=9.9159  , p=0.0070  , df=2
likelihood ratio test: chi2=9.0611  , p=0.0108  , df=2
parameter F test:         F=4.4719  , p=0.0168  , df_denom=46, df_num=2
```

Out[122…

```
{2: ({'ssr_ftest': (4.471886738462427, 0.016799835158385023, 46.0, 2),
   'ssr_chi2test': (9.915922767894946, 0.007027239082455598, 2),
   'lrtest': (9.061117236335633, 0.010774655472992271, 2),
   'params_ftest': (4.47188673846242, 0.016799835158385075, 46.0, 2.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x18f2b9a2bb0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x18f2b9a2c10>,
   array([[0., 0., 1., 0., 0.],
          [0., 0., 0., 1., 0.]])])}
```

The F test statistic turns out to be 4.4719 and the corresponding p-value is 0.0168.

Since the p-value is less than .05,we can reject the null hypothesis of the test and conclude that knowing the GDP is useful for predicting the future HPI.

Hypothesis IV :Interest Rate vs House Price

Data Source : https://www.bankofengland.co.uk/boeapps/database/Bank-Rate.asp

In [50]:
```
#importing various libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import grangercausalitytests, adfuller
from pandas.plotting import lag_plot
from statsmodels.tsa.api import VAR
```

In [11]:
```
#importing interest rate dataset
df_int = pd.read_excel(r'Datasets/Bank_Rate_history.xlsx')
df_int.head()
```

Out[11]:

	Date	Rate
0	1975-01-20 00:00:00	11.25
1	1975-01-27 00:00:00	11.00
2	1975-02-10 00:00:00	10.75
3	1975-02-17 00:00:00	10.50
4	1975-03-10 00:00:00	10.25

In [12]:
```
# checking the statistics
df_int.describe()
```

Out[12]:

	Rate
count	244.000000
mean	9.208934
std	3.707317
min	0.100000
25%	6.250000
50%	9.880000
75%	11.880000
max	17.000000

We can see from the above table , that maximum Bank Rate is 17 whereas the minium is 0.10 with a mean of 9.208 and standard deviation of 3.70

In [13]:
```
#checking the shape of dataset
df_int.shape
```

Out[13]:
```
(244, 2)
```

In [15]:
```
#checking the dtypes
df_int.dtypes
```

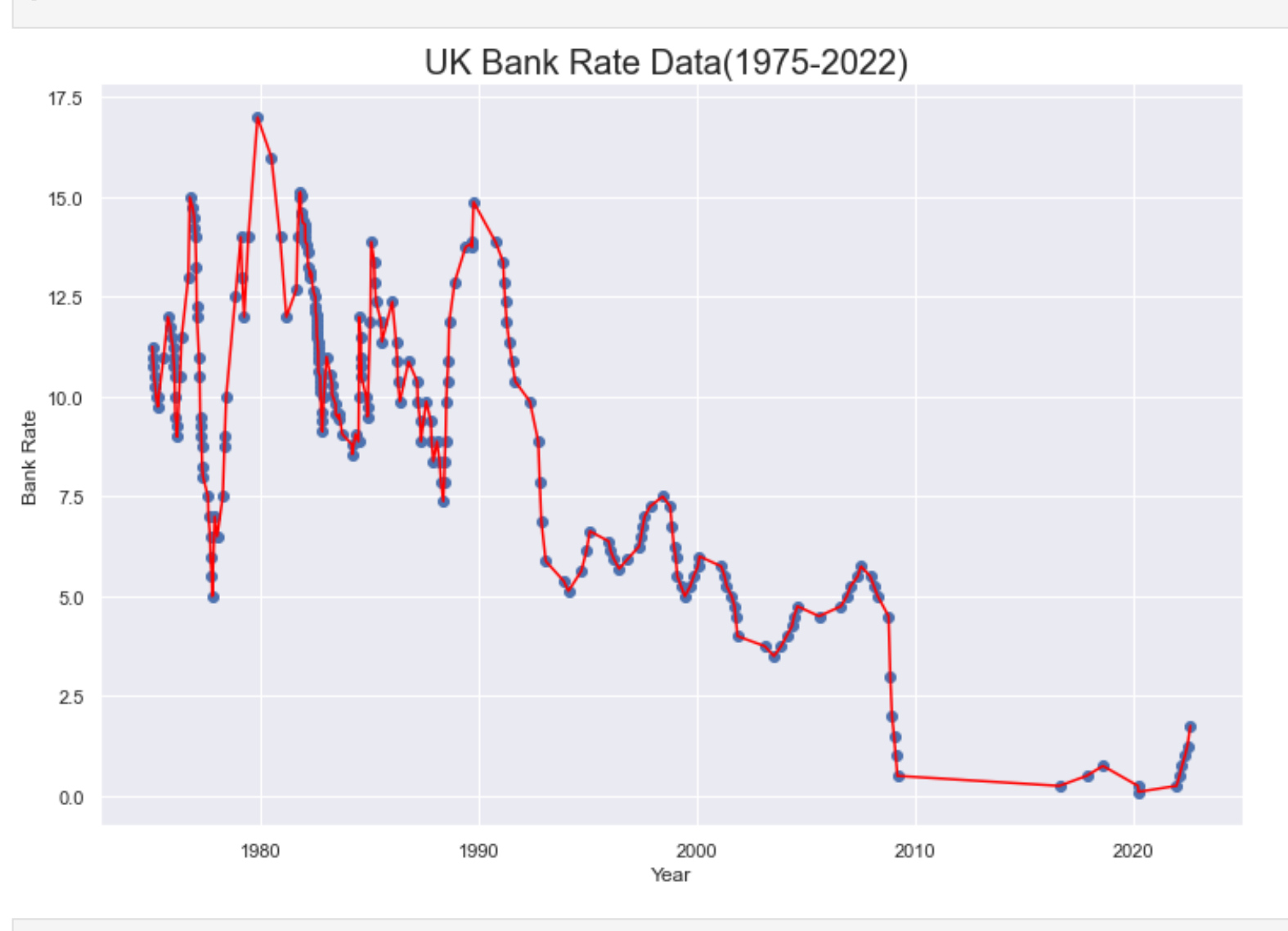
Out[15]:
```
Date      object
Rate     float64
dtype: object
```

In [16]:
```
# converting the date object to datetime.
df_int['Date']= pd.to_datetime(df_int['Date'])
```

In [17]:
```
#checking the dataset's to and from dates
print("Starting date of Bank Rate Data :",df_int.Date.min())
print("Ending date of Bank Rate data :",df_int.Date.max())
```

```
Starting date of Bank Rate Data : 1975-01-20 00:00:00
Ending date of Bank Rate data : 2022-08-04 00:00:00
```

In [18]:
```
#plotting the GDP data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_int, x='Date',y='Rate',err_style='bars',color='red')
plt.scatter(df_int['Date'],df_int['Rate'])
plt.title("UK Bank Rate Data(1975-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('Bank Rate')
plt.show()
```



In [25]:
```
# converting the dataset into monthly dataset
df_int = pd.read_excel(r'Bank_Rate_history.xlsx',sheet_name='monthly')
df_int.head()
```

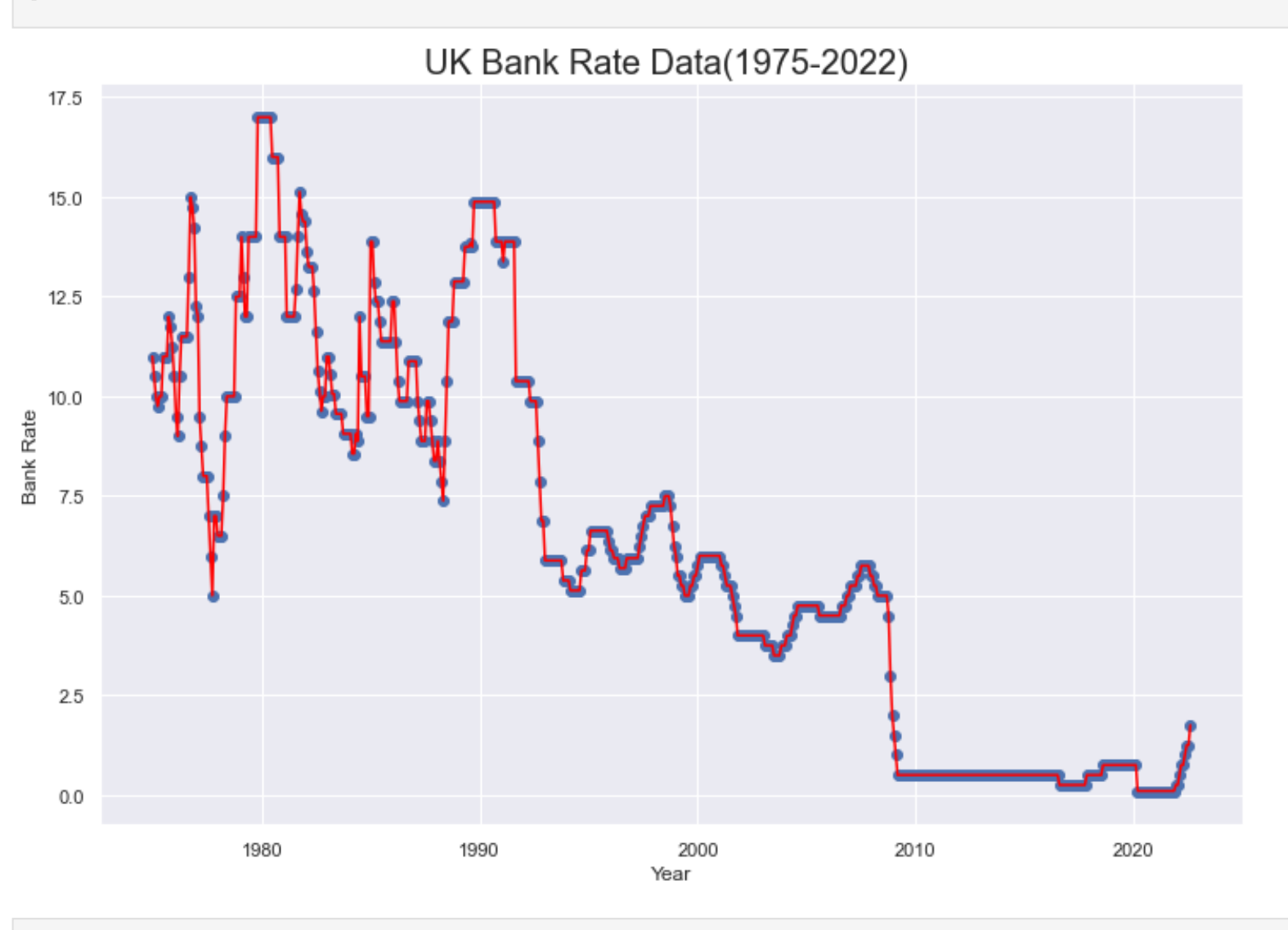
Out[25]:

	Date	Rate
0	1975-01-01	11.00
1	1975-02-01	10.50
2	1975-03-01	10.00
3	1975-04-01	9.75
4	1975-05-01	10.00

In [36]:
```
#checking the dataset's to and from dates
print("Starting date of Bank Rate Data :",df_int.Date.min())
print("Ending date of Bank Rate data :",df_int.Date.max())
```

```
Starting date of Bank Rate Data : 1975-01-01 00:00:00
Ending date of Bank Rate data : 2022-08-01 00:00:00
```

In [26]:
```
#plotting the Bank rate data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_int, x='Date',y='Rate',err_style='bars',color='red')
plt.scatter(df_int['Date'],df_int['Rate'])
plt.title("UK Bank Rate Data(1975-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('Bank Rate')
plt.show()
```



In [29]:
```
# importing the dataset for UK house prices from ONS
df_HP = pd.read_csv(r'indices_dataset/Average-prices-2022-05_ONS.csv')
#since the region_name contains all the regions, we would take the region_name as United Kingdom
df_HP = df_HP[df_HP.Region_Name=='United Kingdom']
#we require only the date and average_price
df_HP = df_HP[['Date','Average_Price']]
df_HP.head()
```

Out[29]:

	Date	Average_Price
9	1968-04-01	3594.602239
19	1968-05-01	3594.602239
29	1968-06-01	3594.602239
39	1968-07-01	3716.453163
49	1968-08-01	3716.453163

In [30]:
```
# checking the statistics
df_HP.describe()
```

Out[30]:

	Average_Price
count	650.000000
mean	92682.855832
std	79576.598379
min	3594.602239
25%	21506.687970
50%	58249.724790
75%	167885.249350
max	283496.357300

We can see from the above table , that maximum Bank Rate is 283496.357 whereas the minium is 3594.60 with a mean of 92682.85 and standard deviation of 79576.59

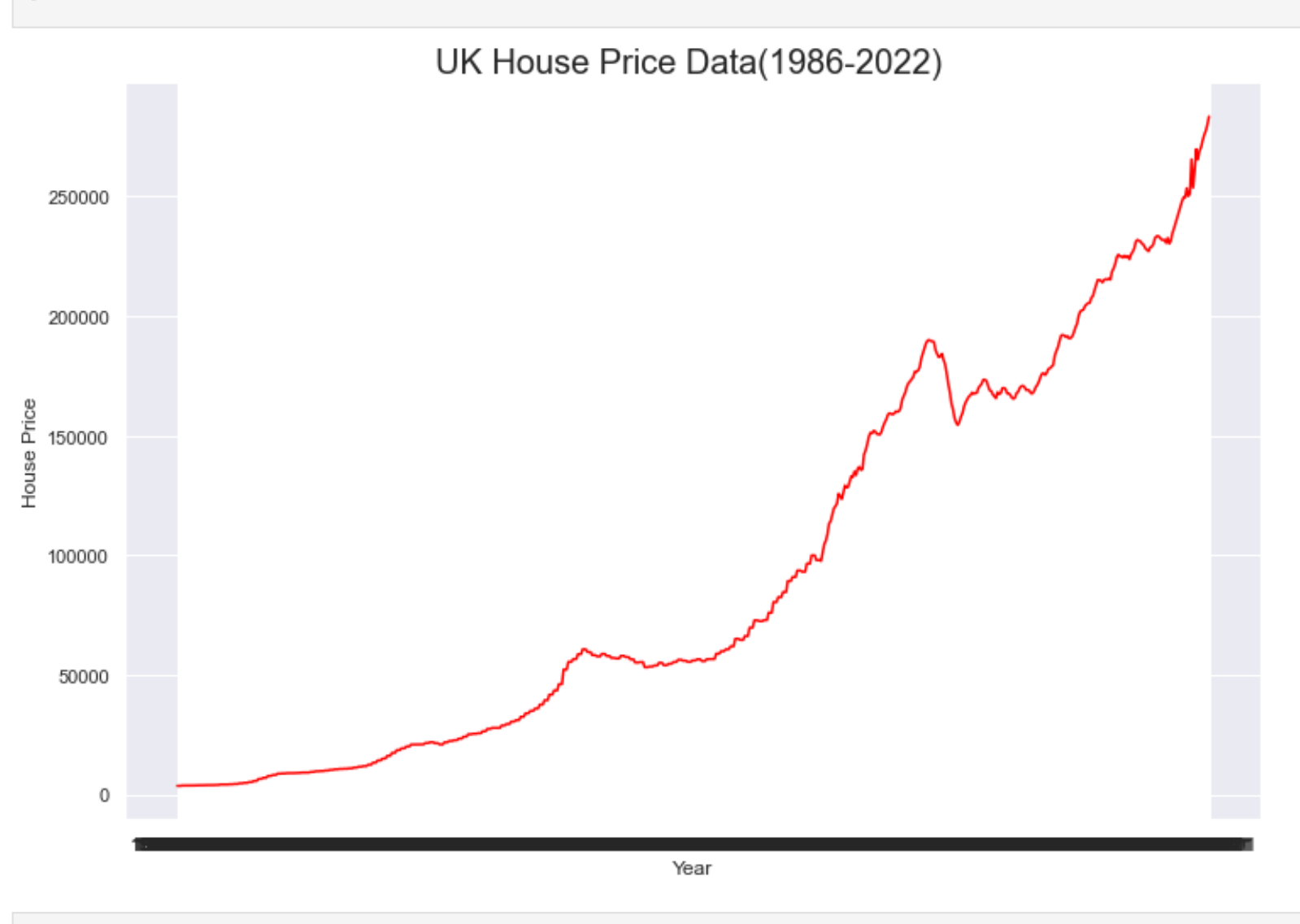
In [31]:
```
#checking the shape of dataset
df_HP.shape
```

Out[31]:
```
(650, 2)
```

In [32]:
```
#checking the dataset's to and from dates
print("Starting date of House Price Data :",df_HP.Date.min())
print("Ending date of House Price data :",df_HP.Date.max())
```

```
Starting date of House Price Data : 1968-04-01
Ending date of House Price data : 2022-05-01
```

In [35]:
```
#plotting the House Price data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_HP, x='Date',y='Average_Price',err_style='bars',color='red')
plt.title("UK House Price Data(1986-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('House Price')
plt.show()
```



In [37]:
```
#since the starting and end dates mismatch among datasets , thus slicing the HP dataset
df_HP = df_HP[df_HP.Date>='1975-01-01']
df_HP
```

Out[37]:

	Date	Average_Price
819	1975-01-01	9179.436227
829	1975-02-01	9179.436227
839	1975-03-01	9179.436227
849	1975-04-01	9423.138074
859	1975-05-01	9423.138074
...	...	...
135249	2022-01-01	273456.582300
135670	2022-02-01	275790.515100
136091	2022-03-01	277525.398100
136512	2022-04-01	280084.228600
136933	2022-05-01	283496.357300

569 rows × 2 columns

In [44]:
```
# combing both the datasets
df_all = pd.DataFrame()
df_all[['Date','HP']]= df_HP[['Date','Average_Price']]
df_all['Rate']=df_int['Rate'][:-3].values
df_all
```

Out[44]:

	Date	HP	Rate
819	1975-01-01	9179.436227	11.00
829	1975-02-01	9179.436227	10.50
839	1975-03-01	9179.436227	10.00
849	1975-04-01	9423.138074	9.75
859	1975-05-01	9423.138074	10.00
...	...	...	...
135249	2022-01-01	273456.582300	0.25
135670	2022-02-01	275790.515100	0.50
136091	2022-03-01	277525.398100	0.75
136512	2022-04-01	280084.228600	0.75
136933	2022-05-01	283496.357300	1.00

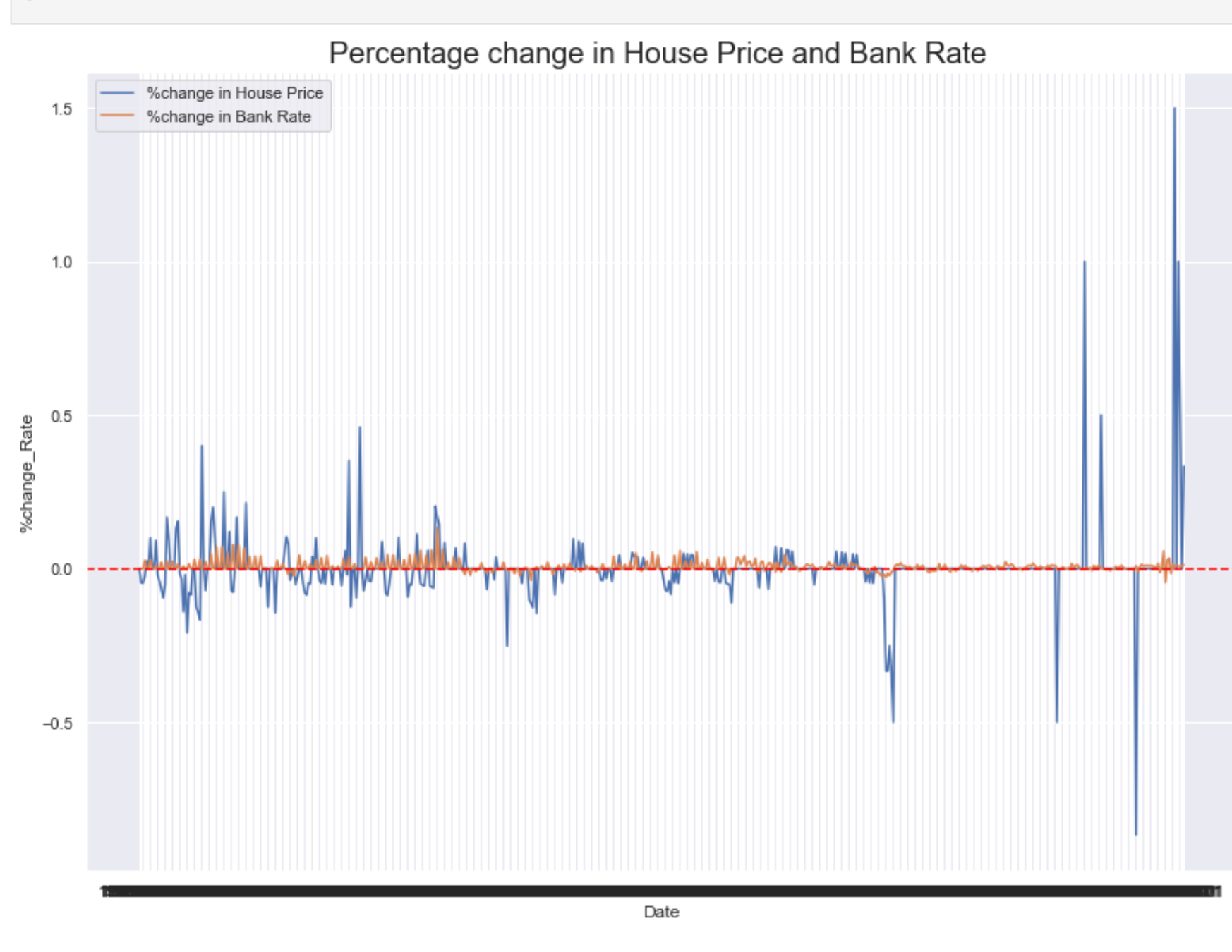
569 rows × 3 columns

In [45]:
```
#calculating the percentage change of both the HP and Rate
df_all['%change_HP'] =df_all['HP'].pct_change()
df_all['%change_Rate'] =df_all['Rate'].pct_change()
df_all.fillna(0,inplace=True)
df_all.head()
```

Out[45]:

	Date	HP	Rate	%change_HP	%change_Rate
819	1975-01-01	9179.436227	11.00	0.000000	0.000000
829	1975-02-01	9179.436227	10.50	0.000000	-0.045455
839	1975-03-01	9179.436227	10.00	0.000000	-0.047619
849	1975-04-01	9423.138074	9.75	0.026549	-0.025000
859	1975-05-01	9423.138074	10.00	0.000000	0.025641

In [49]:
```
#plotting the percentage change together of both house price and Rate
plt.subplots(figsize=(14, 10))
sns.lineplot(data=df_all,x='Date',y='%change_Rate',label='%change in House Price')
sns.lineplot(data=df_all,x='Date',y='%change_HP',label='%change in Bank Rate')
plt.axhline(0, ls='--',c='red')
plt.legend()
plt.title("Percentage change in House Price and Bank Rate ",fontdict={'fontsize': 20})
plt.show()
```



We can see in the above chart that percentage change in house prices is much more than the percentage change in bank rate.

Performing Granger-Causality Test

Source :https://rishi-a.github.io/2020/05/25/granger-causality.html

The Granger Causality test is used to determine whether or not one time series is useful for forecasting another. This test uses the following null and alternative hypotheses:

Null Hypothesis (H0): Time series Bank Rate does not Granger-cause time series HP

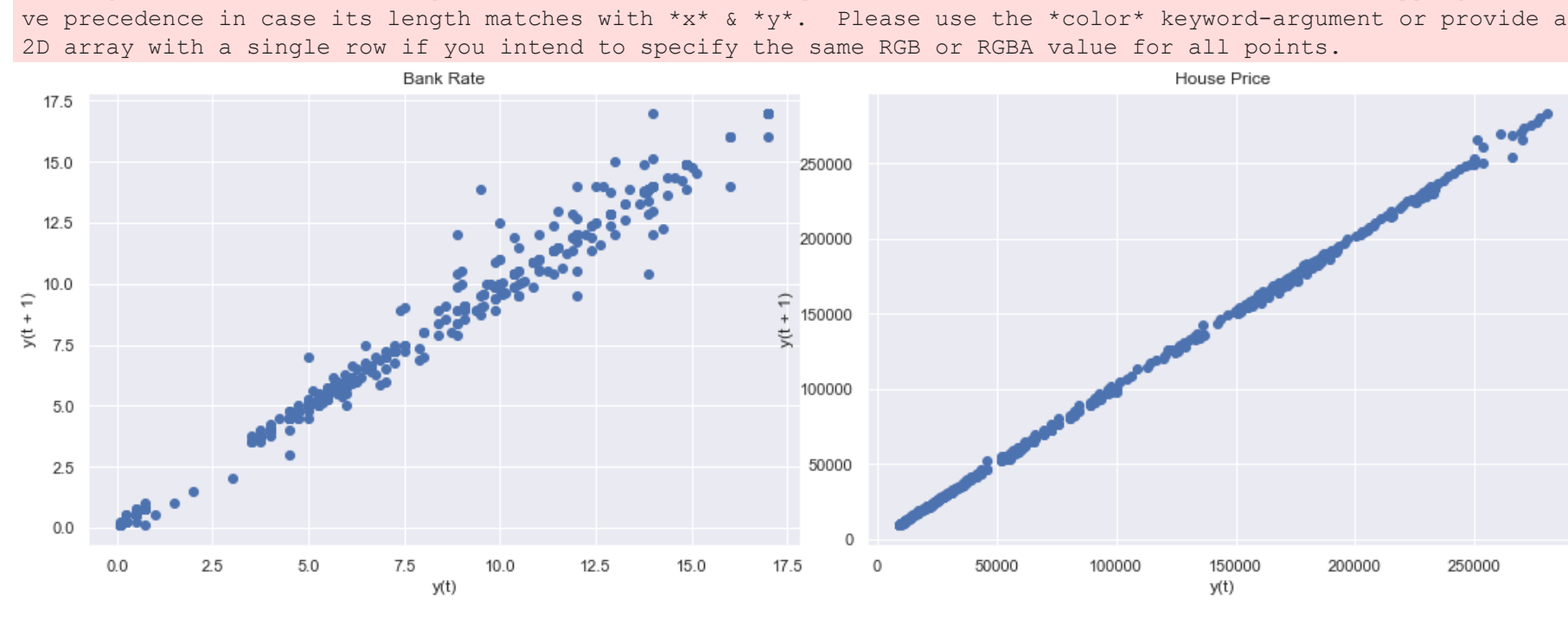
Alternative Hypothesis (HA): Time series Bank Rate Granger-causes time series HP

The term "Granger-causes" means that knowing the value of time series x at a certain lag is useful for predicting the value of time series y at a later time period. This test produces an F test statistic with a corresponding p-value. If the p-value is less than a certain significance level (i.e. α = .05), then we can reject the null hypothesis and conclude that we have sufficient evidence to say that time series x Granger-causes time series y.

Step 1: Test each of the time-series to determine their order of integration. Ideally, this should involve using a test (such as the ADF test) for which the null hypothesis is non-stationarity.

In [51]:
```
#plotting the lag_plots for Bank rate and HP
fig,axes = plt.subplots(1, 2, figsize=(15, 5))
fig.tight_layout()
lag_plot(df_all['Rate'], ax=axes[0])
axes[0].set_title('Bank Rate')
lag_plot(df_all['HP'], ax=axes[1])
axes[1].set_title('House Price')
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



we Can see that both the time series have high autocorrelation.

In [52]:
```
# differencing the data
df1 = df_all.copy()
df_all['Rate'] = df_all['Rate']- df_all['Rate'].shift(1)
df_all['HP'] = df_all['HP']- df_all['HP'].shift(1)
df_all = df_all.dropna()
df_all
```

Out[52]:

	Date	HP	Rate	%change_HP	%change_Rate
829	1975-02-01	0.000000	-0.50	0.000000	-0.045455
839	1975-03-01	0.000000	-0.50	0.000000	-0.047619
849	1975-04-01	243.701847	-0.25	0.026549	-0.025000
859	1975-05-01	0.000000	0.25	0.000000	0.025641
869	1975-06-01	0.000000	0.00	0.000000	0.000000
...	...	...	...	...	...
135249	2022-01-01	2917.312100	0.00	0.010783	0.000000
135670	2022-02-01	2333.932800	0.25	0.008535	1.000000
136091	2022-03-01	1734.883000	0.25	0.006291	0.500000
136512	2022-04-01	2558.830500	0.00	0.009220	0.000000
136933	2022-05-01	3412.128700	0.25	0.012183	0.333333

568 rows × 5 columns

In [53]:
```
#defining a function for dicky fuller test
def adfuller_test(columns):
    result=adfuller(columns)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit ro
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-statio
```

In [54]:
```
#applying Dickey Fulle test on Bank rate
adfuller_test(df_all['Rate'])
```

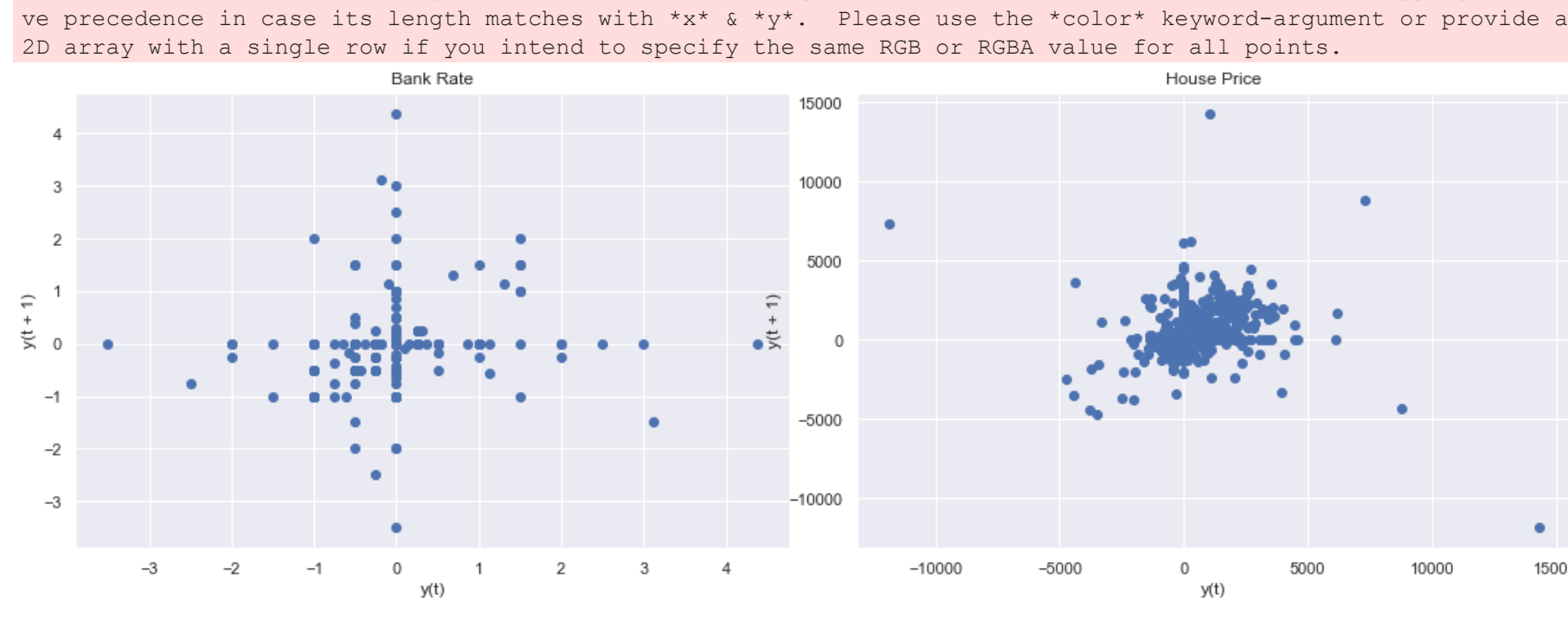
```
ADF Test Statistic : -20.301833325067143
p-value : 0.0
#Lags Used : 0
Number of Observations Used : 567
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stati
onary
```

In [55]:
```
#applying Dickey Fulle test on House Price
adfuller_test(df_all['HP'])
```

```
ADF Test Statistic : -3.3636461382694116
p-value : 0.012262391004210095
#Lags Used : 19
Number of Observations Used : 548
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stati
onary
```

In [56]:
```
#plotting the lag_plots for Bank rate and HP
fig,axes = plt.subplots(1, 2, figsize=(15, 5))
fig.tight_layout()
lag_plot(df_all['Rate'], ax=axes[0])
axes[0].set_title('Bank Rate')
lag_plot(df_all['HP'], ax=axes[1])
axes[1].set_title('House Price')
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



We can see from above tests that ADF Null Hypothesis is rejected: thus data is stationary.Moreover , the above lag plots give confirmation that the data is stationary

Step 2: Let the maximum order of integration for the group of time-series be m. So, if there are two time-series and one is found to be I(1) and the other is I(2), then m = 2. If one is I(0) and the other is I(1), then m = 1, etc. Result: We saw above that for our case m = 1

Step 3: Set up a VAR model in the levels of the data, regardless of the orders of integration of the various time-series. Most importantly, you must not difference the data, no matter what you found at Step 1.//

Step 4: Determine the appropriate maximum lag length for the variables in the VAR, say p, using the usual methods. Specifically, base the choice of p on the usual information criteria, such as AIC, SIC.

In [57]:
```
# applying VAR model and finding the AIC, BIC
df1 = df1.dropna()
model = VAR(df1[['Rate','HP']]) #recall that rawData is w/o difference operation
for i in range(1,13):
    result = model.fit(i)
    try:
        print('Lag Order =', i)
        print('AIC : ', result.aic)
        print('BIC : ', result.bic)

    except:
        continue
```

```
Lag Order = 1
AIC :  13.43081932206397
BIC :  13.476686801839193
Lag Order = 2
AIC :  13.402865650608215
BIC :  13.479415197411173
Lag Order = 3
AIC :  13.406421212783485
BIC :  13.513736260654234
Lag Order = 4
AIC :  13.177204925879442
BIC :  13.31536928545579
Lag Order = 5
AIC :  13.189284368812858
BIC :  13.358382229686896
Lag Order = 6
AIC :  13.200047205667667
BIC :  13.400163138761153
Lag Order = 7
AIC :  13.17367487873984
BIC :  13.404893838698579
Lag Order = 8
AIC :  13.191344032526278
BIC :  13.453751360133714
Lag Order = 9
AIC :  13.199686531257813
BIC :  13.493367955868008
Lag Order = 10
AIC :  13.15260418838588
BIC :  13.477645830376066
Lag Order = 11
AIC :  13.1541093745321
BIC :  13.510597747775003
Lag Order = 12
AIC :  13.17063271248393
BIC :  13.55865472684006
C:\Users\hp\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:578: ValueWarning: An unsupported ind
ex was provided and will be ignored when e.g. forecasting.
  warnings.warn('An unsupported index was provided and will be'
```

Result: The lag-order p = 10 on the basis of Akaike Information Criterion (AIC)

In [67]:
```
#perform Granger-Causality test
grangercausalitytests(df_all[['HP','Rate']], maxlag=10)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=1.4847  , p=0.2235  , df_denom=564, df_num=1
ssr based chi2 test:   chi2=1.4926  , p=0.2218  , df=1
likelihood ratio test: chi2=1.4907  , p=0.2221  , df=1
parameter F test:         F=1.4847  , p=0.2235  , df_denom=564, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=1.0698  , p=0.3438  , df_denom=561, df_num=2
ssr based chi2 test:   chi2=2.1587  , p=0.3398  , df=2
likelihood ratio test: chi2=2.1546  , p=0.3405  , df=2
parameter F test:         F=1.0698  , p=0.3438  , df_denom=561, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=0.1742  , p=0.9138  , df_denom=558, df_num=3
ssr based chi2 test:   chi2=0.5292  , p=0.9124  , df=3
likelihood ratio test: chi2=0.5289  , p=0.9125  , df=3
parameter F test:         F=0.1742  , p=0.9138  , df_denom=558, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.3246  , p=0.8615  , df_denom=555, df_num=4
ssr based chi2 test:   chi2=1.3195  , p=0.8581  , df=4
likelihood ratio test: chi2=1.3179  , p=0.8583  , df=4
parameter F test:         F=0.3246  , p=0.8615  , df_denom=555, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=0.2251  , p=0.9516  , df_denom=552, df_num=5
ssr based chi2 test:   chi2=1.1481  , p=0.9498  , df=5
likelihood ratio test: chi2=1.1469  , p=0.9499  , df=5
parameter F test:         F=0.2251  , p=0.9516  , df_denom=552, df_num=5

Granger Causality
number of lags (no zero) 6
ssr based F test:         F=0.1818  , p=0.9818  , df_denom=549, df_num=6
ssr based chi2 test:   chi2=1.1164  , p=0.9808  , df=6
likelihood ratio test: chi2=1.1153  , p=0.9809  , df=6
parameter F test:         F=0.1818  , p=0.9818  , df_denom=549, df_num=6

Granger Causality
number of lags (no zero) 7
ssr based F test:         F=0.1649  , p=0.9919  , df_denom=546, df_num=7
ssr based chi2 test:   chi2=1.1857  , p=0.9913  , df=7
likelihood ratio test: chi2=1.1844  , p=0.9913  , df=7
parameter F test:         F=0.1649  , p=0.9919  , df_denom=546, df_num=7

Granger Causality
number of lags (no zero) 8
ssr based F test:         F=0.1793  , p=0.9937  , df_denom=543, df_num=8
ssr based chi2 test:   chi2=1.4792  , p=0.9930  , df=8
likelihood ratio test: chi2=1.4772  , p=0.9931  , df=8
parameter F test:         F=0.1793  , p=0.9937  , df_denom=543, df_num=8

Granger Causality
number of lags (no zero) 9
ssr based F test:         F=0.2063  , p=0.9934  , df_denom=540, df_num=9
ssr based chi2 test:   chi2=1.9219  , p=0.9926  , df=9
likelihood ratio test: chi2=1.9186  , p=0.9927  , df=9
parameter F test:         F=0.2063  , p=0.9934  , df_denom=540, df_num=9

Granger Causality
number of lags (no zero) 10
ssr based F test:         F=0.2876  , p=0.9839  , df_denom=537, df_num=10
ssr based chi2 test:   chi2=2.9884  , p=0.9817  , df=10
likelihood ratio test: chi2=2.9804  , p=0.9819  , df=10
parameter F test:         F=0.2876  , p=0.9839  , df_denom=537, df_num=10
```

Out[67]:
```
{1: ({'ssr_ftest': (1.4847382820375674, 0.2235440826155682, 564.0, 1),
   'ssr_chi2test': (1.492635826090959, 0.22180794650042257, 1),
   'lrtest': (1.4906745744865475, 0.22211182530371393, 1),
   'params_ftest': (1.4847382820375719, 0.2235440826155682, 564.0, 1.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c4cb80>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c4c970>,
   array([[0., 1., 0.]])]),
 2: ({'ssr_ftest': (1.069816855342166, 0.3437701868130229, 561.0, 2),
   'ssr_chi2test': (2.158703529852642, 0.3398157347414453, 2),
   'lrtest': (2.154597358472529, 0.34051412224307415, 2),
   'params_ftest': (1.0698168553421747, 0.3437701868129909, 561.0, 2.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c3d250>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c3d6d0>,
   array([[0., 0., 1., 0., 0.],
          [0., 0., 0., 1., 0.]])]),
 3: ({'ssr_ftest': (0.17421220160104614, 0.9138412849317383, 558.0, 3),
   'ssr_chi2test': (0.5291929779816724, 0.9124325947240187, 3),
   'lrtest': (0.5289453050027078, 0.9124877591372796, 3),
   'params_ftest': (0.174212201601041, 0.9138412849317383, 558.0, 3.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c3df10>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c3d7f0>,
   array([[0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 1., 0.]])]),
 4: ({'ssr_ftest': (0.32460411181297727, 0.8615054481790502, 555.0, 4),
   'ssr_chi2test': (1.3194718490992374, 0.8580632910410717, 4),
   'lrtest': (1.317930807130324, 0.8583260445605115, 4),
   'params_ftest': (0.32460411181299703, 0.8615054481790502, 555.0, 4.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c35820>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c35520>,
   array([[0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 5: ({'ssr_ftest': (0.22512710805700625, 0.9516370200300538, 552.0, 5),
   'ssr_chi2test': (1.1480666832979576, 0.9497615692676474, 5),
   'lrtest': (1.1468977063941566, 0.949869225969361, 5),
   'params_ftest': (0.22512710805699307, 0.9516370200300538, 552.0, 5.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c35a60>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c350d0>,
   array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 6: ({'ssr_ftest': (0.18175997143766429, 0.9818207837051683, 549.0, 6),
   'ssr_chi2test': (1.116383649704561, 0.9808132945933585, 6),
   'lrtest': (1.1152762970177719, 0.9808626190862404, 6),
   'params_ftest': (0.18175997143763645, 0.9818207837051736, 549.0, 6.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c353d0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c378e0>,
   array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 7: ({'ssr_ftest': (0.16485672569404172, 0.9918609439598816, 546.0, 7),
   'ssr_chi2test': (1.1857002963379153, 0.9912524265694627, 7),
   'lrtest': (1.1844490419789508, 0.9912805580783169, 7),
   'params_ftest': (0.16485672569402712, 0.9918609439598829, 546.0, 7.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c37b50>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c37df0>,
   array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 8: ({'ssr_ftest': (0.17928101321897344, 0.9936560572180944, 543.0, 8),
   'ssr_chi2test': (1.4791509009594863, 0.9930486760193133, 8),
   'lrtest': (1.4772008631316567, 0.9930800065961779, 8),
   'params_ftest': (0.17928101321897394, 0.9936560572180944, 543.0, 8.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c37790>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c37820>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0.]])]),
 9: ({'ssr_ftest': (0.20628746357987543, 0.9934216790682371, 540.0, 9),
   'ssr_chi2test': (1.9219115356858398, 0.9926430738379584, 9),
   'lrtest': (1.9186152035254054, 0.992690121147244, 9),
   'params_ftest': (0.20628746357987177, 0.9934216790682371, 540.0, 9.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c37370>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c37a00>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0.]])]),
 10: ({'ssr_ftest': (0.2875946193174286, 0.9839042227636366, 537.0, 10),
   'ssr_chi2test': (2.9884133627397604, 0.9816954202504975, 10),
   'lrtest': (2.9804394772454543, 0.9818806422684837, 10),
   'params_ftest': (0.28759461931741315, 0.9839042227636392, 537.0, 10.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c379a0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c37d90>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 1., 0.]])])}
```

The F test statistic turns out to be 1.6911 and the corresponding p-value is 0.0795 for lag=10

Since the p-value is more than .05,we can accept the null hypothesis of the test and conclude that knowing the Bank rate is not useful for predicting the future House Prices.

In [69]:
```
#perform Granger-Causality test on the reverse
grangercausalitytests(df_all[['Rate','HP']], maxlag=[10])
```

```
Granger Causality
number of lags (no zero) 10
ssr based F test:         F=1.1592  , p=0.3160  , df_denom=537, df_num=10
ssr based chi2 test:   chi2=12.0453 , p=0.2820  , df=10
likelihood ratio test: chi2=11.9171 , p=0.2906  , df=10
parameter F test:         F=1.1592  , p=0.3160  , df_denom=537, df_num=10
```

```
Out[69]: {10: ({'ssr_ftest': (1.1591961839679512, 0.3160324131523928, 537.0, 10),
            'ssr_chi2test': (12.045278783130666, 0.2820375925164295, 10),
            'lrtest': (11.917112118736327, 0.29064219595800267, 10),
            'params_ftest': (1.1591961839674083, 0.3160324131527695, 537.0, 10.0)},
           [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db244e0f70>,
            <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1db22c44310>,
            array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
                    0., 0., 0., 0., 0.],
                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
                    0., 0., 0., 0., 0.],
                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
                    0., 0., 0., 0., 0.],
                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
                    0., 0., 0., 0., 0.],
                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
                    0., 0., 0., 0., 0.],
                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
                    0., 0., 0., 0., 0.],
                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                    1., 0., 0., 0., 0.],
                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                    0., 1., 0., 0., 0.],
                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                    0., 0., 1., 0., 0.],
                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                    0., 0., 0., 1., 0.]])])}
```

Since the p-value is more than .05,we can accept the null hypothesis of the test and conclude that knowing the House Prices is not useful for predicting the future Bank Rate.

However studies suggest ,The impact of a rise in interest rates on housing prices will depend on the size and timing of the rise, as well as the supply of available property – with properties in short supply, the current high demand ensures prices remain elevated.

Hypothesis V :Outstanding balances(debt) vs House Price

Data Source : https://www.fca.org.uk/data/mortgage-lending-statistics

In [166…

```
#importing various libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import grangercausalitytests, adfuller
from pandas.plotting import lag_plot
from statsmodels.tsa.api import VAR
```

In [167…

```
#importing the dataset
df_mor = pd.read_excel(r'Datasets/mlar-statistics-q1-2022-summary-long-run.xlsx',sheet_name='Sheet1')
df_mor.head()
```

Out[167…

	Year	Quarter	Arrear_Number_of_loan_accounts	Arrears_Balances_outstanding(Millions)	Loan_Book_Number_of_loan_accounts	Loan_Book_Balar
0	2007	Q1	258659	21916.866	15046772	
1	2007	Q2	267210	23041.773	15473936	
2	2007	Q3	273541	24262.272	15475000	
3	2007	Q4	288546	26236.605	15505441	
4	2008	Q1	299588	28439.189	15619464	

In [168…

```
#checking the statiscs of the dataset
df_mor.describe()
```

Out[168…

	Year	Arrear_Number_of_loan_accounts	Arrears_Balances_outstanding(Millions)	Loan_Book_Number_of_loan_accounts	Loan_Book_Bal
count	61.000000	61.000000	61.000000	6.100000e+01	
mean	2014.131148	250872.983607	23718.952508	1.416759e+07	
std	4.440253	73441.630523	9425.993455	9.160587e+05	
min	2007.000000	152929.000000	13321.940000	1.317537e+07	
25%	2010.000000	185317.000000	14819.821000	1.340325e+07	
50%	2014.000000	228517.000000	21768.696000	1.350203e+07	
75%	2018.000000	303163.000000	30292.753000	1.506973e+07	
max	2022.000000	399365.000000	43572.226000	1.564983e+07	

In [169…

```
#checking the data dtypes
df_mor.dtypes
```

Out[169…

```
Year                                       int64
Quarter                                   object
Arrear_Number_of_loan_accounts             int64
Arrears_Balances_outstanding(Millions)   float64
Loan_Book_Number_of_loan_accounts          int64
Loan_Book_Balances_outstanding(Millions) float64
dtype: object
```

In [170…

```
#checking the dataset's to and from dates
print("Starting year of Mortgage Data :",df_mor.Year.min())
print("Ending year of Mortgage data :",df_mor.Year.max())
```

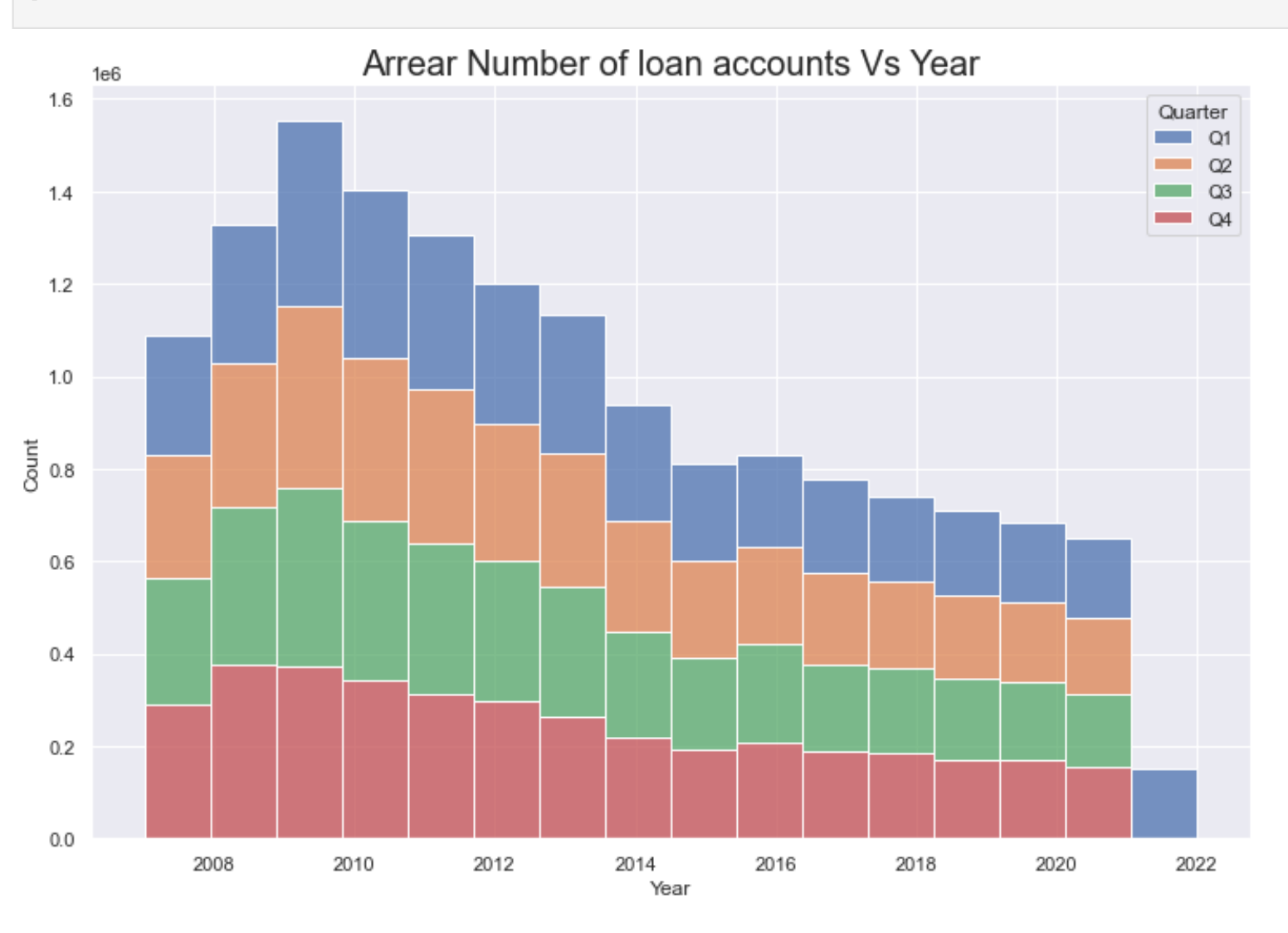
```
Starting year of Mortgage Data : 2007
Ending year of Mortgage data : 2022
```

Columns of Dataset :

- Year : This shows the year
- Quarter : It shows the quarter of the year
- Arrear_Number_of_loan_accounts : It shows the Number of loan accounts which are still pending
- Arrears_Balances_outstanding(Millions) : It shows the total outstanding balance of loans(in millions) which are still pending.
- Loan_Book_Number_of_loan_accounts : It shows the total number of loans in the particular year/quarter from the loan book.
- Loan_Book_Balances_outstanding(Millions) :It shows the total outstanding balance of loans (in millions) in the particular year/quarter from the loan book.

In [171…

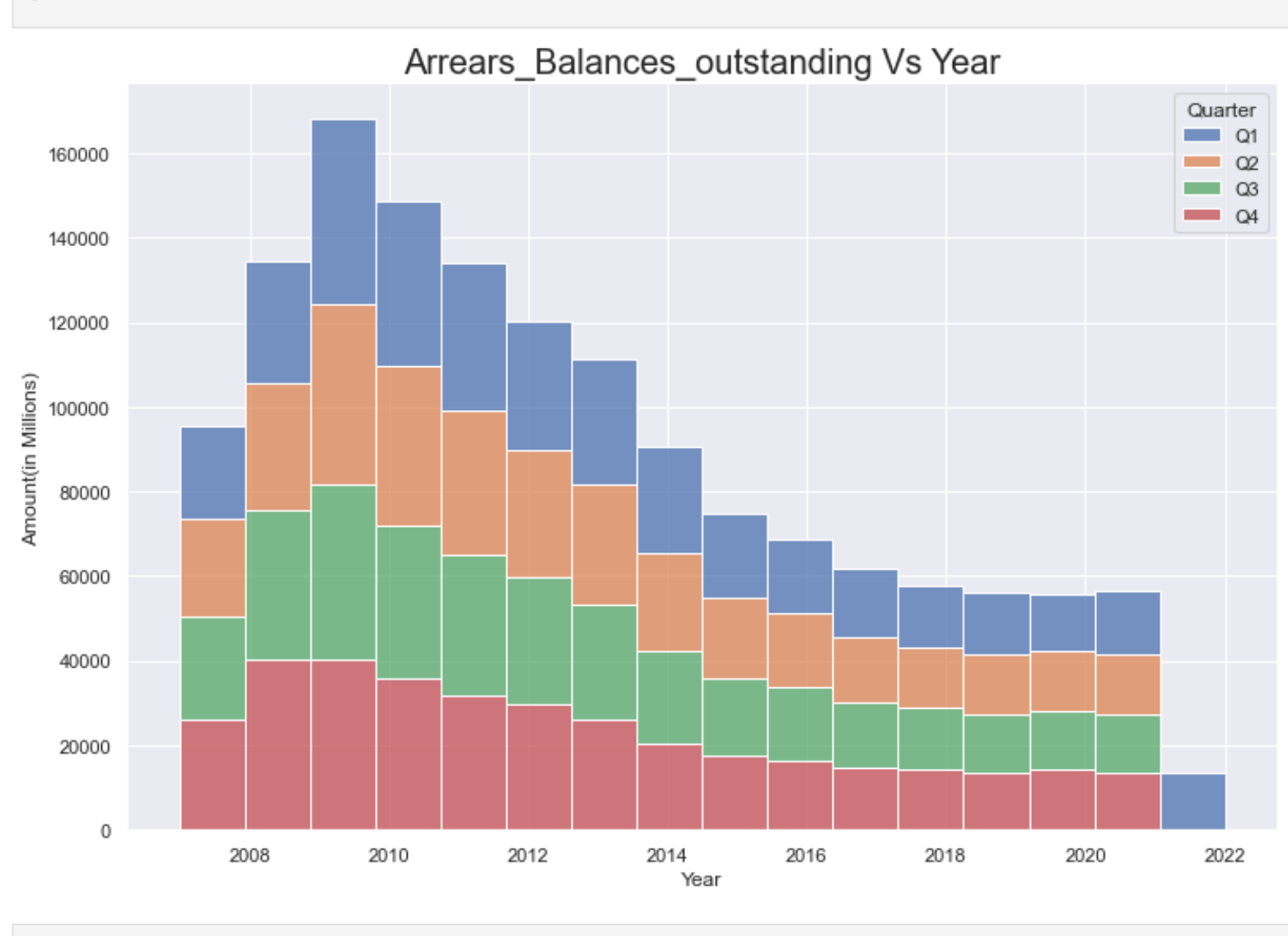
```
#plotting the Arrear_Number_of_loan_accounts with year and quarters
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.histplot(data = df_mor,x='Year',weights='Arrear_Number_of_loan_accounts',hue='Quarter',multiple='stack',bir
plt.title('Arrear Number of loan accounts Vs Year',fontdict={'fontsize':20})
plt.show()
```



We can see in the above chart that number of arrear loans were at the top around 2008 , it was due to the market crash crisis.

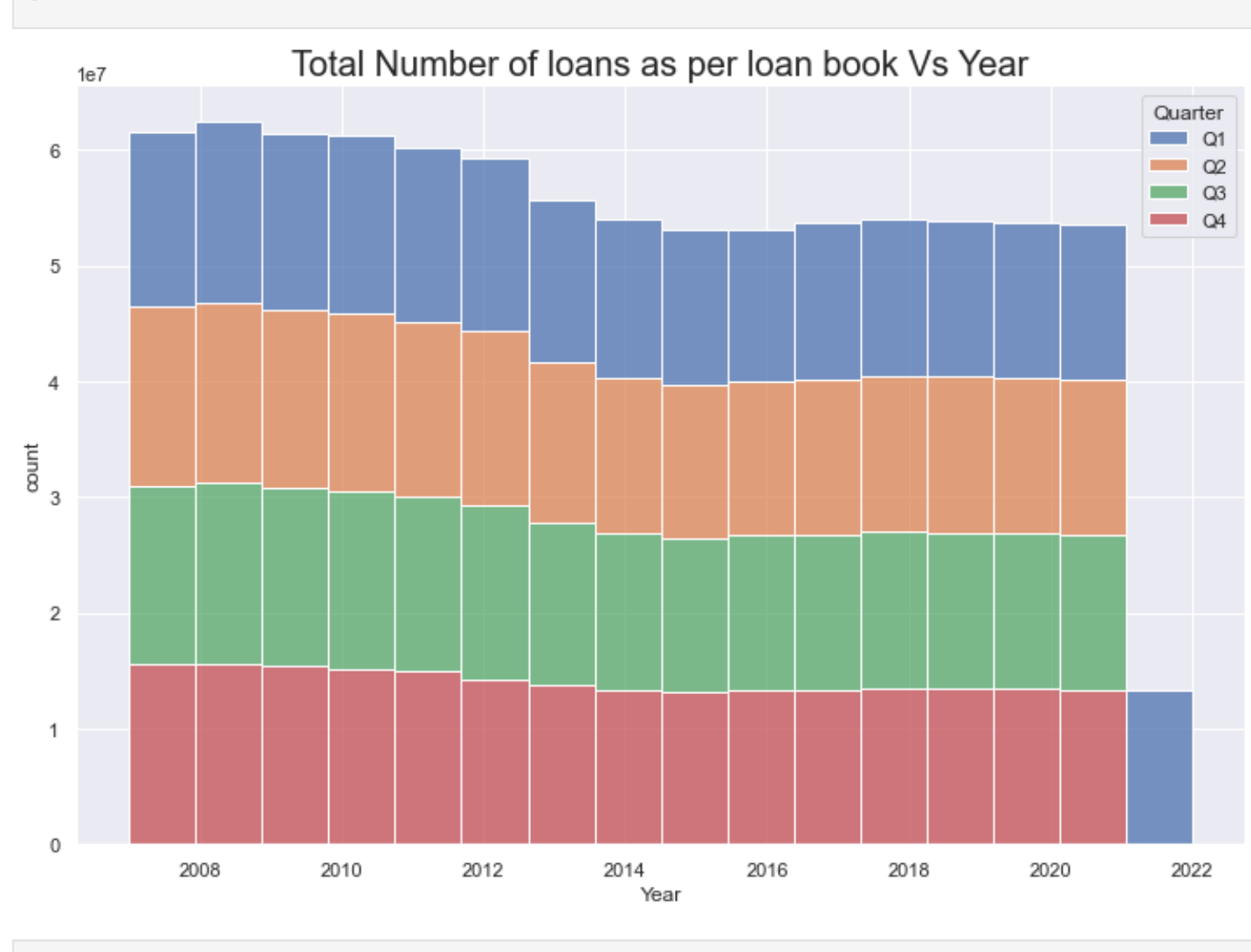
In [172…

```
#plotting the Arrears_Balances_outstanding(Millions) with year and quarters
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.histplot(data = df_mor,x='Year',weights='Arrears_Balances_outstanding(Millions)',hue='Quarter',multiple='st
plt.title('Arrears_Balances_outstanding Vs Year',fontdict={'fontsize':20})
plt.ylabel("Amount(in Millions)")
plt.show()
```



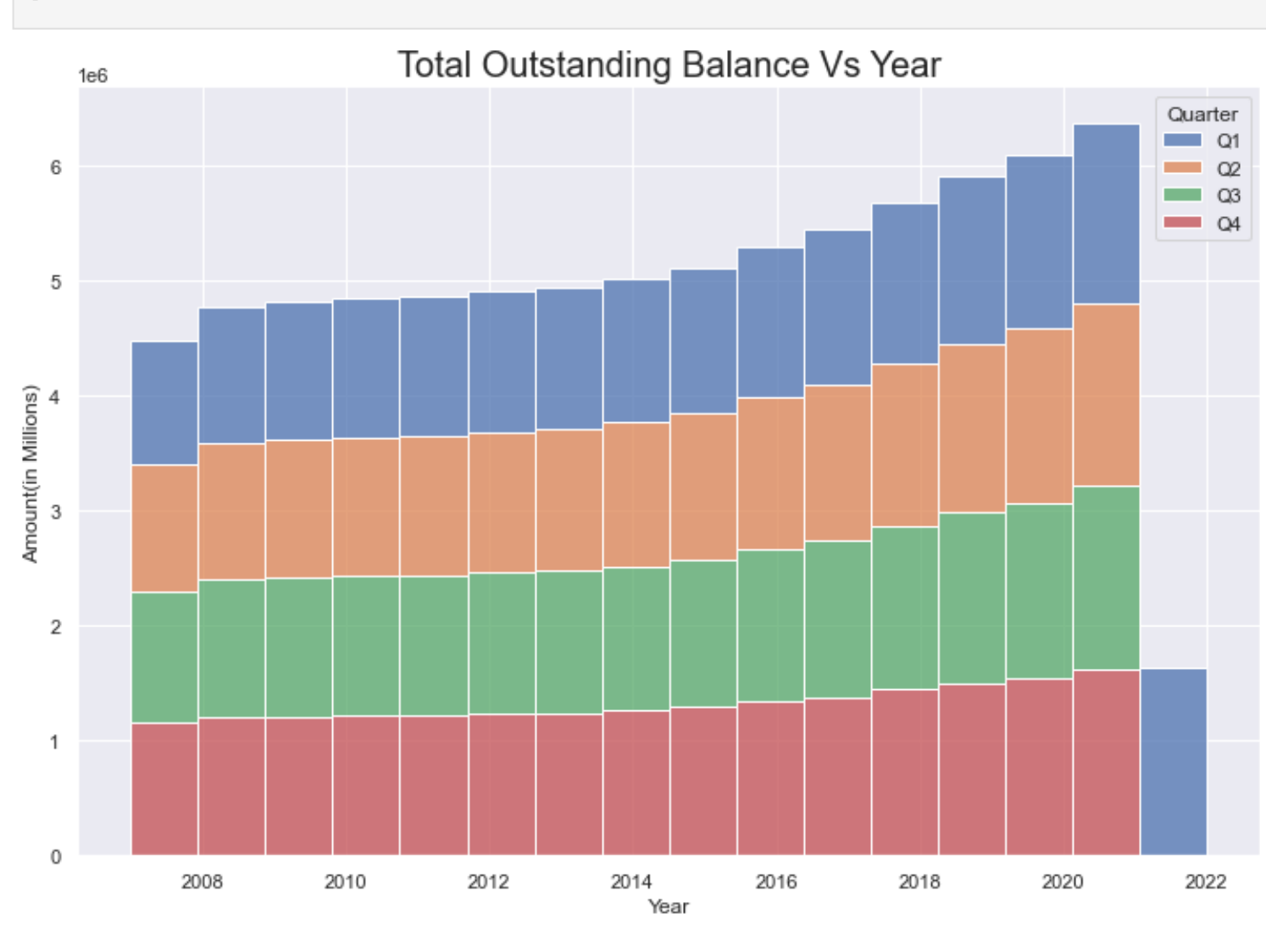
In [173…

```
#plotting the total number of loans with year and quarters
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.histplot(data = df_mor,x='Year',weights='Loan_Book_Number_of_loan_accounts',hue='Quarter',multiple='stack',
plt.title('Total Number of loans as per loan book Vs Year',fontdict={'fontsize':20})
plt.ylabel("count")
plt.show()
```



In [174…

```
#plotting the Loan_Book_Balances_outstanding(Millions) with year and quarters
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.histplot(data = df_mor,x='Year',weights='Loan_Book_Balances_outstanding(Millions)',hue='Quarter',multiple=
plt.title('Total Outstanding Balance Vs Year',fontdict={'fontsize':20})
plt.ylabel("Amount(in Millions)")
plt.show()
```



From the above chart we could clearly see that outstanding balance is increasing per year.

In [175…

```
#grouping by year and quarter
df_grouped = df_mor.groupby('Year').mean()
df_grouped.reset_index(inplace=True)
df_grouped.head()
```

Out[175…

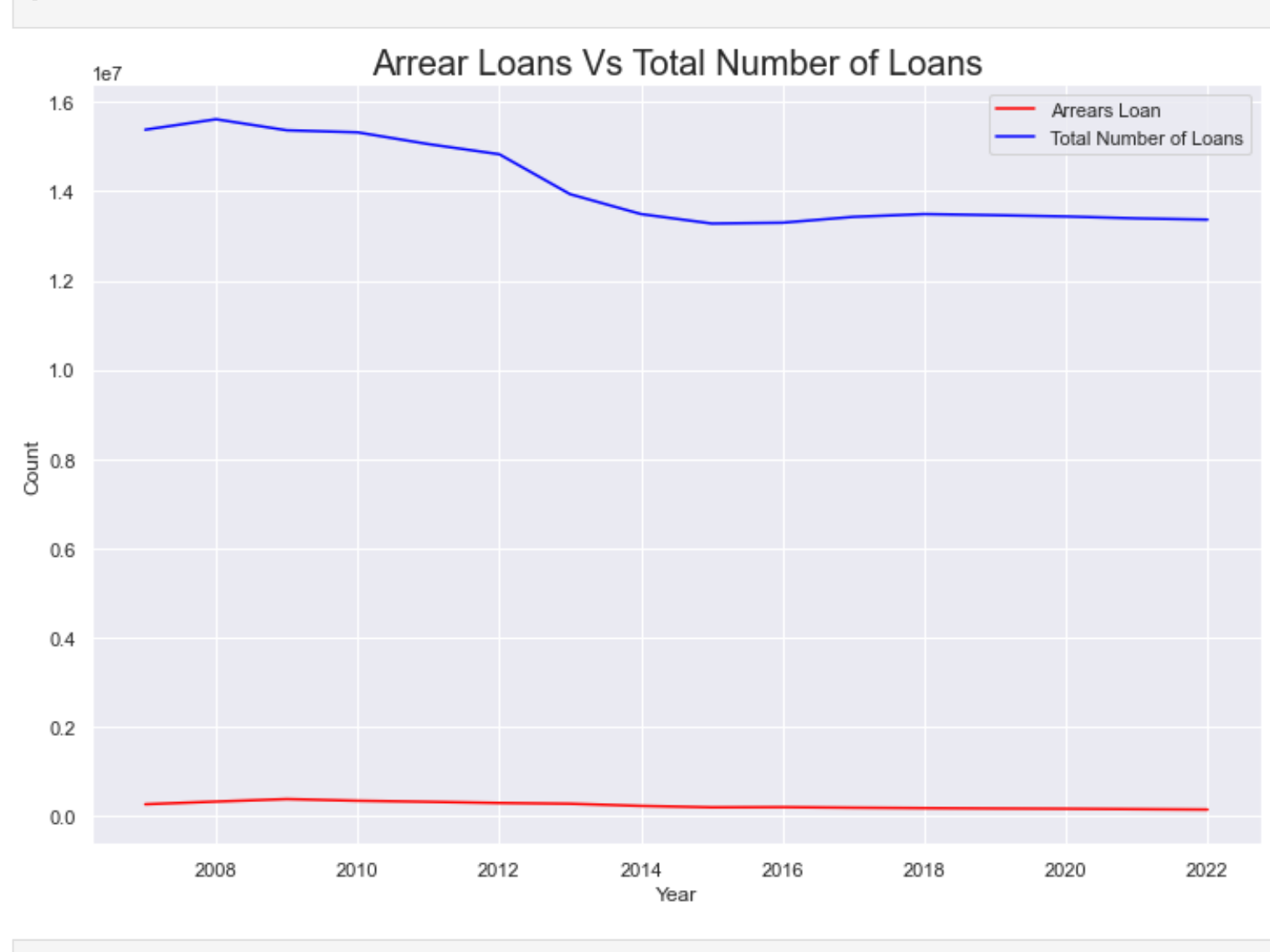
	Year	Arrear_Number_of_loan_accounts	Arrears_Balances_outstanding(Millions)	Loan_Book_Number_of_loan_accounts	Loan_Book_Balances_outst
0	2007	271989.00	23864.3790	15375287.25	
1	2008	331897.50	33555.2775	15609430.25	
2	2009	388116.50	42011.9935	15361548.75	
3	2010	350380.50	37119.9375	15315490.50	
4	2011	326809.75	33516.7295	15052559.00	

In [176…

```
#creating a calculative field of percentage of arrear_loans over total loans
df_grouped['%Arrear_loans'] =(df_grouped['Arrear_Number_of_loan_accounts']/df_grouped['Loan_Book_Number_of_loar
#creating a calculative field of percentage of arrear_loans over total loans
df_grouped['%Arrear_balance'] =(df_grouped['Arrears_Balances_outstanding(Millions)']/df_grouped['Loan_Book_Bala
```

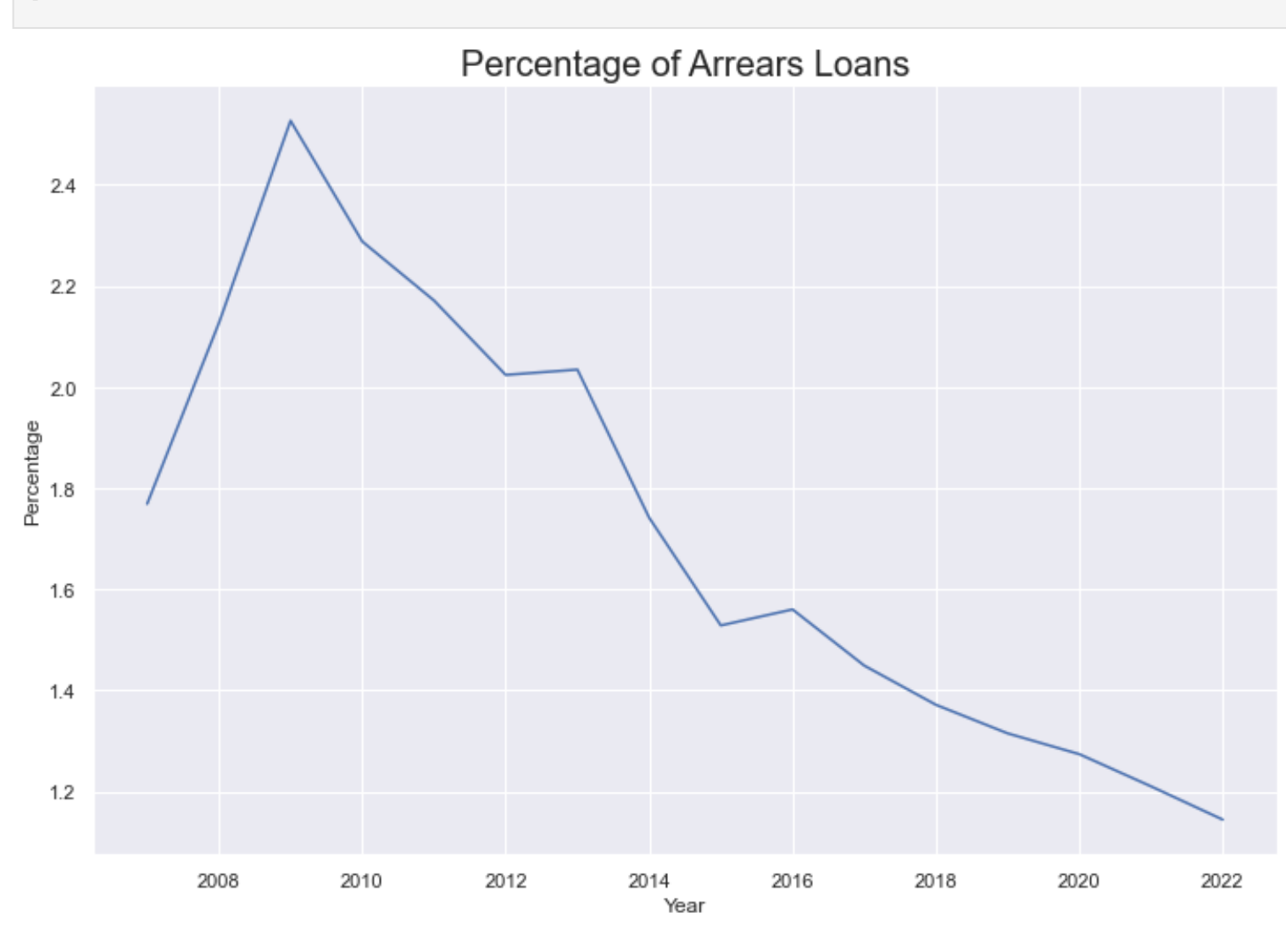
In [177…

```
#plotting the arrear_no_of_loans and loan_book_number_of_loans

plt.figure(figsize=(12,8))
sns.lineplot(data=df_grouped,x='Year',y='Arrear_Number_of_loan_accounts',color='red',label='Arrears Loan')
sns.lineplot(data=df_grouped,x='Year',y='Loan_Book_Number_of_loan_accounts',color='blue',label='Total Number of
plt.title('Arrear Loans Vs Total Number of Loans',fontdict={'fontsize':20})
plt.ylabel("Count")
plt.show()
```



In [178…

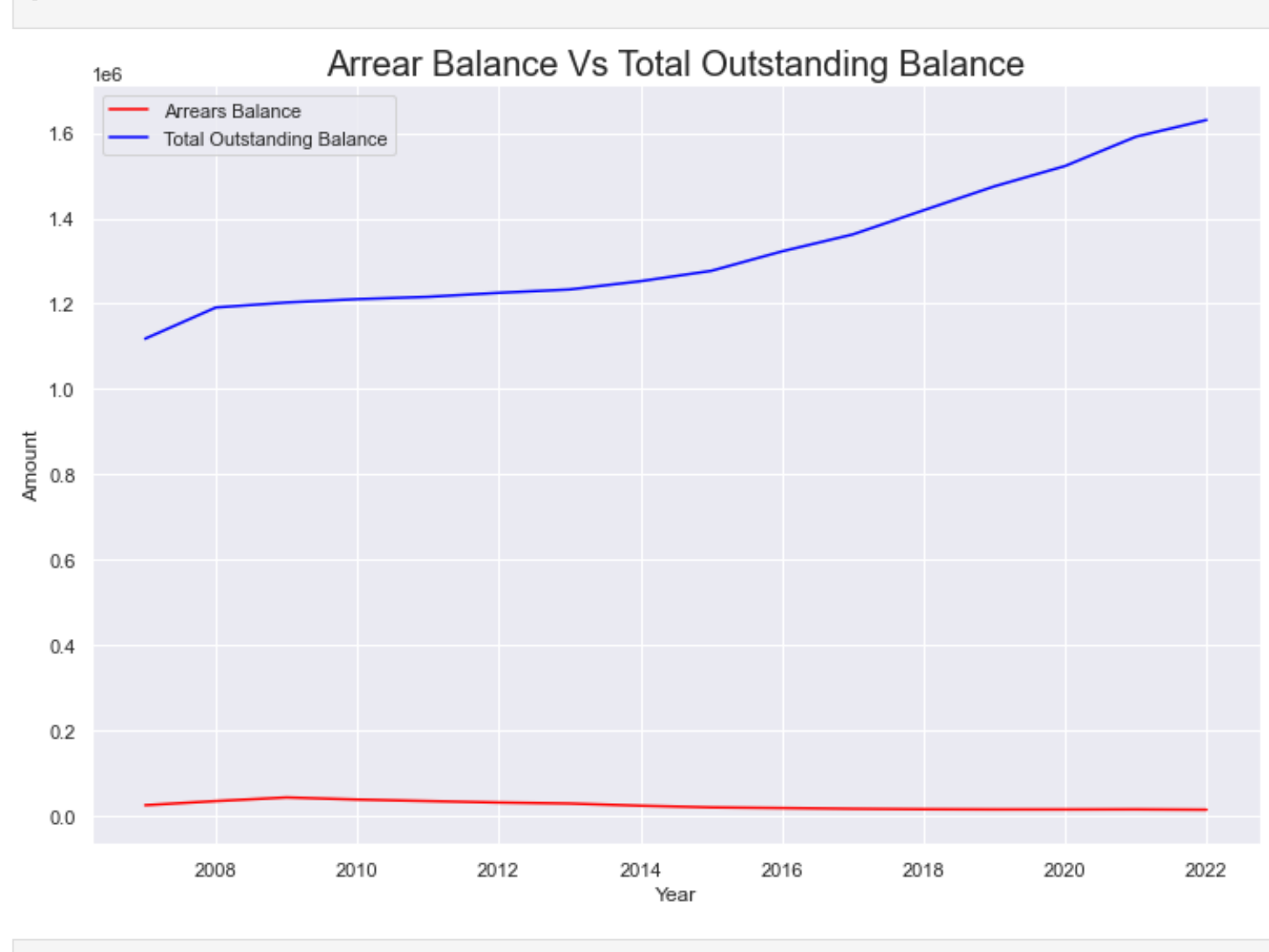
```
#plotting the percentage of arrears loans
plt.figure(figsize=(12,8))
sns.lineplot(data=df_grouped,x='Year',y='%Arrear_loans')
plt.title('Percentage of Arrears Loans',fontdict={'fontsize':20})
plt.ylabel("Percentage")
plt.show()
```



From the above chart, we can see that the from 2009 , the percentage of arrears loans have decreased.While , around 2008 it was the highest due to financial crisis.

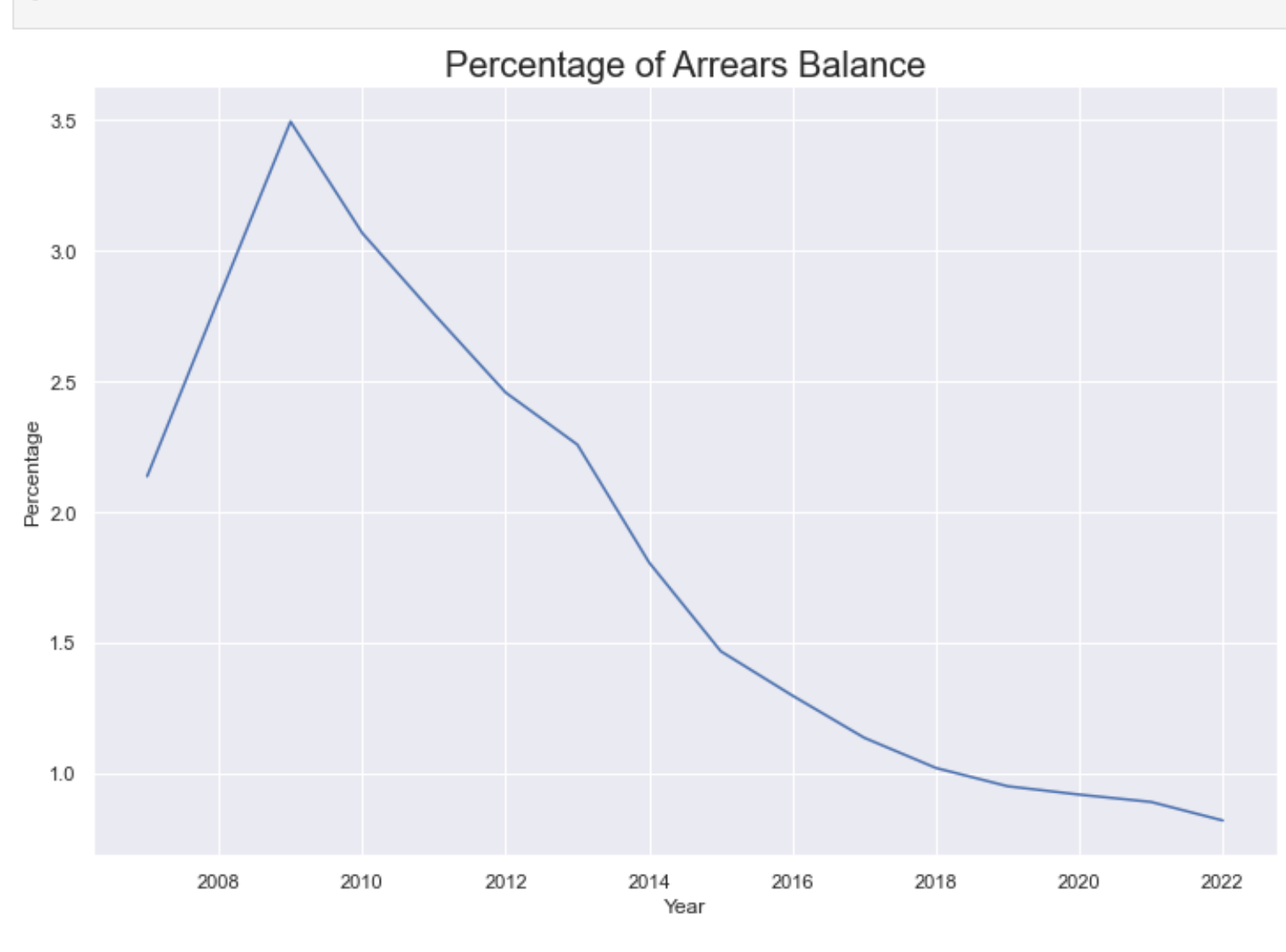
In [179…

```
#plotting the Arrears_Balances_outstanding(Millions) and loan_book_number_of_loans

plt.figure(figsize=(12,8))
sns.lineplot(data=df_grouped,x='Year',y='Arrears_Balances_outstanding(Millions)',color='red',label='Arrears Bal
sns.lineplot(data=df_grouped,x='Year',y='Loan_Book_Balances_outstanding(Millions)',color='blue',label='Total Ou
plt.title('Arrear Balance Vs Total Outstanding Balance',fontdict={'fontsize':20})
plt.ylabel("Amount")
plt.show()
```



In [180…

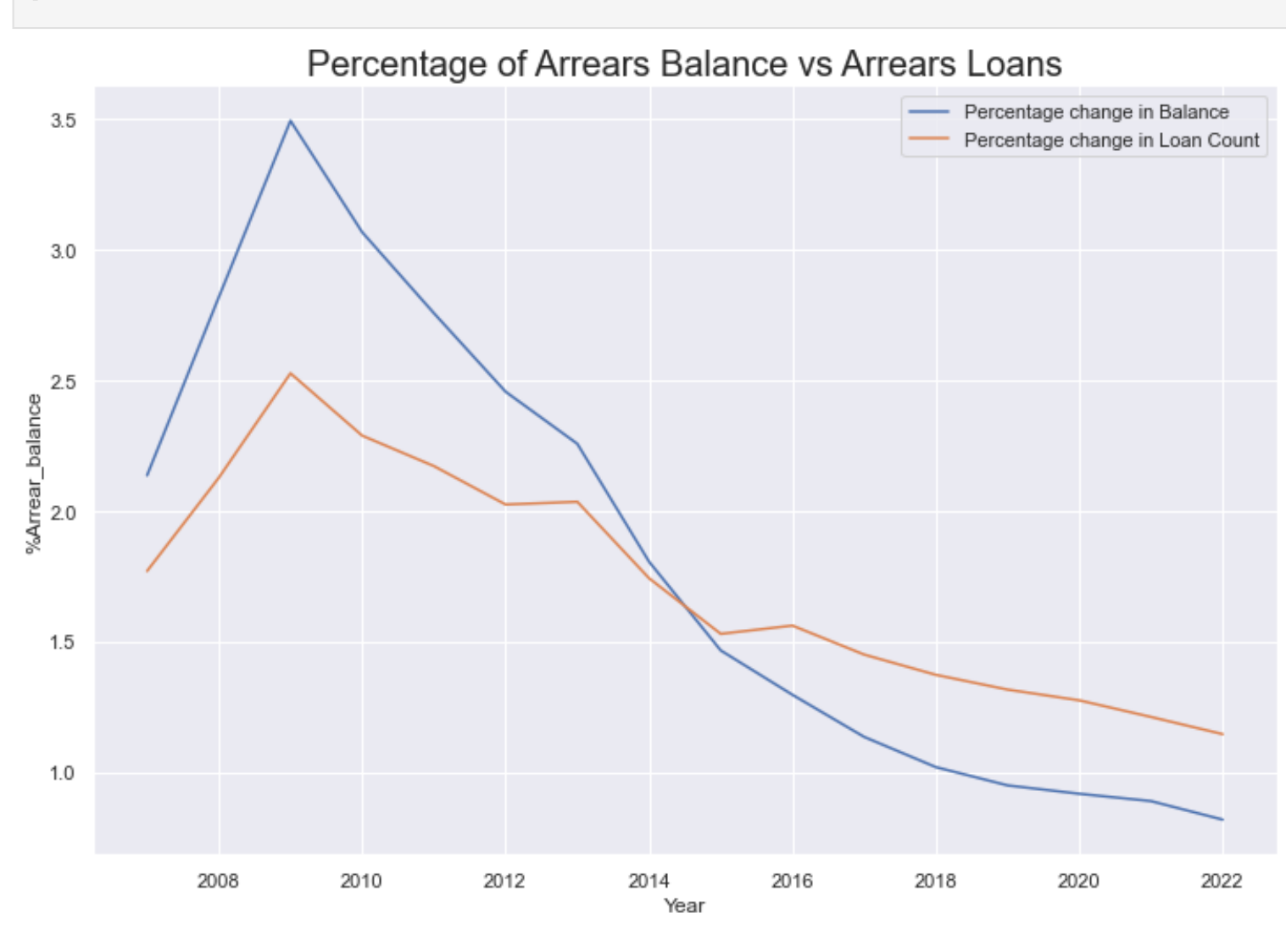
```
#plotting the percentage of arrears loans
plt.figure(figsize=(12,8))
sns.lineplot(data=df_grouped,x='Year',y='%Arrear_balance')
plt.title('Percentage of Arrears Balance',fontdict={'fontsize':20})
plt.ylabel("Percentage")
plt.show()
```



From the above chart, we can see that the from 2009 , the percentage of arrears loans balance have decreased.While , around 2008 it was the highest due to financial crisis.

In [181…

```
#plotting total arrears percentage and total percentage arrears balances
plt.figure(figsize=(12,8))
sns.lineplot(data=df_grouped,x='Year',y='%Arrear_balance',label='Percentage change in Balance')
sns.lineplot(data=df_grouped,x='Year',y='%Arrear_loans',label='Percentage change in Loan Count')
plt.title('Percentage of Arrears Balance vs Arrears Loans',fontdict={'fontsize':20})
# plt.ylabel("Percentage")
plt.show()
```



We can see from the above chart that the arrear balance have reduced however the arrears loan count have increased.

In [182…

```
# importing house price dataset
# importing the dataset for UK house prices from ONS
df_HP = pd.read_csv(r'indices_dataset/Average-prices-2022-05_ONS.csv')
#since the region_name contains all the regions, we would take the region_name as United Kingdom
df_HP = df_HP[df_HP.Region_Name=='United Kingdom']
#we require only the date and average_price
df_HP = df_HP[['Date','Average_Price']]
df_HP.head()
```

Out[182…

	Date	Average_Price
9	1968-04-01	3594.602239
19	1968-05-01	3594.602239
29	1968-06-01	3594.602239
39	1968-07-01	3716.453163
49	1968-08-01	3716.453163

In [183…

```
# checking the statistics
df_HP.describe()
```

Out[183…

	Average_Price
count	650.000000
mean	92682.855832
std	79576.598379
min	3594.602239
25%	21506.687970
50%	58249.724790
75%	167885.249350
max	283496.357300

We can see from the above table , that maximum price is 283496.357 whereas the minium is 3594.60 with a mean of 92682.85 and standard deviation of 79576.59

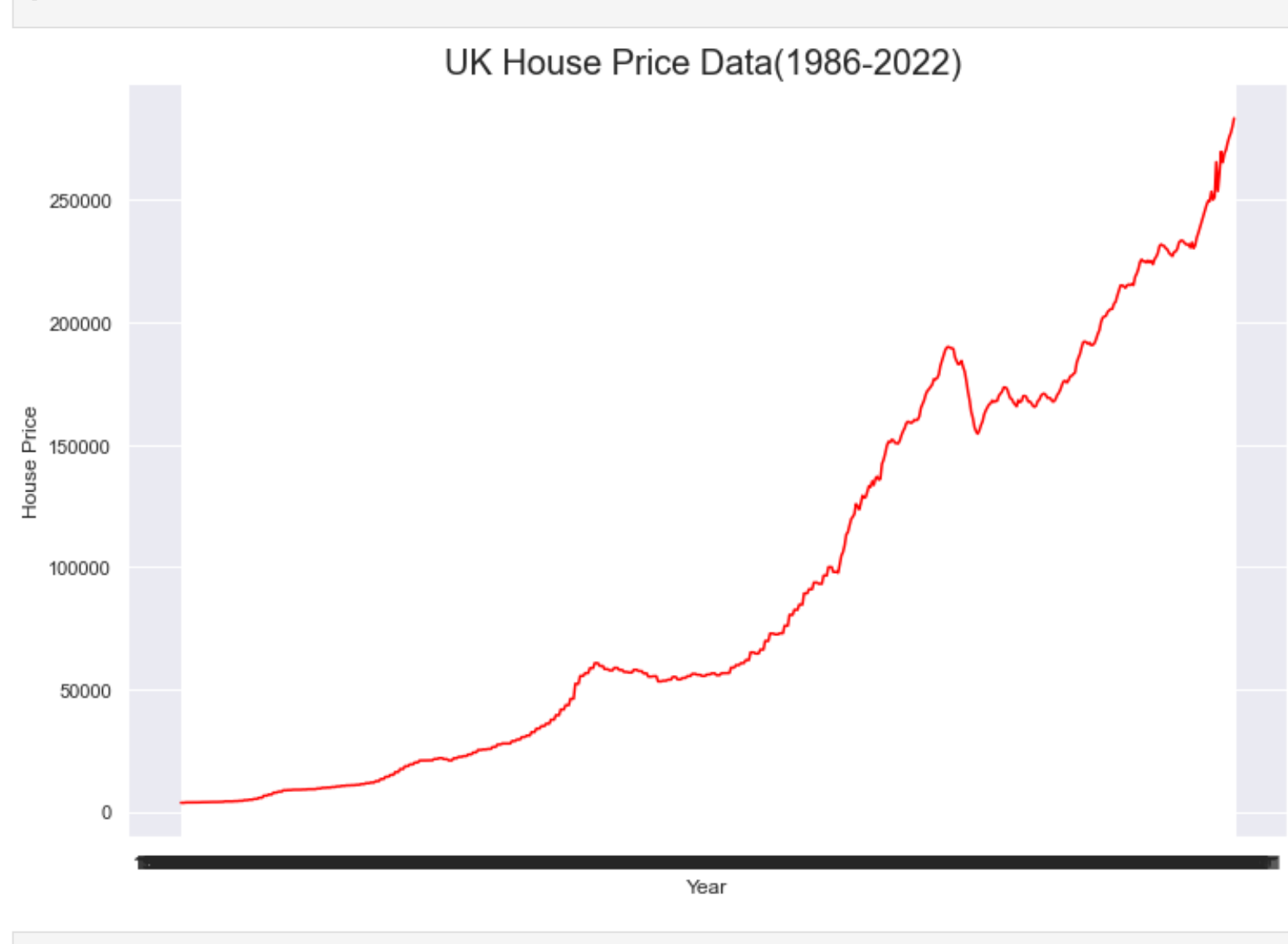
In [184…

```
#checking the dataset's to and from dates
print("Starting date of House Price Data :",df_HP.Date.min())
print("Ending date of House Price data :",df_HP.Date.max())
```

```
Starting date of House Price Data : 1968-04-01
Ending date of House Price data : 2022-05-01
```

In [185…

```
#plotting the House Price data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_HP, x='Date',y='Average_Price',err_style='bars',color='red')
plt.title("UK House Price Data(1986-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('House Price')
plt.show()
```



In [186…

```
#converting the date column into datetime format
df_HP['Date']=pd.to_datetime(df_HP['Date'])
```

In [187…

```
#checking the datatypes of dataset
df_HP.dtypes
```

Out[187…

```
Date             datetime64[ns]
Average_Price           float64
dtype: object
```

In [188…

```
#converting the monthly data into quarterly data
df_qtr = pd.DataFrame(df_HP.set_index('Date').resample('Q')['Average_Price'].mean())
df_qtr
```

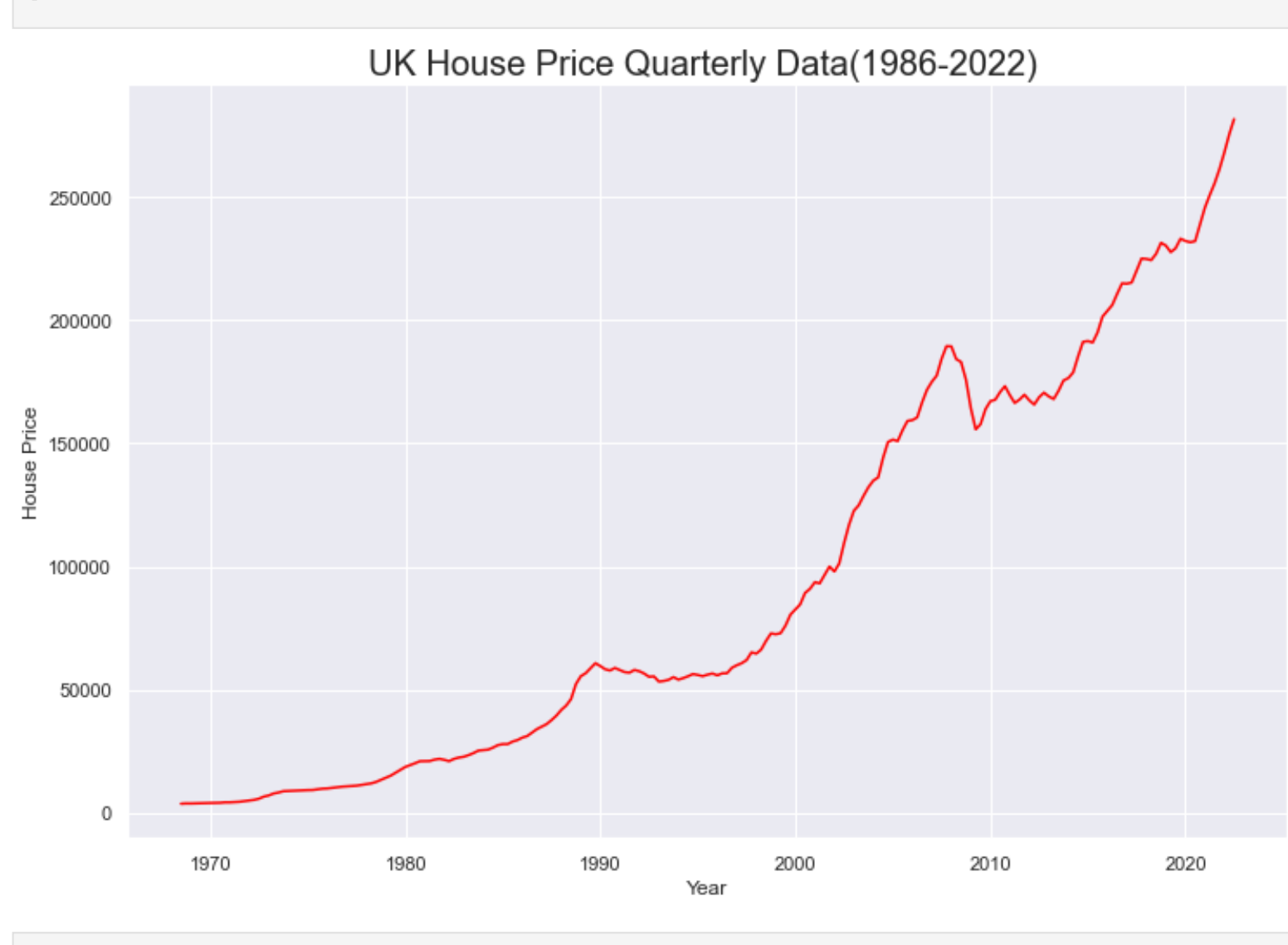
Out[188…

	Average_Price
Date	
1968-06-30	3594.602239
1968-09-30	3716.453163
1968-12-31	3696.144675
1969-03-31	3757.070137
1969-06-30	3817.995599
...	...
2021-06-30	255691.626400
2021-09-30	261520.932267
2021-12-31	268356.800967
2022-03-31	275590.831833
2022-06-30	281790.292950

217 rows × 1 columns

In [189…

```
#plotting the House Price quarterly data
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.lineplot(data = df_qtr, x='Date',y='Average_Price',err_style='bars',color='red')
plt.title("UK House Price Quarterly Data(1986-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('House Price')
plt.show()
```



In [190…

```
#since the dataset of loans is from 2007 Q1 till 2022 Q1 ,thus slicing the house price data as well
start =pd.to_datetime('2006-12-31')
df_qtr =df_qtr[df_qtr.index>start][:-1]
df_qtr
```

Out[190…

	Average_Price
Date	
2007-03-31	177551.810000
2007-06-30	184307.053300
2007-09-30	189502.866233
2007-12-31	189423.837533
2008-03-31	184270.348033
...	...
2021-03-31	250927.062667
2021-06-30	255691.626400
2021-09-30	261520.932267
2021-12-31	268356.800967
2022-03-31	275590.831833

61 rows × 1 columns

In [191…

```
#joining the dataset of loans into house price
df_qtr[['Out_Balance','Ar_Balance']]=df_mor[['Loan_Book_Balances_outstanding(Millions)','Arrears_Balances_outst
df_qtr
```

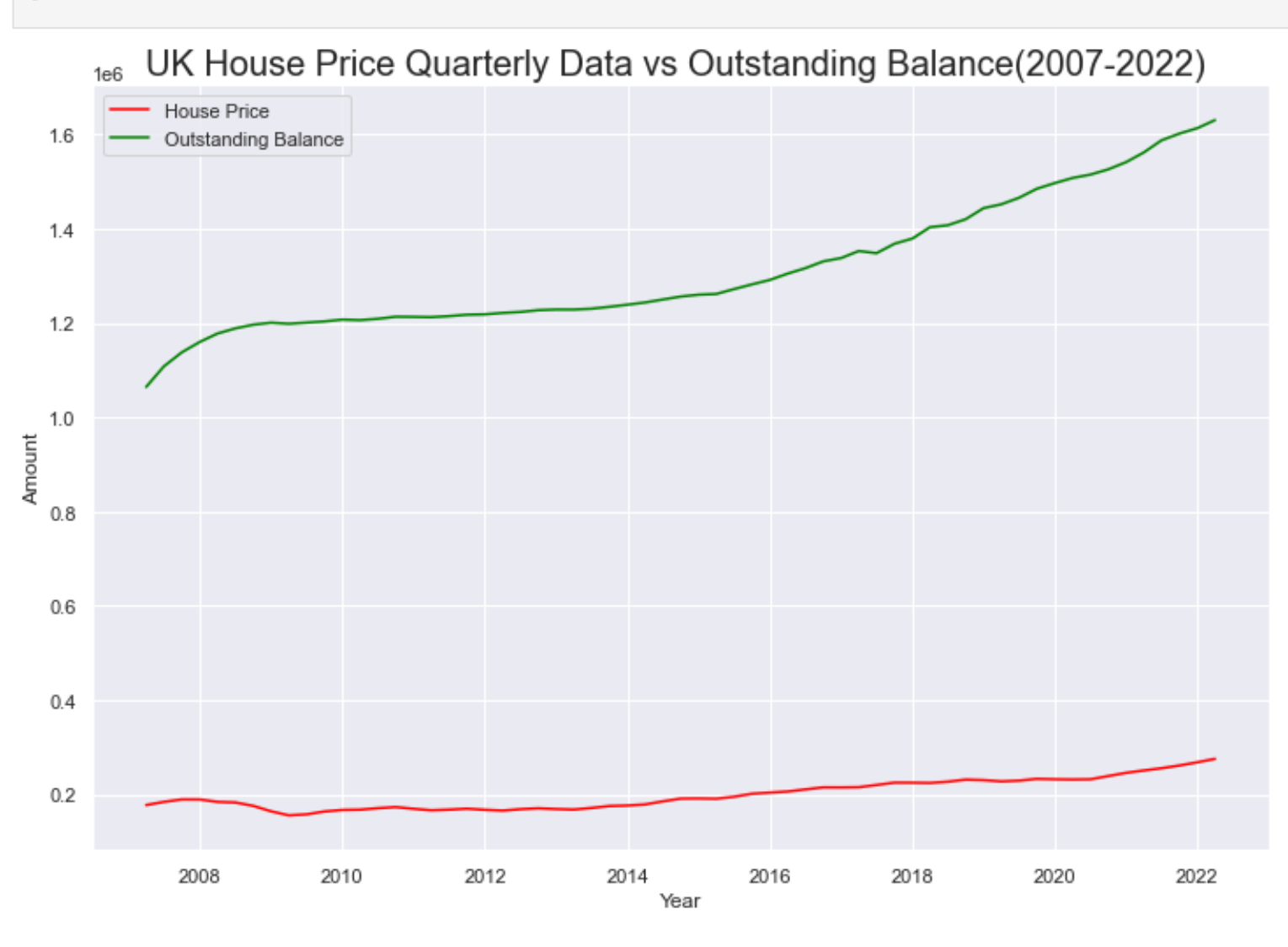
Out[191…

	Average_Price	Out_Balance	Ar_Balance
Date			
2007-03-31	177551.810000	1064363.837	21916.866
2007-06-30	184307.053300	1108071.119	23041.773
2007-09-30	189502.866233	1137898.315	24262.272
2007-12-31	189423.837533	1159861.965	26236.605
2008-03-31	184270.348033	1177761.935	28439.189
...	...	...	...
2021-03-31	250927.062667	1561801.557	15023.426
2021-06-30	255691.626400	1587636.568	14259.491
2021-09-30	261520.932267	1601895.207	13768.081
2021-12-31	268356.800967	1613451.270	13476.106
2022-03-31	275590.831833	1630491.997	13321.940

61 rows × 3 columns

In [192…

```
plt.figure(figsize=(12,8))
sns.lineplot(data = df_qtr, x='Date',y='Average_Price',err_style='bars',color='red',label='House Price')
sns.lineplot(data = df_qtr, x='Date',y='Out_Balance',err_style='bars',color='green',label='Outstanding Balance'
plt.title("UK House Price Quarterly Data vs Outstanding Balance(2007-2022) ",fontdict={'fontsize':20})
plt.xlabel("Year")
plt.ylabel('Amount')
plt.show()
```



Performing Granger-Causality Test Source :https://rishi-a.github.io/2020/05/25/granger-causality.html

The Granger Causality test is used to determine whether or not one time series is useful for forecasting another. This test uses the following null and alternative hypotheses:

Null Hypothesis (H0): Time series Loan data does not Granger-cause time series HP

Alternative Hypothesis (HA): Time series Loan data Granger-causes time series HP

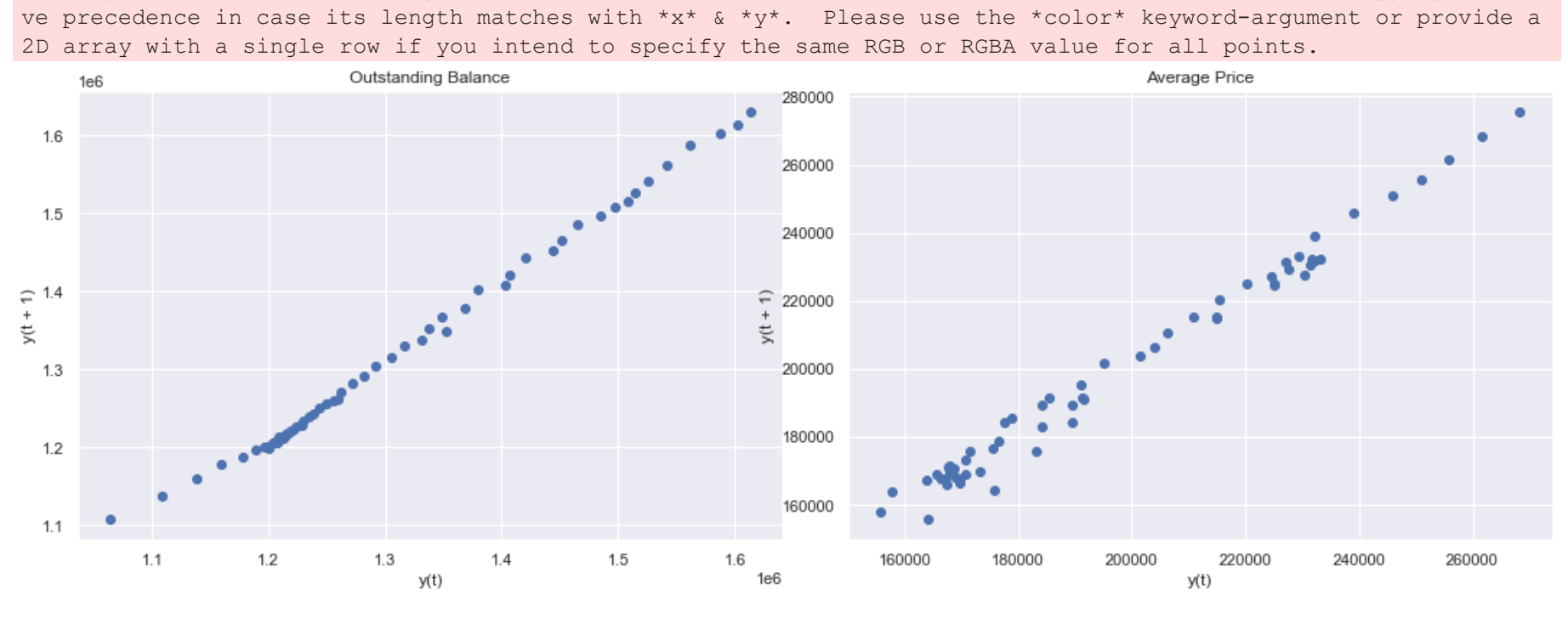
The term "Granger-causes" means that knowing the value of time series x at a certain lag is useful for predicting the value of time series y at a later time period. This test produces an F test statistic with a corresponding p-value. If the p-value is less than a certain significance level (i.e. α = .05), then we can reject the null hypothesis and conclude that we have sufficient evidence to say that time series x Granger-causes time series y.

Step 1: Test each of the time-series to determine their order of integration. Ideally, this should involve using a test (such as the ADF test) for which the null hypothesis is non-stationarity.

In [193…

```
#plotting the lag_plots for Bank rate and HP
fig,axes = plt.subplots(1, 2, figsize=(15, 5))
fig.tight_layout()
lag_plot(df_qtr['Out_Balance'], ax=axes[0])
axes[0].set_title('Outstanding Balance')
lag_plot(df_qtr['Average_Price'], ax=axes[1])
axes[1].set_title('Average Price')
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



we Can see that both the time series have high autocorrelation.

In [194…

```
# differencing the data
df1 = df_qtr.copy()
df_qtr['Out_Balance'] = df_qtr['Out_Balance']- df_qtr['Out_Balance'].shift(1)
df_qtr['Average_Price'] = df_qtr['Average_Price']- df_qtr['Average_Price'].shift(1)
df_qtr = df_qtr.dropna()
df_qtr.head()
```

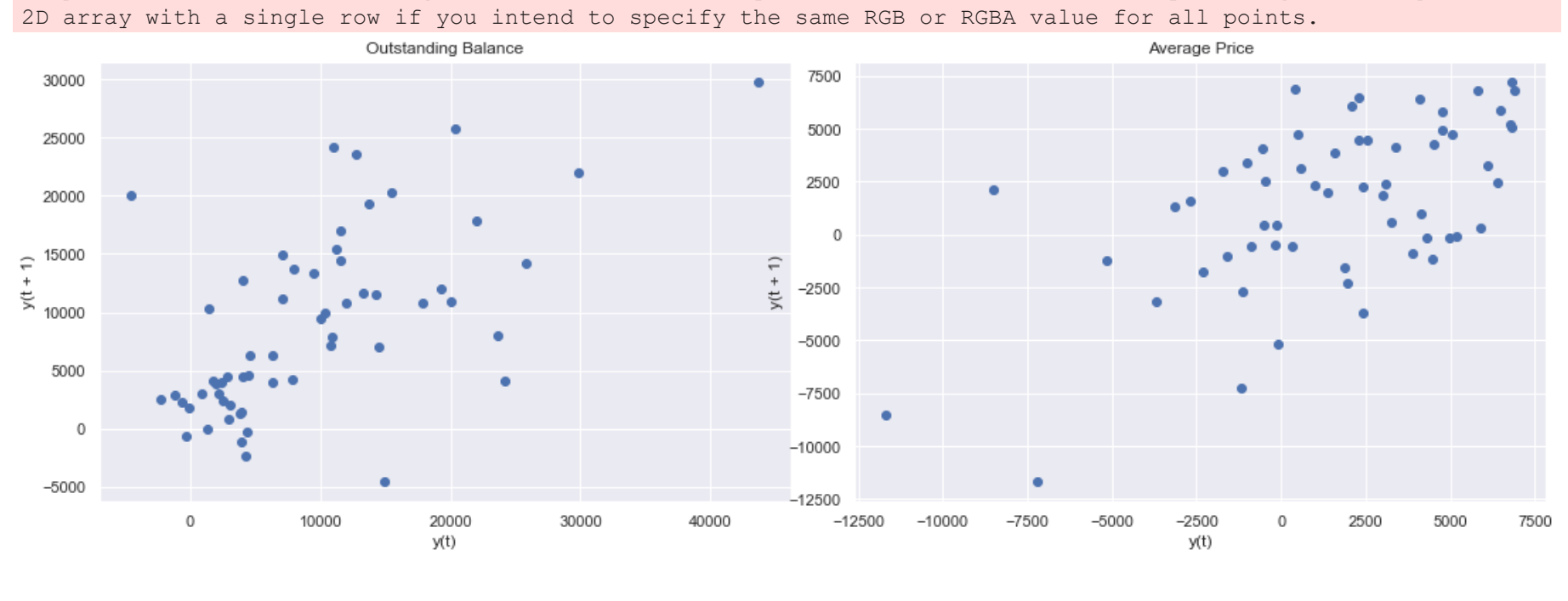
Out[194…

	Average_Price	Out_Balance	Ar_Balance
Date			
2007-06-30	6755.243300	43707.282	23041.773
2007-09-30	5195.812933	29827.196	24262.272
2007-12-31	-79.028700	21963.650	26236.605
2008-03-31	-5153.489500	17899.970	28439.189
2008-06-30	-1188.395133	10854.644	30292.753

In [195…

```
#plotting the lag_plots for Bank rate and HP
fig,axes = plt.subplots(1, 2, figsize=(15, 5))
fig.tight_layout()
lag_plot(df_qtr['Out_Balance'], ax=axes[0])
axes[0].set_title('Outstanding Balance')
lag_plot(df_qtr['Average_Price'], ax=axes[1])
axes[1].set_title('Average Price')
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will ha
ve precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [196…

```python
#defining a function for dicky fuller test
def adfuller_test(columns):
    result=adfuller(columns)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit ro
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-statio
```

In [197…

```python
#applying Dickey Fuller test on loans
adfuller_test(df_qtr['Out_Balance'])
```

```
ADF Test Statistic : -1.2399652020263805
p-value : 0.6561718684286324
#Lags Used : 6
Number of Observations Used : 53
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

In [198…

```python
#applying Dickey Fuller test on House Prices
adfuller_test(df_qtr['Average_Price'])
```

```
ADF Test Statistic : -1.7644808549721533
p-value : 0.3982330189537936
#Lags Used : 6
Number of Observations Used : 53
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

We can see from above tests that ADF Null Hypothesis is rejected: thus data is stationary.Moreover , the above lag plots give confirmation that the data is stationary

Step 2: Let the maximum order of integration for the group of time-series be m. So, if there are two time-series and one is found to be I(1) and the other is I(2), then m = 2. If one is I(0) and the other is I(1), then m = 1, etc. Result: We saw above that for our case m = 1

Step 3: Set up a VAR model in the levels of the data, regardless of the orders of integration of the various time-series. Most importantly, you must not difference the data, no matter what you found at Step 1.//

Step 4: Determine the appropriate maximum lag length for the variables in the VAR, say p, using the usual methods. Specifically, base the choice of p on the usual information criteria, such as AIC, SIC.

In [199…

```python
# applying VAR model and finding the AIC, BIC
df1 = df1.dropna()
model = VAR(df1[['Out_Balance','Average_Price']]) #recall that rawData is w/o difference operation
for i in range(1,13):
    result = model.fit(i)
    try:
        print('Lag Order =', i)
        print('AIC : ', result.aic)
        print('BIC : ', result.bic)

    except:
        continue
```

```
Lag Order = 1
AIC :  33.825561173371256
BIC :  34.03499562959346
Lag Order = 2
AIC :  33.1939311485623
BIC :  33.54605613905479
Lag Order = 3
AIC :  32.93735251853675
BIC :  33.43470083142726
Lag Order = 4
AIC :  32.6379409287026
BIC :  33.283115013281936
Lag Order = 5
AIC :  32.73649753978139
BIC :  33.532171418284484
Lag Order = 6
AIC :  32.14110390394195
BIC :  33.09002504605185
Lag Order = 7
AIC :  32.142771813056754
BIC :  33.2477629500369
Lag Order = 8
AIC :  31.99805415903867
BIC :  33.26201500924192
Lag Order = 9
AIC :  31.965274343407735
BIC :  33.39118321467878
Lag Order = 10
AIC :  31.943893784076977
BIC :  33.53480901102642
Lag Order = 11
AIC :  31.471710760432895
BIC :  33.23077192542679
Lag Order = 12
AIC :  31.54998207412326
BIC :  33.48041094974635
```

Result: The lag-order p = 11 on the basis of Akaike Information Criterion (AIC)

In [200…

```python
#perform Granger-Causality test
grangercausalitytests(df_qtr[['Average_Price','Out_Balance']], maxlag=11)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=1.8126  , p=0.1836  , df_denom=56, df_num=1
ssr based chi2 test:   chi2=1.9097  , p=0.1670  , df=1
likelihood ratio test: chi2=1.8795  , p=0.1704  , df=1
parameter F test:         F=1.8126  , p=0.1836  , df_denom=56, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.3693  , p=0.6930  , df_denom=53, df_num=2
ssr based chi2 test:   chi2=0.8082  , p=0.6676  , df=2
likelihood ratio test: chi2=0.8027  , p=0.6694  , df=2
parameter F test:         F=0.3693  , p=0.6930  , df_denom=53, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=1.5875  , p=0.2041  , df_denom=50, df_num=3
ssr based chi2 test:   chi2=5.4293  , p=0.1429  , df=3
likelihood ratio test: chi2=5.1860  , p=0.1587  , df=3
parameter F test:         F=1.5875  , p=0.2041  , df_denom=50, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.6503  , p=0.6295  , df_denom=47, df_num=4
ssr based chi2 test:   chi2=3.0993  , p=0.5413  , df=4
likelihood ratio test: chi2=3.0166  , p=0.5551  , df=4
parameter F test:         F=0.6503  , p=0.6295  , df_denom=47, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=1.2193  , p=0.3161  , df_denom=44, df_num=5
ssr based chi2 test:   chi2=7.6209  , p=0.1784  , df=5
likelihood ratio test: chi2=7.1372  , p=0.2106  , df=5
parameter F test:         F=1.2193  , p=0.3161  , df_denom=44, df_num=5

Granger Causality
number of lags (no zero) 6
ssr based F test:         F=0.7999  , p=0.5757  , df_denom=41, df_num=6
ssr based chi2 test:   chi2=6.3208  , p=0.3882  , df=6
likelihood ratio test: chi2=5.9774  , p=0.4257  , df=6
parameter F test:         F=0.7999  , p=0.5757  , df_denom=41, df_num=6

Granger Causality
number of lags (no zero) 7
ssr based F test:         F=0.9197  , p=0.5022  , df_denom=38, df_num=7
ssr based chi2 test:   chi2=8.9793  , p=0.2541  , df=7
likelihood ratio test: chi2=8.2950  , p=0.3073  , df=7
parameter F test:         F=0.9197  , p=0.5022  , df_denom=38, df_num=7

Granger Causality
number of lags (no zero) 8
ssr based F test:         F=1.4231  , p=0.2215  , df_denom=35, df_num=8
ssr based chi2 test:   chi2=16.9142 , p=0.0310  , df=8
likelihood ratio test: chi2=14.6442 , p=0.0664  , df=8
parameter F test:         F=1.4231  , p=0.2215  , df_denom=35, df_num=8

Granger Causality
number of lags (no zero) 9
ssr based F test:         F=1.5907  , p=0.1602  , df_denom=32, df_num=9
ssr based chi2 test:   chi2=22.8165 , p=0.0066  , df=9
likelihood ratio test: chi2=18.8576 , p=0.0264  , df=9
parameter F test:         F=1.5907  , p=0.1602  , df_denom=32, df_num=9

Granger Causality
number of lags (no zero) 10
ssr based F test:         F=1.6837  , p=0.1327  , df_denom=29, df_num=10
ssr based chi2 test:   chi2=29.0297 , p=0.0012  , df=10
likelihood ratio test: chi2=22.8900 , p=0.0112  , df=10
parameter F test:         F=1.6837  , p=0.1327  , df_denom=29, df_num=10

Granger Causality
number of lags (no zero) 11
ssr based F test:         F=1.7096  , p=0.1269  , df_denom=26, df_num=11
ssr based chi2 test:   chi2=35.4419 , p=0.0002  , df=11
likelihood ratio test: chi2=26.6679 , p=0.0052  , df=11
parameter F test:         F=1.7096  , p=0.1269  , df_denom=26, df_num=11
```

Out[200…

```
{1: ({'ssr_ftest': (1.8126036540852901, 0.18362073716802496, 56.0, 1),
   'ssr_chi2test': (1.9097074212684304, 0.16699576239712646, 1),
   'lrtest': (1.8794519308135023, 0.170396452447467, 1),
   'params_ftest': (1.812603654085283, 0.1836207371680268, 56.0, 1.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0edbe0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aed9ec0820>,
   array([[0., 1., 0.]])]),
 2: ({'ssr_ftest': (0.3692792716081632, 0.6929967553476781, 53.0, 2),
   'ssr_chi2test': (0.8082338774820176, 0.6675660523706741, 2),
   'lrtest': (0.8026542561051428, 0.6694310355442078, 2),
   'params_ftest': (0.36927927160817203, 0.6929967553476741, 53.0, 2.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0a4d60>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0a4d30>,
   array([[0., 0., 1., 0., 0.],
          [0., 0., 0., 1., 0.]])]),
 3: ({'ssr_ftest': (1.5875061910556032, 0.204064214075811, 50.0, 3),
   'ssr_chi2test': (5.429271173410163, 0.1429307207412319, 3),
   'lrtest': (5.186030251607463, 0.15867103633152216, 3),
   'params_ftest': (1.5875061910555932, 0.20406421407581296, 50.0, 3.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0ed790>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0ed970>,
   array([[0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 1., 0.]])]),
 4: ({'ssr_ftest': (0.6503056516448661, 0.6295082636643949, 47.0, 4),
   'ssr_chi2test': (3.099329063158511, 0.5413427036898034, 4),
   'lrtest': (3.016601338420969, 0.5550510599430135, 4),
   'params_ftest': (0.6503056516448629, 0.629508263664396, 47.0, 4.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0f05b0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0f07f0>,
   array([[0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 5: ({'ssr_ftest': (1.219349282227344, 0.3160940276275042, 44.0, 5),
   'ssr_chi2test': (7.620933013920901, 0.17840133162886848, 5),
   'lrtest': (7.137153659364458, 0.21063823076315075, 5),
   'params_ftest': (1.2193492822273666, 0.3160940276274943, 44.0, 5.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0c1310>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b03d0>,
   array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 6: ({'ssr_ftest': (0.799856666899458, 0.5756855397500065, 41.0, 6),
   'ssr_chi2test': (6.3208185384249855, 0.38822763674348754, 6),
   'lrtest': (5.977435336782605, 0.4257225421705936, 6),
   'params_ftest': (0.799856666899463, 0.5756855397500023, 41.0, 6.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b0610>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b02e0>,
   array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 7: ({'ssr_ftest': (0.9197176434352683, 0.5022100659327848, 38.0, 7),
   'ssr_chi2test': (8.979348571433805, 0.25414213943620895, 7),
   'lrtest': (8.294994407821832, 0.3073017364702035, 7),
   'params_ftest': (0.9197176434352634, 0.502210065932789, 38.0, 7.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b0a00>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b0910>,
   array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 8: ({'ssr_ftest': (1.423071099815329, 0.2215257302590751, 35.0, 8),
   'ssr_chi2test': (16.914216500662192, 0.03101483552814374, 8),
   'lrtest': (14.64417621846951, 0.0664448731983788, 8),
   'params_ftest': (1.423071099815361, 0.22152573025906247, 35.0, 8.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b0cd0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b03a0>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0.]])]),
 9: ({'ssr_ftest': (1.5906934508138169, 0.16022042722871963, 32.0, 9),
   'ssr_chi2test': (22.816509185110686, 0.006621829774947346, 9),
   'lrtest': (18.857595553553665, 0.026431325366399524, 9),
   'params_ftest': (1.5906934508127692, 0.16022042722903979, 32.0, 9.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b0c10>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b0ee0>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0.]])]),
 10: ({'ssr_ftest': (1.6837227862866957, 0.13271850690180956, 29.0, 10),
   'ssr_chi2test': (29.029703211839582, 0.001232322331731564, 10),
   'lrtest': (22.890038319781297, 0.011159823552724301, 10),
   'params_ftest': (1.6837227862877149, 0.13271850690154433, 29.0, 10.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2b0fd0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2ca190>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 1., 0.]])]),
 11: ({'ssr_ftest': (1.7096256038622808, 0.12687758398244703, 26.0, 11),
   'ssr_chi2test': (35.44185386468344, 0.00020945616915490106, 11),
   'lrtest': (26.667901092791908, 0.005156428596696458, 11),
   'params_ftest': (1.7096256038615878, 0.12687758398262, 26.0, 11.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2caee0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2caa00>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 0., 1., 0.]])])}
```

The F test statistic turns out to be 1.7096 and the corresponding p-value is 0.1269 for lag=11

Since the p-value is more than .05,we can accept the null hypothesis of the test and conclude that knowing the Outstanding Loans is not useful for predicting the future House Prices

Trying reverse check

In [201…

```python
#perform Granger-Causality test
grangercausalitytests(df_qtr[['Out_Balance','Average_Price']], maxlag=11)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=6.5854  , p=0.0130  , df_denom=56, df_num=1
ssr based chi2 test:   chi2=6.9382  , p=0.0084  , df=1
likelihood ratio test: chi2=6.5597  , p=0.0104  , df=1
parameter F test:         F=6.5854  , p=0.0130  , df_denom=56, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=6.1703  , p=0.0039  , df_denom=53, df_num=2
ssr based chi2 test:   chi2=13.5047 , p=0.0012  , df=2
likelihood ratio test: chi2=12.1406 , p=0.0023  , df=2
parameter F test:         F=6.1703  , p=0.0039  , df_denom=53, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=4.7081  , p=0.0057  , df_denom=50, df_num=3
ssr based chi2 test:   chi2=16.1018 , p=0.0011  , df=3
likelihood ratio test: chi2=14.1817 , p=0.0027  , df=3
parameter F test:         F=4.7081  , p=0.0057  , df_denom=50, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=3.6058  , p=0.0121  , df_denom=47, df_num=4
ssr based chi2 test:   chi2=17.1849 , p=0.0018  , df=4
likelihood ratio test: chi2=14.9877 , p=0.0047  , df=4
parameter F test:         F=3.6058  , p=0.0121  , df_denom=47, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=2.4211  , p=0.0505  , df_denom=44, df_num=5
ssr based chi2 test:   chi2=15.1322 , p=0.0098  , df=5
likelihood ratio test: chi2=13.3677 , p=0.0202  , df=5
parameter F test:         F=2.4211  , p=0.0505  , df_denom=44, df_num=5

Granger Causality
number of lags (no zero) 6
ssr based F test:         F=2.7282  , p=0.0253  , df_denom=41, df_num=6
ssr based chi2 test:   chi2=21.5593 , p=0.0015  , df=6
likelihood ratio test: chi2=18.1404 , p=0.0059  , df=6
parameter F test:         F=2.7282  , p=0.0253  , df_denom=41, df_num=6

Granger Causality
number of lags (no zero) 7
ssr based F test:         F=2.6852  , p=0.0231  , df_denom=38, df_num=7
ssr based chi2 test:   chi2=26.2157 , p=0.0005  , df=7
likelihood ratio test: chi2=21.2998 , p=0.0034  , df=7
parameter F test:         F=2.6852  , p=0.0231  , df_denom=38, df_num=7

Granger Causality
number of lags (no zero) 8
ssr based F test:         F=1.9992  , p=0.0757  , df_denom=35, df_num=8
ssr based chi2 test:   chi2=23.7622 , p=0.0025  , df=8
likelihood ratio test: chi2=19.5705 , p=0.0121  , df=8
parameter F test:         F=1.9992  , p=0.0757  , df_denom=35, df_num=8

Granger Causality
number of lags (no zero) 9
ssr based F test:         F=1.4747  , p=0.1995  , df_denom=32, df_num=9
ssr based chi2 test:   chi2=21.1527 , p=0.0120  , df=9
likelihood ratio test: chi2=17.6949 , p=0.0389  , df=9
parameter F test:         F=1.4747  , p=0.1995  , df_denom=32, df_num=9

Granger Causality
number of lags (no zero) 10
ssr based F test:         F=1.1655  , p=0.3521  , df_denom=29, df_num=10
ssr based chi2 test:   chi2=20.0953 , p=0.0284  , df=10
likelihood ratio test: chi2=16.8916 , p=0.0768  , df=10
parameter F test:         F=1.1655  , p=0.3521  , df_denom=29, df_num=10

Granger Causality
number of lags (no zero) 11
ssr based F test:         F=1.3518  , p=0.2532  , df_denom=26, df_num=11
ssr based chi2 test:   chi2=28.0249 , p=0.0032  , df=11
likelihood ratio test: chi2=22.1631 , p=0.0231  , df=11
parameter F test:         F=1.3518  , p=0.2532  , df_denom=26, df_num=11
```

Out[201…

```
{1: ({'ssr_ftest': (6.5854448398302265, 0.012983817483576118, 56.0, 1),
   'ssr_chi2test': (6.938236527678273, 0.008437225436222471, 1),
   'lrtest': (6.559681937627374, 0.010431500964808604, 1),
   'params_ftest': (6.585444839830196, 0.012983817483576321, 56.0, 1.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0f0b20>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda09ec10>,
   array([[0., 1., 0.]])]),
 2: ({'ssr_ftest': (6.1702575633462375, 0.0038991762192021463, 53.0, 2),
   'ssr_chi2test': (13.504714666946484, 0.0011681227178144613, 2),
   'lrtest': (12.140581829894472, 0.0023105009535590655, 2),
   'params_ftest': (6.170257563346023, 0.0038991762192028393, 53.0, 2.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0f0c70>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c6100>,
   array([[0., 0., 1., 0., 0.],
          [0., 0., 0., 1., 0.]])]),
 3: ({'ssr_ftest': (4.708122225163655, 0.005689224763593615, 50.0, 3),
   'ssr_chi2test': (16.1017780100597, 0.0010807798169014205, 3),
   'lrtest': (14.181681031679545, 0.002667999932961011, 3),
   'params_ftest': (4.708122225163776, 0.005689224763592848, 50.0, 3.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c6040>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c65e0>,
   array([[0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 1., 0.]])]),
 4: ({'ssr_ftest': (3.605756216254806, 0.012095961541912875, 47.0, 4),
   'ssr_chi2test': (17.18488069023567, 0.0017794249638166128, 4),
   'lrtest': (14.987681013230713, 0.004726835858192697, 4),
   'params_ftest': (3.605756216254701, 0.012095961541914576, 47.0, 4.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2bcd00>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c6820>,
   array([[0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 5: ({'ssr_ftest': (2.4211491238475844, 0.05046268369345787, 44.0, 5),
   'ssr_chi2test': (15.132182024047403, 0.009812226571900905, 5),
   'lrtest': (13.367672495171746, 0.020166419323311376, 5),
   'params_ftest': (2.4211491238475826, 0.05046268369345788, 44.0, 5.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c6160>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda09eaf0>,
   array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 6: ({'ssr_ftest': (2.7281858882942696, 0.0253194982390619, 41.0, 6),
   'ssr_chi2test': (21.559322629447394, 0.0014548116755077588, 6),
   'lrtest': (18.14043769331556, 0.005890652985679399, 6),
   'params_ftest': (2.728185888294336, 0.02531949823905905, 41.0, 6.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c33d0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c3760>,
   array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 7: ({'ssr_ftest': (2.685169946346582, 0.023146270038842457, 38.0, 7),
   'ssr_chi2test': (26.215738160383736, 0.00046085740671403686, 7),
   'lrtest': (21.299803222694436, 0.003350863273570627, 7),
   'params_ftest': (2.68516994634659, 0.023146270038842138, 38.0, 7.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c3670>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c3e50>,
   array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])]),
 8: ({'ssr_ftest': (1.999224210932811, 0.07568693774851468, 35.0, 8),
   'ssr_chi2test': (23.76220776422998, 0.0025118363729859225, 8),
   'lrtest': (19.57050531420896, 0.012089033917528082, 8),
   'params_ftest': (1.9992242109327025, 0.07568693774853019, 35.0, 8.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2c31f0>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2bd580>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0.]])]),
 9: ({'ssr_ftest': (1.474700767068744, 0.19954933386775872, 32.0, 9),
   'ssr_chi2test': (21.152739127642295, 0.01198923384286289, 9),
   'lrtest': (17.694940422222658, 0.038882326059164925, 9),
   'params_ftest': (1.474700767069113, 0.19954933386762028, 32.0, 9.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2bd670>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2bd700>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0.]])]),
 10: ({'ssr_ftest': (1.1655268456684227, 0.3521397483054272, 29.0, 10),
   'ssr_chi2test': (20.095290442559012, 0.028364174745327382, 10),
   'lrtest': (16.8916301469892, 0.0767965869262592, 10),
   'params_ftest': (1.165526845668418, 0.3521397483054298, 29.0, 10.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0e2d00>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda0e2670>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 1., 0.]])]),
 11: ({'ssr_ftest': (1.351849957977892, 0.2531967271920684, 26.0, 11),
   'ssr_chi2test': (28.02488951346476, 0.0032090465977463786, 11),
   'lrtest': (22.163107285389856, 0.023139643967026603, 11),
   'params_ftest': (1.3518499579778187, 0.2531967271921033, 26.0, 11.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2bda30>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2aeda2bda00>,
   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
           0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           1., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 1., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 1., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0., 0., 0., 0., 1., 0.]])])}
```

Since the p-value is more than .05,we can accept the null hypothesis of the test and conclude that knowing the Outstanding Loans is not useful for predicting the future House Prices

In [ ]:

In [ ]:

In [ ]:

Hypothesis VI: Stock Market vs House Price Index

In [1]:
```
#importing various libraries
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy import stats
import seaborn as sns
```

In [2]:
```
#downloading the data from yahoo finance
df_stock=yf.download(tickers='^FTAS',start='2005-01-01',end='2021-12-01',interval="1mo")
```
```
[*********************100%***********************]  1 of 1 completed
```

In [3]:
```
#checking the shape
df_stock.shape
```
Out[3]:
```
(204, 6)
```

In [4]:
```
#printing the top 5 rows
df_stock.head()
```
Out[4]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2005-01-01	2410.850098	2449.860107	2393.300049	2441.219971	2441.219971	0
2005-02-01	2441.270020	2543.479980	2441.270020	2495.459961	2495.459961	0
2005-03-01	2495.459961	2529.159912	2452.340088	2457.729980	2457.729980	0
2005-04-01	2457.729980	2504.800049	2385.709961	2397.050049	2397.050049	0
2005-05-01	2397.129883	2497.510010	2397.129883	2483.350098	2483.350098	0

In [5]:
```
#checking the dtypes
df_stock.dtypes
```
Out[5]:
```
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

In [6]:
```
#picking only the close column
df_stock = df_stock[['Adj Close']]
df_stock.head()
```
Out[6]:

	Adj Close
Date	
2005-01-01	2441.219971
2005-02-01	2495.459961
2005-03-01	2457.729980
2005-04-01	2397.050049
2005-05-01	2483.350098

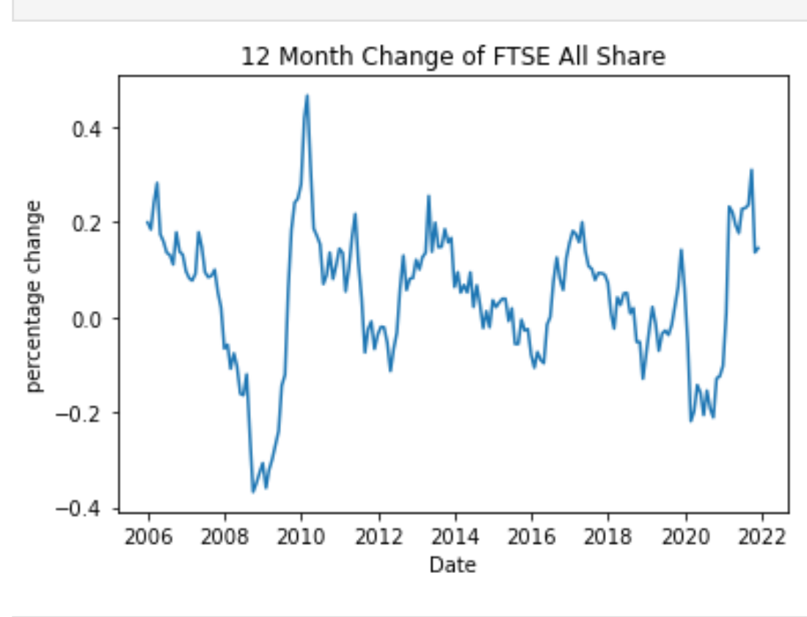
In [7]:
```
#creating a row of percentage change in 12 months
df_stock['12m_change'] = df_stock.pct_change(periods=12)
df_stock
```
```
C:\Users\hp\AppData\Local\Temp/ipykernel_20796/1674755348.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df_stock['12m_change'] = df_stock.pct_change(periods=12)
```
Out[7]:

	Adj Close	12m_change
Date		
2005-01-01	2441.219971	NaN
2005-02-01	2495.459961	NaN
2005-03-01	2457.729980	NaN
2005-04-01	2397.050049	NaN
2005-05-01	2483.350098	NaN
...	...	...
2021-08-01	4109.959961	0.229629
2021-09-01	4058.959961	0.236639
2021-10-01	4129.160156	0.310316
2021-11-01	4025.870117	0.136330
2021-12-01	4208.020020	0.145467

204 rows × 2 columns

In [8]:
```
#dropping the null values
df_stock = df_stock.dropna()
```

In [9]:
```
#plotting a graph of percentage change over time
plt.plot(df_stock['12m_change'])
plt.title('12 Month Change of FTSE All Share')
plt.xlabel("Date")
plt.ylabel("percentage change")
plt.show()
```



In [10]:
```
#reseting the index of the dataframe
df_stock.reset_index(inplace=True)
```

In [11]:
```
# importing the house price index data
df_hp = pd.read_csv("UK-HPI-full-file-2021-12.csv")
df_hp
```
Out[11]:

	Date	RegionName	AreaCode	AveragePrice	Index	IndexSA	1m%Change	12m%Change	AveragePriceSA	SalesVolume
0	01/01/2004	Aberdeenshire	S12000034	81693.66964	40.864214	NaN	NaN	NaN	NaN	388.0
1	01/02/2004	Aberdeenshire	S12000034	81678.76231	40.856757	NaN	-0.018248	NaN	NaN	326.0
2	01/03/2004	Aberdeenshire	S12000034	83525.09702	41.780317	NaN	2.260483	NaN	NaN	453.0
3	01/04/2004	Aberdeenshire	S12000034	84333.67900	42.184780	NaN	0.968071	NaN	NaN	571.0
4	01/05/2004	Aberdeenshire	S12000034	86379.95396	43.208353	NaN	2.426403	NaN	NaN	502.0
...	...	...	...	...	...	...	...	...	...	...
134824	01/08/2021	Yorkshire and The Humber	E12000003	188016.42450	138.076091	135.074056	4.131720	9.951090	183928.5929	5925.0
134825	01/09/2021	Yorkshire and The Humber	E12000003	196554.37790	144.346220	142.216028	4.541068	14.390502	193653.7235	5744.0
134826	01/10/2021	Yorkshire and The Humber	E12000003	189337.96220	139.046606	137.102525	-3.671460	8.590378	186690.7317	2538.0
134827	01/11/2021	Yorkshire and The Humber	E12000003	193306.58140	141.961093	139.622055	2.096050	8.510908	190121.5441	NaN
134828	01/12/2021	Yorkshire and The Humber	E12000003	196876.82130	144.583017	141.773263	1.846931	9.793483	193050.8152	NaN

134829 rows × 54 columns

In [12]:
```
#converting the date column into datetime format
df_hp['Date'] = pd.to_datetime(df_hp['Date'])
```

In [13]:
```
#selecting only date, index and percentage change columns
df_hp = df_hp[['Date','Index','12m%Change']]
df_hp
```
Out[13]:

	Date	Index	12m%Change
0	2004-01-01	40.864214	NaN
1	2004-01-02	40.856757	NaN
2	2004-01-03	41.780317	NaN
3	2004-01-04	42.184780	NaN
4	2004-01-05	43.208353	NaN
...	...	...	...
134824	2021-01-08	138.076091	9.951090
134825	2021-01-09	144.346220	14.390502
134826	2021-01-10	139.046606	8.590378
134827	2021-01-11	141.961093	8.510908
134828	2021-01-12	144.583017	9.793483

134829 rows × 3 columns

In [14]:
```
#grouping the dataframe by date and taking the mean
df_grouped = df_hp.groupby(['Date']).mean()
df_grouped
```
Out[14]:

	Index	12m%Change
Date		
1968-01-04	1.987761	NaN
1968-01-05	1.987761	NaN
1968-01-06	1.987761	NaN
1968-01-07	2.040551	NaN
1968-01-08	2.040551	NaN
...	...	...
2021-01-08	136.318513	10.393528
2021-01-09	137.304337	10.145492
2021-01-10	139.442037	10.848571
2021-01-11	141.162944	11.214669
2021-01-12	141.910027	10.887216

645 rows × 2 columns

In [15]:
```
#picking the date from start='2005-12-01',end='2021-12-01'
start='2005-12-01'
end='2021-12-01'
dates = (df_grouped.index > start) & (df_grouped.index <= end)
df_grouped = df_grouped[dates]
df_grouped
```
Out[15]:

	Index	12m%Change
Date		
2006-01-01	85.943123	5.505685
2006-01-02	85.978339	6.023725
2006-01-03	86.181003	6.289991
2006-01-04	87.359124	6.721187
2006-01-05	88.432401	6.869807
...	...	...
2021-01-08	136.318513	10.393528
2021-01-09	137.304337	10.145492
2021-01-10	139.442037	10.848571
2021-01-11	141.162944	11.214669
2021-01-12	141.910027	10.887216

192 rows × 2 columns

In [16]:
```
#reseting the index
df_grouped.reset_index(inplace=True)
```

In [17]:
```
#changing the date format
# date in YYYY-MM-DD format
df_grouped['Date'] = df_grouped['Date'].dt.strftime('%Y-%d-%m')
```
```
C:\Users\hp\AppData\Local\Temp/ipykernel_20796/1852874333.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df_grouped['Date'] = df_grouped['Date'].dt.strftime('%Y-%d-%m')
```

In [18]:
```
#copying the dataset columns
df_grouped[['stock_price','12m_change_stock']]= df_stock[['Adj Close','12m_change']]
```
```
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\frame.py:3641: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  self[k1] = value[k2]
```

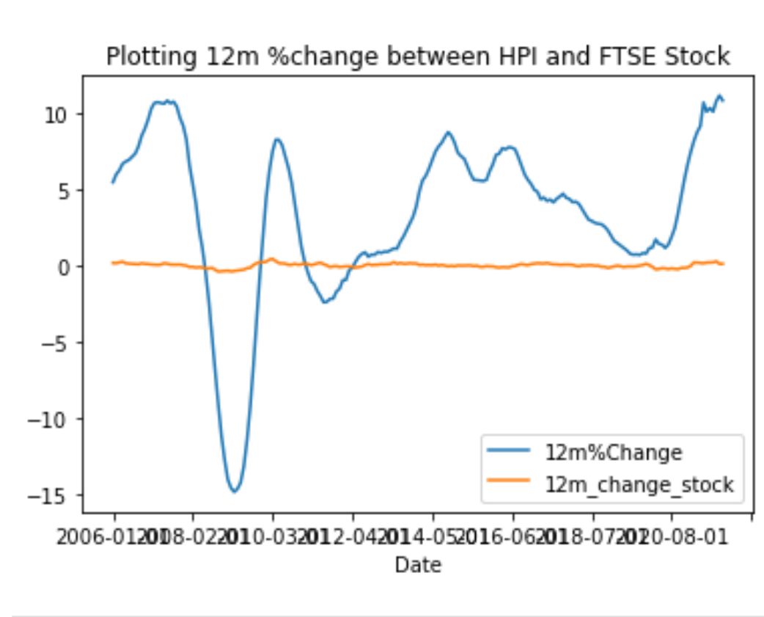
In [19]:
```
#setting the index as date
df_grouped.set_index('Date',inplace=True)
df_grouped
```
Out[19]:

	Index	12m%Change	stock_price	12m_change_stock
Date				
2006-01-01	85.943123	5.505685	2928.560059	0.199630
2006-02-01	85.978339	6.023725	2956.120117	0.184599
2006-03-01	86.181003	6.289991	3047.959961	0.240152
2006-04-01	87.359124	6.721187	3074.260010	0.282518
2006-05-01	88.432401	6.869807	2916.850098	0.174563
...	...	...	...	...
2021-08-01	136.318513	10.393528	4109.959961	0.229629
2021-09-01	137.304337	10.145492	4058.959961	0.236639
2021-10-01	139.442037	10.848571	4129.160156	0.310316
2021-11-01	141.162944	11.214669	4025.870117	0.136330
2021-12-01	141.910027	10.887216	4208.020020	0.145467

192 rows × 4 columns

Checking relationship between 12 month percentage change

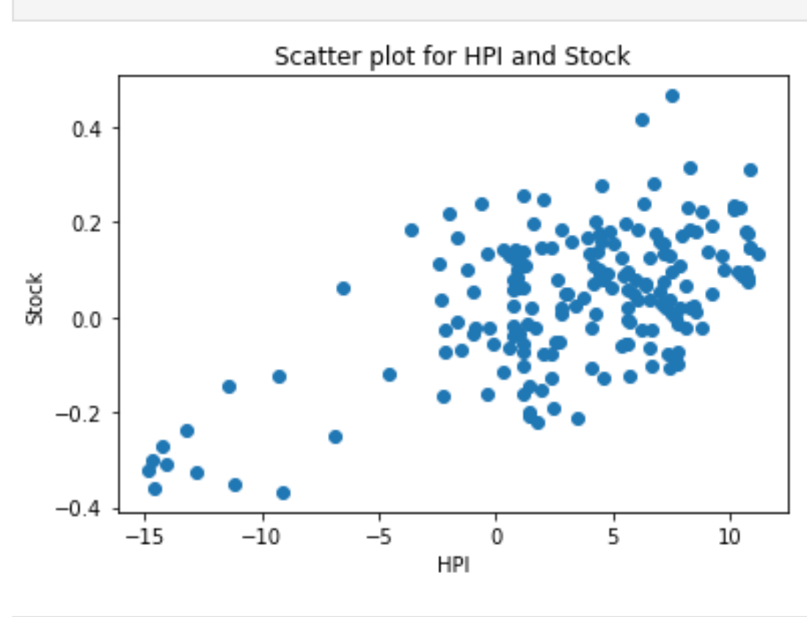
In [20]:
```
#plotting the percentage changes of both time series
df_grouped[['12m%Change','12m_change_stock']].plot(title='Plotting 12m %change between HPI and FTSE Stock')
```
Out[20]:
```
<AxesSubplot:title={'center':'Plotting 12m %change between HPI and FTSE Stock'}, xlabel='Date'>
```



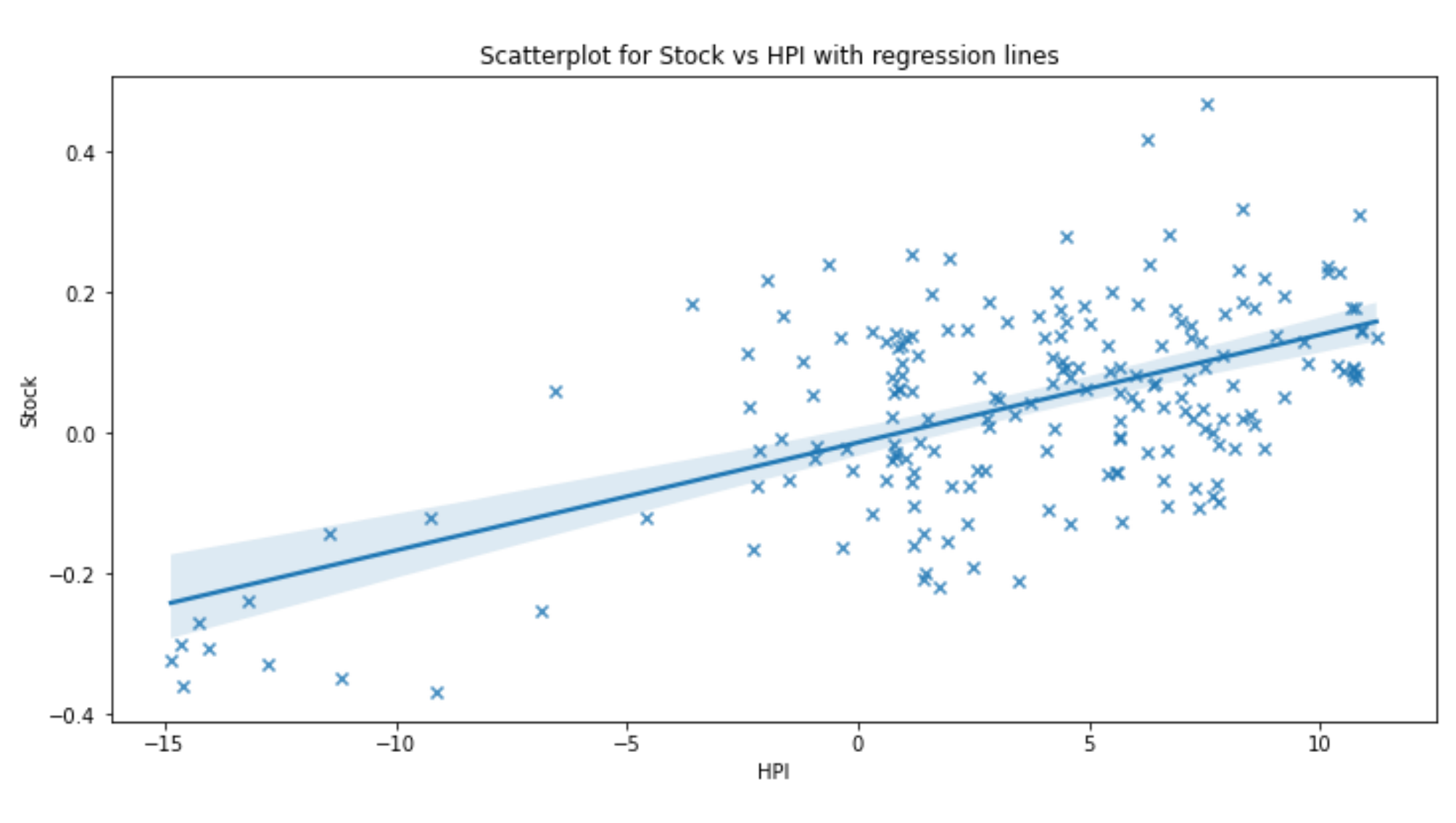
In [21]:
```
#checking the correlation between HPI and stock
df_grouped[['12m%Change','12m_change_stock']].corr()
```
Out[21]:

	12m%Change	12m_change_stock
12m%Change	1.000000	0.581861
12m_change_stock	0.581861	1.000000

In [22]:
```
#plotting the scatter plot chart
plt.scatter(df_grouped['12m%Change'],df_grouped['12m_change_stock'])
plt.xlabel("HPI")
plt.ylabel('Stock')
plt.title('Scatter plot for HPI and Stock')
plt.show()
```



In [33]:
```
#using regression plot

plt.figure(figsize=(12,6))

# Scatterplot with regression lines
sns.regplot(x='12m%Change', y='12m_change_stock', data=df_grouped,
           marker='x')

# Aesthetics
plt.title('Scatterplot for Stock vs HPI with regression lines ')
plt.xlabel('HPI')
plt.ylabel('Stock')
```
Out[33]:
```
Text(0, 0.5, 'Stock')
```



In [37]:
```
#Slope of regression line

slope, intercept, r_value, p_value, std_err = stats.linregress(df_grouped['12m%Change'],df_grouped['12m_change_

print('The slope of the regression line is: ','{:.2f}'.format(slope))
```
```
The slope of the regression line is:  0.02
```

Cross correlation Function

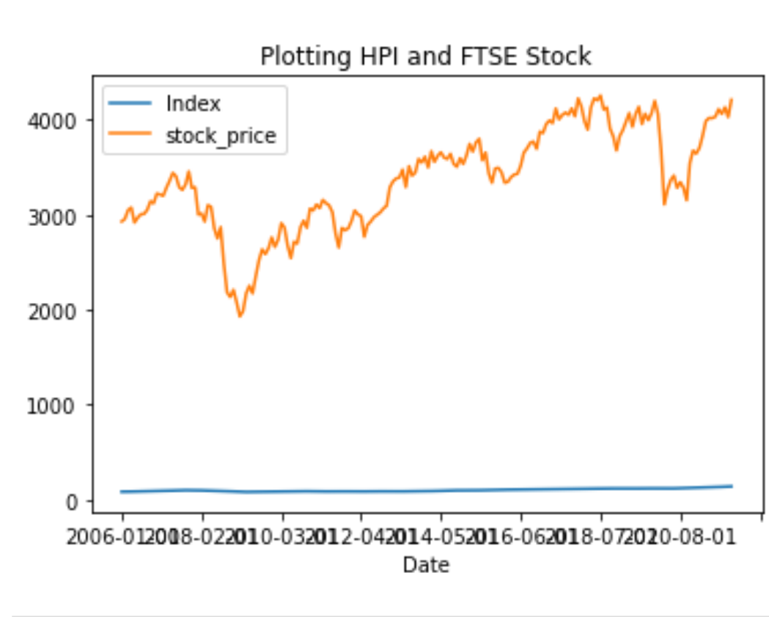
Cross correlation is a way to measure the degree of similarity between a time series and a lagged version of another time series.

This type of correlation is useful to calculate because it can tell us if the values of one time series are predictive of the future values of another time series. In other words, it can tell us if one time series is a leading indicator for another time series.

In [30]:
```
#calculate cross correlation
sm.tsa.stattools.ccf(df_grouped['12m_change_stock'],df_grouped['12m%Change'], adjusted=False)
```
Out[30]:
```
array([ 0.58186145,  0.53065811,  0.4653062 ,  0.38683361,  0.29951717,
        0.2078333 ,  0.11595235,  0.02558685, -0.06048453, -0.14128062,
       -0.21546138, -0.27952549, -0.33369477, -0.37953069, -0.41504219,
       -0.44184856, -0.45845478, -0.46961768, -0.47292798, -0.46860857,
       -0.45725494, -0.43685247, -0.41048138, -0.37658422, -0.33641226,
       -0.28866215, -0.23733111, -0.18297252, -0.13063434, -0.07870477,
       -0.0274309 ,  0.02083362,  0.06302266,  0.09993237,  0.13287632,
        0.15772804,  0.17425013,  0.18133418,  0.18059297,  0.17196931,
        0.1581168 ,  0.13808712,  0.11191173,  0.07940318,  0.0433899 ,
        0.00400336, -0.03650098, -0.07566137, -0.11312533, -0.14632779,
       -0.17528244, -0.19724703, -0.21083391, -0.2164104 , -0.21592552,
       -0.20930215, -0.19630738, -0.18087066, -0.16438383, -0.14634539,
       -0.1264883 , -0.10868407, -0.09288432, -0.08075829, -0.07151349,
       -0.06459588, -0.05844607, -0.05018603, -0.03947942, -0.02522884,
       -0.0095355 ,  0.00993595,  0.03101779,  0.05548804,  0.0828801 ,
        0.11443472,  0.14716572,  0.18007979,  0.21018245,  0.23591617,
        0.25543591,  0.2674044 ,  0.27338334,  0.27107111,  0.26244983,
        0.24601461,  0.22348224,  0.1943751 ,  0.16096062,  0.12286813,
        0.08371778,  0.043463  ,  0.00493991, -0.03065529, -0.06211509,
       -0.08876387, -0.11049864, -0.1255562 , -0.13669047, -0.14150916,
       -0.14212   , -0.13699924, -0.12776282, -0.11330214, -0.09690583,
       -0.07805755, -0.05775459, -0.03719798, -0.01433663,  0.00677601,
        0.02813215,  0.04596795,  0.06022372,  0.07039848,  0.07723106,
        0.07975642,  0.07939018,  0.07633156,  0.06960149,  0.06211464,
        0.05223044,  0.04450688,  0.03889516,  0.03708535,  0.04163807,
        0.0520377 ,  0.06809647,  0.08933312,  0.11383632,  0.14037008,
        0.16877756,  0.19616295,  0.21993089,  0.23875598,  0.25045593,
        0.25389471,  0.24755581,  0.23005172,  0.20319728,  0.16708424,
        0.12358969,  0.0760876 ,  0.02659309, -0.0220635 , -0.06921391,
       -0.11311758, -0.15213405, -0.18513066, -0.21196212, -0.23057988,
       -0.24253508, -0.24781846, -0.24510179, -0.23772604, -0.22589292,
       -0.21122939, -0.19315111, -0.17466834, -0.15483879, -0.13411649,
       -0.11324806, -0.09232855, -0.07249307, -0.05319551, -0.0361414 ,
       -0.02081652, -0.00750727,  0.00305534,  0.0116624 ,  0.02018347,
        0.02785343,  0.03527622,  0.04076194,  0.04377455,  0.04618433,
        0.04801486,  0.04860101,  0.04904793,  0.04901864,  0.04765385,
        0.04595232,  0.04379559,  0.04032445,  0.03444375,  0.02930478,
        0.0250354 ,  0.02099433,  0.01631142,  0.01201455,  0.00775119,
        0.00332114,  0.00154131])
```

Checking the relationship between the actual values

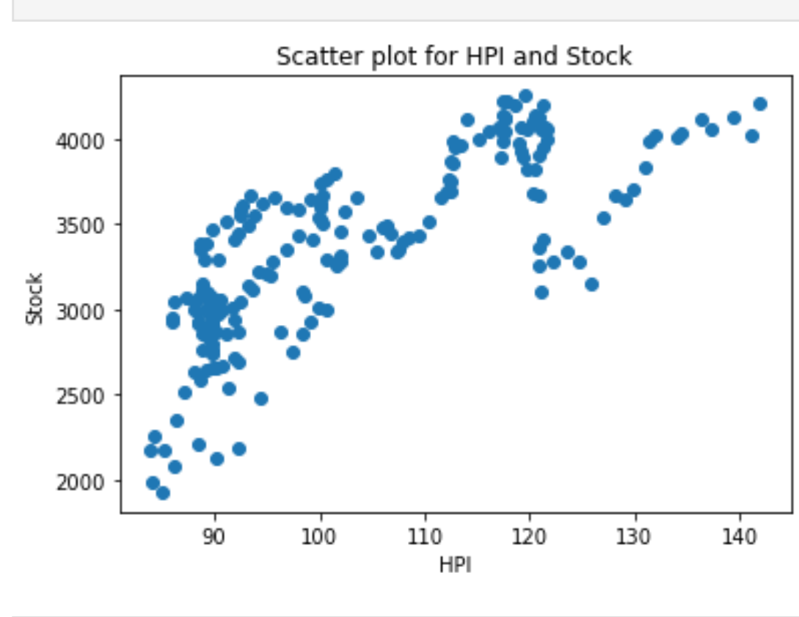
In [24]:
```
#plotting the actual values of the series
df_grouped[['Index','stock_price']].plot(title='Plotting HPI and FTSE Stock')
```
Out[24]:
```
<AxesSubplot:title={'center':'Plotting HPI and FTSE Stock'}, xlabel='Date'>
```



In [25]:
```
#checking the correlation between HPI and stock
df_grouped[['Index','stock_price']].corr()
```
Out[25]:

	Index	stock_price
Index	1.000000	0.770861
stock_price	0.770861	1.000000

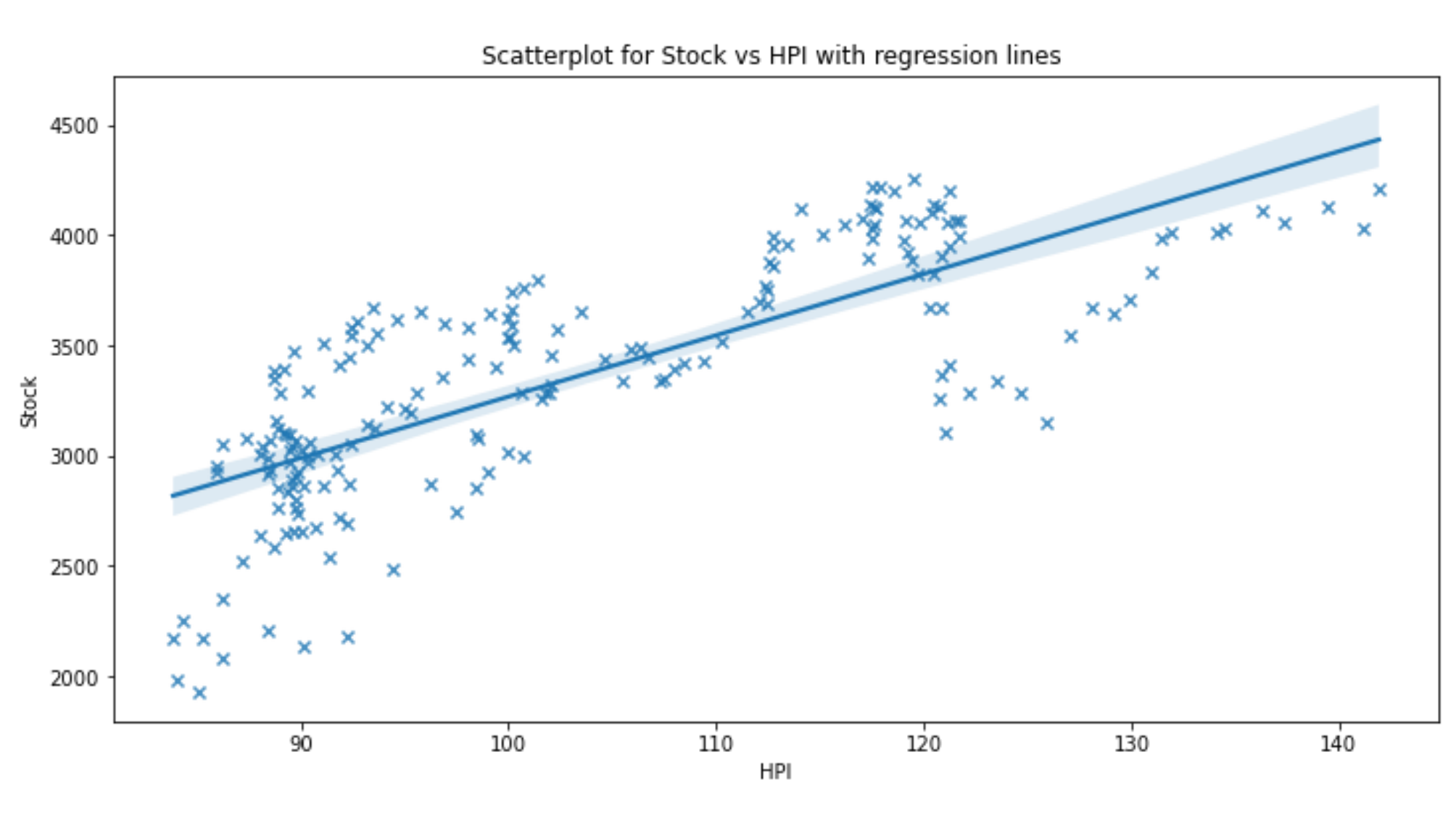
In [26]:
```
#plotting the scatter plot chart
plt.scatter(df_grouped['Index'],df_grouped['stock_price'])
plt.xlabel("HPI")
plt.ylabel('Stock')
plt.title('Scatter plot for HPI and Stock')
plt.show()
```



In [29]:
```
#calculate cross correlation
sm.tsa.stattools.ccf(df_grouped['stock_price'],df_grouped['Index'], adjusted=False)
```
Out[29]:
```
array([ 0.77086053,  0.74730546,  0.7231462 ,  0.69929122,  0.67539481,
        0.64900091,  0.62334213,  0.59780914,  0.5732612 ,  0.5490662 ,
        0.52595665,  0.50316913,  0.48240387,  0.46261196,  0.4443779 ,
        0.4287249 ,  0.41537202,  0.4041194 ,  0.3939091 ,  0.38390703,
        0.37447349,  0.36676098,  0.36134047,  0.35445331,  0.34839327,
        0.33991299,  0.33267102,  0.32516662,  0.31984141,  0.31440431,
        0.30666947,  0.29792074,  0.29014673,  0.27702898,  0.26028305,
        0.24181335,  0.22344699,  0.20267502,  0.1798304 ,  0.15687073,
        0.13542196,  0.11410628,  0.09103414,  0.06968551,  0.04989547,
        0.03094089,  0.01164034, -0.007102  , -0.02426888, -0.04200334,
       -0.05769773, -0.07052029, -0.08333756, -0.09770696, -0.11309076,
       -0.12561616, -0.13750672, -0.14707508, -0.15536931, -0.16440987,
       -0.17068028, -0.17686356, -0.18184582, -0.18719967, -0.19159157,
       -0.19622878, -0.201211  , -0.20654732, -0.21419238, -0.22334641,
       -0.2298667 , -0.23645294, -0.24280488, -0.24805882, -0.25151583,
       -0.2549746 , -0.2587333 , -0.26480115, -0.26952753, -0.27380786,
       -0.27797707, -0.28274655, -0.28787612, -0.29388942, -0.30043562,
       -0.30585799, -0.3111553 , -0.3165963 , -0.32235292, -0.32719144,
       -0.33418738, -0.33805845, -0.34274309, -0.34641996, -0.34762067,
       -0.34927599, -0.34956534, -0.35085413, -0.34944593, -0.34834062,
       -0.34592477, -0.34211673, -0.33794448, -0.33291526, -0.32682154,
       -0.32133987, -0.31592794, -0.30952316, -0.30373185, -0.29744705,
       -0.28957067, -0.28281519, -0.27517797, -0.26719476, -0.26178423,
       -0.25537082, -0.25162811, -0.2489996 , -0.24482159, -0.24109998,
       -0.2390489 , -0.23918845, -0.23977633, -0.23998582, -0.23968081,
       -0.2389032 , -0.23644471, -0.23188337, -0.22651776, -0.22030208,
       -0.21383869, -0.20891338, -0.20274327, -0.19811502, -0.19348302,
       -0.18960413, -0.18717459, -0.18412319, -0.18334737, -0.18295548,
       -0.18267854, -0.18275897, -0.18186818, -0.18191478, -0.17991624,
       -0.17926194, -0.17969088, -0.18026281, -0.17680025, -0.17063942,
       -0.16347221, -0.154416  , -0.14539682, -0.13487876, -0.12612387,
       -0.11796687, -0.11160284, -0.1043963 , -0.09659194, -0.08827711,
       -0.07971268, -0.07364489, -0.06659153, -0.05899934, -0.05363976,
       -0.04668378, -0.04042939, -0.03378651, -0.02609126, -0.0209112 ,
       -0.02067437, -0.02739817, -0.03261333, -0.03693345, -0.04063881,
       -0.04572168, -0.05005037, -0.05495666, -0.06099848, -0.06266881,
       -0.06299781, -0.06348074, -0.06304539, -0.0610588 , -0.05694769,
       -0.05214021, -0.04698249, -0.04101479, -0.03357824, -0.02626322,
       -0.01749361, -0.0098156 ])
```

Notice that the correlation between the two time series becomes less and less positive as the number of lags increases. This tells us that Stock Market price during a given month is quite predictive of house price index

In [38]:
```
#using regression plot

plt.figure(figsize=(12,6))

# Scatterplot with regression lines
sns.regplot(x='Index', y='stock_price', data=df_grouped,
           marker='x')

# Aesthetics
plt.title('Scatterplot for Stock vs HPI with regression lines ')
plt.xlabel('HPI')
plt.ylabel('Stock')
```
Out[38]:
```
Text(0, 0.5, 'Stock')
```



In [40]:
```
#Slope of regression line

slope, intercept, r_value, p_value, std_err = stats.linregress(df_grouped['Index'],df_grouped['stock_price'])

print('The slope of the regression line is: ','{:.2f}'.format(slope))
```
```
The slope of the regression line is:  27.80
```

We can see that Stock market has a positive impact on the House Price Index

In [ ]:

In [ ]:

In [ ]:

EDA: 1

EDA on Data from Land Registry

In [115…

```
#importing various libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from geopy.geocoders import Nominatim
import geocoder
import folium
```

In [10]:

```
#importing the dataset
df = pd.read_csv(r'indices_dataset/UK-HPI-full-file-2022-05_land_registry.csv')
df.head()
```

Out[10]:

	Date	RegionName	AreaCode	AveragePrice	Index	IndexSA	1m%Change	12m%Change	AveragePriceSA	SalesVolume	...	Nev
0	1/1/2004	Aberdeenshire	S12000034	81693.66964	40.864214	NaN	NaN	NaN	NaN	388.0	...	88436.
1	1/2/2004	Aberdeenshire	S12000034	81678.76231	40.856757	NaN	-0.018248	NaN	NaN	326.0	...	88606.
2	1/3/2004	Aberdeenshire	S12000034	83525.09702	41.780317	NaN	2.260483	NaN	NaN	453.0	...	90296.
3	1/4/2004	Aberdeenshire	S12000034	84333.67900	42.184780	NaN	0.968071	NaN	NaN	571.0	...	90319.
4	1/5/2004	Aberdeenshire	S12000034	86379.95396	43.208353	NaN	2.426403	NaN	NaN	502.0	...	91989.

5 rows × 54 columns

In [11]:

```
#describing the dataset ( checking statiscs)
df.describe()
```

Out[11]:

	AveragePrice	Index	IndexSA	1m%Change	12m%Change	AveragePriceSA	SalesVolume	DetachedPrice	DetachedIndex
count	1.369340e+05	136934.000000	4599.000000	136491.000000	131894.000000	4599.000000	132435.000000	1.306370e+05	130637.000000
mean	1.679390e+05	81.277471	83.839599	0.577060	6.619196	158460.884039	1216.563499	2.961552e+05	82.775005
std	1.132823e+05	35.083942	32.324541	1.993859	8.926884	82089.189229	7591.577874	2.574826e+05	33.360480
min	2.712016e+03	1.096815	18.312610	-30.297781	-35.786566	40405.313840	2.000000	4.064610e+04	12.494361
25%	9.426384e+04	46.953837	57.519430	-0.425305	1.511545	109592.343300	139.000000	1.640175e+05	52.143149
50%	1.472749e+05	89.003520	90.292480	0.533856	5.635033	148884.457100	214.000000	2.376951e+05	88.512634
75%	2.105152e+05	106.513165	106.852440	1.520537	10.844191	194692.982000	373.000000	3.437461e+05	105.838528
max	1.463378e+06	221.485238	158.076988	35.286646	98.437940	527792.225700	163312.666700	4.232089e+06	197.988967

8 rows × 51 columns

In [12]:

```
#checking the length of dataset
df.shape
```

Out[12]:

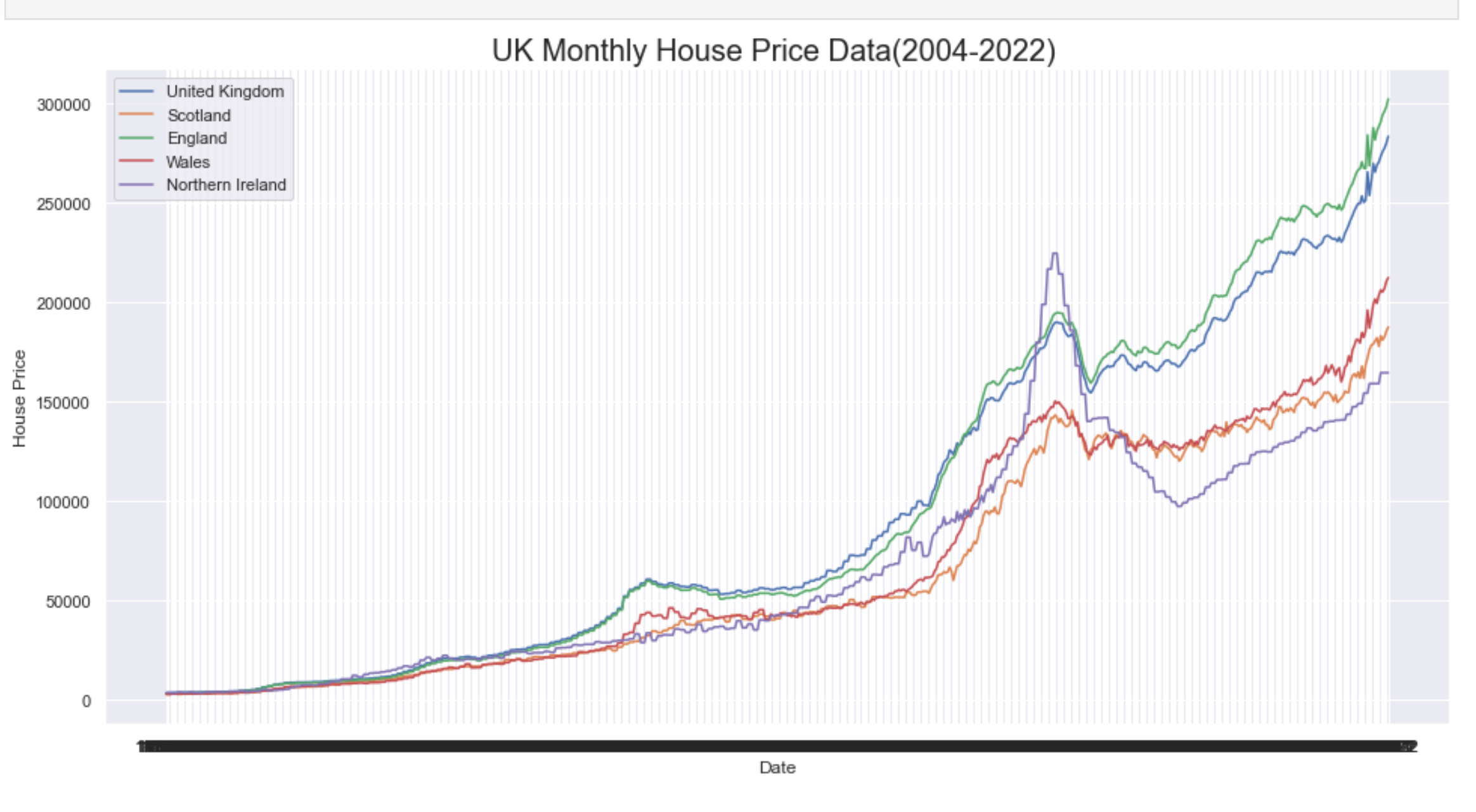
```
(136934, 54)
```

In [15]:

```
#creating a separate dataframe which contains the countries data
df_uk = df[df.RegionName=='United Kingdom']
df_scot = df[df.RegionName=='Scotland']
df_eng = df[df.RegionName=='England']
df_wales= df[df.RegionName=='Wales']
df_NI = df[df.RegionName=='Northern Ireland']
```

In [22]:

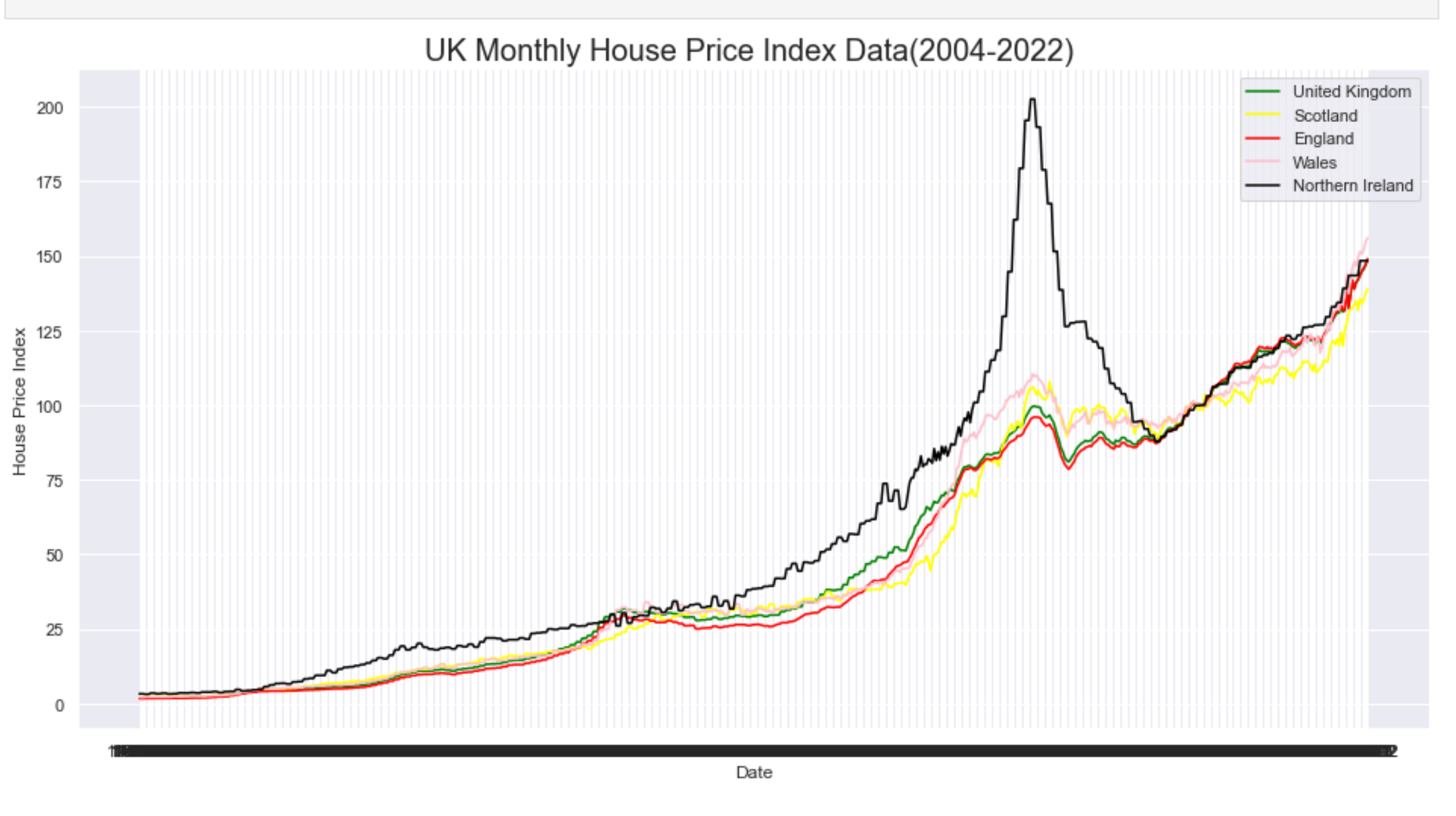
```
# plotting the average House Price of Different countries over years

sns.set_theme(style="darkgrid")
plt.figure(figsize=(16,8))
sns.lineplot(data = df_uk, x='Date',y='AveragePrice', label='United Kingdom')
sns.lineplot(data = df_scot, x='Date',y='AveragePrice', label='Scotland')
sns.lineplot(data = df_eng, x='Date',y='AveragePrice', label='England')
sns.lineplot(data = df_wales, x='Date',y='AveragePrice', label='Wales')
sns.lineplot(data = df_NI, x='Date',y='AveragePrice', label='Northern Ireland')
plt.title("UK Monthly House Price Data(2004-2022) ",fontdict={'fontsize':20})
plt.xlabel("Date")
plt.ylabel('House Price')
plt.show()
```



We can see from the above chart that ENGLAND has the highest average house prices , whereas Northern Ireland has the lowest.

In [26]:

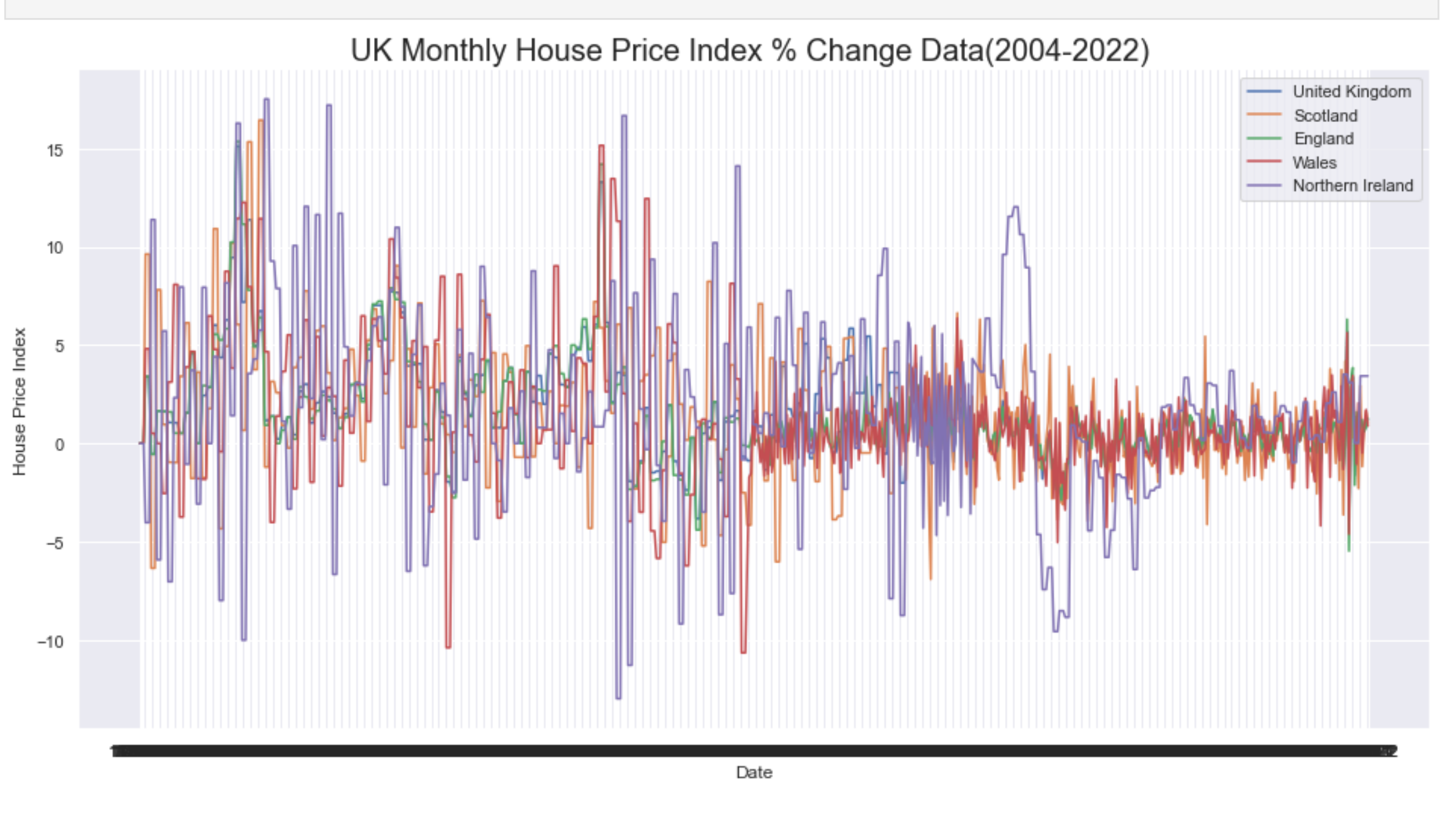
```
#plotting the House price indexes for all countries in UK.


plt.figure(figsize=(16,8))
sns.lineplot(data = df_uk, x='Date',y='Index', label='United Kingdom',color='green')
sns.lineplot(data = df_scot, x='Date',y='Index', label='Scotland',color='yellow')
sns.lineplot(data = df_eng, x='Date',y='Index', label='England',color='red')
sns.lineplot(data = df_wales, x='Date',y='Index', label='Wales',color='pink')
sns.lineplot(data = df_NI, x='Date',y='Index', label='Northern Ireland',color='black')
plt.title("UK Monthly House Price Index Data(2004-2022) ",fontdict={'fontsize':20})
plt.xlabel("Date")
plt.ylabel('House Price Index')
plt.show()
```



We can see from the above chart , that Northern Ireland has the highest House Price Index.

In [28]:

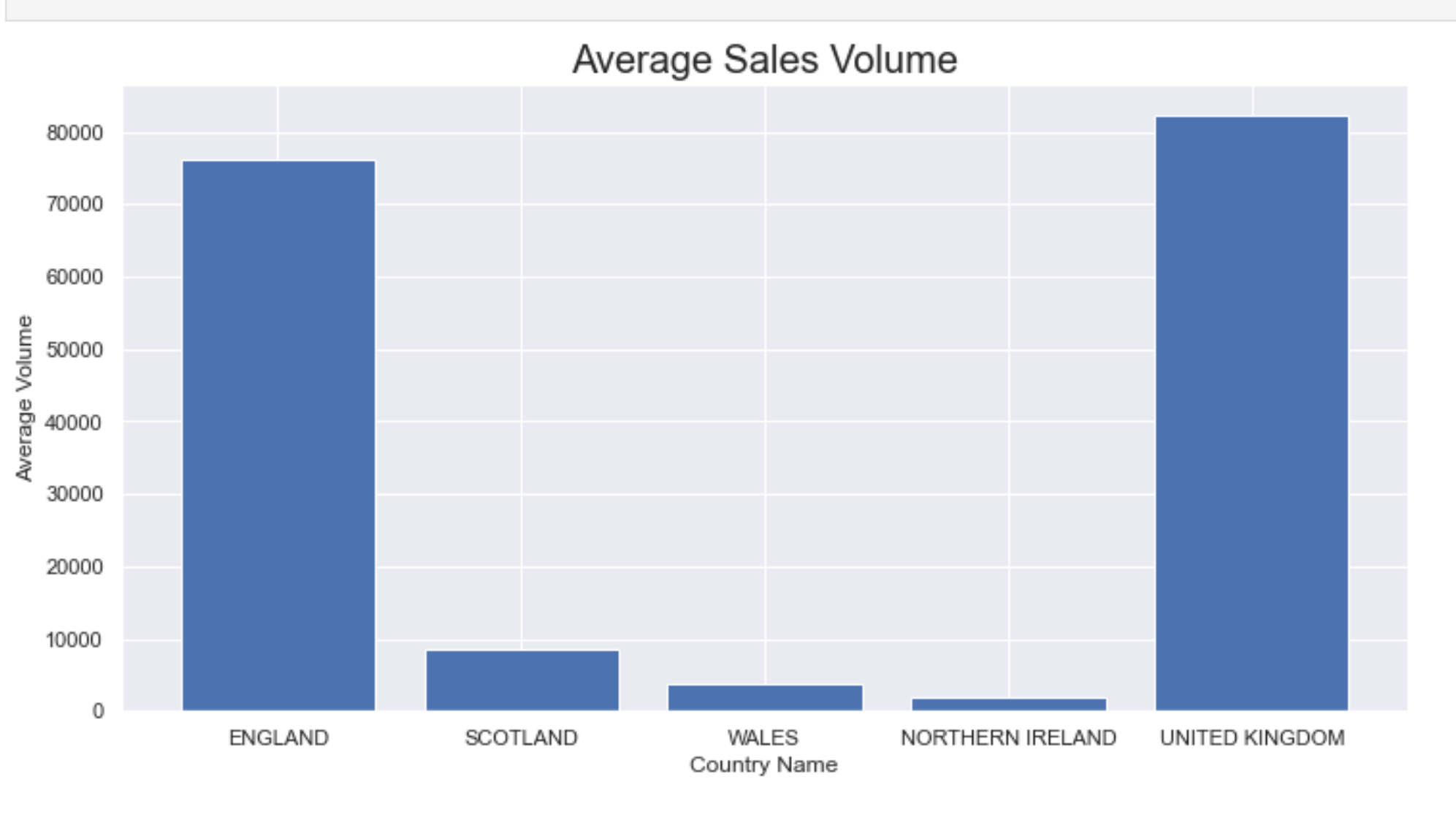
```
#plotting the House price indexes percentage change for all countries in UK.


plt.figure(figsize=(16,8))
sns.lineplot(data = df_uk, x='Date',y='1m%Change', label='United Kingdom')
sns.lineplot(data = df_scot, x='Date',y='1m%Change', label='Scotland')
sns.lineplot(data = df_eng, x='Date',y='1m%Change', label='England')
sns.lineplot(data = df_wales, x='Date',y='1m%Change', label='Wales')
sns.lineplot(data = df_NI, x='Date',y='1m%Change', label='Northern Ireland')
plt.title("UK Monthly House Price Index % Change Data(2004-2022) ",fontdict={'fontsize':20})
plt.xlabel("Date")
plt.ylabel('House Price Index')
plt.show()
```



We can see from the above chart the maximum fluctuations are observed in Northern Ireland.

In [37]:

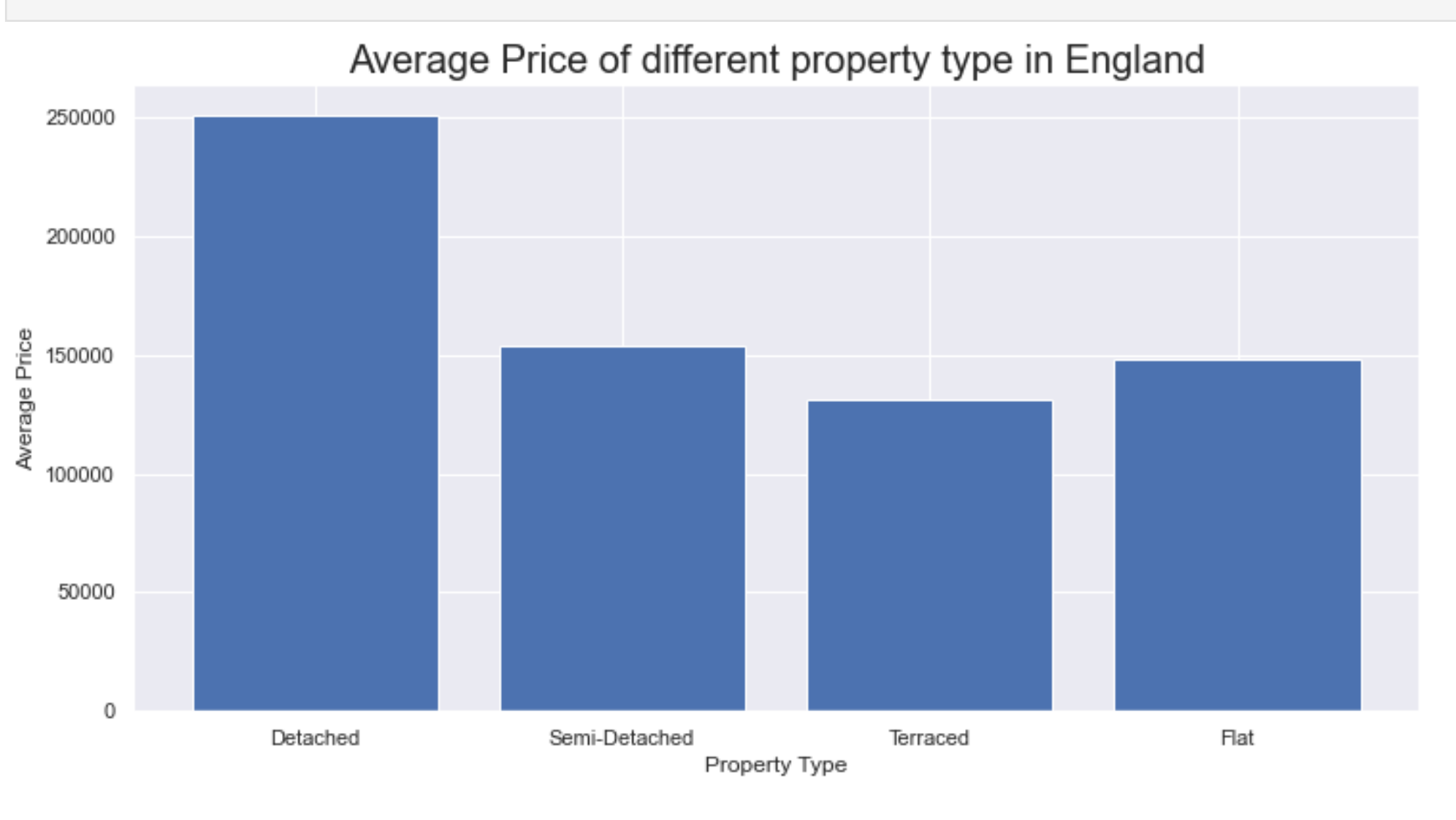
```
#plotting the average sales volume
country_name = ['ENGLAND', 'SCOTLAND','WALES','NORTHERN IRELAND','UNITED KINGDOM']
sales_lst = [df_eng['SalesVolume'].mean(),df_scot['SalesVolume'].mean(),df_wales['SalesVolume'].mean(),
             df_NI['SalesVolume'].mean(),df_uk['SalesVolume'].mean()]
plt.figure(figsize=(12,6))
plt.bar(country_name,sales_lst)
plt.title('Average Sales Volume',fontdict={'fontsize':20})
plt.xlabel("Country Name")
plt.ylabel('Average Volume')
plt.show()
```



We can see that England has the highest sales while Northern Ireland has the lowest.

In [38]:

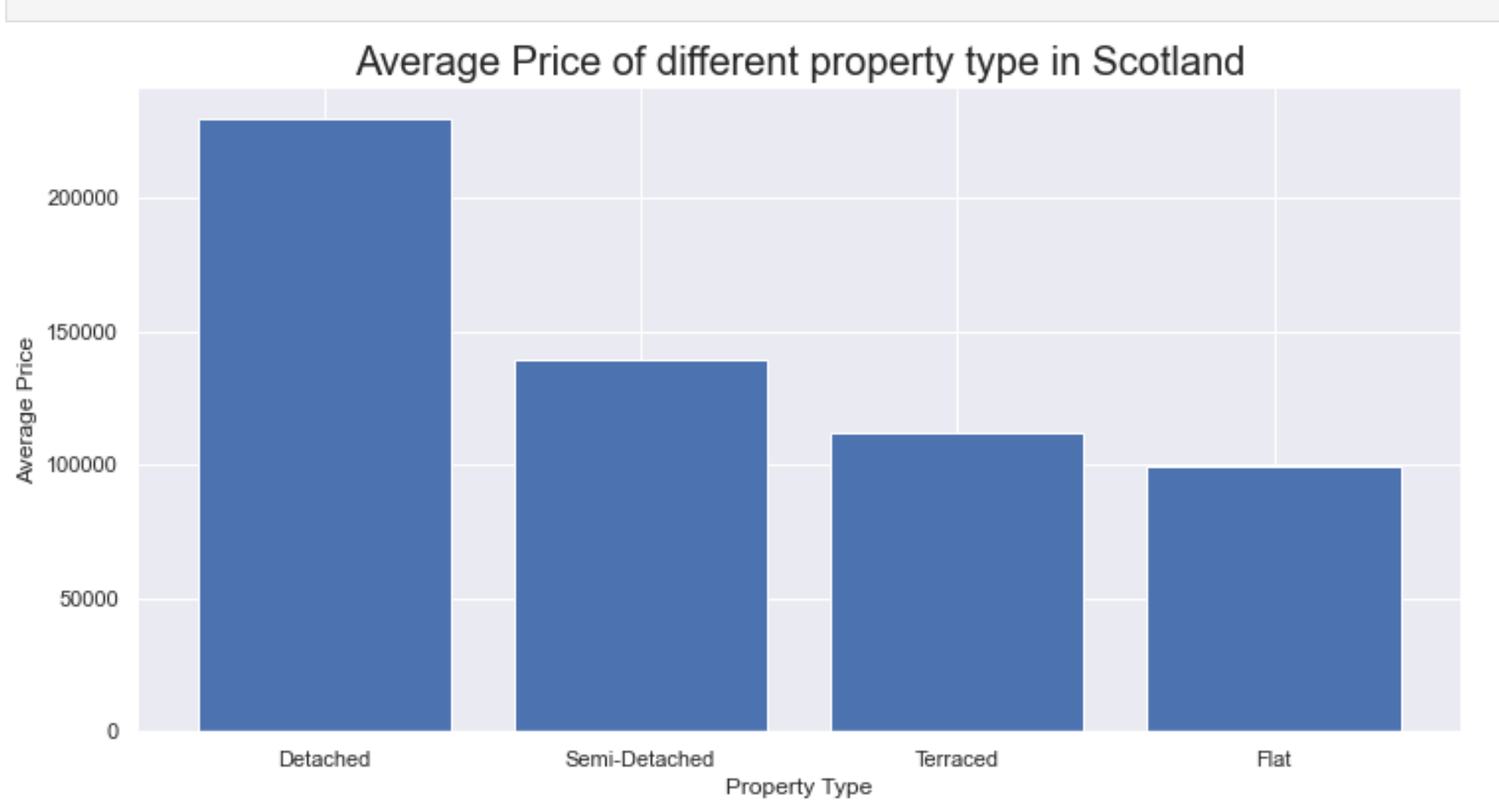
```
# plotting the average price of types of properties in United Kingdom
property_type =['Detached', 'Semi-Detached', 'Terraced','Flat']
eng_property = [df_eng['DetachedPrice'].mean(),df_eng['SemiDetachedPrice'].mean(),
               df_eng['TerracedPrice'].mean(),df_eng['FlatPrice'].mean()]
plt.figure(figsize=(12,6))
plt.bar(property_type,eng_property)
plt.title('Average Price of different property type in England',fontdict={'fontsize':20})
plt.xlabel("Property Type")
plt.ylabel('Average Price')
plt.show()
```



We can see from the above chart that the detached property has the highest price whereas terraced property is the lowest in England.

In [39]:

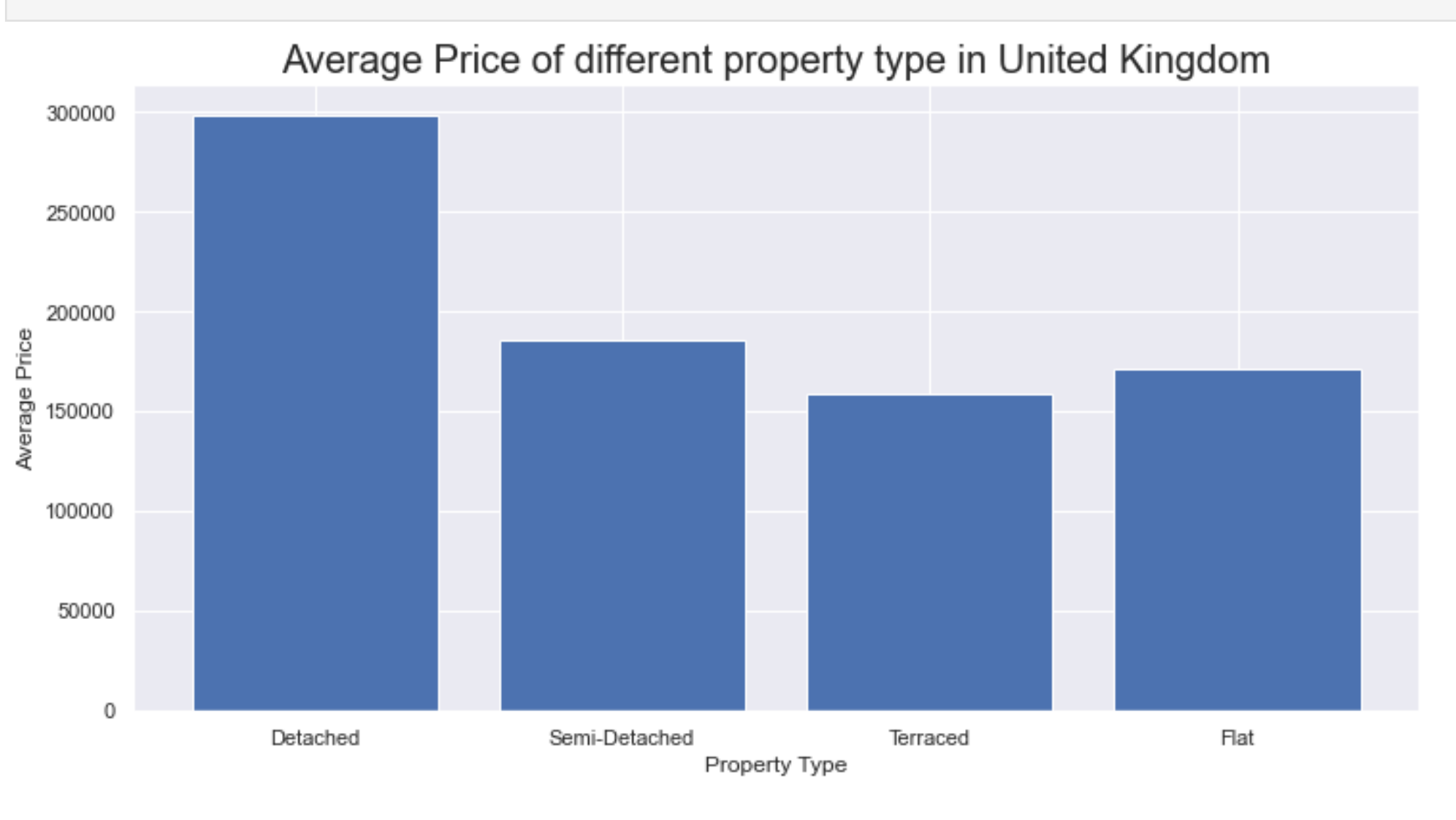
```
# plotting the average price of types of properties in Scotland

scot_property = [df_scot['DetachedPrice'].mean(),df_scot['SemiDetachedPrice'].mean(),
               df_scot['TerracedPrice'].mean(),df_scot['FlatPrice'].mean()]
plt.figure(figsize=(12,6))
plt.bar(property_type,scot_property)
plt.title('Average Price of different property type in Scotland',fontdict={'fontsize':20})
plt.xlabel("Property Type")
plt.ylabel('Average Price')
plt.show()
```



We can see from the above chart that the detached property has the highest price whereas Flat property is the lowest in Scotland.

In [40]:

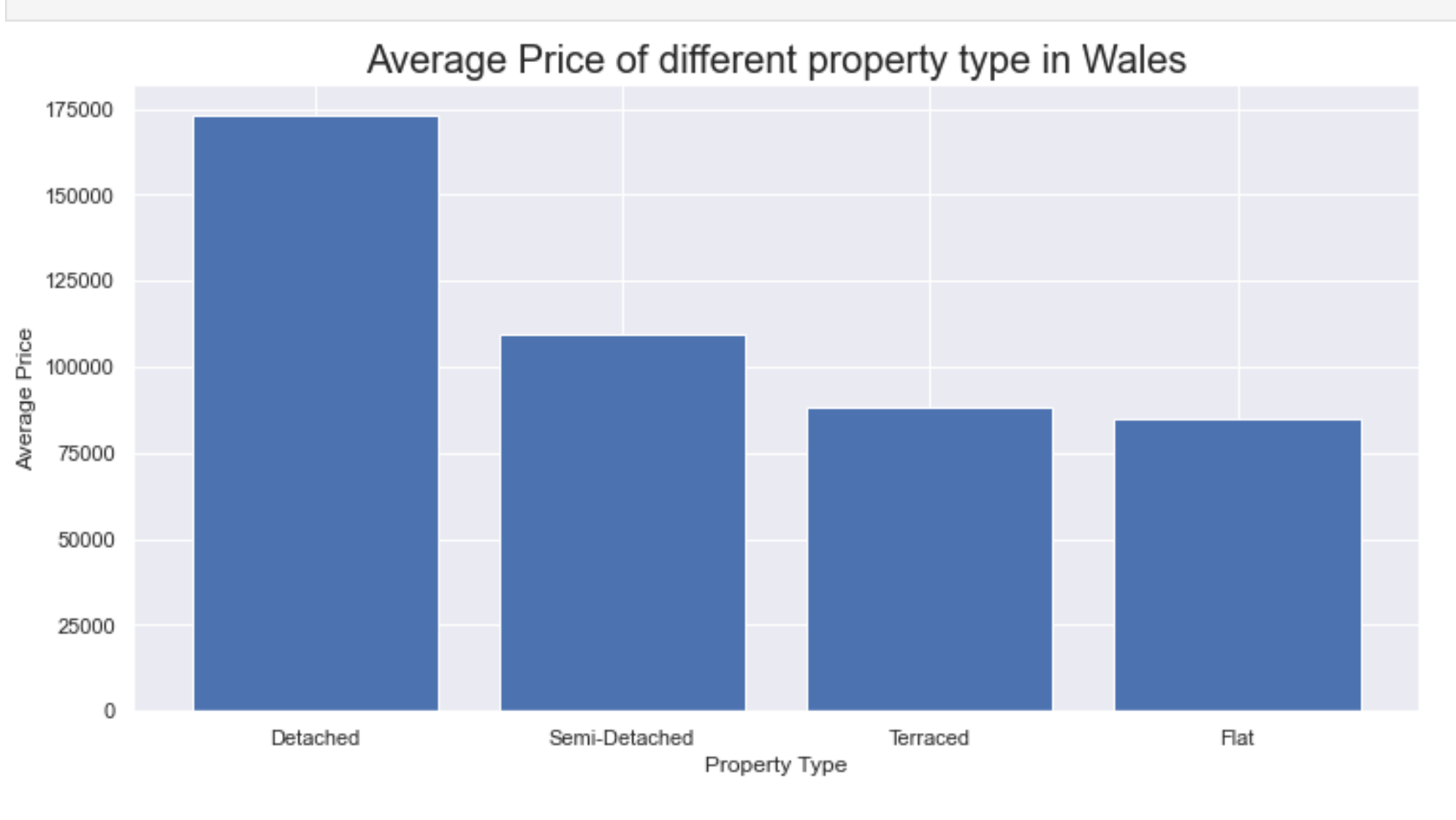
```
# plotting the average price of types of properties in United Kingdom

uk_property = [df_uk['DetachedPrice'].mean(),df_uk['SemiDetachedPrice'].mean(),
               df_uk['TerracedPrice'].mean(),df_uk['FlatPrice'].mean()]
plt.figure(figsize=(12,6))
plt.bar(property_type,uk_property)
plt.title('Average Price of different property type in United Kingdom',fontdict={'fontsize':20})
plt.xlabel("Property Type")
plt.ylabel('Average Price')
plt.show()
```



We can see from the above chart that the detached property has the highest price whereas terraced property is the lowest in United Kingdom.

In [41]:

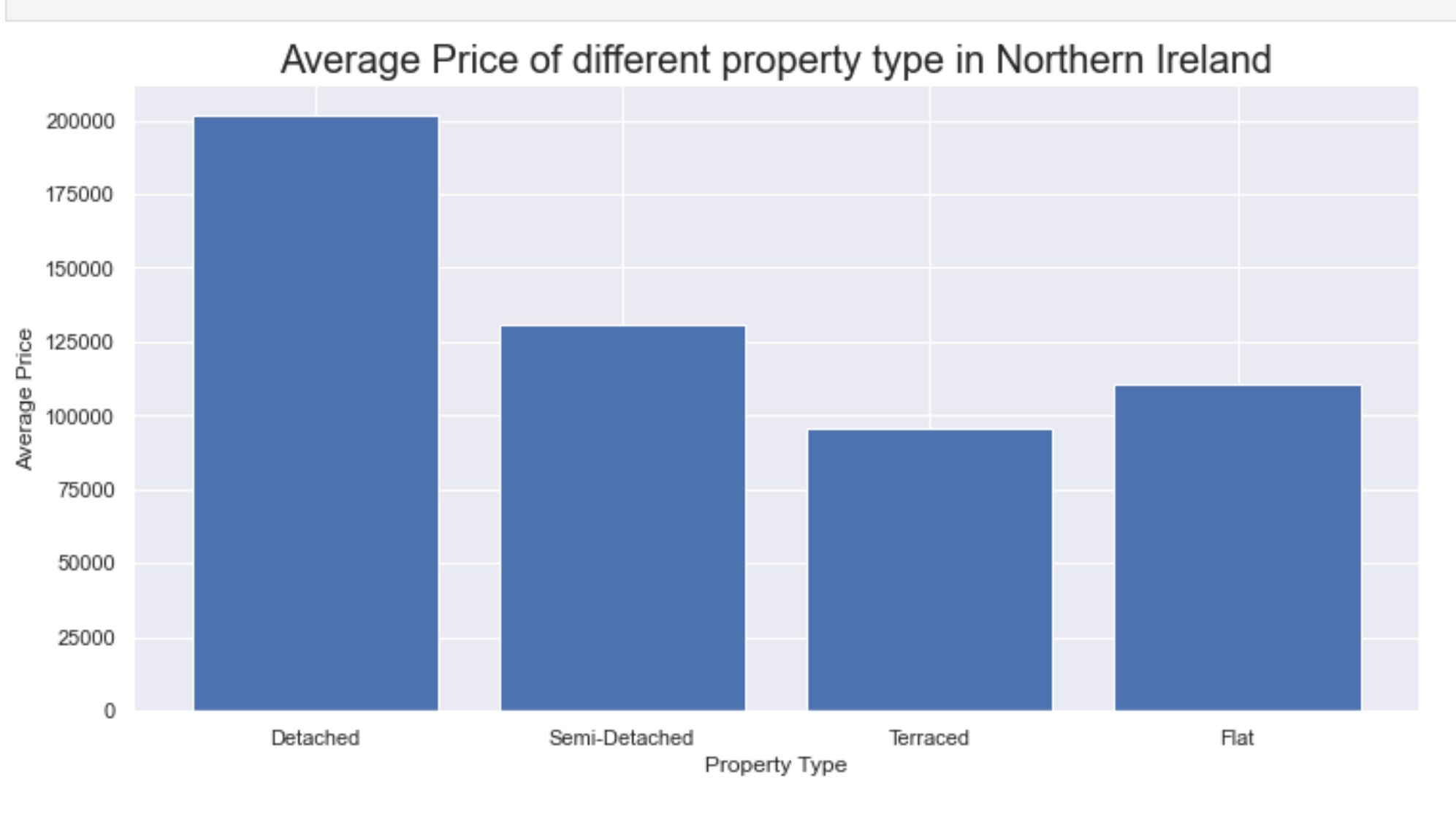
```
# plotting the average price of types of properties in Wales

wales_property = [df_wales['DetachedPrice'].mean(),df_wales['SemiDetachedPrice'].mean(),
               df_wales['TerracedPrice'].mean(),df_wales['FlatPrice'].mean()]
plt.figure(figsize=(12,6))
plt.bar(property_type,wales_property)
plt.title('Average Price of different property type in Wales',fontdict={'fontsize':20})
plt.xlabel("Property Type")
plt.ylabel('Average Price')
plt.show()
```



We can see from the above chart that the detached property has the highest price whereas Flat property is the lowest in Wales.

In [42]:

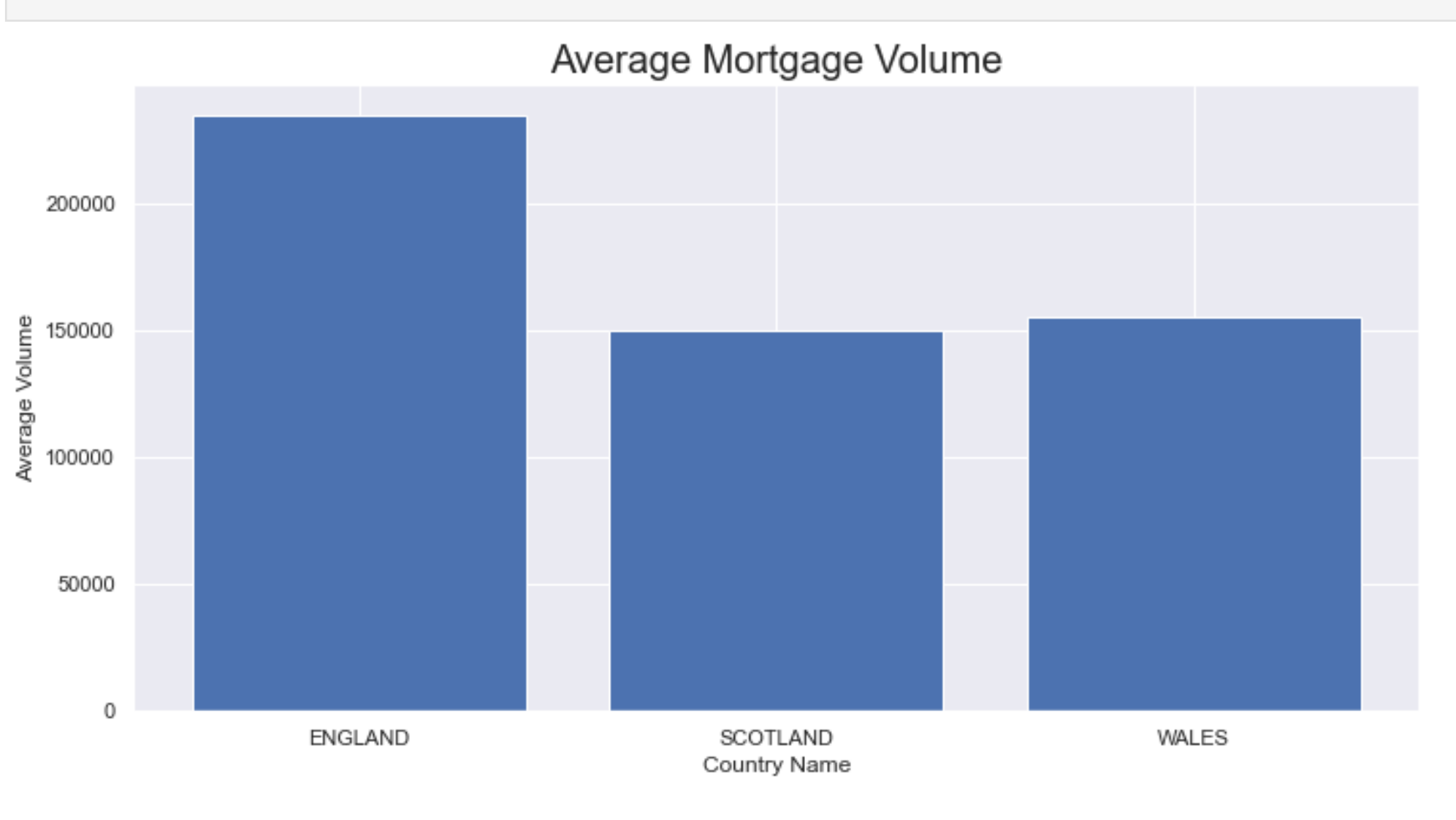
```
# plotting the average price of types of properties in Northern Ireland

ni_property = [df_NI['DetachedPrice'].mean(),df_NI['SemiDetachedPrice'].mean(),
               df_NI['TerracedPrice'].mean(),df_NI['FlatPrice'].mean()]
plt.figure(figsize=(12,6))
plt.bar(property_type,ni_property)
plt.title('Average Price of different property type in Northern Ireland',fontdict={'fontsize':20})
plt.xlabel("Property Type")
plt.ylabel('Average Price')
plt.show()
```



We can see from the above chart that the detached property has the highest price whereas terraced property is the lowest in Northern Ireland.

In [43]:

```
#plotting mortgage prices among the countries

country_name = ['ENGLAND', 'SCOTLAND','WALES','NORTHERN IRELAND','UNITED KINGDOM']
sales_lst = [df_eng['MortgagePrice'].mean(),df_scot['MortgagePrice'].mean(),df_wales['MortgagePrice'].mean(),
             df_NI['MortgagePrice'].mean(),df_uk['MortgagePrice'].mean()]
plt.figure(figsize=(12,6))
plt.bar(country_name,sales_lst)
plt.title('Average Mortgage Volume',fontdict={'fontsize':20})
plt.xlabel("Country Name")
plt.ylabel('Average Volume')
plt.show()
```



We can see that England has the highest mortgage prices while scotland has the lowest.

Plotting the prices over the geographical map

In [127…

```
#importing the dataset
df_all = pd.read_excel(r'UK-HPI-full-file-2022-05_land_registry.xlsx',sheet_name='Sheet1')
df_all.head()
```

Out[127…

	Date	AreaCode	RegionName	Country	AveragePrice	Index	DetachedPrice	DetachedIndex	SemiDetachedPrice	SemiDetachedIndex
0	2022-01-05	S12000034	Aberdeenshire	Scotland	202557.5499	101.321865	291818.5518	103.898171	176281.0867	101.981358
1	2022-01-05	E07000223	Adur	England	374366.8221	149.953080	589634.8687	156.572932	427194.4734	154.299567
2	2022-01-05	E07000026	Allerdale	England	168799.3463	124.578590	255977.2284	129.374073	165069.7650	125.412416
3	2022-01-05	E07000032	Amber Valley	England	212442.2828	147.562038	309674.3669	150.595518	191401.0786	148.138008
4	2022-01-05	S12000041	Angus	Scotland	171135.5138	123.672084	288151.8865	129.609512	181340.2010	126.305327

In [128…

```
#separting the cities as per countries and making different dataframes
df_e = df_all[df_all.Country=='England']
df_w = df_all[df_all.Country=='Wales']
df_s = df_all[df_all.Country=='Scotland']
df_n = df_all[df_all.Country=='Northern Ireland']
```

In [129…

```
#calling nominatim function
geolocator = Nominatim(user_agent="UK")
```

In [134…

```
#getting the co-ordinates of the cities of england
lat=[]
long=[]
for name in df_e['RegionName']:
    address = str(name+", England")
    location = geocoder.arcgis(address)
    lat.append(location.latlng[0])
    long.append(location.latlng[1])
df_e['lat']=lat
df_e['long']=long
```

```
C:\Users\hp\AppData\Local\Temp/ipykernel_17256/1934058786.py:9: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy

C:\Users\hp\AppData\Local\Temp/ipykernel_17256/1934058786.py:10: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
```

In [135…

```
#getting the co-ordinates of the cities of wales
lat=[]
long=[]
for name in df_w['RegionName']:
    address = str(name+", Wales")
    location = geocoder.arcgis(address)
    lat.append(location.latlng[0])
    long.append(location.latlng[1])
df_w['lat']=lat
df_w['long']=long
```

```
C:\Users\hp\AppData\Local\Temp/ipykernel_17256/3199315468.py:9: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy

C:\Users\hp\AppData\Local\Temp/ipykernel_17256/3199315468.py:10: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
```

In [136…

```
#getting the co-ordinates of the cities of scotland
lat=[]
long=[]
for name in df_s['RegionName']:
    address = str(name+", Scotland")
    location = geocoder.arcgis(address)
    lat.append(location.latlng[0])
    long.append(location.latlng[1])
df_s['lat']=lat
df_s['long']=long
```

```
C:\Users\hp\AppData\Local\Temp/ipykernel_17256/965660312.py:9: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy

C:\Users\hp\AppData\Local\Temp/ipykernel_17256/965660312.py:10: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
```

In [137…

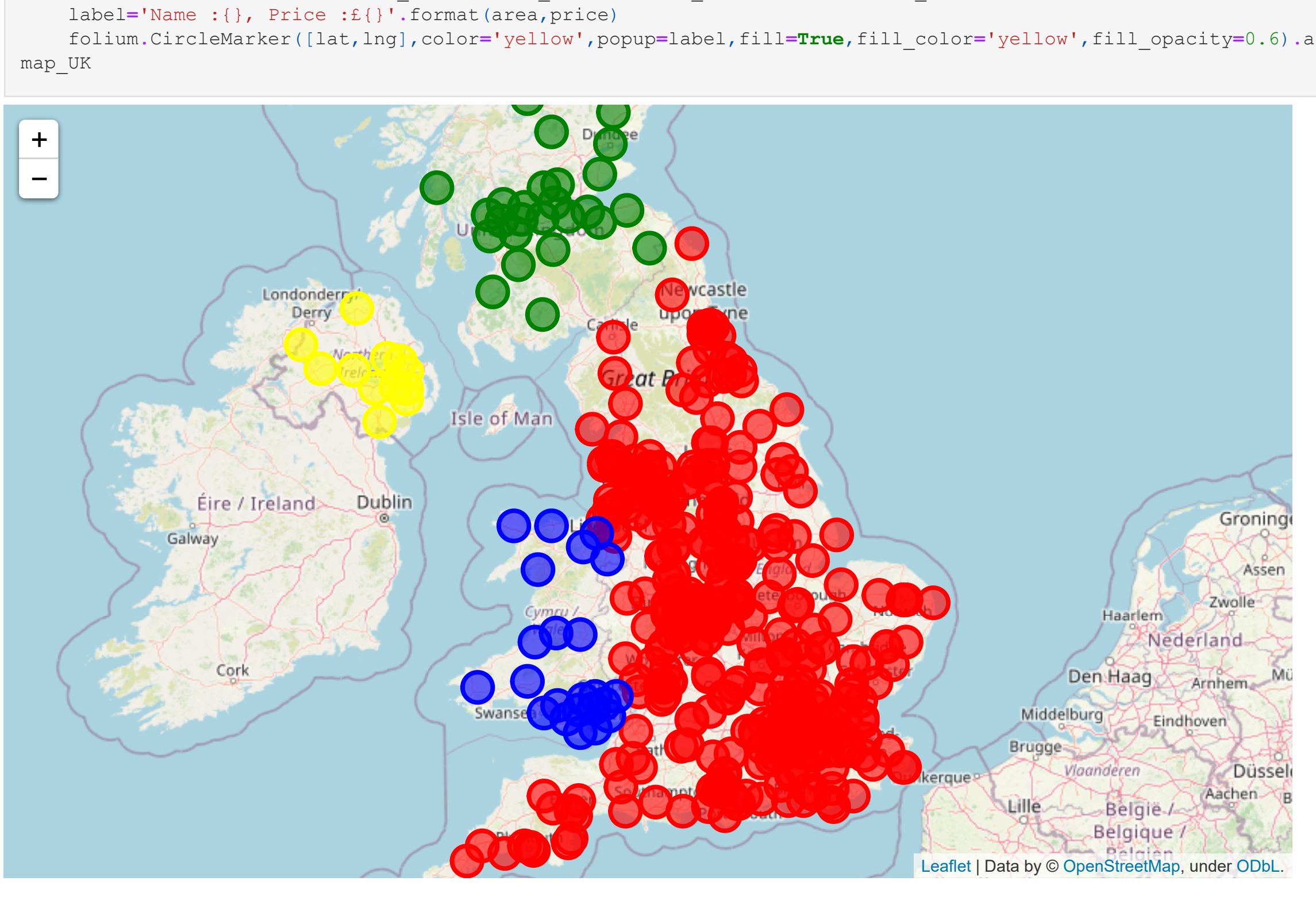
```
#getting the co-ordinates of the cities of northern ireland
lat=[]
long=[]
for name in df_n['RegionName']:
    address = str(name+", Northern Ireland")
    location = geocoder.arcgis(address)
    lat.append(location.latlng[0])
    long.append(location.latlng[1])
df_n['lat']=lat
df_n['long']=long
```

```
C:\Users\hp\AppData\Local\Temp/ipykernel_17256/3120937460.py:9: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy

C:\Users\hp\AppData\Local\Temp/ipykernel_17256/3120937460.py:10: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
```

In [141…

```
#plotting a map
map_UK=folium.Map(location=[53.584797 , -2.438721],zoom_start=6)
for lat,lng,area,price in zip(df_e['lat'],df_e['long'],df_e['RegionName'],df_e['AveragePrice']):
    label='Name :{}, Price :£{}'.format(area,price)
    folium.CircleMarker([lat,lng],color='red',popup=label,fill=True,fill_color='red',fill_opacity=0.6).add_to(m
for lat,lng,area,price in zip(df_w['lat'],df_w['long'],df_w['RegionName'],df_w['AveragePrice']):
    label='Name :{}, Price :£{}'.format(area,price)
    folium.CircleMarker([lat,lng],color='Blue',popup=label,fill=True,fill_color='blue',fill_opacity=0.6).add_tc
for lat,lng,area,price in zip(df_s['lat'],df_s['long'],df_s['RegionName'],df_s['AveragePrice']):
    label='Name :{}, Price :£{}'.format(area,price)
    folium.CircleMarker([lat,lng],color='green',popup=label,fill=True,fill_color='green',fill_opacity=0.6).add_
for lat,lng,area,price in zip(df_n['lat'],df_n['long'],df_n['RegionName'],df_n['AveragePrice']):
    label='Name :{}, Price :£{}'.format(area,price)
    folium.CircleMarker([lat,lng],color='yellow',popup=label,fill=True,fill_color='yellow',fill_opacity=0.6).ac
map_UK
```

Out[141…



EDA : 2

Exploratory Data Analysis on API data

In [76]:
```
#importing various libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import json
import requests
import time
from geopy.geocoders import Nominatim
import geocoder
import folium
```

In [7]:
```
# Connecting to the Property Data API
# Connect to API and dislay response
response = requests.get('https://api.propertydata.co.uk/prices?key=FKU1VNNTNJ&postcode=W149JH&bedrooms=2')
print(response)
```

```
<Response [200]>
```

Response is 200 , which tells us our request was successful.

In [6]:
```
# API Key for property data API
API_KEY ='FKU1VNNTNJ'
```

In [8]:
```
# printing the response of the prices with random postcode
print(response.json())
```

```
{'status': 'success', 'postcode': 'W14 9JH', 'postcode_type': 'full', 'url': 'https://propertydata.co.uk/draw?i
nput=W14+9JH', 'bedrooms': 2, 'data': {'points_analysed': 20, 'radius': '0.09', 'average': 657495, '70pc_rang
e': [575000, 725000], '80pc_range': [550000, 875000], '90pc_range': [550000, 925000], '100pc_range': [525000, 9
50000], 'raw_data': [{'price': 650000, 'lat': '51.48887000', 'lng': '-0.20776000', 'bedrooms': 2, 'type': 'fla
t', 'distance': '0.00'}, {'price': 575000, 'lat': '51.48884800', 'lng': '-0.20701200', 'bedrooms': 2, 'type':
'flat', 'distance': '0.03'}, {'price': 615000, 'lat': '51.48851000', 'lng': '-0.20742000', 'bedrooms': 2, 'typ
e': 'flat', 'distance': '0.03'}, {'price': 640000, 'lat': '51.48932000', 'lng': '-0.20804000', 'bedrooms': 2,
'type': 'flat', 'distance': '0.03'}, {'price': 725000, 'lat': '51.48843100', 'lng': '-0.20775400', 'bedrooms':
2, 'type': 'flat', 'distance': '0.03'}, {'price': 699950, 'lat': '51.48928000', 'lng': '-0.20793700', 'bedroom
s': 2, 'type': 'flat', 'distance': '0.03'}, {'price': 550000, 'lat': '51.48941000', 'lng': '-0.20832000', 'bedr
ooms': 2, 'type': 'flat', 'distance': '0.04'}, {'price': 600000, 'lat': '51.48813700', 'lng': '-0.20781600', 'b
edrooms': 2, 'type': 'flat', 'distance': '0.05'}, {'price': 925000, 'lat': '51.48948300', 'lng': '-0.20827400',
'bedrooms': 2, 'type': 'flat', 'distance': '0.05'}, {'price': 650000, 'lat': '51.48824900', 'lng': '-0.2066900
0', 'bedrooms': 2, 'type': 'flat', 'distance': '0.06'}, {'price': 700000, 'lat': '51.48941200', 'lng': '-0.2066
6900', 'bedrooms': 2, 'type': 'flat', 'distance': '0.06'}, {'price': 625000, 'lat': '51.48817000', 'lng': '-0.2
0689500', 'bedrooms': 2, 'type': 'flat', 'distance': '0.06'}, {'price': 645000, 'lat': '51.48822500', 'lng': '-
0.20872100', 'bedrooms': 2, 'type': 'flat', 'distance': '0.06'}, {'price': 675000, 'lat': '51.48877800', 'lng':
'-0.20643200', 'bedrooms': 2, 'type': 'flat', 'distance': '0.06'}, {'price': 525000, 'lat': '51.48978800', 'ln
g': '-0.20864000', 'bedrooms': 2, 'type': 'flat', 'distance': '0.07'}, {'price': 700000, 'lat': '51.48785100',
'lng': '-0.20757600', 'bedrooms': 2, 'type': 'flat', 'distance': '0.07'}, {'price': 675000, 'lat': '51.4881990
0', 'lng': '-0.20637900', 'bedrooms': 2, 'type': 'flat', 'distance': '0.08'}, {'price': 875000, 'lat': '51.4881
8700', 'lng': '-0.20931800', 'bedrooms': 2, 'type': 'flat', 'distance': '0.08'}, {'price': 550000, 'lat': '51.4
8891000', 'lng': '-0.20955000', 'bedrooms': 2, 'type': 'flat', 'distance': '0.08'}, {'price': 950000, 'lat': '5
1.48808300', 'lng': '-0.20628500', 'bedrooms': 2, 'type': 'flat', 'distance': '0.09'}]}, 'process_time': '2.1
0'}
```

In [122…
```
# making a list of region list
region_list =[ 'north_east','north_west','east_midlands','west_midlands','east_of_england','greater_london','sc
              'south_west','wales','scotland','northern_ireland']
reg_eng=['north_east','north_west','east_midlands','west_midlands','east_of_england','greater_london','south_ea
              'south_west']
```

In [ ]:
```
#creating a dataframe of the all regions
df_list = {name: pd.DataFrame() for name in region_list}
```

In [71]:
```
#downloading the data using API calls
for region_name in region_list:
    url ='https://api.propertydata.co.uk/postcode-key-stats?key={}&region={}'.format(API_KEY,region_name)
    response = requests.get(url)
    data = response.json()
    records =[]
    for row in data['data']:
        records.append(row)
    df_list[region_name] = pd.DataFrame.from_dict(records, orient='columns')
    #putting your program to sleep so as to give time for next api call
    time.sleep(5)
```

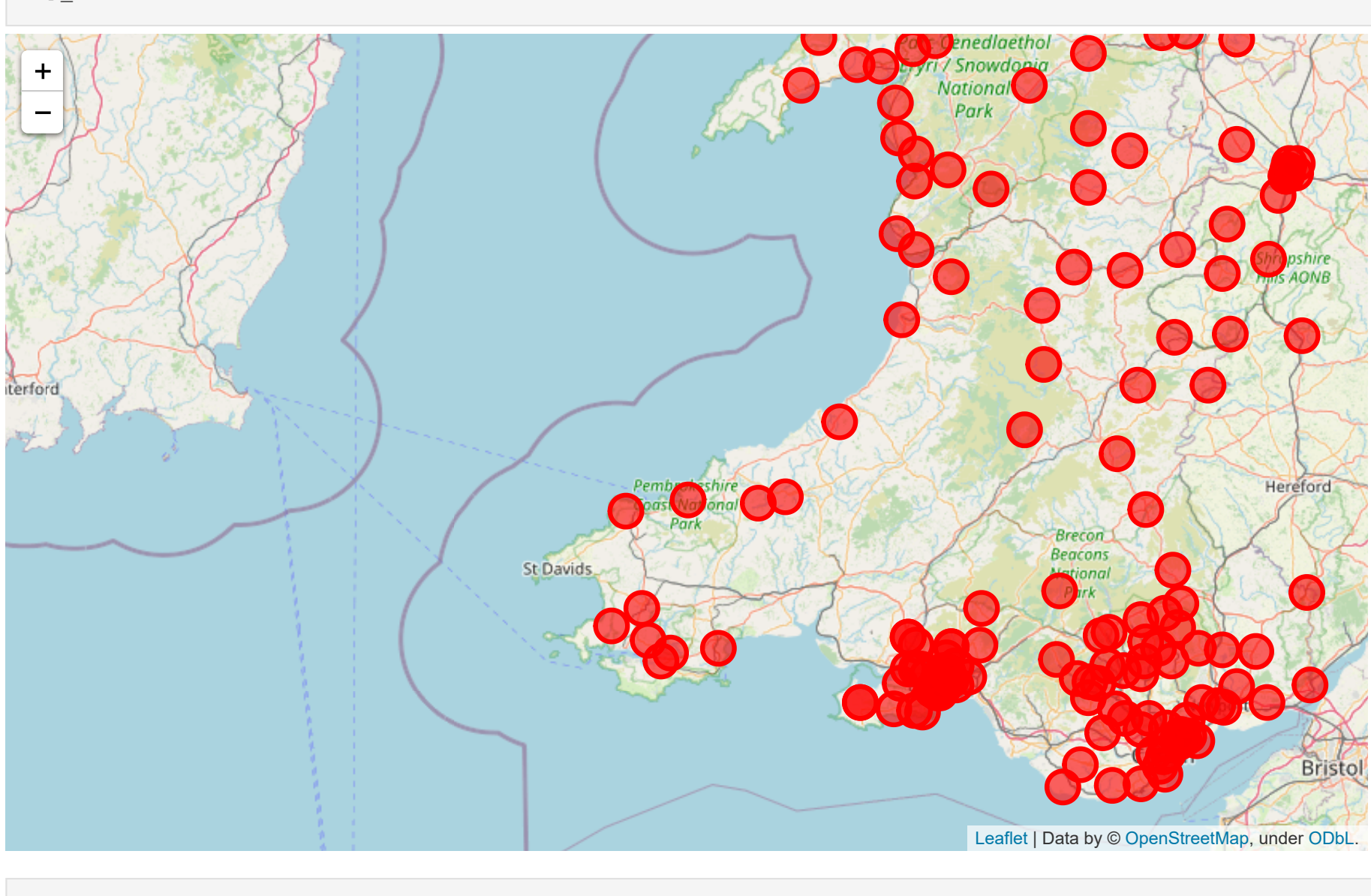
In [73]:
```
#checking all the dataframes if data is loaded correctly
df_list
```

Out[73]:
```
{'north_east':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0       DH1   244163.1          199.5          201.9      4.3%     -2.9%      7.0%
 1       DH2   156124.2          148.2          130.2      4.3%     10.3%     12.2%
 2       DH3   172700.4          173.2          130.5      3.9%    -12.0%     -5.4%
 3       DH4   167187.7          151.5          127.3      4.0%     -3.7%     -2.9%
 4       DH5   130662.1          136.5          116.5      4.6%     -5.5%     -2.0%
 ..      ...        ...            ...            ...       ...       ...       ...
 212    YO60   543633.0          273.8            NaN      None    -23.8%    -15.0%
 213    YO61   449382.2          276.1            NaN      None     -8.6%      4.3%
 214    YO62   369793.7          264.3            NaN      None     13.5%     23.6%
 215     YO7   307837.7          235.5          184.1      3.1%     -3.6%      7.6%
 216     YO8   249070.4          223.7          170.3      3.6%     -1.2%      8.3%

     growth_5y  sales_per_month turnover
 0        9.0%             30.0      21%
 1       10.6%             21.0      21%
 2        5.9%             18.0      16%
 3        1.3%             25.0      19%
 4       -7.0%             18.0      15%
 ..        ...              ...      ...
 212      0.4%              2.0       7%
 213     12.6%              9.0      14%
 214     29.8%              8.0      14%
 215     12.2%             16.0      13%
 216     17.3%             41.0      27%

 [217 rows x 10 columns],
 'north_west':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0       BB1   166661.3          139.9          138.7      4.3%     -3.9%     13.6%
 1      BB10   150239.9          109.1          115.8      4.0%      6.5%     17.8%
 2      BB11   109015.3           95.0          105.1      5.0%     16.7%     24.4%
 3      BB12   177420.7          150.1          120.1      3.5%     -8.5%     16.5%
 4      BB18   185276.3          142.5          134.9      3.8%      0.4%     16.4%
 ..      ...        ...            ...            ...       ...       ...       ...
 298     WN5   176996.8          161.9          151.5      4.5%      6.0%     21.4%
 299     WN6   212604.7          200.6          160.5      3.9%      0.9%      3.8%
 300     WN7   166781.5          161.3          150.3      4.7%     -5.0%     13.2%
 301     WN8   176953.2          160.0          164.0      4.8%     -6.2%     -0.9%
 302     M50   220151.5          341.6          257.1      6.1%     -2.2%     39.6%

     growth_5y  sales_per_month turnover
 0       17.8%             36.0      24%
 1       23.9%             33.0      21%
 2       40.2%             25.0      24%
 3       16.7%             37.0      24%
 4       10.6%             13.0      16%
 ..        ...              ...      ...
 298     22.1%             36.0      37%
 299     17.8%             34.0      43%
 300     21.4%             43.0      22%
 301      3.3%             28.0      22%
 302     23.2%              5.0       3%

 [303 rows x 10 columns],
 'east_midlands':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0       DE1   166925.5         178.20          152.4      4.7%     -9.6%      2.3%
 1      DE11   242013.5         229.55          174.4      3.7%      0.3%     10.7%
 2      DE12   314516.1         237.50          166.4      2.8%      0.3%      9.3%
 3      DE13   335195.5         262.30          187.2      2.9%     -1.5%     10.1%
 4      DE14   181259.5         179.40          153.3      4.4%      1.2%      9.1%
 ..      ...        ...            ...            ...       ...       ...       ...
 158     S75   244819.2         190.30          149.8      3.2%     -2.7%      3.4%
 159      S8   238046.4         239.25          185.0      4.0%     -2.1%      8.6%
 160     S80   175730.0         159.45          123.4      3.7%     -0.6%     17.4%
 161     S81   251895.1         180.80          166.9      3.4%      3.3%      8.8%
 162      S9   142377.0         142.40            NaN      None      9.8%     22.9%

     growth_5y  sales_per_month turnover
 0       13.9%               12      17%
 1       20.6%               46      22%
 2       21.6%               22      27%
 3       11.9%               34      17%
 4        9.5%               30      25%
 ..        ...              ...      ...
 158     21.6%               42      30%
 159     22.5%               31      27%
 160     25.7%               20      16%
 161     31.6%               33      15%
 162     35.7%               15      33%

 [163 rows x 10 columns],
 'west_midlands':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0        B1   235152.2         345.75          236.4      5.2%      0.1%      0.2%
 1       B10   202268.8         153.90            NaN      None      6.9%     27.6%
 2       B11   198979.6         179.70          175.4      4.6%      5.1%     13.2%
 3       B12   198207.2         199.20          199.6      5.2%     -2.7%     26.3%
 4       B13   294563.4         249.50          204.2      3.6%     -8.2%     10.4%
 ..      ...        ...            ...            ...       ...       ...       ...
 217     WV5   364493.9         267.90            NaN      None     -2.8%      5.9%
 218     WV6   291788.5         229.80          188.1      3.4%     -6.5%     10.5%
 219     WV7   355063.8         254.30            NaN      None     21.0%     47.0%
 220     WV8   327726.4         254.50          234.2      3.7%     -6.5%     14.3%
 221     WV9   257297.9         240.20            NaN      None     16.0%     34.1%

     growth_5y  sales_per_month turnover
 0        8.0%              9.0       3%
 1       43.7%              8.0      19%
 2       35.0%             19.0      41%
 3       36.4%              3.0       4%
 4       23.3%             22.0      23%
 ..        ...              ...      ...
 217     13.9%             10.0      22%
 218     21.4%             26.0      16%
 219     61.5%              3.0       9%
 220     30.9%             12.0      22%
 221     52.6%              6.0      35%

 [222 rows x 10 columns],
 'east_of_england':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0       AL1   530173.0         613.15          333.5      3.3%      0.5%     17.6%
 1      AL10   363455.7         386.80          349.6      5.0%     -1.0%      8.4%
 2       AL2   599550.8         483.35          299.3      2.6%      0.3%      6.1%
 3       AL3   625663.6         606.00          396.0      3.3%      1.3%      9.7%
 4       AL4   681031.4         544.35          318.1      2.4%      2.0%      9.9%
 ..      ...        ...            ...            ...       ...       ...       ...
 231     SS5   483386.6         364.10          340.4      3.7%     -0.5%     13.1%
 232     SS6   439511.9         382.95          312.2      3.7%      3.6%     12.4%
 233     SS7   442210.6         371.05          257.8      3.0%      3.1%     15.8%
 234     SS8   342011.9         346.10          230.7      3.5%      9.2%     17.3%
 235     SS9   454327.5         428.60          273.1      3.1%      6.9%     18.5%

     growth_5y  sales_per_month turnover
 0        9.1%             37.0      21%
 1       10.3%             15.0      19%
 2       11.1%             18.0      16%
 3       16.9%             20.0      17%
 4       13.5%             16.0      17%
 ..        ...              ...      ...
 231     21.1%             18.0      18%
 232     20.8%             28.0      24%
 233     18.7%             40.0      23%
 234     20.8%             31.0      32%
 235     29.4%             56.0      25%

 [236 rows x 10 columns],
 'greater_london':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0       BR1   481747.7         506.25          340.6      3.7%     -5.8%     11.1%
 1       BR2   572258.4         514.80          340.0      3.1%     -3.2%      9.2%
 2       BR3   543353.9         533.80          331.7      3.2%     -6.1%      3.2%
 3       BR4   715835.7         491.10            NaN      None      5.1%     21.5%
 4       BR5   478807.1         490.70          345.4      3.8%      0.9%     24.4%
 ..      ...        ...            ...            ...       ...       ...       ...
 270     WD3   658019.5         567.30          356.4      2.8%      9.7%     17.4%
 271     WD4   622975.7         491.10          296.4      2.5%     -5.6%     20.7%
 272     WD5   536621.1         497.80            NaN      None      2.8%     13.6%
 273     WD6   477905.2         519.05          339.5      3.7%      1.2%     23.5%
 274     WD7   913618.1         579.90          514.2      2.9%     22.2%     23.7%

     growth_5y  sales_per_month turnover
 0        9.9%             50.0      25%
 1        7.4%             41.0      23%
 2       -0.1%             44.0      22%
 3       26.7%             15.0      32%
 4       25.5%             25.0      20%
 ..        ...              ...      ...
 270     17.5%             28.0      17%
 271      4.0%              8.0      10%
 272      6.4%              7.0      20%
 273     17.6%             22.0      19%
 274     26.5%             10.0      18%

 [275 rows x 10 columns],
 'south_east':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0       BN1   482630.3         509.00          347.4      3.7%     -5.8%      9.9%
 1      BN10   395825.8         339.00          255.4      3.4%      3.7%     15.6%
 2      BN11   304452.1         358.20          242.7      4.1%     -9.9%     11.6%
 3      BN12   468757.7         431.45          284.7      3.2%      8.2%     25.7%
 4      BN13   411139.1         383.90          268.3      3.4%      3.5%     11.7%
 ..      ...        ...            ...            ...       ...       ...       ...
 311     TN5   629422.1         386.20            NaN      None    -11.9%      5.7%
 312     TN6   547117.3         367.00          287.1      2.7%      1.3%     20.9%
 313     TN7   999117.7         408.50            NaN      None   -100.0%   -100.0%
 314     TN8   497999.7         408.10            NaN      None     -4.8%     10.0%
 315     TN9   394248.3         418.10          268.3      3.5%     -7.8%      4.7%

     growth_5y  sales_per_month turnover
 0       14.4%               71       8%
 1       23.0%               16      15%
 2       22.5%               37      23%
 3       34.9%               23      29%
 4       16.6%               27      25%
 ..        ...              ...      ...
 311      4.9%                7      24%
 312     20.1%               21      20%
 313   -100.0%                2      11%
 314     15.3%               11      20%
 315      8.9%               21      20%

 [316 rows x 10 columns],
 'south_west':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0       BA1   515211.8         448.15          305.0      3.1%     -0.3%      5.2%
 1      BA10   388690.5         280.70            NaN      None     39.8%     96.1%
 2      BA11   394495.3         338.35          260.6      3.4%     -0.8%     18.5%
 3      BA12   365043.9         336.55          223.2      3.2%      3.6%     16.7%
 4      BA13   324774.5         260.90          200.9      3.2%      0.9%      9.0%
 ..      ...        ...            ...            ...       ...       ...       ...
 270     TR5   560833.3         361.90            NaN      None      4.2%     32.8%
 271     TR6   457968.8         333.10            NaN      None     -9.6%     13.9%
 272     TR7   382467.9         341.65          224.2      3.0%      6.1%     33.4%
 273     TR8   340237.1         287.10            NaN      None     -1.8%      6.6%
 274     TR9   290600.0         232.50            NaN      None     -0.4%     28.7%

     growth_5y  sales_per_month turnover
 0       13.1%             32.0      16%
 1       64.2%              3.0      16%
 2       28.9%             23.0      17%
 3       26.1%             30.0      29%
 4       16.6%             22.0      15%
 ..        ...              ...      ...
 270     32.1%              5.0      19%
 271     28.3%              4.0      14%
 272     44.9%             20.0      16%
 273     28.7%             11.0      10%
 274     35.2%              7.0      22%

 [275 rows x 10 columns],
 'wales':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0      CF10   191179.2         249.40          227.7      6.2%      0.0%      4.5%
 1      CF11   241035.3         290.05          240.3      5.2%      0.8%     18.2%
 2      CF14   351226.1         306.65          242.7      3.6%      7.8%     20.6%
 3      CF15   371554.0         265.30            NaN      None      5.8%     15.8%
 4      CF23   353603.0         306.45          242.7      3.6%      1.9%     27.5%
 ..      ...        ...            ...            ...       ...       ...       ...
 175     SY5   395348.6         243.30            NaN      None      2.7%     22.3%
 176     SY6   465336.5         270.90            NaN      None      0.0%     25.7%
 177     SY7   418178.7         227.90            NaN      None      9.8%     24.8%
 178     SY8   358226.5         250.80          167.5      2.4%    -14.9%      8.1%
 179     SY9   348571.1         233.10            NaN      None   -100.0%   -100.0%

     growth_5y  sales_per_month turnover
 0       10.0%             18.0      15%
 1       37.4%             34.0      25%
 2       36.7%             59.0      30%
 3       20.0%             15.0      31%
 4       30.5%             35.0      26%
 ..        ...              ...      ...
 175     19.7%             15.0      18%
 176     41.5%              5.0      14%
 177     37.7%              6.0      10%
 178      6.4%             12.0      10%
 179   -100.0%              2.0      15%

 [180 rows x 10 columns],
 'scotland':     outcode  avg_price  avg_price_psf  avg_rent avg_yield growth_1y growth_3y  \
 0      AB10   110448.1          193.9          142.1      6.7%      3.3%     -0.7%
 1      AB11   100371.4          181.3          132.4      6.9%    -11.7%     -6.7%
 2      AB12   241096.4          199.2            NaN      None      0.6%     -4.6%
 3      AB15   327879.3          232.9          297.3      4.7%     -4.4%     -6.8%
 4      AB16   107772.7          149.2            NaN      None     -1.0%     -8.0%
 ..      ...        ...            ...            ...       ...       ...       ...
 303     TD6   226304.4          176.8            NaN      None      None     25.8%
 304     TD7   184750.0          132.1            NaN      None      None     24.2%
 305     TD8   223181.8          119.6            NaN      None      None     39.6%
 306     TD9   101759.2          106.9            NaN      None      None     23.3%
 307     G75   153042.1          140.5          142.0      4.8%      1.4%      0.3%

     growth_5y  sales_per_month turnover
 0        None             49.0     104%
 1        None             44.0      75%
 2        None             40.0      79%
 3        None             54.0      67%
 4        None             25.0      79%
 ..        ...              ...      ...
 303      None             10.0      86%
 304      None              7.0      28%
 305      None              9.0      56%
 306      None             27.0      32%
 307      None             50.0      44%

 [308 rows x 10 columns],
 'northern_ireland':   outcode  avg_price  avg_price_psf avg_rent avg_yield growth_1y growth_3y  \
 0    BT19        NaN              0     None      None      None      None
 1    BT23   191154.2              0     None      None      None      None
 2    BT24        NaN              0     None      None      None      None
 3    BT28   185915.0              0     None      None      None      None
 4    BT34   209465.0              0     None      None      None      None
 5    BT35   191179.2              0     None      None      None      None
 6    BT38   192745.6              0     None      None      None      None
 7     BT4        NaN              0     None      None      None      None
 8     BT5        NaN              0     None      None      None      None
 9     BT6        NaN              0     None      None      None      None

   growth_5y sales_per_month turnover
 0      None            None        %
 1      None            None        %
 2      None            None        %
 3      None            None        %
 4      None            None        %
 5      None            None        %
 6      None            None        %
 7      None            None        %
 8      None            None        %
 9      None            None        %  }
```

In [98]:
```
#calling nominatim function
geolocator = Nominatim(user_agent="UK")
#getting the co-ordinates of the cities of england
for area in df_list:
    lat=[]
    long=[]
    for code in df_list[area].outcode:
        address = str(code+", United Kingdom")
        location = geocoder.arcgis(address)
        lat.append(location.latlng[0])
        long.append(location.latlng[1])
    df_list[area]['lat']=lat
    df_list[area]['long']=long
```

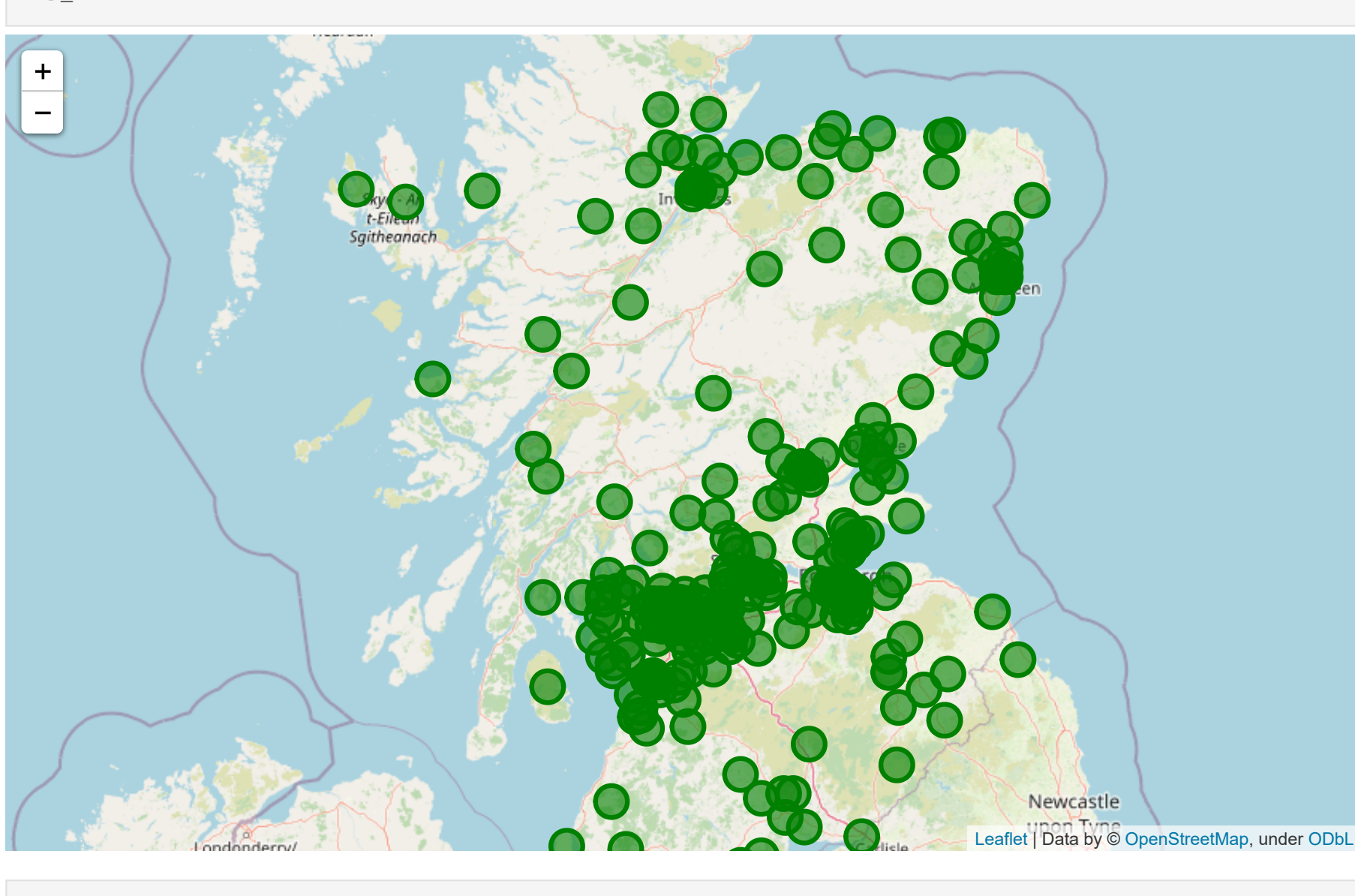
In [132…
```
#plotting the prices of wales
#plotting a map
map_UK=folium.Map(location=[52.1307 , -4.7837],zoom_start=8)
for lat,lng,area,price in zip(df_list['wales']['lat'],df_list['wales']['long'],df_list['wales']['outcode'],df_l
    label='AreaCode :{}, Price :£{}'.format(area,price)
    folium.CircleMarker([lat,lng],color='red',popup=label,fill=True,fill_color='red',fill_opacity=0.6).add_to(m
map_UK
```

Out[132…


+
−
Eryri / Snowdonia National Park
Shropshire Hills AONB
Pembrokeshire Coast National Park
St Davids
Hereford
Brecon Beacons National Park
Bristol
Leaflet | Data by © OpenStreetMap, under ODbL.

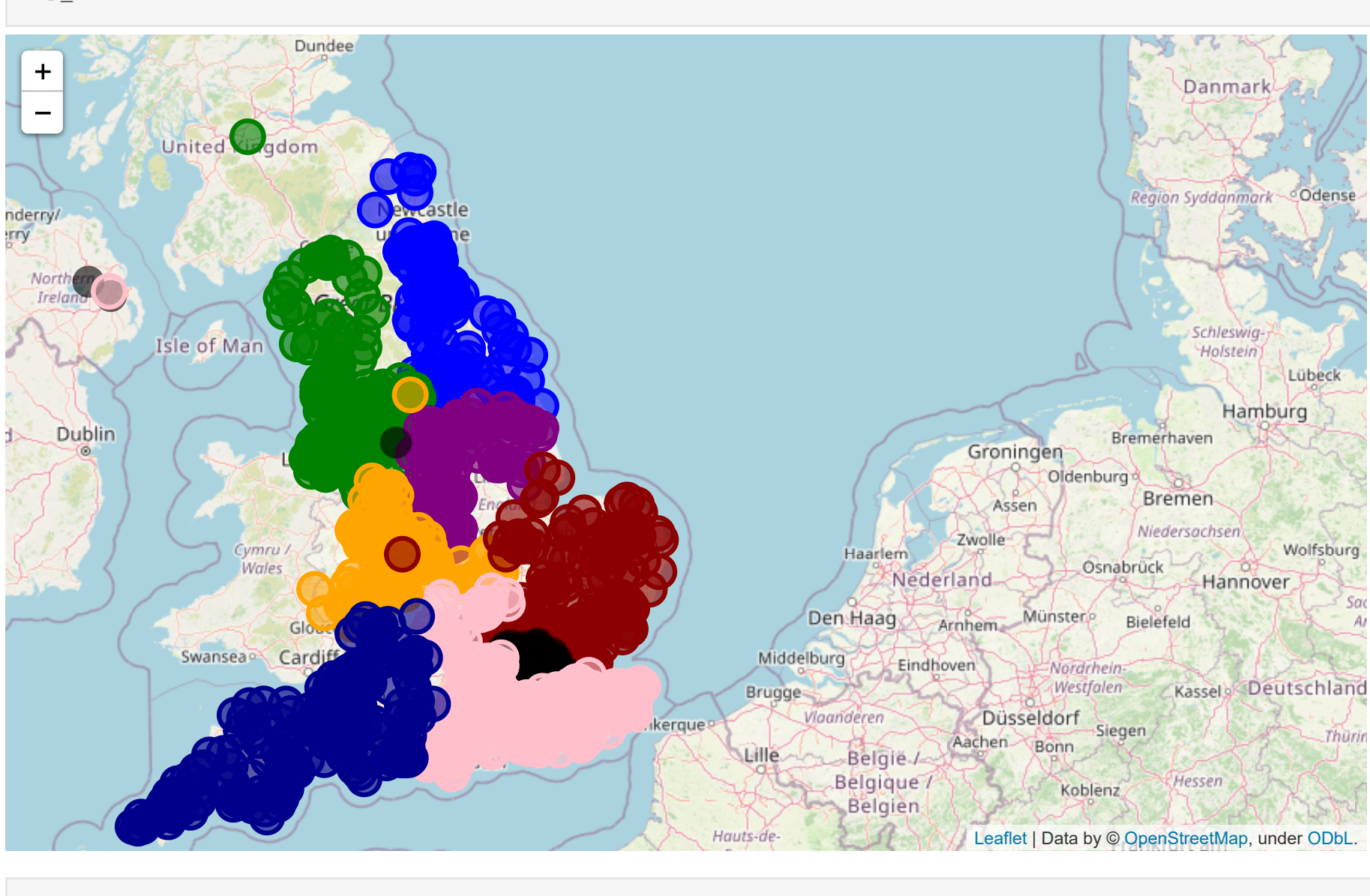
In [121…
```
#plotting the prices of scotland
#plotting a map
map_UK=folium.Map(location=[56.4907 , -4.2026],zoom_start=7)
for lat,lng,area,price in zip(df_list['scotland']['lat'],df_list['scotland']['long'],df_list['scotland']['outc
    label='AreaCode :{}, Price :£{}'.format(area,price)
    folium.CircleMarker([lat,lng],color='green',popup=label,fill=True,fill_color='green',fill_opacity=0.6).add_
map_UK
```

Out[121…



In [133…
```
#plotting the prices of england
#plotting a map
map_UK=folium.Map(location=[53.3555 , 2.1743],zoom_start=6)
color_code =[ 'blue', 'green', 'purple', 'orange', 'darkred', 'lightred', 'pink', 'darkblue']
for name,color in zip(reg_eng,color_code):
    for lat,lng,area,price in zip(df_list[name]['lat'],df_list[name]['long'],df_list[name]['outcode'],df_list[n
        label='AreaCode :{}, Price :£{}'.format(area,price)
        folium.CircleMarker([lat,lng],color=color,popup=label,fill=True,fill_color=color,fill_opacity=0.6).add_
map_UK
```

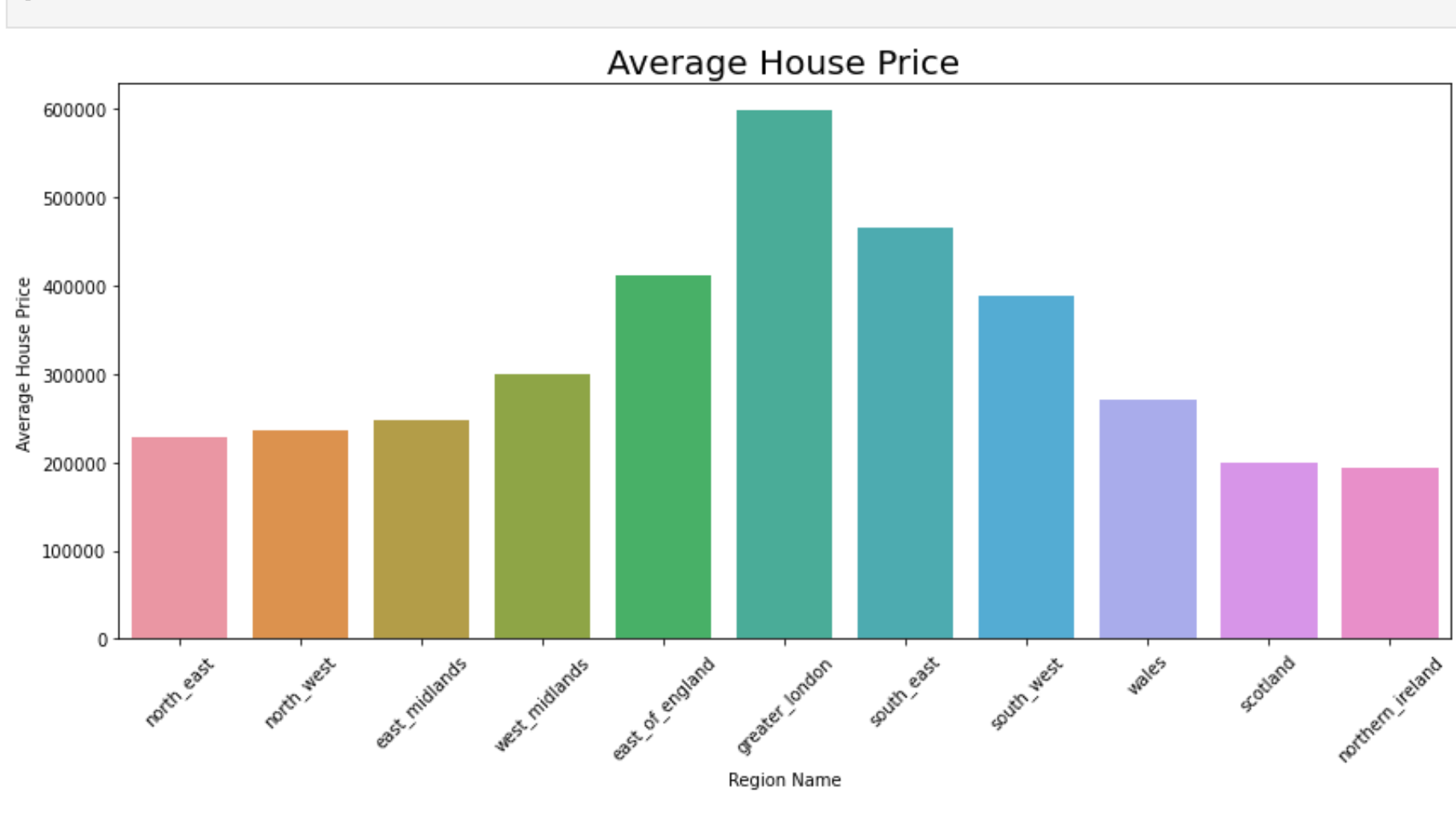
Out[133…



In [137…
```
#calculating the average prices of the countries
reg_price =[]
for reg in region_list:
    reg_price.append(df_list[reg]['avg_price'].mean())
reg_price
```

Out[137…
```
[227462.599537037,
 235224.32433333338,
 247548.39141104312,
 298845.42850678734,
 410298.54553191504,
 598243.173818182,
 464399.2794303794,
 388698.70402930374,
 270287.2359999999,
 198494.53214285712,
 194091.8]
```

In [142…
```
#plotting the average prices region wise
plt.figure(figsize=(14,6))
sns.barplot(x=region_list,y=reg_price)
plt.title('Average House Price',fontdict={'fontsize':20})
plt.xlabel("Region Name")
plt.ylabel('Average House Price(in pounds)')
plt.xticks(rotation = 45)
plt.show()
```

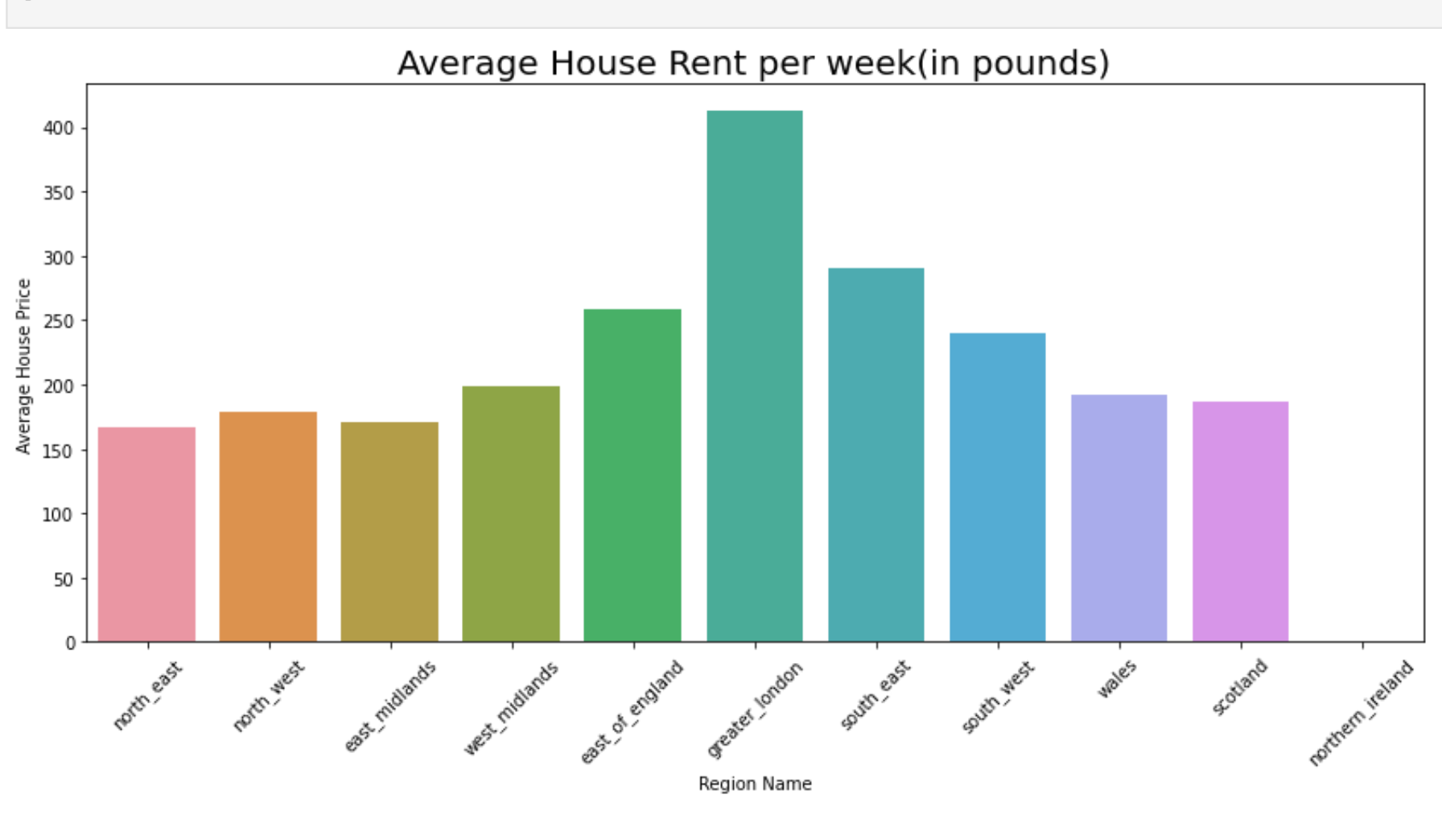


We can see from the above chart that the greater london region has the highest average prices amoung all regions in UK, whereas northern ireland has the lowest.

In [146…
```
#calculating the average rents of the regions
reg_rent =[]
for reg in region_list:
    reg_rent.append(df_list[reg]['avg_rent'].mean())
reg_rent
```

Out[146…
```
[167.23034482758626,
 179.2582159624413,
 171.43833333333336,
 198.2769230769231,
 259.1965116279069,
 412.9697674418603,
 290.91729957805904,
 240.33733333333328,
 192.64883720930237,
 187.1318181818182,
 nan]
```

In [147…
```
#plotting the average rents region wise
plt.figure(figsize=(14,6))
sns.barplot(x=region_list,y=reg_rent)
plt.title('Average House Rent per week(in pounds)',fontdict={'fontsize':20})
plt.xlabel("Region Name")
plt.ylabel('Average House Price')
plt.xticks(rotation = 45)
plt.show()
```

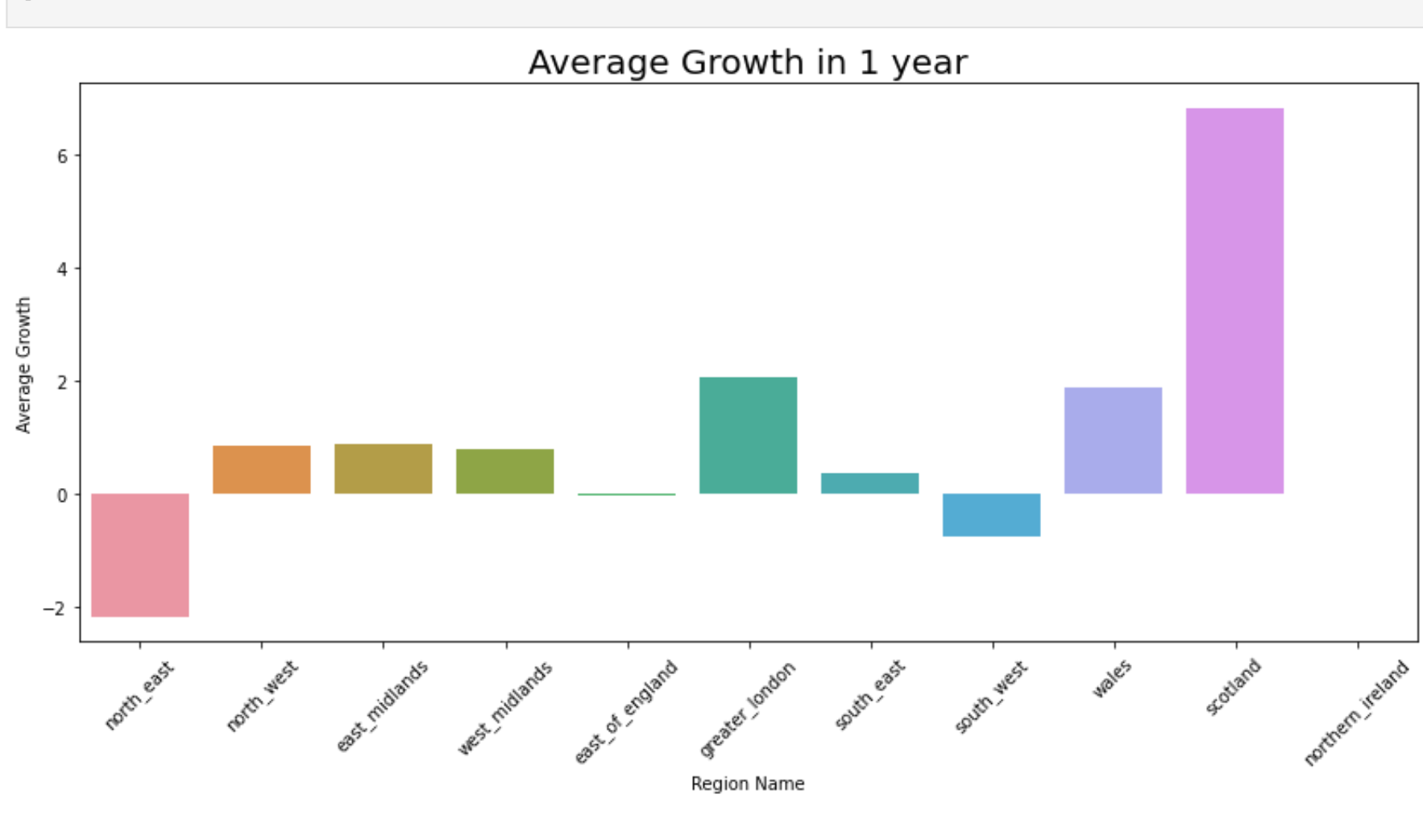


We can see from the above chart that the greater london region has the highest average prices amoung all regions in UK, whereas east midlands has the lowest. Moreover , no data is provided for northern ireland.

In [175…
```
#calculating the average rents of the regions
reg_growth_1 =[]
for reg in region_list:
    reg_growth_1.append(df_list[reg]['growth_1y'].str.strip(",%").astype(float).mean())

reg_growth_1
```

Out[175…
```
[-2.175462962962963,
 0.835548172757475,
 0.8803680981595097,
 0.7790909090909092,
 -0.04067796610169442,
 2.051094890510949,
 0.3724683544303802,
 -0.7556363636363631,
 1.8759776536312847,
 6.815957446808511,
 nan]
```

In [176…
```
#plotting the average growth in a year region wise
fig =plt.figure(figsize=(14,6))
sns.barplot(x=region_list,y=reg_growth_1)
plt.title('Average Growth in 1 year',fontdict={'fontsize':20})
plt.xlabel("Region Name")
plt.ylabel('Average Growth')
plt.xticks(rotation = 45)
plt.show()
```

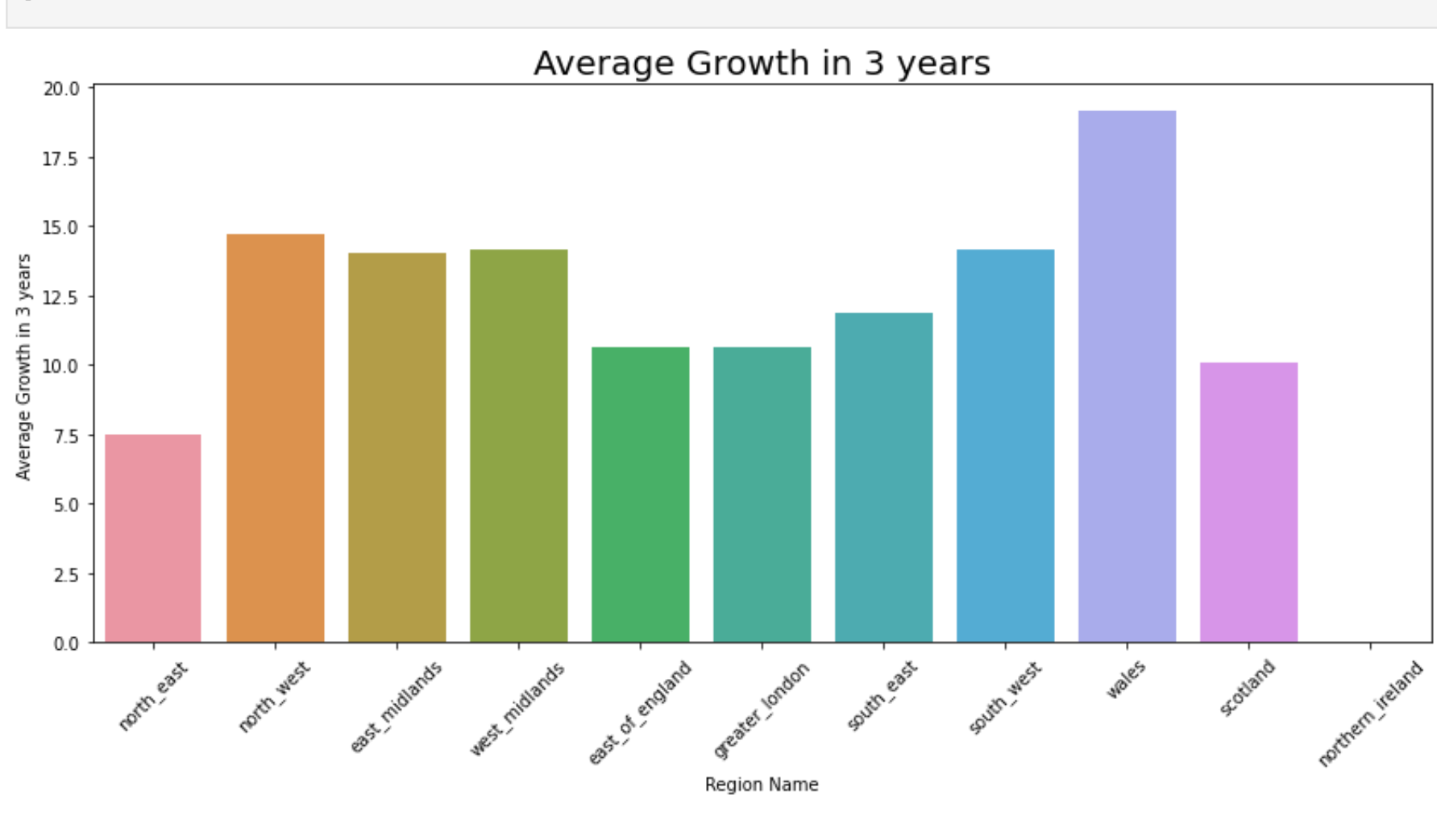


We can see from the above chart that the scotland has the highest average growth in one year time , while north east region has the lowest.Moreover, northern ireland data was not provided.

In [179…
```
#calculating the average rents of the regions
reg_growth_2 =[]
for reg in region_list:
    reg_growth_2.append(df_list[reg]['growth_3y'].str.strip(",%").astype(float).mean())

reg_growth_2
```

Out[179…
```
[7.508796296296289,
 14.699003322259134,
 14.017791411042952,
 14.17488584474886,
 10.625000000000004,
 10.661090909090902,
 11.846202531645565,
 14.12956204379562,
 19.159550561797747,
 10.082736156351796,
 nan]
```

In [180…
```
#plotting the average growth in a three years region wise
fig =plt.figure(figsize=(14,6))
sns.barplot(x=region_list,y=reg_growth_2)
plt.title('Average Growth in 3 years',fontdict={'fontsize':20})
plt.xlabel("Region Name")
plt.ylabel('Average Growth in 3 years')
plt.xticks(rotation = 45)
plt.show()
```

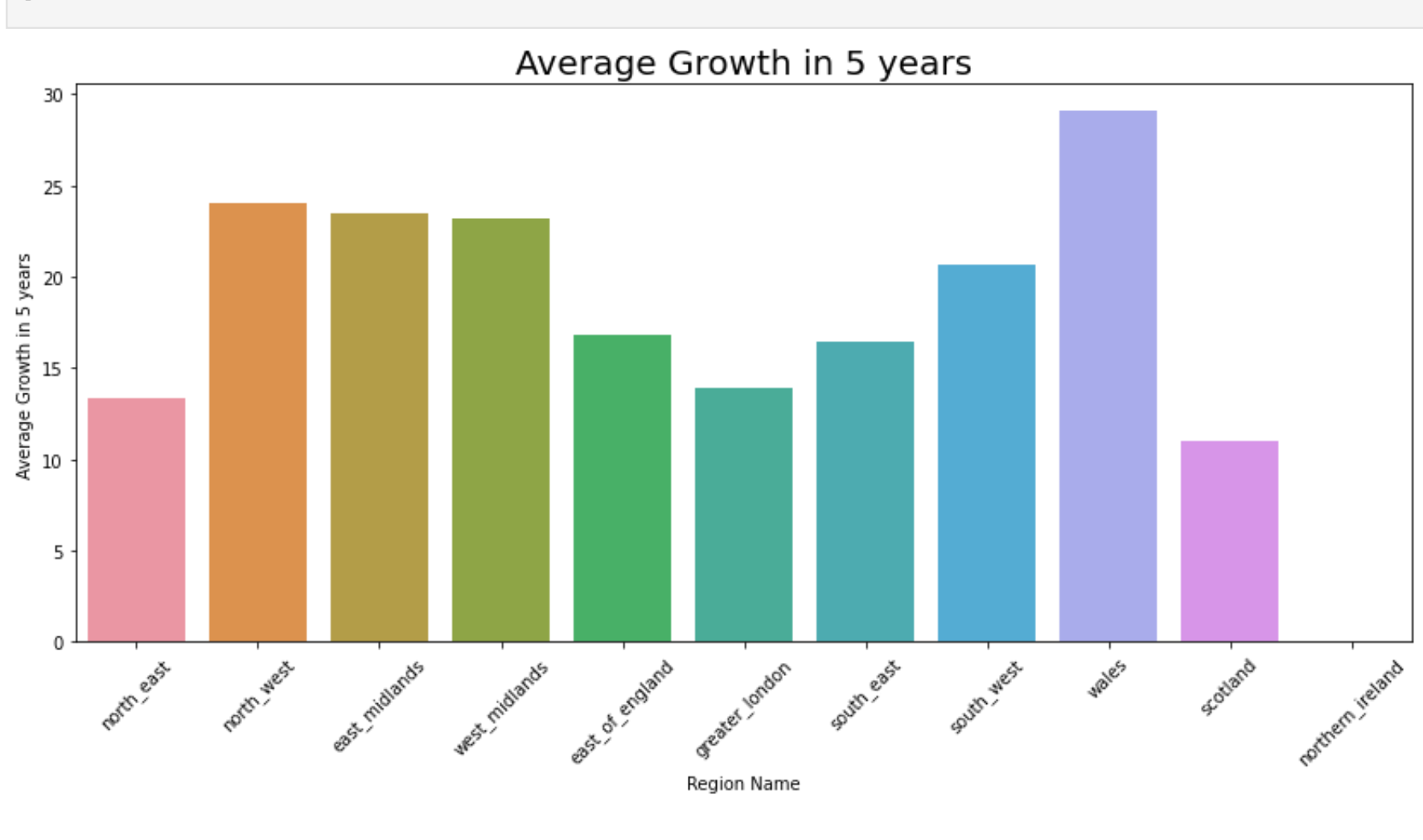


We can see from the above chart that the wales has the highest average growth in three years time , while north east region has the lowest.Moreover, northern ireland data was not provided.

In [181…
```
#calculating the average rents of the regions
reg_growth_3 =[]
for reg in region_list:
    reg_growth_3.append(df_list[reg]['growth_5y'].str.strip(",%").astype(float).mean())

reg_growth_3
```

Out[181…
```
[13.372685185185185,
 24.052333333333334,
 23.444171779141097,
 23.216289592760173,
 16.796610169491526,
 13.964102564102564,
 16.48006329113924,
 20.684363636363628,
 29.11761363636364,
 10.966666666666667,
 nan]
```

In [182…
```
#plotting the average growth in a 5 years region wise
fig =plt.figure(figsize=(14,6))
sns.barplot(x=region_list,y=reg_growth_3)
plt.title('Average Growth in 5 years',fontdict={'fontsize':20})
plt.xlabel("Region Name")
plt.ylabel('Average Growth in 5 years')
plt.xticks(rotation = 45)
plt.show()
```



We can see from the above chart that the Wales has the highest average growth in five years time , while scotland region has the lowest.Moreover, northern ireland data was not provided.

In [184…
```
df_list.keys()
```

Out[184…
```
dict_keys(['north_east', 'north_west', 'east_midlands', 'west_midlands', 'east_of_england', 'greater_london',
'south_east', 'south_west', 'wales', 'scotland', 'northern_ireland'])
```

In [187…
```
with pd.ExcelWriter("all_places.xlsx") as writer:
    df_list['north_east'].to_excel(writer,sheet_name="north_east")
    df_list['north_west'].to_excel(writer,sheet_name="north_west")
    df_list['east_midlands'].to_excel(writer,sheet_name="east_midlands")
    df_list['west_midlands'].to_excel(writer,sheet_name="west_midlands")
    df_list['east_of_england'].to_excel(writer,sheet_name="east_of_england")
    df_list['greater_london'].to_excel(writer,sheet_name="greater_london")
    df_list['south_east'].to_excel(writer,sheet_name="south_east")
    df_list['south_west'].to_excel(writer,sheet_name="south_west")
    df_list['wales'].to_excel(writer,sheet_name="wales")
    df_list['scotland'].to_excel(writer,sheet_name="scotland")
    df_list['northern_ireland'].to_excel(writer,sheet_name="northern_ireland")
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Comparison of Different Indices

https://www.gov.uk/government/publications/about-the-uk-house-price-index/comparing-house-price-indices-in-the-uk

Source : https://ifs.org.uk/bns/bn146.pdf

There are several major house price indices in the UK. These indices differ both in terms of what they are trying to measure and in terms of the data on which they are based. We focus on the following five indices, because they all seek to measure the price of housing using actual sales data and are relatively long-standing; hence, we can evaluate differences in trends in the series over a longer period:

TOC:

- Indices produced by Halifax
- Indices produced by the Nationwide Building Society
- An index produced by LSL Property Services / Acadata (LSL Acad),which is based on Land Registry data.
- The Land Registry house price index, which is based on a complete register of all residential sales in England and Wales

In [134…

```
#importing libraries
import pandas as pd
import seaborn as sns
from datetime import datetime
import matplotlib.pyplot as plt
```

HALIFAX House Price Index Data

In [75]:

```
#importing the halifax hpi data
df_halifax = pd.read_excel(r'indices_dataset\halifax_HPI.xlsx')
df_halifax.head()
```

Out[75]:

	Date	Price	Index
0	2019-01-01	223691	397.8
1	2019-02-01	236800	403.8
2	2019-03-01	233181	402.6
3	2019-04-01	236619	400.3
4	2019-05-01	237837	399.7

In [76]:

```
#describing the dataset(checking the statistical information)
df_halifax.describe()
```

Out[76]:

	Price	Index
count	42.000000	42.000000
mean	252375.261905	434.535714
std	18242.371306	32.052357
min	223691.000000	397.800000
25%	237664.000000	406.275000
50%	250163.500000	430.650000
75%	262651.250000	453.150000
max	294845.000000	508.400000

We can see :
1. The House Price Max is 294,845 and Min is 223,691 with a standard deviation of 18242.37 and mean as 252375.26
2. Index Max is 508.4 and Min is 397.8 with a standard deviation of 32.052 and mean as 434.53

In [77]:

```
#checking for any null values
df_halifax.isna().sum()
```

Out[77]:

```
Date     0
Price    0
Index    0
dtype: int64
```

In [78]:

```
#checking the data types of variables
df_halifax.dtypes
```

Out[78]:

```
Date     datetime64[ns]
Price             int64
Index           float64
dtype: object
```

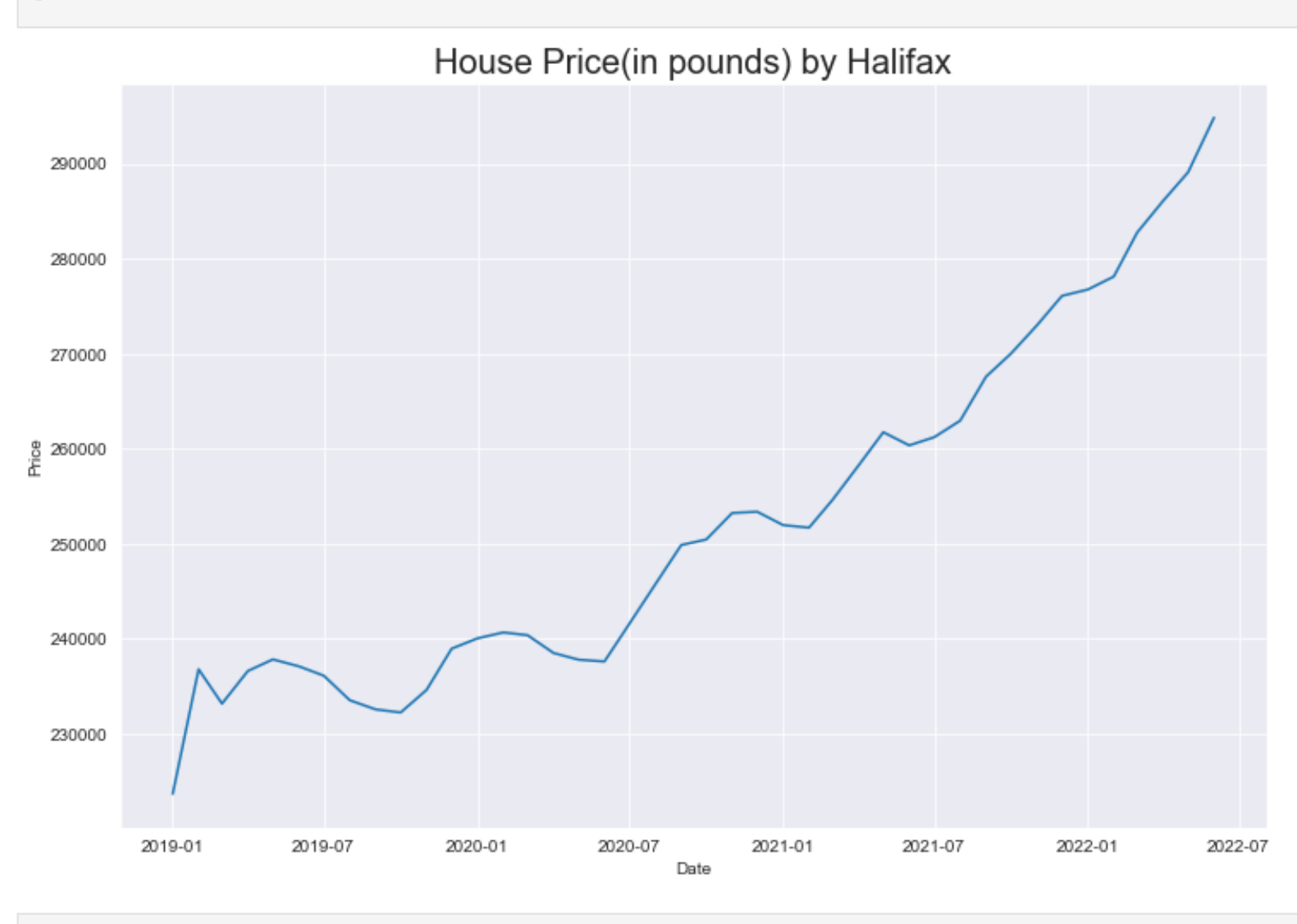
In [79]:

```
#to check the date range
date_list = list(df_halifax['Date'])
print("The starting date of dataset is :",date_list[0])
print("The Ending date of dataset is :",date_list[-1])
```

```
The starting date of dataset is : 2019-01-01 00:00:00
The Ending date of dataset is : 2022-06-01 00:00:00
```

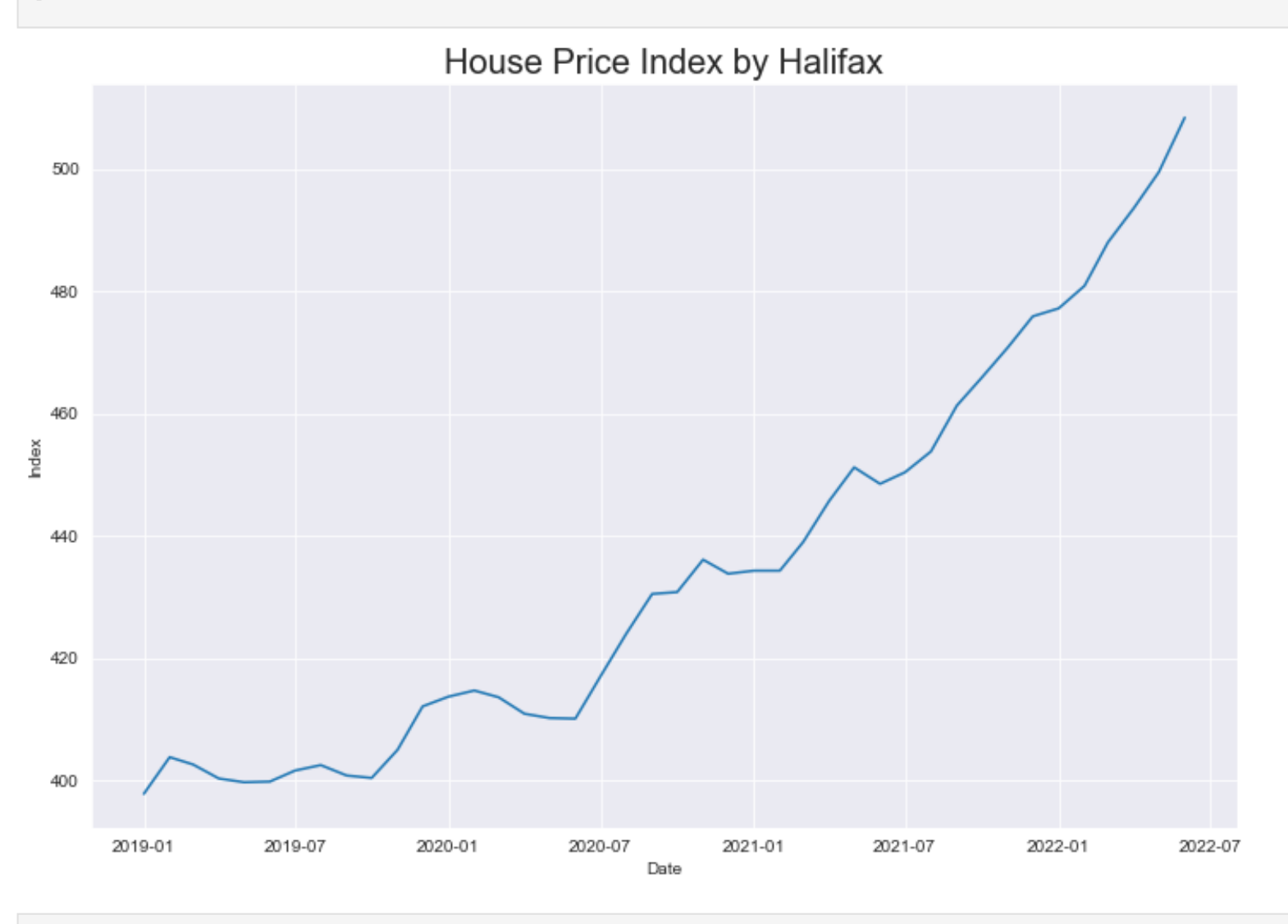
In [80]:

```
#Plotting the house price of halifax
sns.set_style("darkgrid")
plt.figure(figsize = (12,8))
sns.lineplot(data = df_halifax ,x='Date',y='Price')
plt.title("House Price(in pounds) by Halifax",fontdict={'fontsize': 20})
plt.show()
```



In [81]:

```
#Plotting the Index of halifax
plt.figure(figsize = (12,8))
sns.lineplot(data = df_halifax ,x='Date',y='Index')
plt.title("House Price Index by Halifax",fontdict={'fontsize': 20})
plt.show()
```



In [82]:

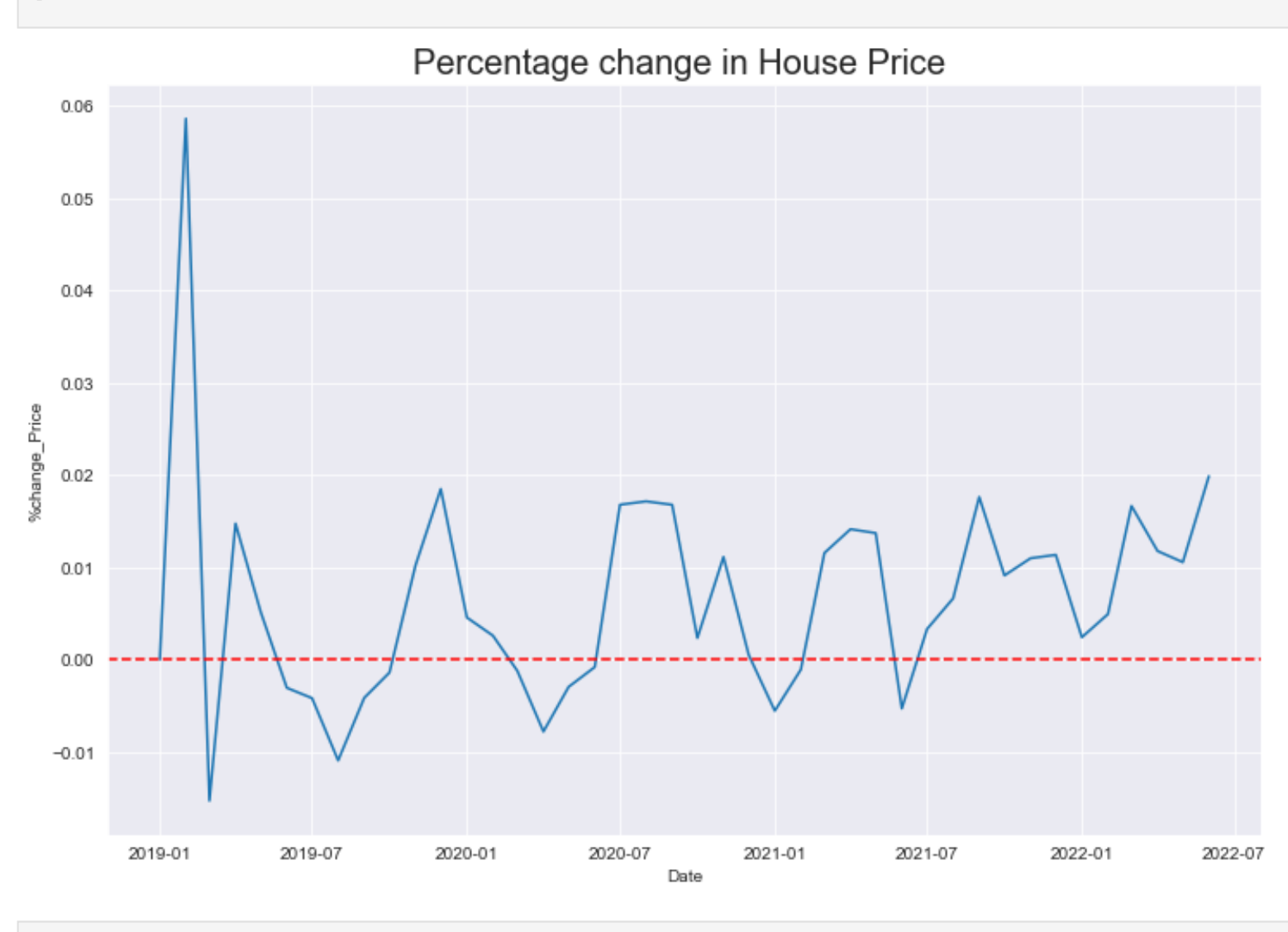
```
#calculating the percentage change in house price and index
df_halifax['%change_HPI'] =df_halifax['Index'].pct_change()
df_halifax['%change_Price'] =df_halifax['Price'].pct_change()
df_halifax.fillna(0,inplace=True)
df_halifax.head()
```

Out[82]:

	Date	Price	Index	%change_HPI	%change_Price
0	2019-01-01	223691	397.8	0.000000	0.000000
1	2019-02-01	236800	403.8	0.015083	0.058603
2	2019-03-01	233181	402.6	-0.002972	-0.015283
3	2019-04-01	236619	400.3	-0.005713	0.014744
4	2019-05-01	237837	399.7	-0.001499	0.005148

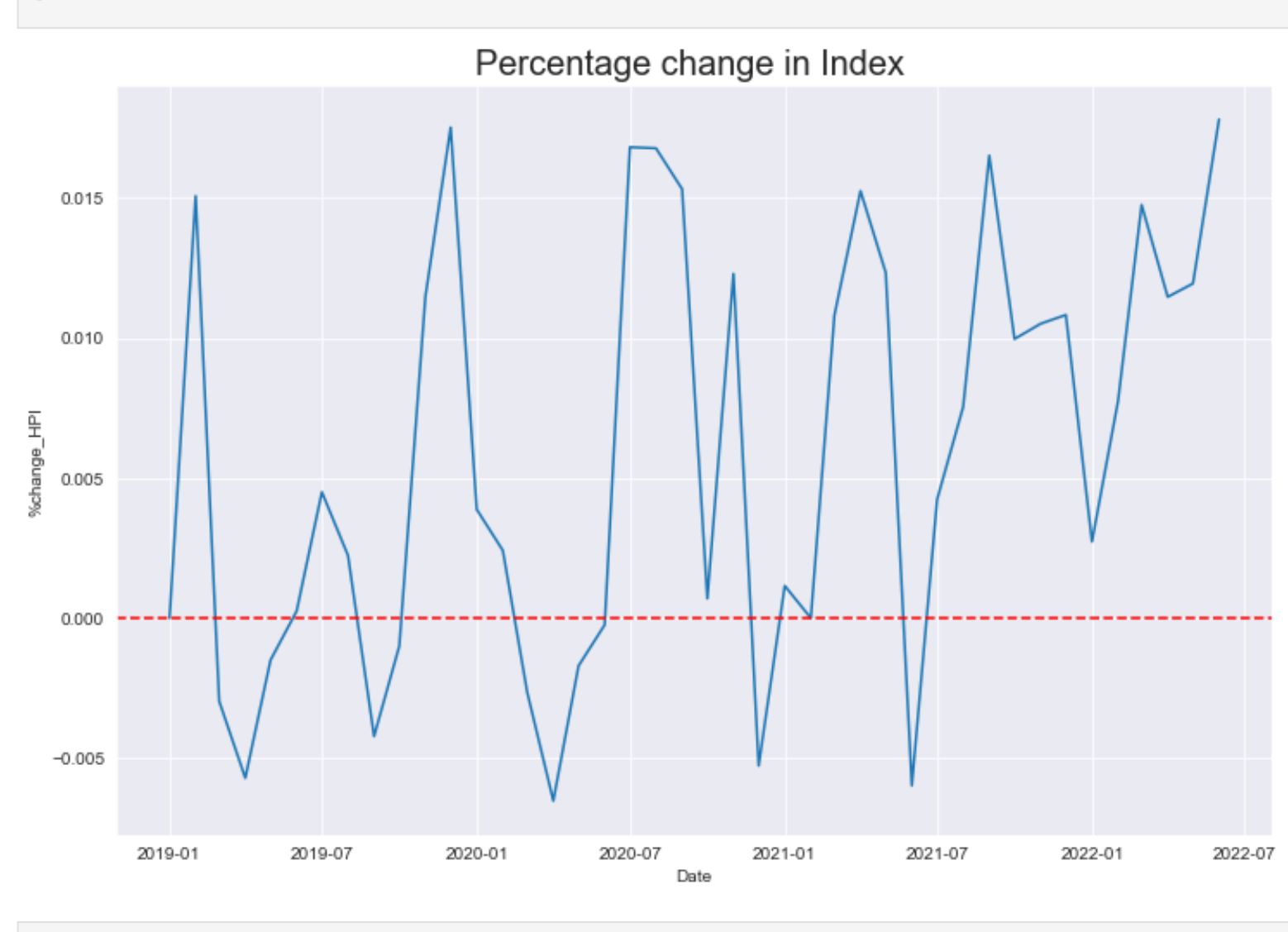
In [83]:

```
# plotting the percentage change in house prices
plt.figure(figsize = (12,8))
sns.lineplot(data=df_halifax,x='Date',y='%change_Price')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in House Price ",fontdict={'fontsize': 20})
plt.show()
```



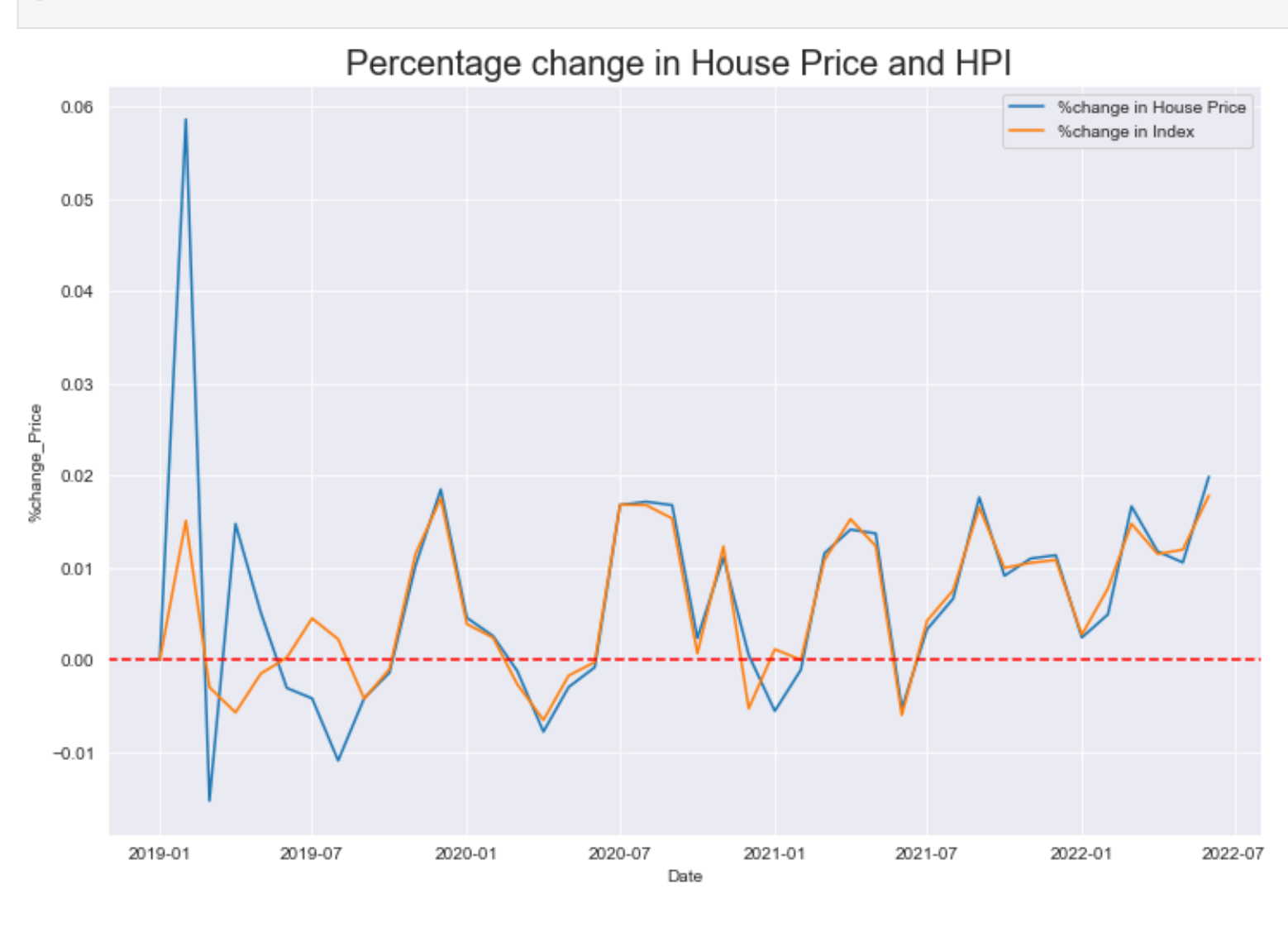
In [84]:

```
#plotting the percentage change in index
plt.figure(figsize = (12,8))
sns.lineplot(data=df_halifax,x='Date',y='%change_HPI')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in Index ",fontdict={'fontsize': 20})
plt.show()
```



In [85]:

```
#plotting the percentage change together of both house price and Index
plt.figure(figsize = (12,8))
sns.lineplot(data=df_halifax,x='Date',y='%change_Price',label='%change in House Price')
sns.lineplot(data=df_halifax,x='Date',y='%change_HPI',label='%change in Index')
plt.axhline(0, ls='--',c='red')
plt.legend()
plt.title("Percentage change in House Price and HPI ",fontdict={'fontsize': 20})
plt.show()
```



Nationwide House Price Index Data

In [86]:

```
#importing nationwide data
df_nationwide = pd.read_excel(r'indices_dataset\nationwide_prices.xls')
df_nationwide.head()
```

Out[86]:

	Date	Average House Price	INDEX	Seasonally Adjusted Index	Monthly % Change (SA)	Year % Change	Latest 3 months on previous 3 months
0	1991-01-01	53051.721106	105.832511	107.127299	NaN	NaN	NaN
1	1991-02-01	53496.798746	106.720393	107.494996	0.003432	NaN	NaN
2	1991-03-01	52892.861606	105.515603	106.525649	-0.009018	NaN	NaN
3	1991-04-01	53677.435270	107.080744	107.126622	0.005642	NaN	NaN
4	1991-05-01	54385.726747	108.493710	107.266757	0.001308	NaN	NaN

In [87]:

```
#dropping non-required columns
df_nationwide = df_nationwide[['Date','Average House Price','INDEX']]
df_nationwide.head()
```

Out[87]:

	Date	Average House Price	INDEX
0	1991-01-01	53051.721106	105.832511
1	1991-02-01	53496.798746	106.720393
2	1991-03-01	52892.861606	105.515603
3	1991-04-01	53677.435270	107.080744
4	1991-05-01	54385.726747	108.493710

In [88]:

```
##describing the dataset(checking the statistical information)
df_nationwide.describe()
```

Out[88]:

	Average House Price	INDEX
count	379.000000	379.000000
mean	139109.281995	277.508143
std	64082.182943	127.837103
min	49601.664241	98.950016
25%	66378.483270	132.417977
50%	158572.967796	316.336115
75%	188502.181703	376.041697
max	271613.044812	541.838982

We can see :
1. The House Price Max is 271,613 and Min is 49,601 with a standard deviation of 64082.18 and mean as 139109.28
2. Index Max is 541.83 and Min is 98.95 with a standard deviation of 127.83 and mean as 277.50

In [89]:

```
#checking for any null values
df_nationwide.isna().sum()
```

Out[89]:

```
Date                   0
Average House Price    0
INDEX                  0
dtype: int64
```

In [90]:

```
#checking the data types of variables
df_nationwide.dtypes
```

Out[90]:

```
Date                   datetime64[ns]
Average House Price           float64
INDEX                         float64
dtype: object
```

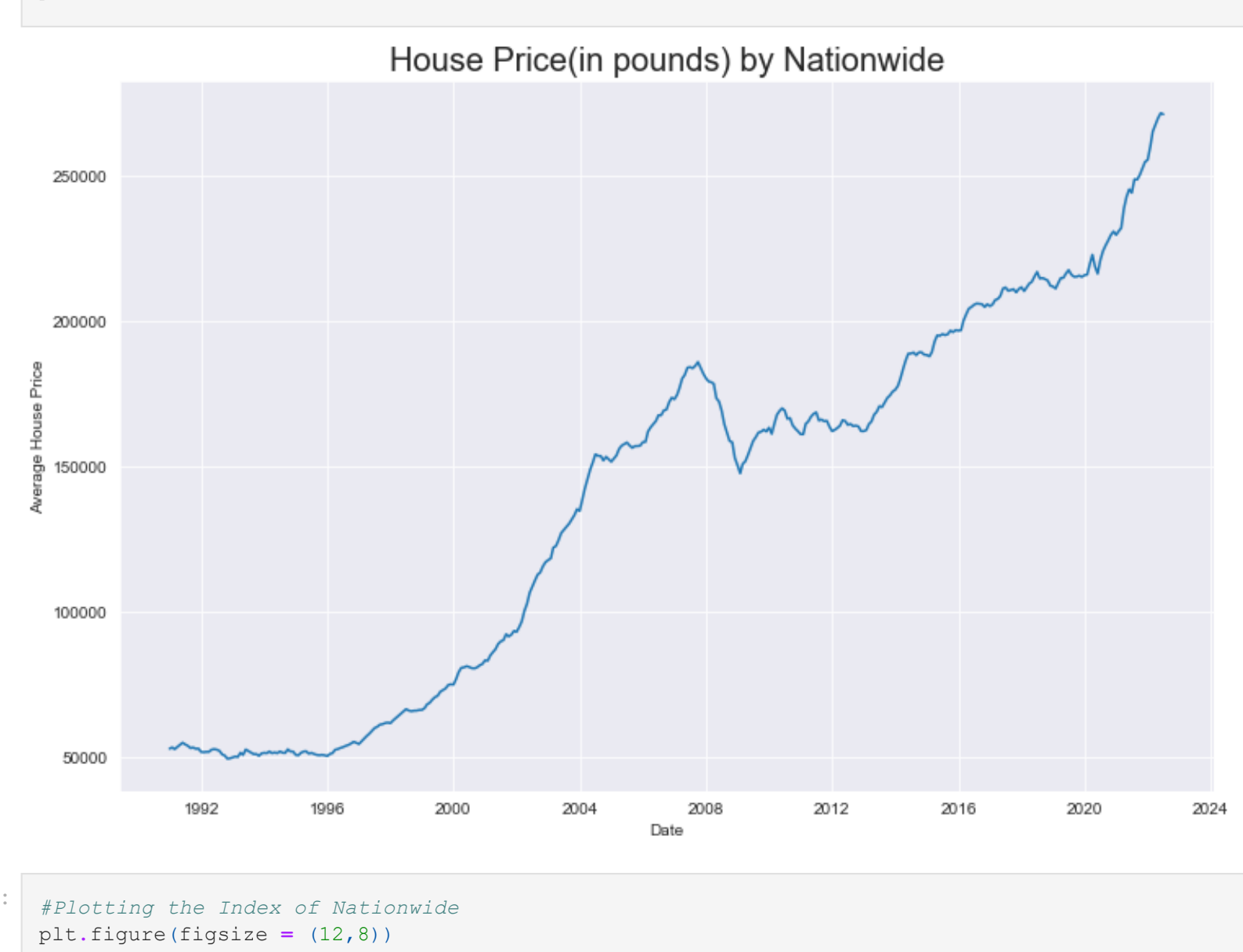
In [91]:

```
#to check the date range
date_list = list(df_nationwide['Date'])
print("The starting date of dataset is :",date_list[0])
print("The Ending date of dataset is :",date_list[-1])
```

```
The starting date of dataset is : 1991-01-01 00:00:00
The Ending date of dataset is : 2022-07-01 00:00:00
```

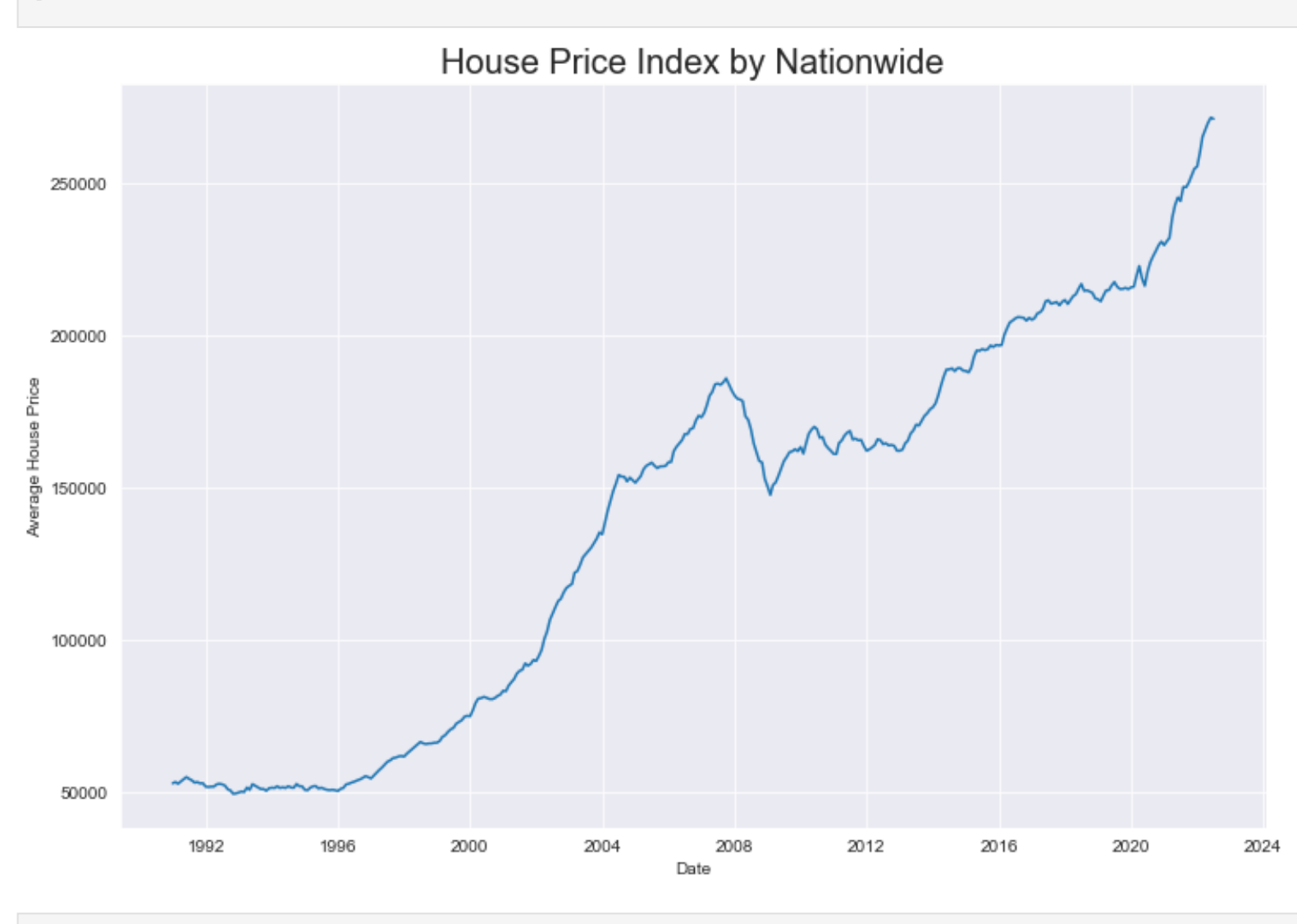
In [92]:

```
#Plotting the house price of nationwide

plt.figure(figsize = (12,8))
sns.lineplot(data = df_nationwide ,x='Date',y='Average House Price')
plt.title("House Price(in pounds) by Nationwide",fontdict={'fontsize': 20})
plt.show()
```



In [93]:

```
#Plotting the Index of Nationwide
plt.figure(figsize = (12,8))
sns.lineplot(data = df_nationwide ,x='Date',y='Average House Price')
plt.title("House Price Index by Nationwide",fontdict={'fontsize': 20})
plt.show()
```



In [94]:

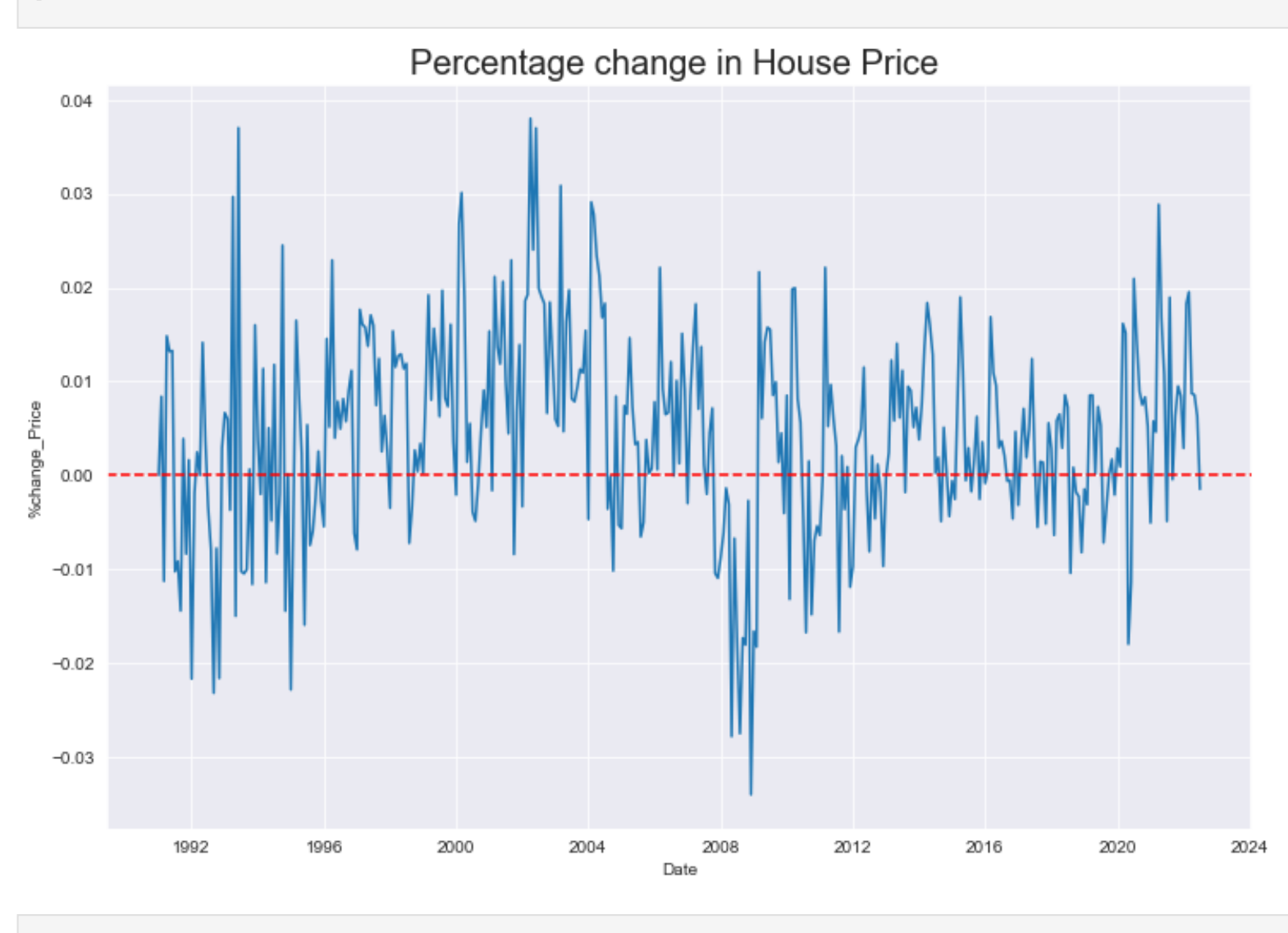
```
#calculating the percentage change in house price and index
df_nationwide['%change_HPI'] =df_nationwide['INDEX'].pct_change()
df_nationwide['%change_Price'] =df_nationwide['Average House Price'].pct_change()
df_nationwide.fillna(0,inplace=True)
df_nationwide.head()
```

Out[94]:

	Date	Average House Price	INDEX	%change_HPI	%change_Price
0	1991-01-01	53051.721106	105.832511	0.000000	0.000000
1	1991-02-01	53496.798746	106.720393	0.008390	0.008390
2	1991-03-01	52892.861606	105.515603	-0.011289	-0.011289
3	1991-04-01	53677.435270	107.080744	0.014833	0.014833
4	1991-05-01	54385.726747	108.493710	0.013195	0.013195

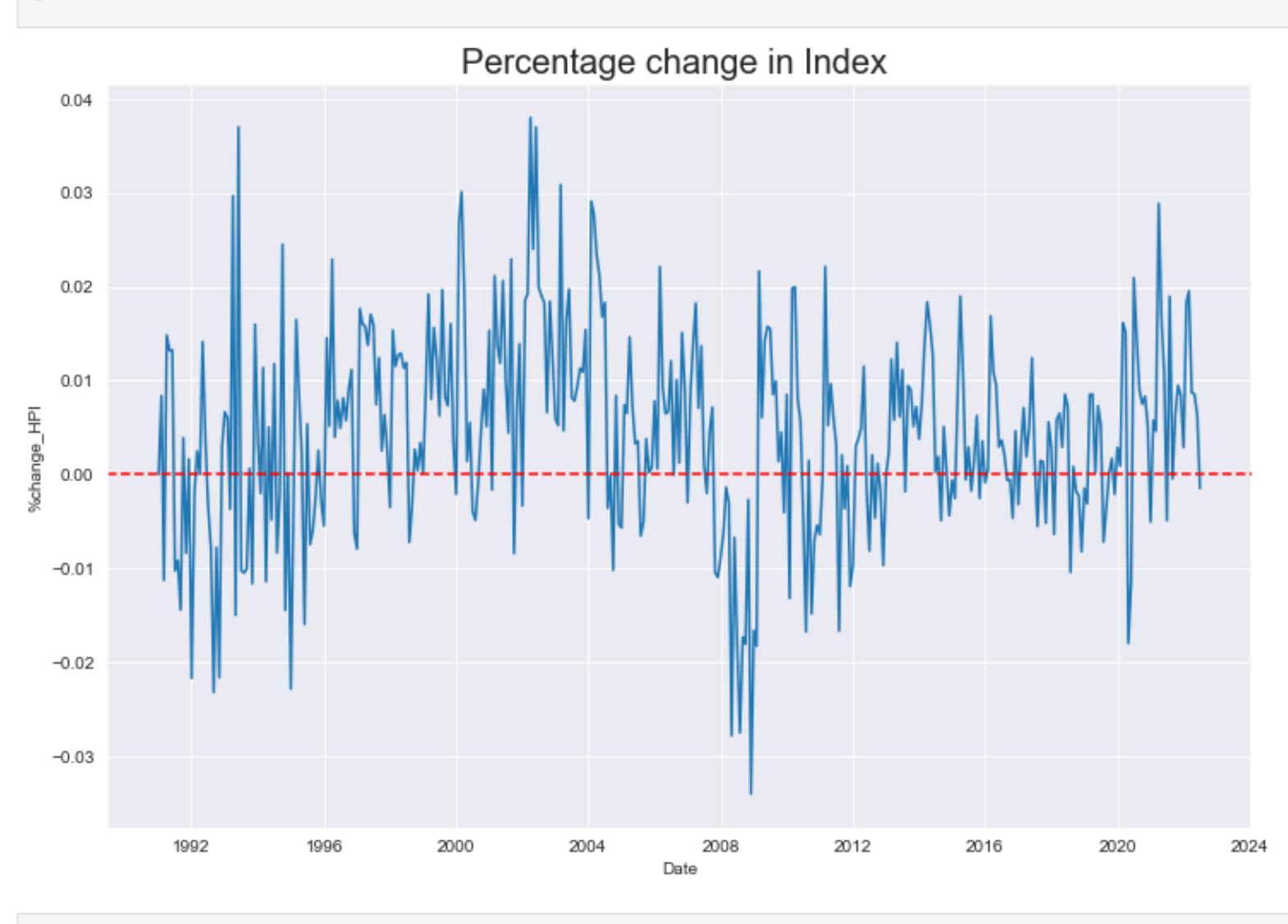
In [95]:

```
# plotting the percentage change in house prices
plt.figure(figsize = (12,8))
sns.lineplot(data=df_nationwide,x='Date',y='%change_Price')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in House Price ",fontdict={'fontsize': 20})
plt.show()
```



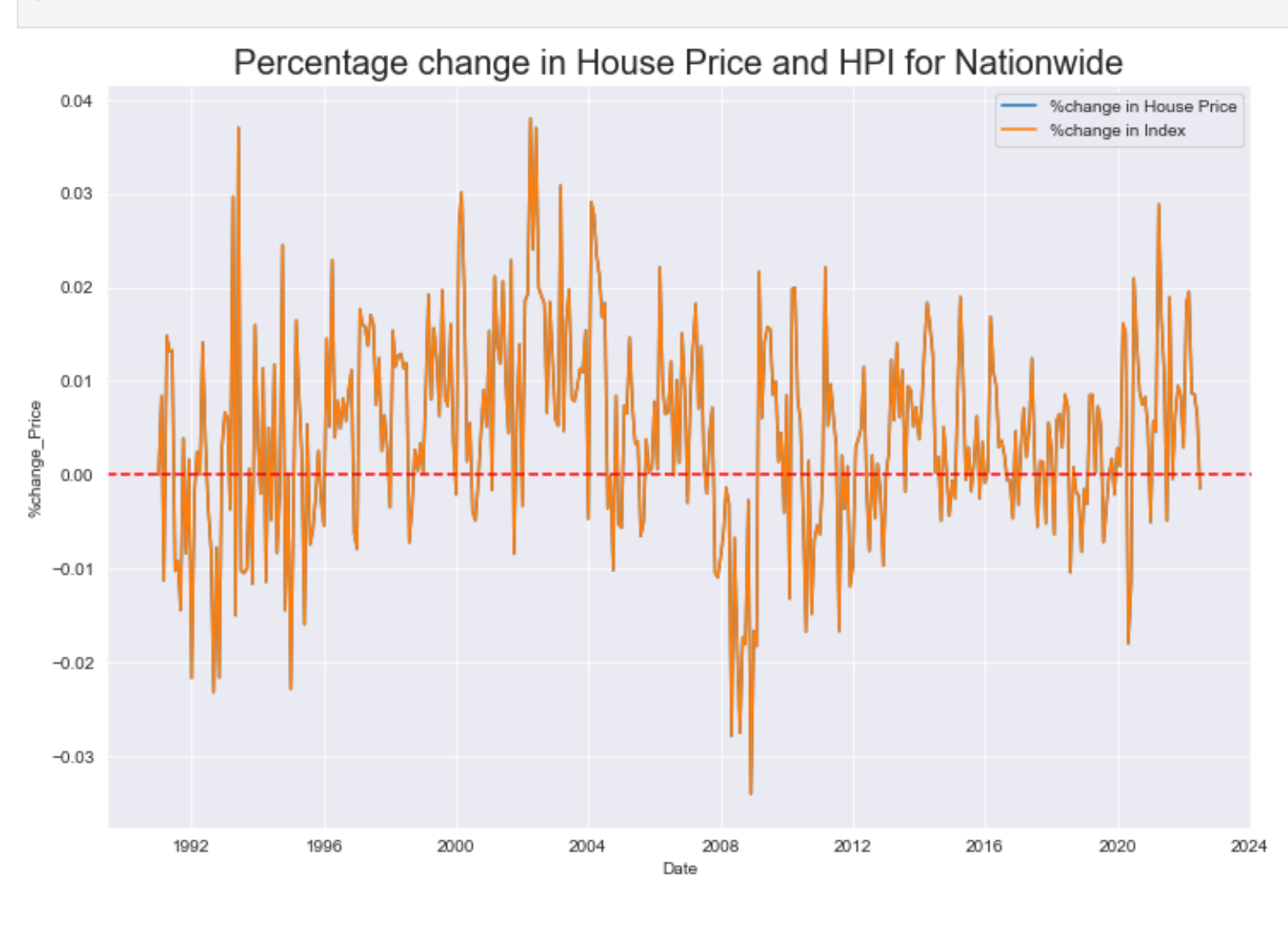
In [96]:

```
#plotting the percentage change in index
plt.figure(figsize = (12,8))
sns.lineplot(data=df_nationwide,x='Date',y='%change_HPI')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in Index ",fontdict={'fontsize': 20})
plt.show()
```



In [97]:

```
#plotting the percentage change together of both house price and Index
plt.figure(figsize = (12,8))
sns.lineplot(data=df_nationwide,x='Date',y='%change_Price',label='%change in House Price')
sns.lineplot(data=df_nationwide,x='Date',y='%change_HPI',label='%change in Index')
plt.axhline(0, ls='--',c='red')
plt.legend()
plt.title("Percentage change in House Price and HPI for Nationwide",fontdict={'fontsize': 20})
plt.show()
```



LSL Acad E&W (Chartered Surveyors) House Price Data

In [98]:

```
#reading the data
df_surveyors = pd.read_excel(r'indices_dataset\e.surv-Acadata-EW-HPI-National-and-Regional-Formatted-Data-from-
                            sheet_name='DATA',usecols='D:F',nrows=217,header=8)
df_surveyors
```

Out[98]:

	Month	Average House Price	Index
0	2005-01-01	189930.269310	194.627352
1	2005-02-01	191085.625651	195.719519
2	2005-03-01	190886.853827	195.655161
3	2005-04-01	190951.982629	195.753681
4	2005-05-01	190420.781021	195.194145
...	...	...	...
203	2021-12-01	353619.030454	338.249738
204	2022-01-01	357377.940073	341.845274
205	2022-02-01	366206.760484	350.290369
206	2022-03-01	368853.774168	352.822336
207	2022-04-01	372435.582968	356.248469

208 rows × 3 columns

In [99]:

```
##describing the dataset(checking the statistical information)
df_surveyors.describe()
```

Out[99]:

	Average House Price	Index
count	208.000000	208.000000
mean	256548.743381	255.690249
std	48198.023179	40.076397
min	189930.269310	194.627352
25%	220525.584800	224.776352
50%	234265.686038	240.139593
75%	303664.547252	292.142244
max	372435.582968	356.248469

We can see :
1. The House Price Max is 372,435 and Min is 189,930 with a standard deviation of 48,198.02 and mean as 256,548.74
2. Index Max is 356.24 and Min is 194.62 with a standard deviation of 40.07 and mean as 255.69

In [100…

```
#checking for any null values
df_surveyors.isna().sum()
```

Out[100…

```
Month                  0
Average House Price    0
Index                  0
dtype: int64
```

In [101…

```
#checking the dtypes of dataset
df_surveyors.dtypes
```

Out[101…

```
Month                  datetime64[ns]
Average House Price           float64
Index                         float64
dtype: object
```

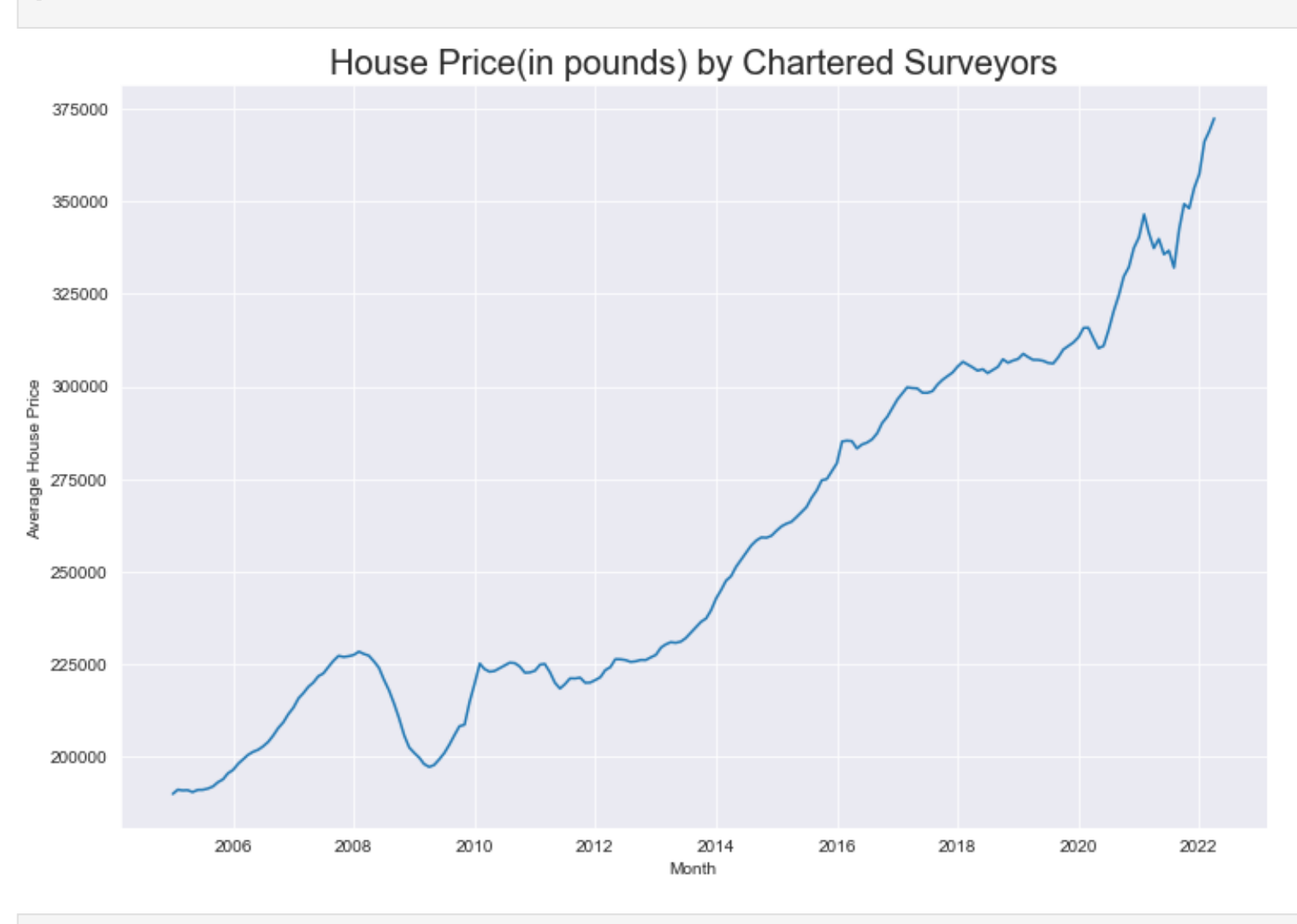
In [102…

```
#to check the date range
date_list = list(df_surveyors['Month'])
print("The starting date of dataset is :",date_list[0])
print("The Ending date of dataset is :",date_list[-1])
```

```
The starting date of dataset is : 2005-01-01 00:00:00
The Ending date of dataset is : 2022-04-01 00:00:00
```

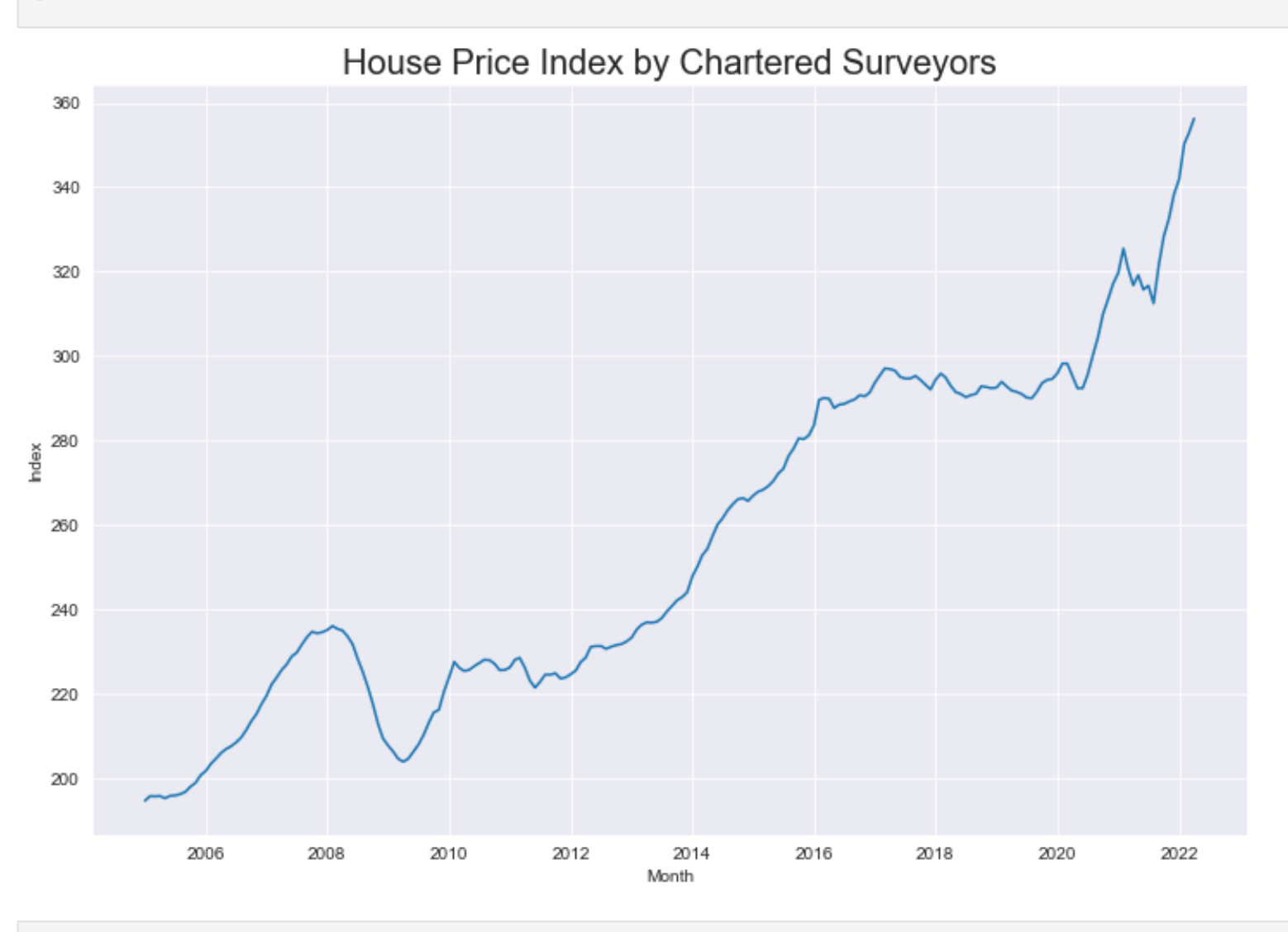
In [103…

```
#Plotting the house price of nationwide

plt.figure(figsize = (12,8))
sns.lineplot(data = df_surveyors ,x='Month',y='Average House Price ')
plt.title("House Price(in pounds) by Chartered Surveyors",fontdict={'fontsize': 20})
plt.show()
```



In [104…

```
#Plotting the Index of Nationwide
plt.figure(figsize = (12,8))
sns.lineplot(data = df_surveyors ,x='Month',y='Index')
plt.title("House Price Index by Chartered Surveyors",fontdict={'fontsize': 20})
plt.show()
```



In [105…

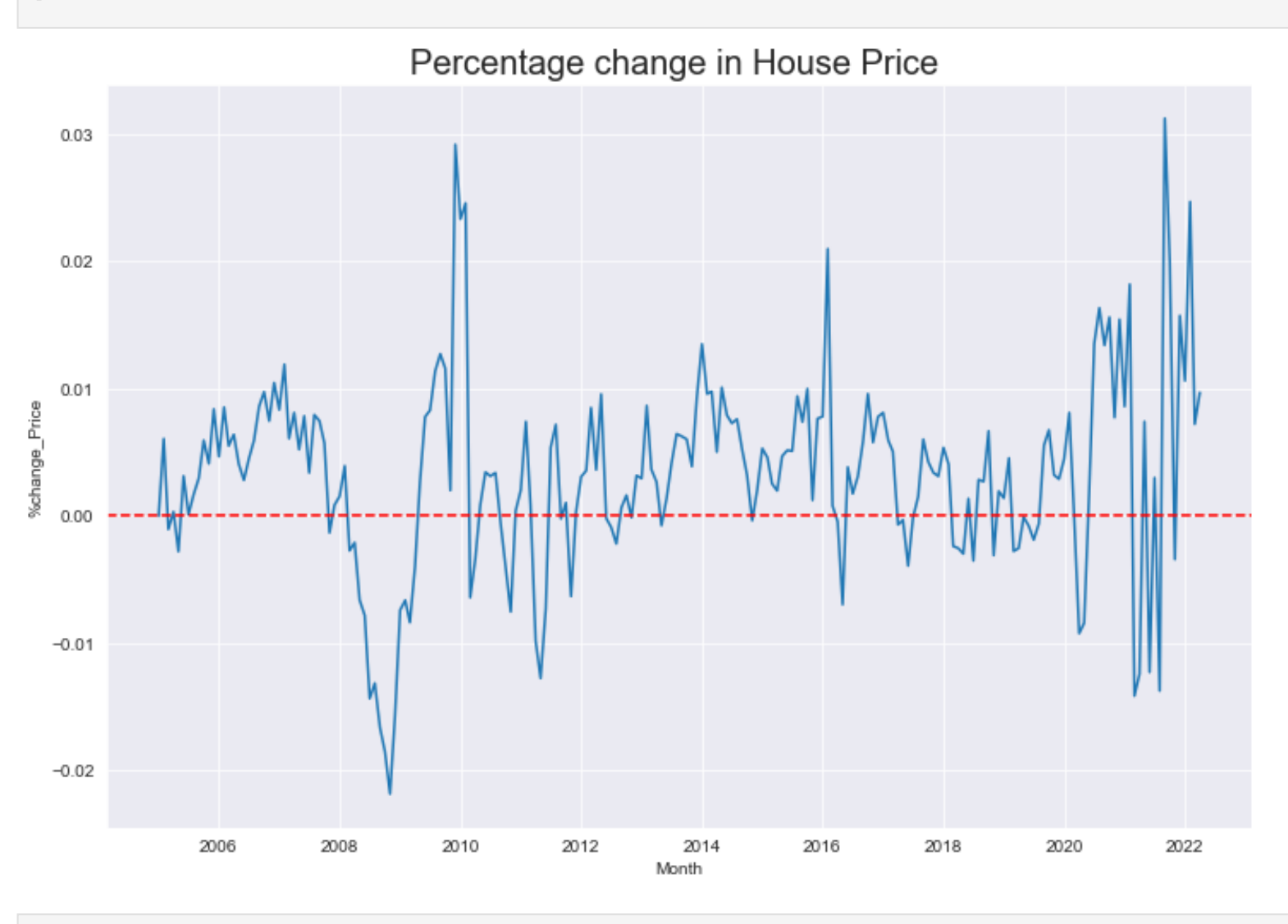
```
#calculating the percentage change in house price and index
df_surveyors['%change_HPI'] =df_surveyors['Index'].pct_change()
df_surveyors['%change_Price'] =df_surveyors['Average House Price '].pct_change()
df_surveyors.fillna(0,inplace=True)
df_surveyors.head()
```

Out[105…

	Month	Average House Price	Index	%change_HPI	%change_Price
0	2005-01-01	189930.269310	194.627352	0.000000	0.000000
1	2005-02-01	191085.625651	195.719519	0.005612	0.006083
2	2005-03-01	190886.853827	195.655161	-0.000329	-0.001040
3	2005-04-01	190951.982629	195.753681	0.000504	0.000341
4	2005-05-01	190420.781021	195.194145	-0.002858	-0.002782

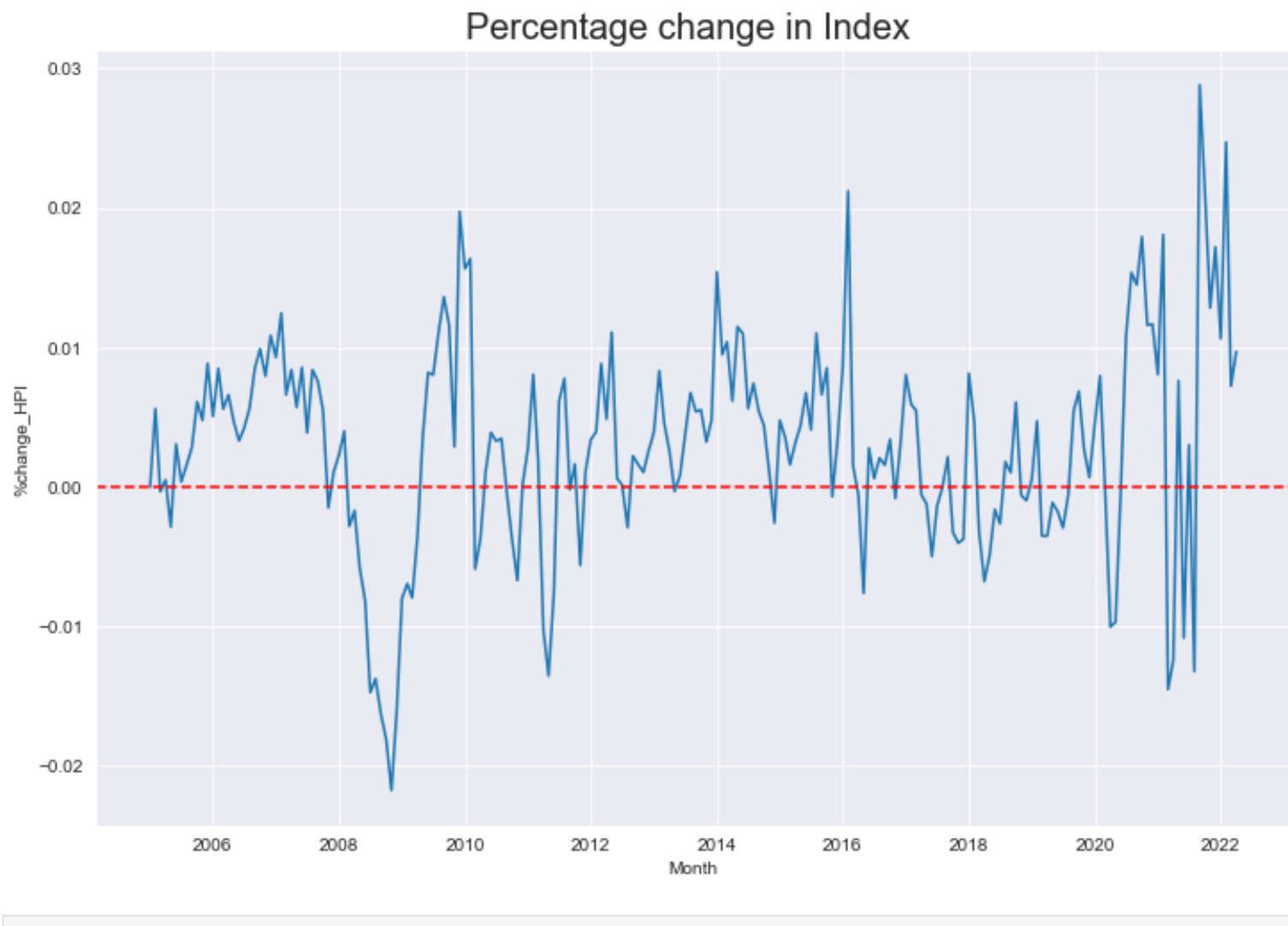
In [106…

```
# plotting the percentage change in house prices
plt.figure(figsize = (12,8))
sns.lineplot(data=df_surveyors,x='Month',y='%change_Price')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in House Price ",fontdict={'fontsize': 20})
plt.show()
```



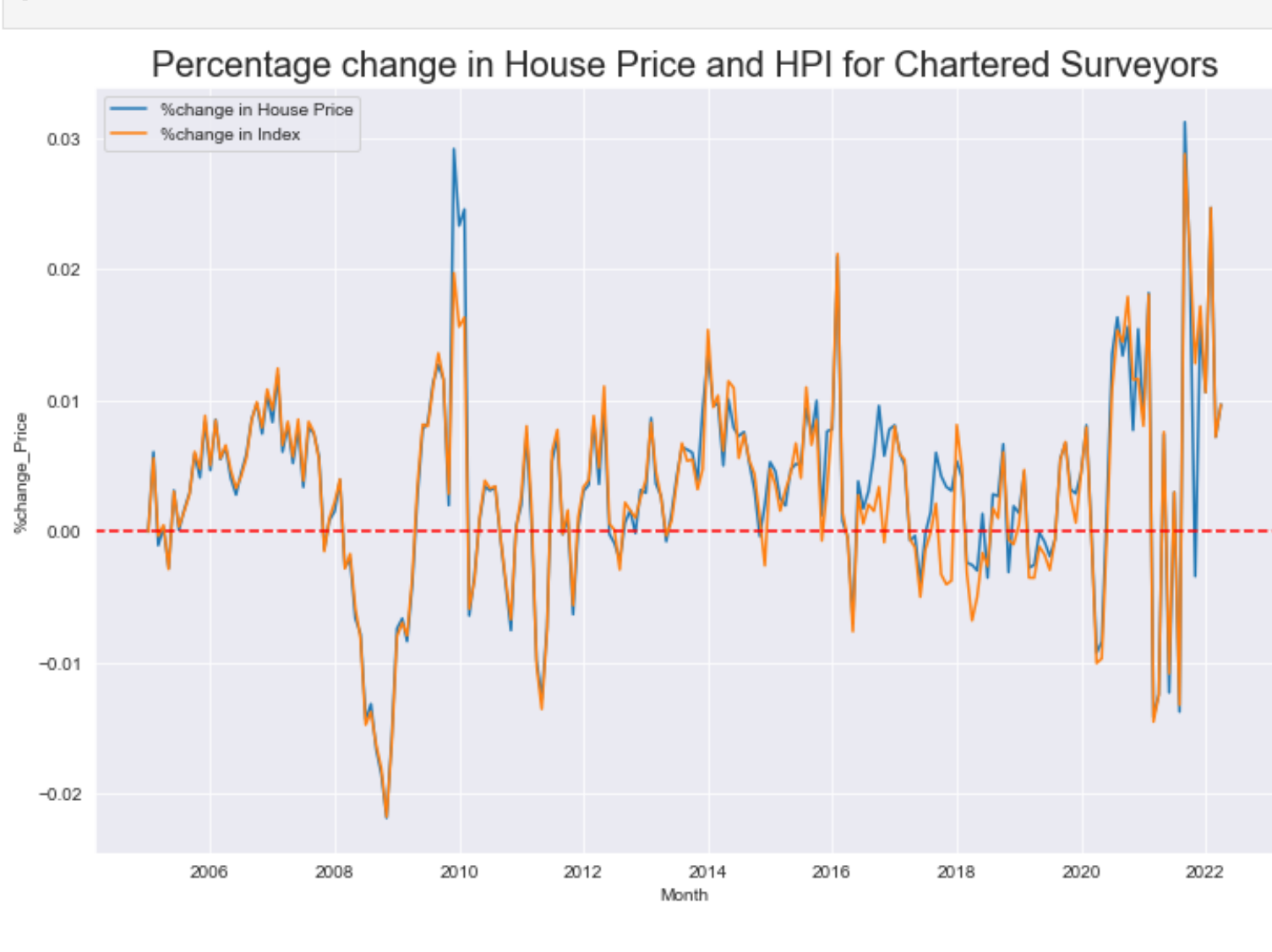
In [107…

```
#plotting the percentage change in index
plt.figure(figsize = (12,8))
sns.lineplot(data=df_surveyors,x='Month',y='%change_HPI')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in Index ",fontdict={'fontsize': 20})
plt.show()
```



In [108…

```
#plotting the percentage change together of both house price and Index
plt.figure(figsize = (12,8))
sns.lineplot(data=df_surveyors,x='Month',y='%change_Price',label='%change in House Price')
sns.lineplot(data=df_surveyors,x='Month',y='%change_HPI',label='%change in Index')
plt.axhline(0, ls='--',c='red')
plt.legend()
plt.title("Percentage change in House Price and HPI for Chartered Surveyors",fontdict={'fontsize': 20})
plt.show()
```



Land Registry House Price Data

In [109…

```
# importing the dataset for house prices
df_LR = pd.read_csv(r'indices_dataset/UK-HPI-full-file-2022-05_land_registry.csv')
df_LR.head()
```

Out[109…

	Date	RegionName	AreaCode	AveragePrice	Index	IndexSA	1m%Change	12m%Change	AveragePriceSA	SalesVolume	...	Nev
0	1/1/2004	Aberdeenshire	S12000034	81693.66964	40.864214	NaN	NaN	NaN	NaN	388.0	...	88436.
1	1/2/2004	Aberdeenshire	S12000034	81678.76231	40.856757	NaN	-0.018248	NaN	NaN	326.0	...	88606.
2	1/3/2004	Aberdeenshire	S12000034	83525.09702	41.780317	NaN	2.260483	NaN	NaN	453.0	...	90296.
3	1/4/2004	Aberdeenshire	S12000034	84333.67900	42.184780	NaN	0.968071	NaN	NaN	571.0	...	90319.
4	1/5/2004	Aberdeenshire	S12000034	86379.95396	43.208353	NaN	2.426403	NaN	NaN	502.0	...	91989.

5 rows × 54 columns

In [110…

```
#since the region_name contains all the regions, we would take the region_name as United Kingdom
df_LR = df_LR[df_LR.RegionName=='United Kingdom']
#we require only the date and average_price
df_LR = df_LR[['Date','Index','AveragePrice']]
df_LR
```

Out[110…

	Date	Index	AveragePrice
120732	1/4/1968	1.885299	3594.602239
120733	1/5/1968	1.885299	3594.602239
120734	1/6/1968	1.885299	3594.602239
120735	1/7/1968	1.949207	3716.453163
120736	1/8/1968	1.949207	3716.453163
...	...	...	...
121377	1/1/2022	143.422582	273456.582300
121378	1/2/2022	144.646684	275790.515100
121379	1/3/2022	145.556596	277525.398100
121380	1/4/2022	146.898652	280084.228600
121381	1/5/2022	148.688246	283496.357300

650 rows × 3 columns

In [111…

```
#describing the dataset(checking the statistical information)
df_LR.describe()
```

Out[111…

	Index	AveragePrice
count	650.000000	650.000000
mean	48.610337	92682.855832
std	41.736355	79576.598379
min	1.885299	3594.602239
25%	11.279839	21506.687970
50%	30.550845	58249.724790
75%	88.052504	167885.249350
max	148.688246	283496.357300

We can see :
1. The House Price Max is 283,496 and Min is 3,594 with a standard deviation of 79576.59 and mean as 92682.85
2. Index Max is 1.88 and Min is 148.68 with a standard deviation of 41.73 and mean as 48.61

In [112…

```
#checking the data types of variables
df_LR.dtypes
```

Out[112…

```
Date             object
Index           float64
AveragePrice    float64
dtype: object
```

In [113…

```
#converting the date object to different datetime format
df_LR['Date']=df_LR['Date'].astype('datetime64[ns]').dt.strftime('%Y-%d-%m')
```

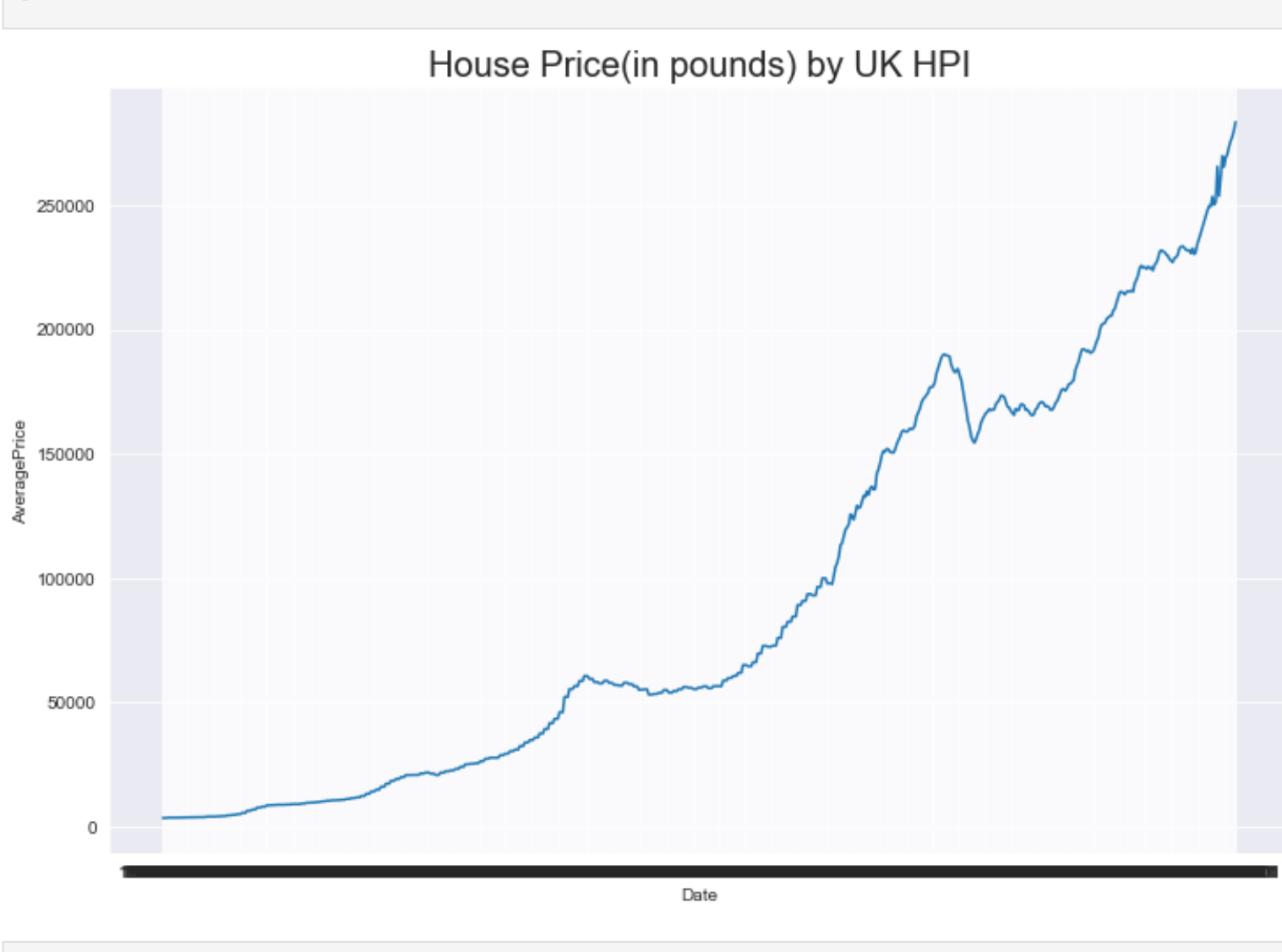
In [114…

```
#to check the date range
date_list = list(df_LR['Date'])
print("The starting date of dataset is :",date_list[0])
print("The Ending date of dataset is :",date_list[-1])
```

```
The starting date of dataset is : 1968-04-01
The Ending date of dataset is : 2022-05-01
```

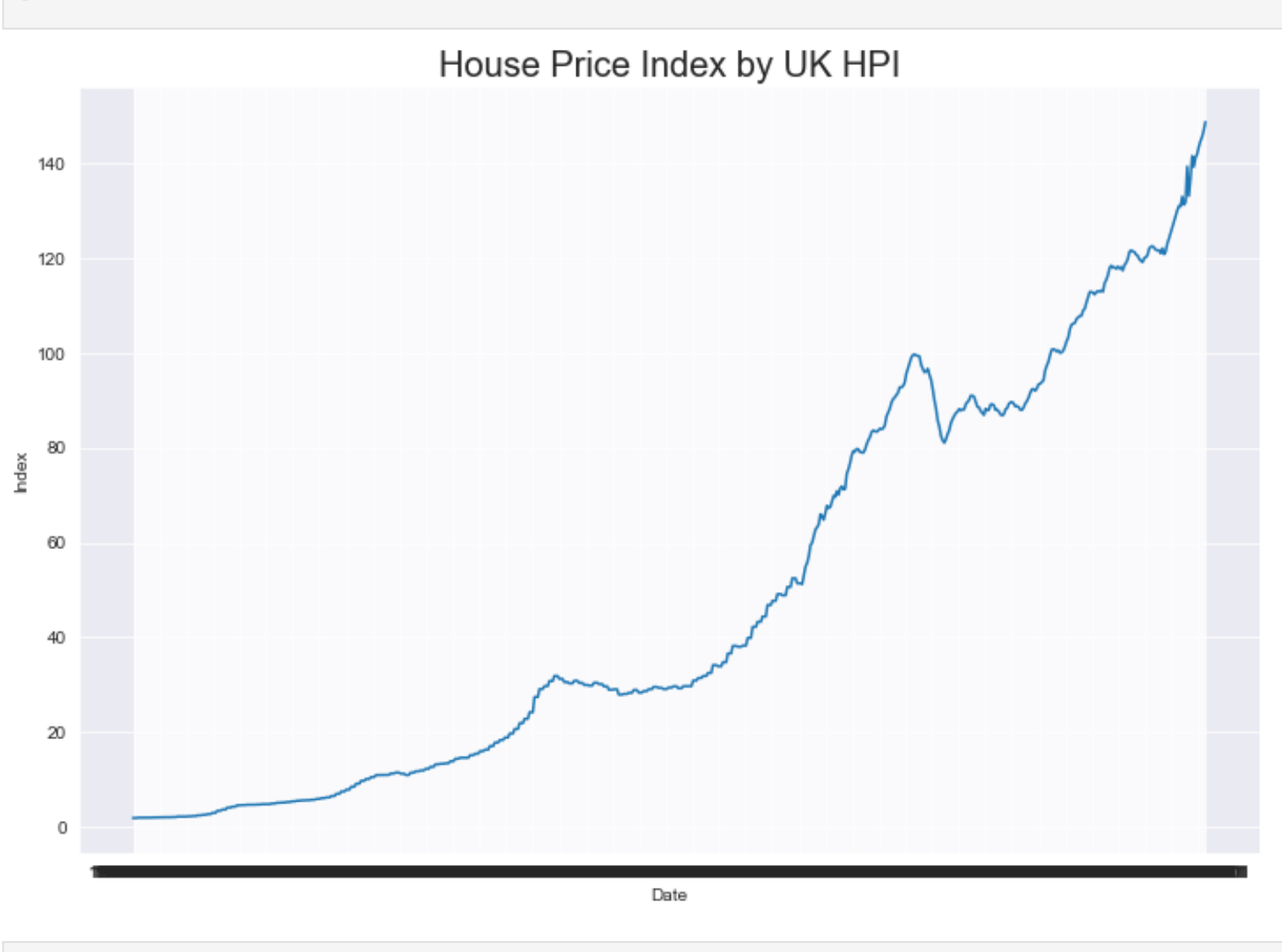
In [115…

```
#Plotting the house price of Land Registry

plt.figure(figsize = (12,8))
sns.lineplot(data = df_LR ,x='Date',y='AveragePrice')
plt.title("House Price(in pounds) by UK HPI",fontdict={'fontsize': 20})
plt.show()
```



In [116…

```
#Plotting the Index of Land Registry
plt.figure(figsize = (12,8))
sns.lineplot(data = df_LR ,x='Date',y='Index')
plt.title("House Price Index by UK HPI",fontdict={'fontsize': 20})
plt.show()
```



In [117…

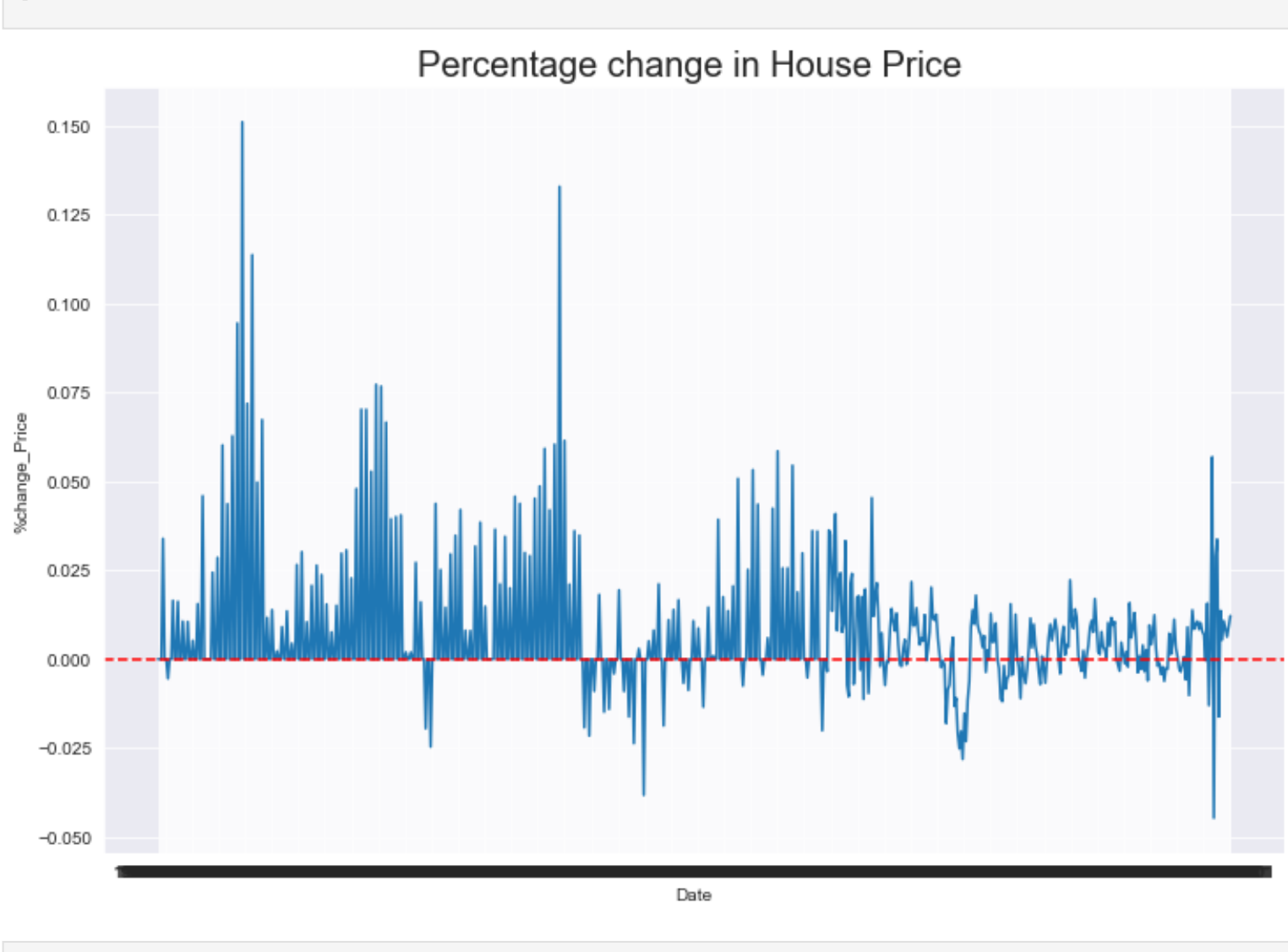
```
#calculating the percentage change in house price and index
df_LR['%change_HPI'] =df_LR['Index'].pct_change()
df_LR['%change_Price'] =df_LR['AveragePrice'].pct_change()
df_LR.fillna(0,inplace=True)
df_LR.head()
```

Out[117…

	Date	Index	AveragePrice	%change_HPI	%change_Price
120732	1968-04-01	1.885299	3594.602239	0.000000	0.000000
120733	1968-05-01	1.885299	3594.602239	0.000000	0.000000
120734	1968-06-01	1.885299	3594.602239	0.000000	0.000000
120735	1968-07-01	1.949207	3716.453163	0.033898	0.033898
120736	1968-08-01	1.949207	3716.453163	0.000000	0.000000

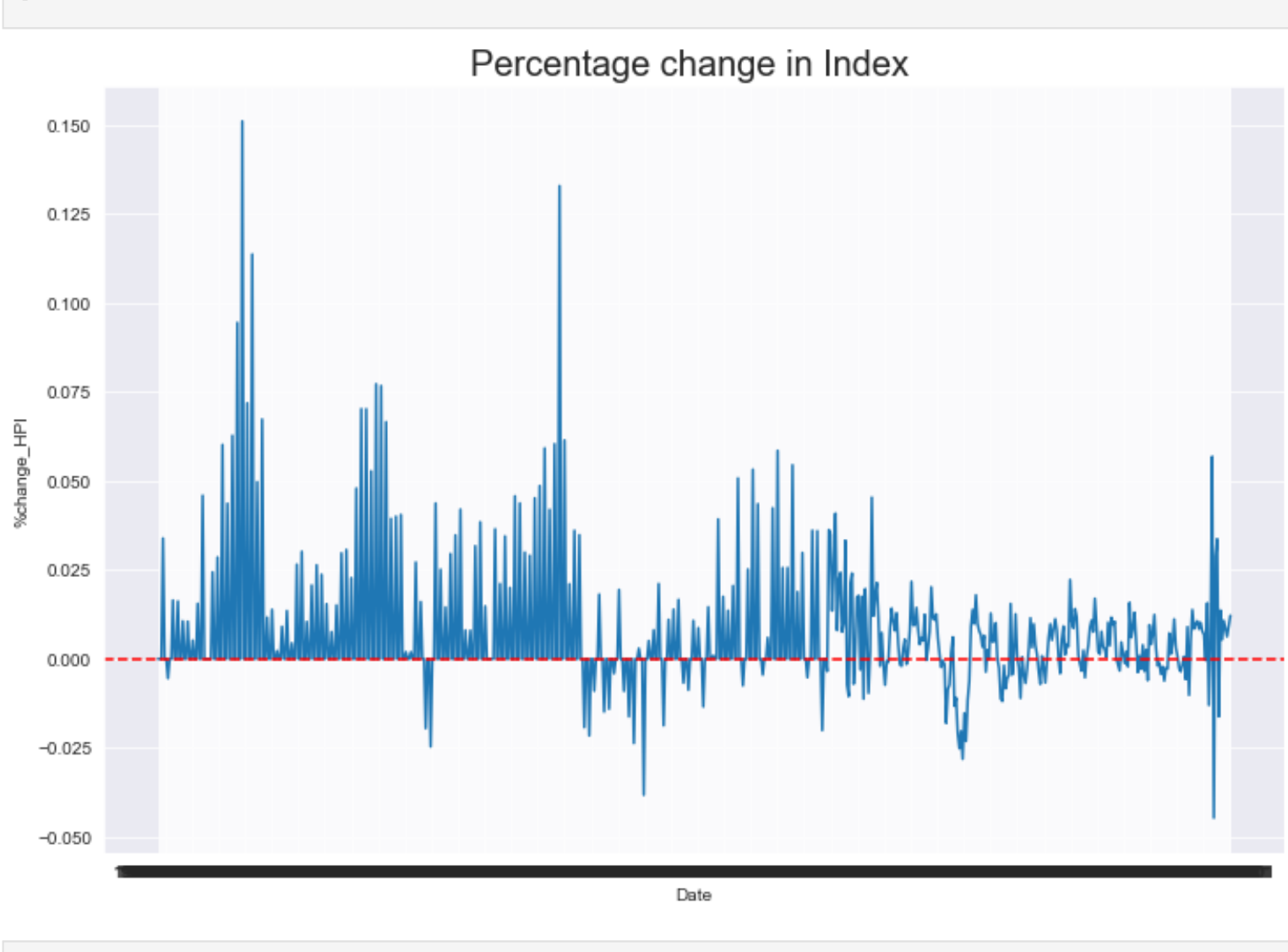
In [118…

```
# plotting the percentage change in house prices
plt.figure(figsize = (12,8))
sns.lineplot(data=df_LR,x='Date',y='%change_Price')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in House Price ",fontdict={'fontsize': 20})
plt.show()
```



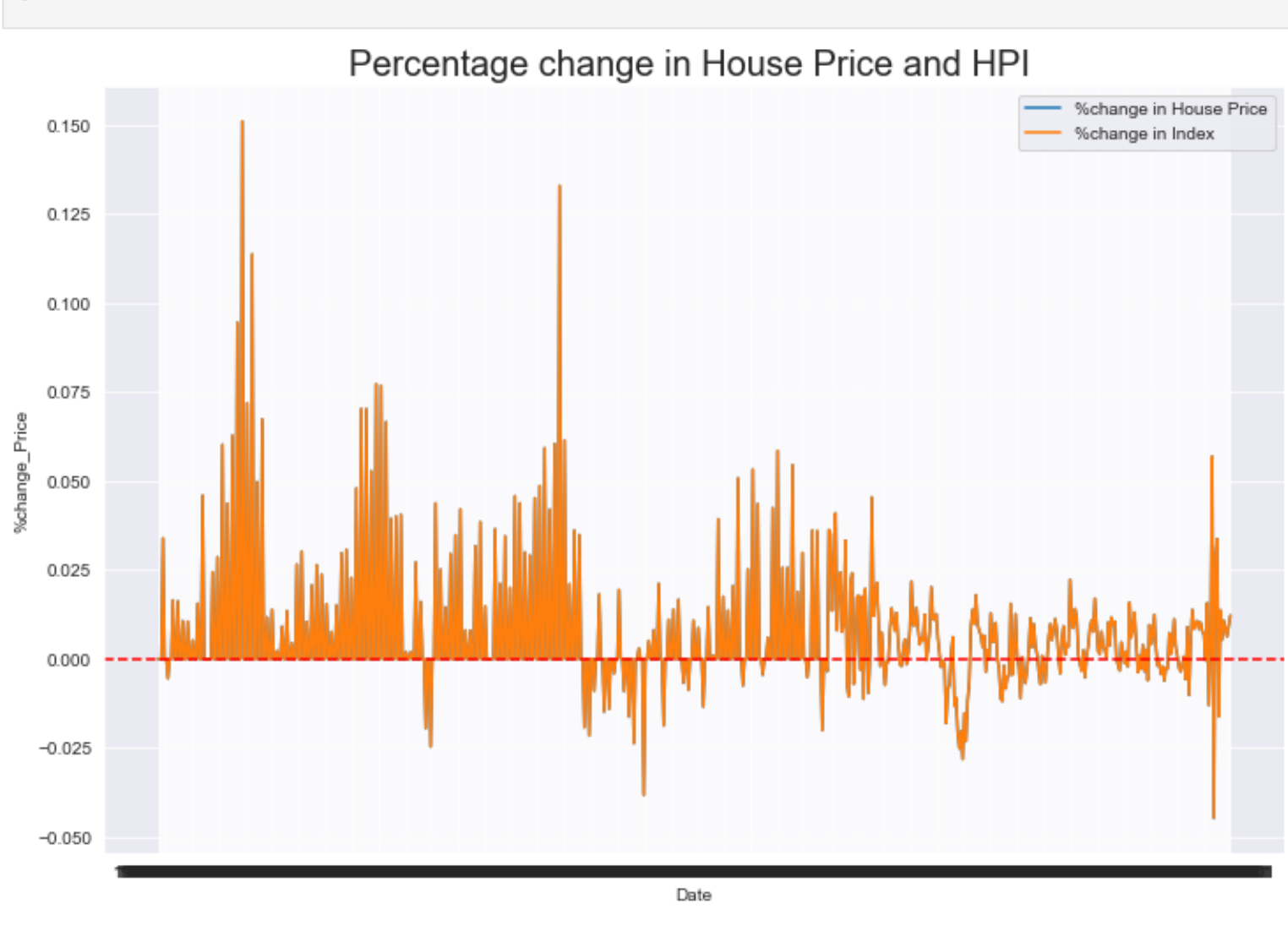
In [119…

```
#plotting the percentage change in index
plt.figure(figsize = (12,8))
sns.lineplot(data=df_LR,x='Date',y='%change_HPI')
plt.axhline(0, ls='--',c='red')
plt.title("Percentage change in Index ",fontdict={'fontsize': 20})
plt.show()
```



In [120…

```
#plotting the percentage change together of both house price and Index
plt.figure(figsize = (12,8))
sns.lineplot(data=df_LR,x='Date',y='%change_Price',label='%change in House Price')
sns.lineplot(data=df_LR,x='Date',y='%change_HPI',label='%change in Index')
plt.axhline(0, ls='--',c='red')
plt.legend()
plt.title("Percentage change in House Price and HPI ",fontdict={'fontsize': 20})
plt.show()
```



Comparing all the indices together

In [184…

```
#since the dates of the all datasets are different , thus , taking dates only from 2019 to 2022
START = '2019-01-01'
END = '2022-04-01'
```

In [202…

```
# adding halifax data into the merged dataset
df_all = pd.DataFrame()
df_all[['Date','Halifax_Price','Halifax_Index','Halifax_%change_HPI','Halifax_%change_Price']] = df_halifax
df_all.set_index('Date',inplace=True)
df_all =  df_all.loc[START:END]
df_all.head()
```

Out[202…

	Halifax_Price	Halifax_Index	Halifax_%change_HPI	Halifax_%change_Price
Date				
2019-01-01	223691	397.8	0.000000	0.000000
2019-02-01	236800	403.8	0.015083	0.058603
2019-03-01	233181	402.6	-0.002972	-0.015283
2019-04-01	236619	400.3	-0.005713	0.014744
2019-05-01	237837	399.7	-0.001499	0.005148

In [203…

```
# adding the nationwide data into the merged dataset
temp =df_nationwide[(df_nationwide['Date']>=START) & (df_nationwide['Date']<=END)]
df_all[['Nationwide_Price','Nationwide_Index','Nationwide_%change_HPI','Nationwide_%change_Price']]=temp[['Aver
df_all.head()
```

Out[203…

	Halifax_Price	Halifax_Index	Halifax_%change_HPI	Halifax_%change_Price	Nationwide_Price	Nationwide_Index	Nationwide_%change_HPI
Date							
2019-01-01	223691	397.8	0.000000	0.000000	211966.181814	422.849868	-0.001485
2019-02-01	236800	403.8	0.015083	0.058603	211303.681381	421.528250	-0.003126
2019-03-01	233181	402.6	-0.002972	-0.015283	213102.320257	425.116343	0.008512
2019-04-01	236619	400.3	-0.005713	0.014744	214919.604264	428.741630	0.008528
2019-05-01	237837	399.7	-0.001499	0.005148	214945.636469	428.793561	0.000121

In [204…

```
# adding the LSL data into the merged dataset
temp =df_surveyors[(df_surveyors['Month']>=START) & (df_surveyors['Month']<=END)]
df_all[['LSL_Price','LSL_Index','LSL_%change_HPI','LSL_%change_Price']]=temp[['Average House Price ','Index','%
df_all.head()
```

Out[204…

	Halifax_Price	Halifax_Index	Halifax_%change_HPI	Halifax_%change_Price	Nationwide_Price	Nationwide_Index	Nationwide_%change_HPI
Date							
2019-01-01	223691	397.8	0.000000	0.000000	211966.181814	422.849868	-0.001485
2019-02-01	236800	403.8	0.015083	0.058603	211303.681381	421.528250	-0.003126
2019-03-01	233181	402.6	-0.002972	-0.015283	213102.320257	425.116343	0.008512
2019-04-01	236619	400.3	-0.005713	0.014744	214919.604264	428.741630	0.008528
2019-05-01	237837	399.7	-0.001499	0.005148	214945.636469	428.793561	0.000121

In [208…

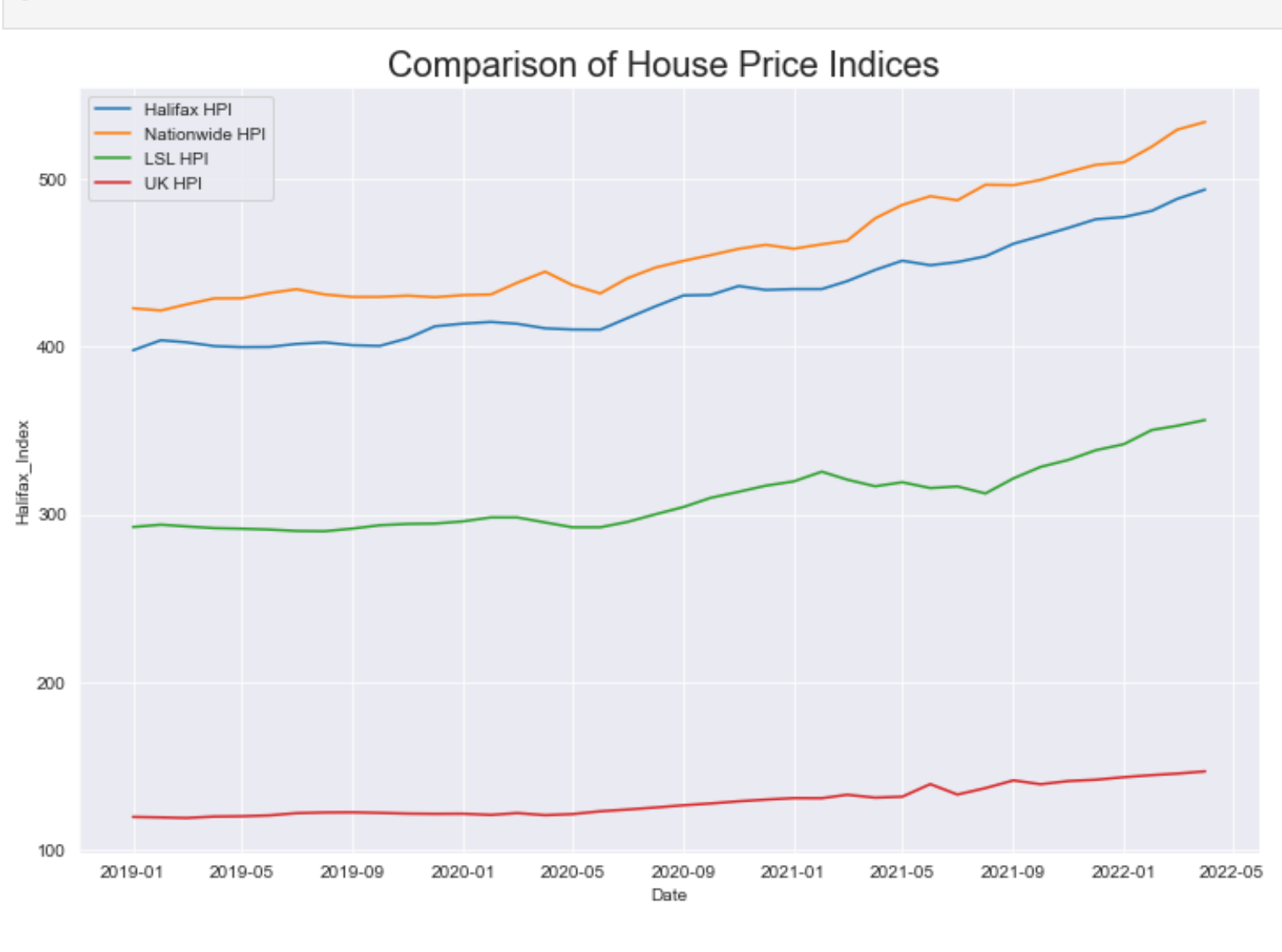
```
# adding the Land Registry data into the merged dataset
temp =df_LR[(df_LR['Date']>=START) & (df_LR['Date']<=END)]
df_all[['LR_Price','LR_Index','LR_%change_HPI','LR_%change_Price']]=temp[['AveragePrice','Index','%change_HPI',
df_all.head()
```

Out[208…

	Halifax_Price	Halifax_Index	Halifax_%change_HPI	Halifax_%change_Price	Nationwide_Price	Nationwide_Index	Nationwide_%change_HPI
Date							
2019-01-01	223691	397.8	0.000000	0.000000	211966.181814	422.849868	-0.001485
2019-02-01	236800	403.8	0.015083	0.058603	211303.681381	421.528250	-0.003126
2019-03-01	233181	402.6	-0.002972	-0.015283	213102.320257	425.116343	0.008512
2019-04-01	236619	400.3	-0.005713	0.014744	214919.604264	428.741630	0.008528
2019-05-01	237837	399.7	-0.001499	0.005148	214945.636469	428.793561	0.000121

In [214…

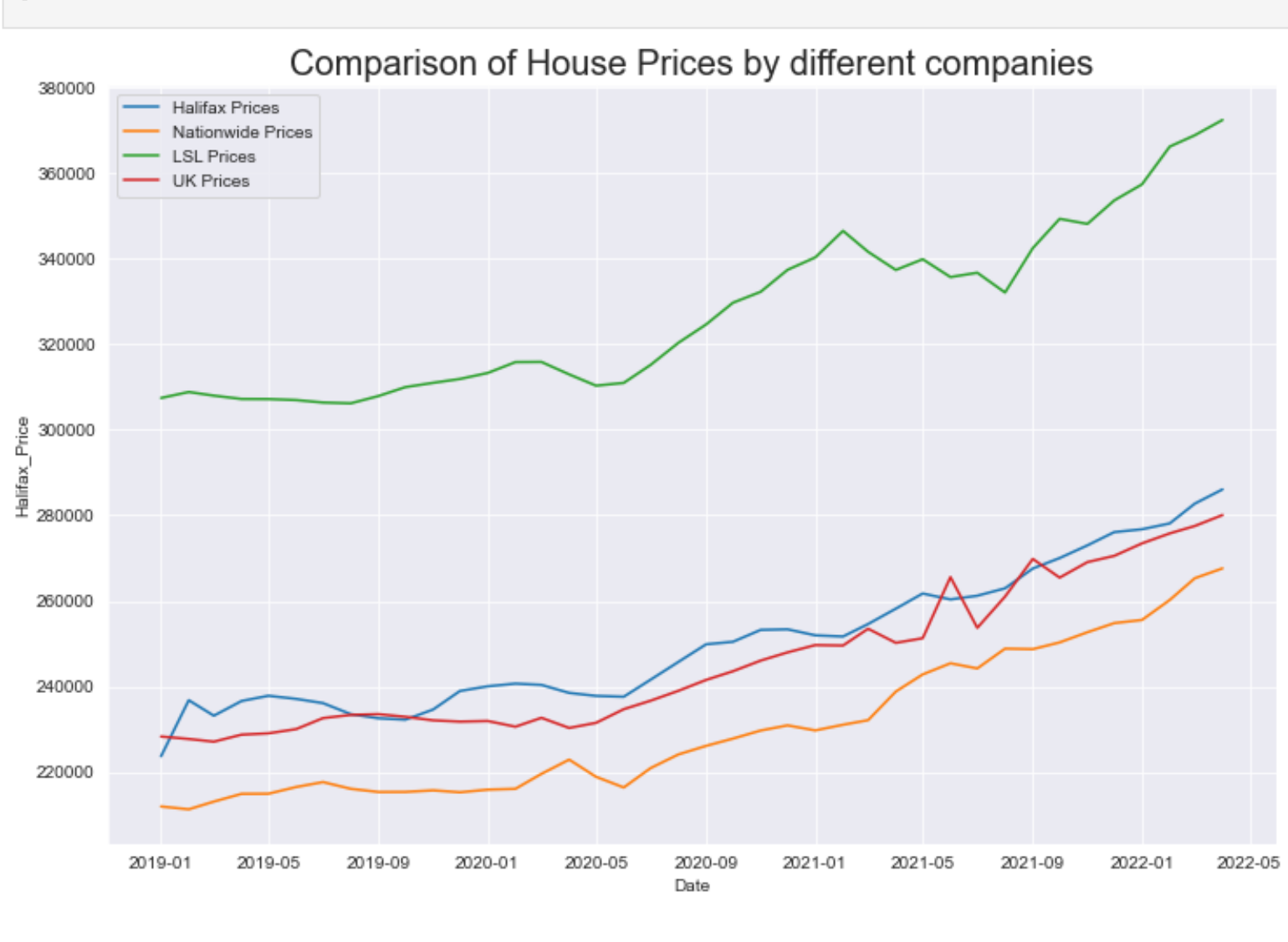
```
#plotting the house prices index by different companies together

plt.figure(figsize = (12,8))
sns.lineplot(data = df_all ,x='Date',y='Halifax_Index', label ='Halifax HPI')
sns.lineplot(data = df_all ,x='Date',y='Nationwide_Index', label ='Nationwide HPI')
sns.lineplot(data = df_all ,x='Date',y='LSL_Index', label ='LSL HPI')
sns.lineplot(data = df_all ,x='Date',y='LR_Index', label ='UK HPI')
plt.title("Comparison of House Price Indices ",fontdict={'fontsize': 20})
plt.show()
```



We can see in the above chart that the UK HPI (Land Registry) is the lowest while Nationwide HPI is the highest

In [217…

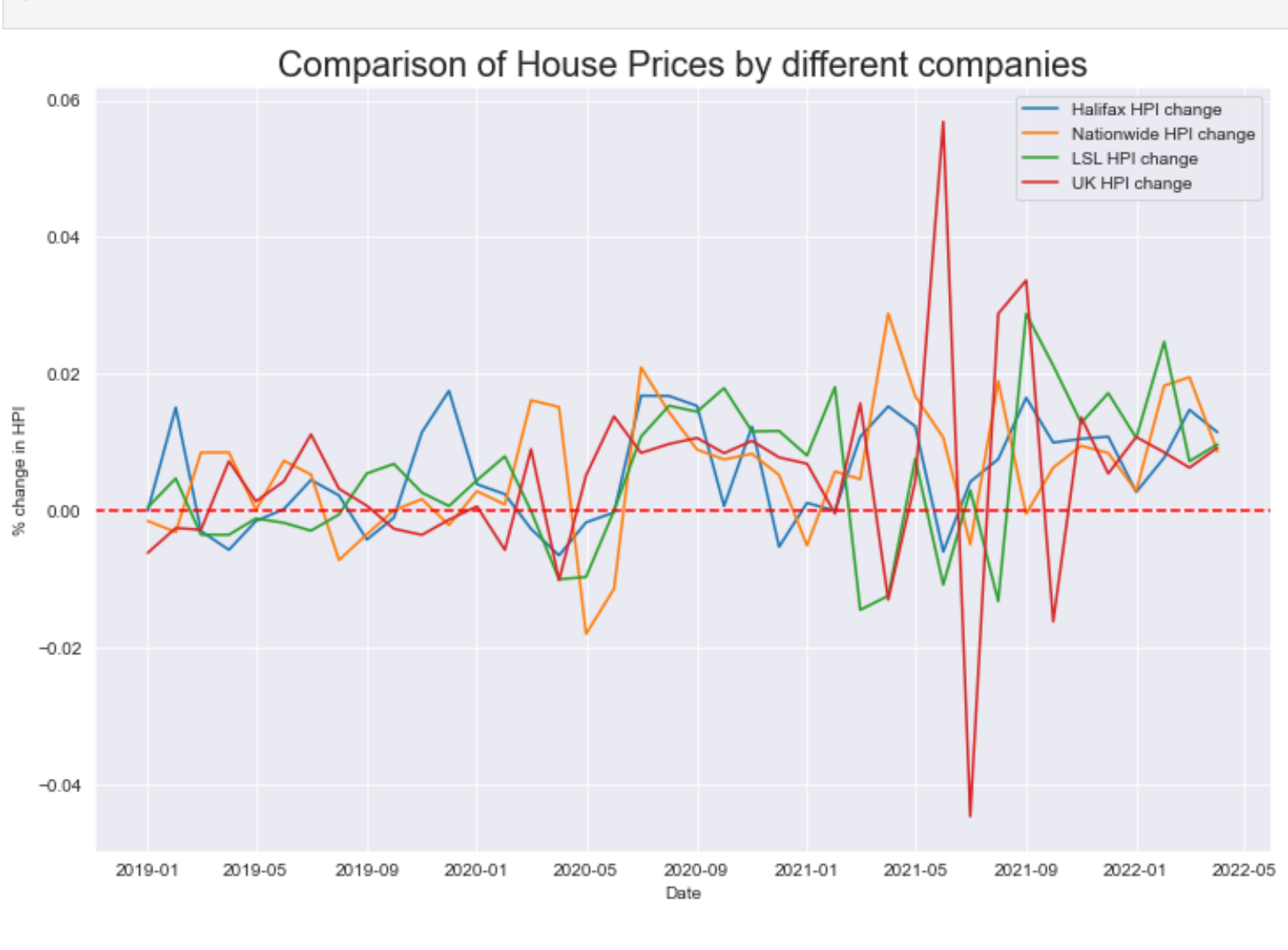
```
#plotting the house prices by different companies together

plt.figure(figsize = (12,8))
sns.lineplot(data = df_all ,x='Date',y='Halifax_Price', label ='Halifax Prices')
sns.lineplot(data = df_all ,x='Date',y='Nationwide_Price', label ='Nationwide Prices')
sns.lineplot(data = df_all ,x='Date',y='LSL_Price', label ='LSL Prices')
sns.lineplot(data = df_all ,x='Date',y='LR_Price', label ='UK Prices')
plt.title("Comparison of House Prices by different companies",fontdict={'fontsize': 20})
plt.show()
```



We can see in the above chart that the LSL Prices(Land Surveyors) is the highest while Nationwide HPI is the lowest.

In [239…

```
#plotting the %change in HPI

plt.figure(figsize = (12,8))
sns.lineplot(data = df_all ,x='Date',y='Halifax_%change_HPI', label ='Halifax HPI change')
sns.lineplot(data = df_all ,x='Date',y='Nationwide_%change_HPI', label ='Nationwide HPI change')
sns.lineplot(data = df_all ,x='Date',y='LSL_%change_HPI', label ='LSL HPI change')
sns.lineplot(data = df_all ,x='Date',y='LR_%change_HPI', label ='UK HPI change')
plt.axhline(0, ls='--',c='red')
plt.xlabel("Date")
plt.ylabel("% change in HPI")
plt.title("Comparison of House Prices by different companies",fontdict={'fontsize': 20})
plt.show()
```



We can see there is a huge % fluctuation in UK HPI , during year 2021, may be it is due to COVID-19 situation
Moreover , we can see that fluctuation for all indices is around 0.02

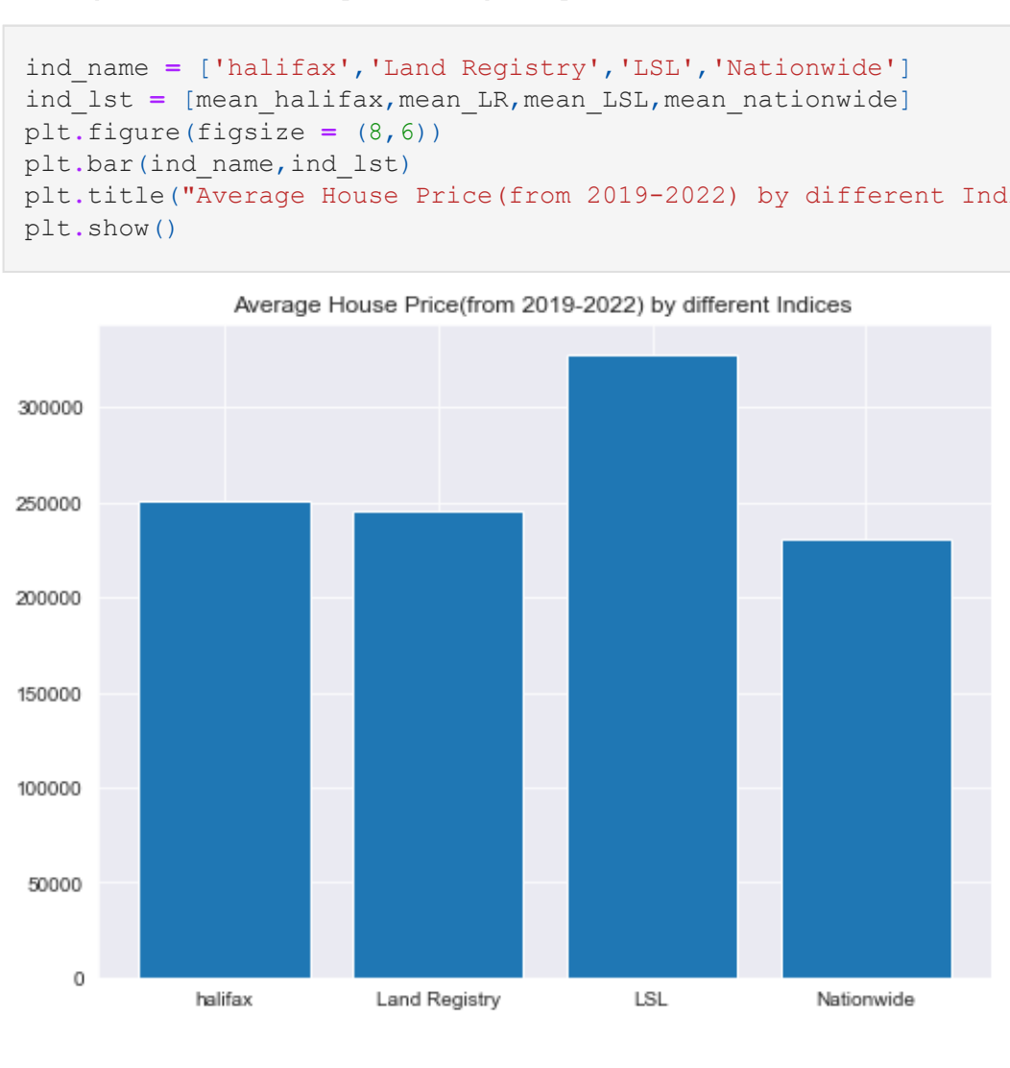
In [235…

```
# calculating the average house prices from year 2019
mean_halifax = df_all['Halifax_Price'].mean()
print("Average House Price by Halifax",mean_halifax)
mean_nationwide = df_all['Nationwide_Price'].mean()
print("Average House Price by Nationwide",mean_nationwide)
mean_LSL = df_all['LSL_Price'].mean()
print("Average House Price by LSL",mean_LSL)
mean_LR = df_all['LR_Price'].mean()
print("Average House Price by Land Registry",mean_LR)
```

```
Average House Price by Halifax 250395.425
Average House Price by Nationwide 230402.65679764157
Average House Price by LSL 327657.8222333012
Average House Price by Land Registry 245757.55771499994
```

In [236…

```
ind_name = ['halifax','Land Registry','LSL','Nationwide']
ind_lst = [mean_halifax,mean_LR,mean_LSL,mean_nationwide]
plt.figure(figsize = (8,6))
plt.bar(ind_name,ind_lst)
plt.title("Average House Price(from 2019-2022) by different Indices ")
plt.show()
```



we can see in the above chart that the average house prices is the highest in LSL whereas lowest in Nationwide

Table1. UK house price indices summarised

Index	Data	Coverage (property type)	Geographical coverage	Time in house-sale cycle	Adjustment methodology	Target concept
Nationwide	Nationwide mortgage lending	Excludes cash sales	UK	Mortgage offer	Hedonic regression	Price of typically transacted property, based on Nationwide and external sales data from the previous two years
Halifax	Halifax mortgage lending	Excludes cash sales	UK	Mortgage offer	Hedonic regression	Price of property typically sold by Halifax in 1983
LSL Acad	Land Registry price paid data	All registered sales	England & Wales; also Scotland and Wales separately	Registration of sale with Land Registry	Mix adjustment	Until Dec 2013: price of property typical of sales Jan 2000 to Dec 2003 Since Jan 2014: price of property typical of sales Jan 2010 to Dec 2013
Land Registry	Repeat sales data	Excludes properties that have not sold at least twice since 1995 (i.e. excludes all new properties)	England & Wales	Registration of sale with Land Registry	Repeat sales regression	Average increase in house prices (geometric mean price change applied to 'average' house in April 2000)

Source : https://ifs.org.uk/bns/bn146.pdf

Classification Model(K-Means) applied on the interests of People on local Venues

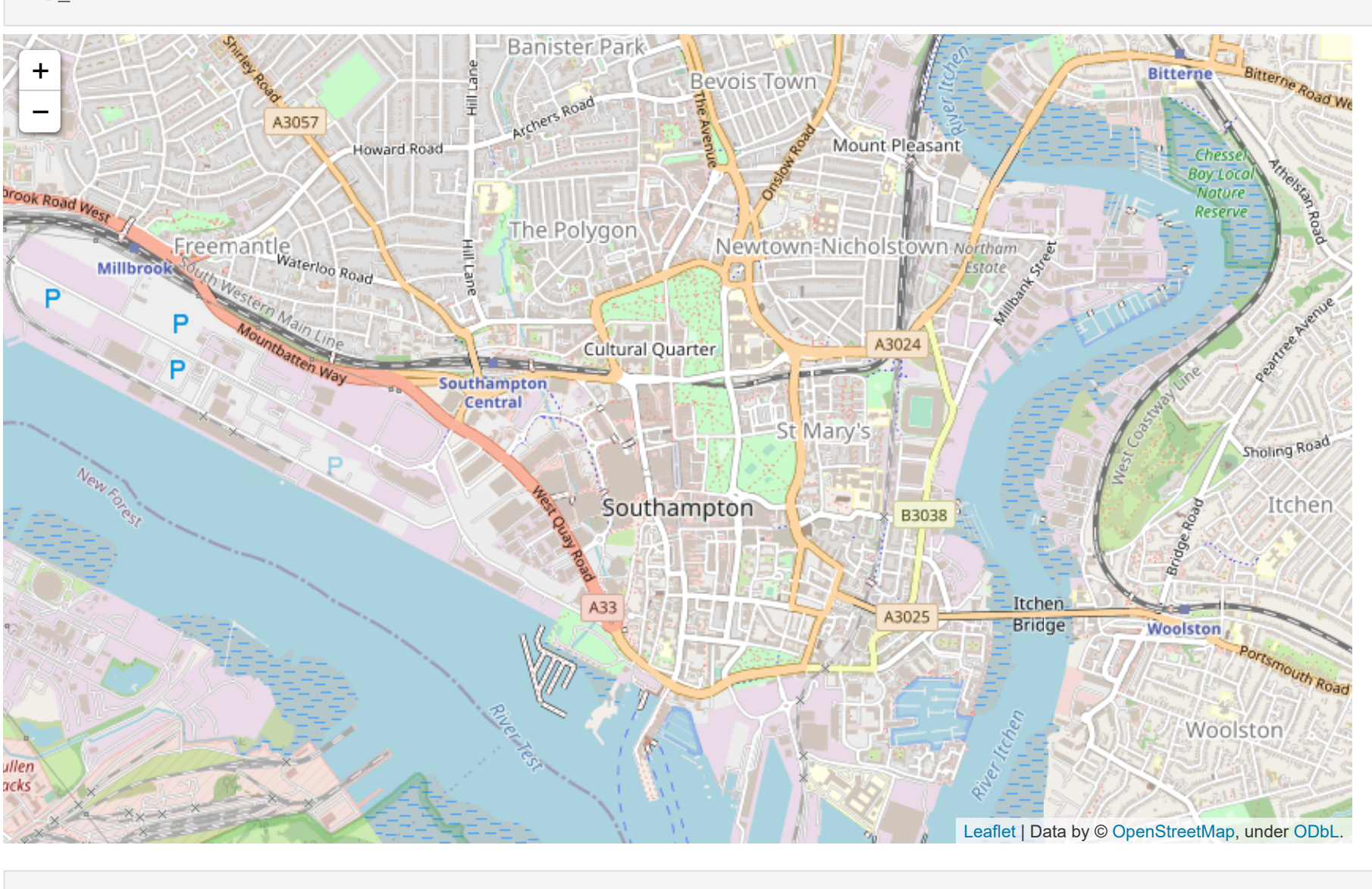
In [38]:
```
#importing various libraries
import requests
import pandas as pd
import numpy as np
import random
from geopy.geocoders import Nominatim
from IPython.display import Image
from IPython.core.display import HTML
from pandas.io.json import json_normalize
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.cm as cm
import matplotlib.colors as colors
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
import folium
import geocoder
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#User enters the location to do some analysis
loc = input("Enter the location :")
```
```
Enter the location :Southampton
```

In [3]:
```
#getting the co-ordinates of the location
geolocator = Nominatim(user_agent="uk")
address = str(loc +", United Kingdom")
location = geocoder.arcgis(address)
latlang = location.latlng
```

In [4]:
```
#plotting the map of the location entered
map_loc=folium.Map(location=[latlang[0],latlang[1]],zoom_start=14)
map_loc
```

Out[4]:



+
−
Banister Park
Bevois Town
Mount Pleasant
Bitterne
Bitterne Road West
A3057
Shirley Road
Howard Road
Archers Road
Hill Lane
The Avenue
Onslow Road
River Itchen
Chessel Bay Local Nature Reserve
Athelstan Road
Freemantle
Millbrook
Waterloo Road
The Polygon
Newtown-Nicholstown
Northam Estate
Millbank Street
South Western Main Line
Mountbatten Way
Southampton Central
Cultural Quarter
A3024
Peartree Avenue
West Coastway Line
St Mary's
Sholing Road
New Forest
West Quay Road
Southampton
B3038
Itchen
Bridge Road
A33
A3025
Itchen Bridge
Woolston
Portsmouth Road
River Test
Woolston
Leaflet | Data by © OpenStreetMap, under ODbL.

In [5]:
```
#Importing the dataset which contains all the neighboorhoods of the locations
loc_neighborhoods = pd.read_excel(r'top25.xlsx',sheet_name='Sheet2')
#selecting the neighborhoods from the city selected
loc_neighborhoods=loc_neighborhoods[loc_neighborhoods.city==loc]
loc_neighborhoods.drop('city',inplace=True,axis=1)
loc_neighborhoods.head()
```

Out[5]:

	neighborhood
402	Bargate
403	Bassett
404	Bevois
405	Bitterne
406	Bitterne Park

In [7]:
```
# defining the function for geopy library to get latitude and longitude for all the neighborhoods
def getNeighborhoodLatLong(Neighborhoods):

    #Neighborhood with lat , long
    latlang = []
    # defining the user agent for geopy
    geolocator = Nominatim(user_agent="uk_neighborhoods")

    # for every unique  of Neighborhood  get the lattitude and longitude
    for  neigh in Neighborhoods:
        address = str(neigh +", {}".format(loc))
        location = geocoder.arcgis(address)
        latlang.append(location.latlng)

    return(latlang)
```

In [8]:
```
#applying the function on neighboorhoods to get the neighboorhoods
loc_neigh = getNeighborhoodLatLong(Neighborhoods = loc_neighborhoods["neighborhood"])
```

In [9]:
```
#checking the top 5 values of locations
loc_neigh[:5]
```

Out[9]:
```
[[50.90265000000005, -1.4041899999999714],
 [50.947819490224404, -1.4045911432397613],
 [50.91969010144991, -1.3953344241759458],
 [50.914540000000045, -1.3384599999999941],
 [50.91483967933845, -1.3632775884247788]]
```

In [10]:
```
#splitting the co-ordinates of the locations
latitude= []
longitude = []
for lat, lng in loc_neigh:
    latitude.append(lat)
    longitude.append(lng)

print(latitude[0:5])
print(longitude[0:5])
```
```
[50.90265000000005, 50.947819490224404, 50.91969010144991, 50.914540000000045, 50.91483967933845]
[-1.4041899999999714, -1.4045911432397613, -1.3953344241759458, -1.3384599999999941, -1.3632775884247788]
```

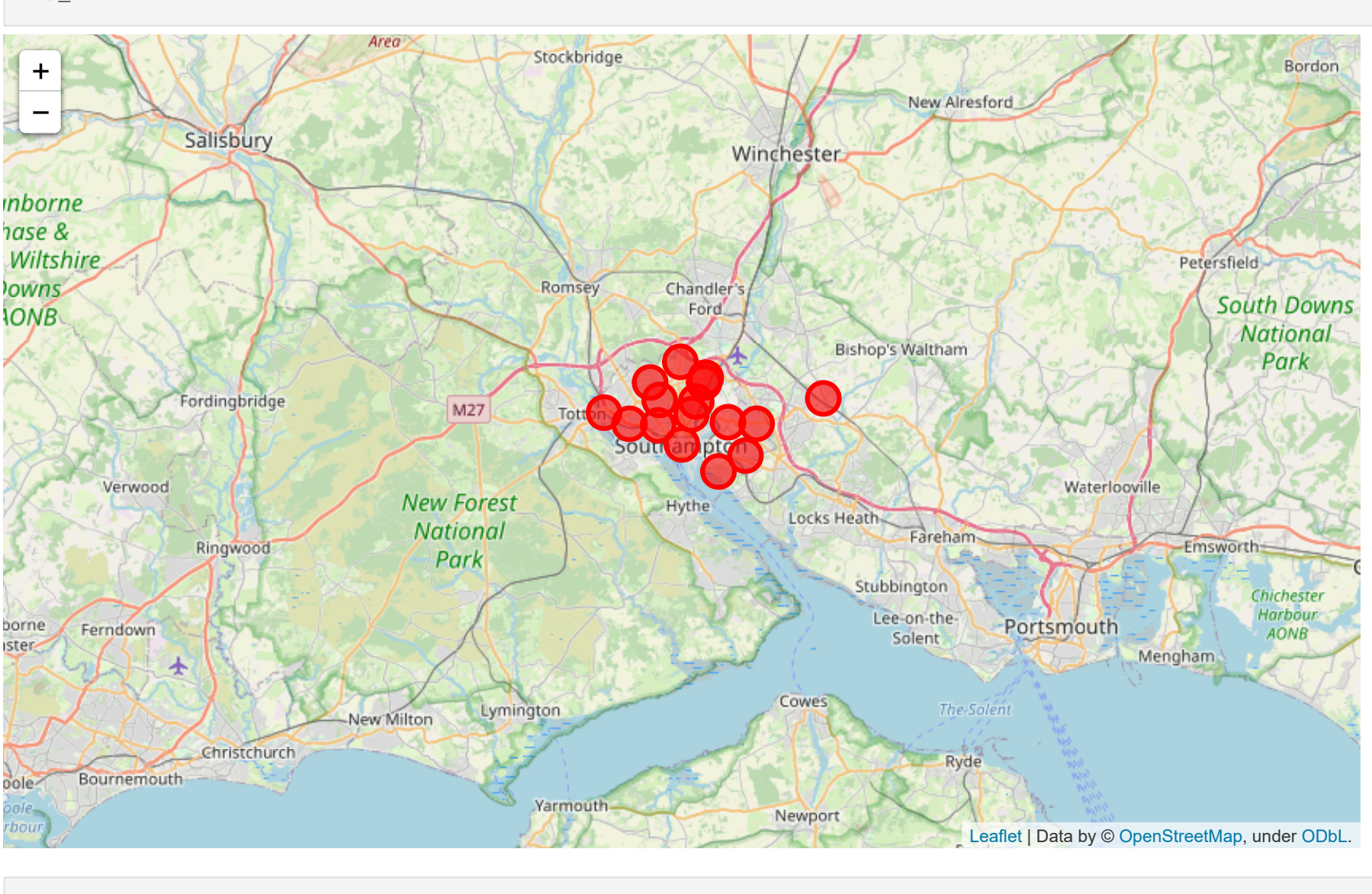
In [11]:
```
#appending the values of co-ordinates to the dataframe
loc_neighborhoods["Latitude"] = latitude
loc_neighborhoods["Longitude"] = longitude

loc_neighborhoods.head()
```

Out[11]:

	neighborhood	Latitude	Longitude
402	Bargate	50.902650	-1.404190
403	Bassett	50.947819	-1.404591
404	Bevois	50.919690	-1.395334
405	Bitterne	50.914540	-1.338460
406	Bitterne Park	50.914840	-1.363278

In [12]:
```
#plotting the locations and areas within the location
map_loc=folium.Map(location=[latlang[0],latlang[1]],zoom_start=10)
for lat,lng,neighborhood in zip(loc_neighborhoods['Latitude'],loc_neighborhoods['Longitude'],loc_neighborhoods[
    label='{}'.format(neighborhood)
    folium.CircleMarker([lat,lng],color='red',popup=label,fill=True,fill_color='red',fill_opacity=0.6).add_to(m
map_loc
```

Out[12]:



+
−
Area
Stockbridge
Bordon
New Alresford
Salisbury
Winchester
nborne
hase &
Wiltshire
owns
AONB
Petersfield
Romsey
Chandler's Ford
South Downs National Park
Bishop's Waltham
Fordingbridge
M27
Tott
Sout   pton
Verwood
New Forest National Park
Hythe
Waterlooville
Locks Heath
Fareham
Emsworth
Ringwood
Stubbington
Chichester Harbour AONB
Lee-on-the-Solent
Portsmouth
Ferndown
Mengham
Cowes
The Solent
New Milton
Lymington
Christchurch
Ryde
Bournemouth
Yarmouth
Newport
Leaflet | Data by © OpenStreetMap, under ODbL.

In [13]:
```
#the key ids used in foursquare API to fetch the data
CLIENT_ID = 'WXER4KXQTW0IHEGA0HLO2EMDOOWEZKAX4YFQLRDMZ2TSBRVM'  # your Foursquare ID
CLIENT_SECRET = '4NMKXNOKJBU4FR312SHLECHR1WSY0ONMQP2JXUYOZ0TWKTJG'  # your Foursquare Secret
VERSION = '20210118'  # Foursquare API version taking last month api for analysis

print('Your credentails:')
print('CLIENT_ID: ' + CLIENT_ID)
print('CLIENT_SECRET:' + CLIENT_SECRET)
```
```
Your credentails:
CLIENT_ID: WXER4KXQTW0IHEGA0HLO2EMDOOWEZKAX4YFQLRDMZ2TSBRVM
CLIENT_SECRET:4NMKXNOKJBU4FR312SHLECHR1WSY0ONMQP2JXUYOZ0TWKTJG
```

In [14]:
```
# API KEY for foursquare API
API_KEY='fsq3lREtx31InFlFo6N72vvhvlJsyukU2DtVPxkju8HMVGU='
```

In [17]:
```
#defining a function to get the nearby venues of the locations

def getNearbyVenues(names, latitudes, longitudes, radius=500):
    LIMIT = 1000

    venues_list = []
    for name, lat, lng in zip(names, latitudes, longitudes):
        print(name)

        # create the API request URL
        url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius
            CLIENT_ID, CLIENT_SECRET, VERSION, lat, lng, radius, LIMIT)

        # make the GET request
        results = requests.get(url).json()["response"]['groups'][0]['items']

        # return only relevant information for each nearby venue
        venues_list.append([
            (name, lat, lng, v['venue']['name'], v['venue']['location']['lat'],
             v['venue']['location']['lng'],
             v['venue']['categories'][0]['name']) for v in results
        ])

    nearby_venues = pd.DataFrame(
        [item for venue_list in venues_list for item in venue_list])
    nearby_venues.columns = [
        'Neighborhood', 'Neighborhood Latitude', 'Neighborhood Longitude',
        'Venue', 'Venue Latitude', 'Venue Longitude', 'Venue Category'
    ]

    return (nearby_venues)
```

In [18]:
```
#applying the function on all the areas to get the venues
loc_venues = getNearbyVenues(names=loc_neighborhoods['neighborhood'],
                                latitudes=loc_neighborhoods['Latitude'],
                                longitudes=loc_neighborhoods['Longitude'])
```
```
Bargate
Bassett
Bevois
Bitterne
Bitterne Park
Coxford
Freemantle
Harefield
Millbrook
Peartree
Portswood
Redbridge
Shirley
Sholing
Swaythling
Woolston
```

In [19]:
```
#printing the location venues and shape
print(loc_venues.shape)
loc_venues
```
```
(175, 7)
```

Out[19]:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Bargate	50.902650	-1.40419	Showcase Cinema de Lux	50.902088	-1.407260	Movie Theater
1	Bargate	50.902650	-1.40419	Apple Southampton	50.904082	-1.405050	Electronics Store
2	Bargate	50.902650	-1.40419	Cafe Thrive	50.902995	-1.401872	Café
3	Bargate	50.902650	-1.40419	Pret A Manger	50.904087	-1.404636	Sandwich Place
4	Bargate	50.902650	-1.40419	Waterstones	50.903190	-1.405865	Bookstore
...	...	...	...	...	...	...	...
170	Woolston	50.888353	-1.37232	Weston Shore	50.886414	-1.374114	Beach
171	Woolston	50.888353	-1.37232	Co-op Food	50.889407	-1.367750	Grocery Store
172	Woolston	50.888353	-1.37232	Forest Flame Disco	50.888281	-1.376445	Entertainment Service
173	Woolston	50.888353	-1.37232	Domino's Pizza	50.889232	-1.368329	Pizza Place
174	Woolston	50.888353	-1.37232	Ice Cream Vendors By Weston Seaside	50.885014	-1.370202	Food Stand

175 rows × 7 columns

In [20]:
```
#counting the count of areas in neighboorhoods
loc_neighborhoods["neighborhood"].value_counts().count()
```

Out[20]:
```
16
```

In [21]:
```
#counting the number of areas in venues
loc_venues["Neighborhood"].value_counts().count()
```

Out[21]:
```
16
```

In [22]:
```
#creating a dataframe in which the areas in neighboorhod is there in venues
loc_neighborhoods = loc_neighborhoods.loc[loc_neighborhoods["neighborhood"].isin(loc_venues["Neighborhood"])]
#counting the locations
loc_neighborhoods["neighborhood"].value_counts().count()
```

Out[22]:
```
16
```

In [23]:
```
#checking the top5 rows
loc_neighborhoods.head()
```

Out[23]:

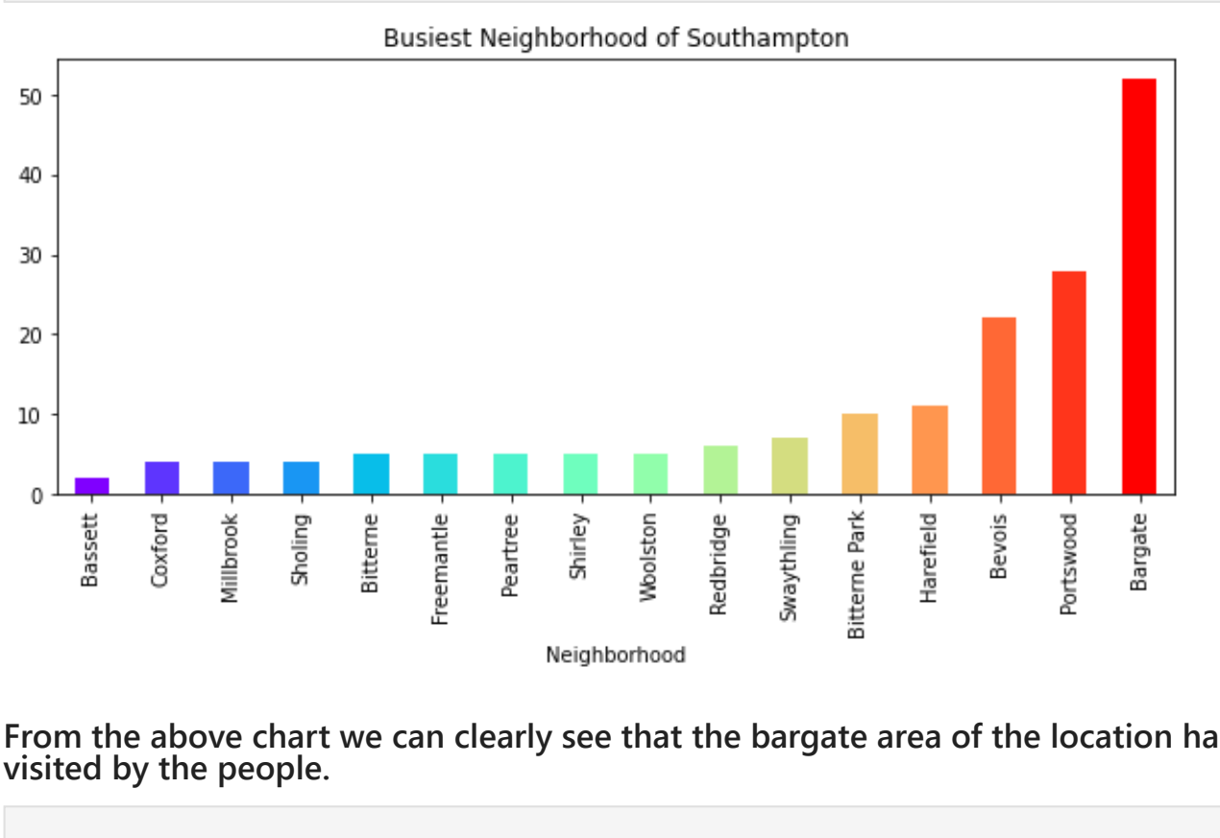
	neighborhood	Latitude	Longitude
402	Bargate	50.902650	-1.404190
403	Bassett	50.947819	-1.404591
404	Bevois	50.919690	-1.395334
405	Bitterne	50.914540	-1.338460
406	Bitterne Park	50.914840	-1.363278

In [24]:
```
#grouping the dataframe with repect to neighborhood
loc_venues.groupby('Neighborhood').count()
```

Out[24]:

	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
Neighborhood						
Bargate	52	52	52	52	52	52
Bassett	2	2	2	2	2	2
Bevois	22	22	22	22	22	22
Bitterne	5	5	5	5	5	5
Bitterne Park	10	10	10	10	10	10
Coxford	4	4	4	4	4	4
Freemantle	5	5	5	5	5	5
Harefield	11	11	11	11	11	11
Millbrook	4	4	4	4	4	4
Peartree	5	5	5	5	5	5
Portswood	28	28	28	28	28	28
Redbridge	6	6	6	6	6	6
Shirley	5	5	5	5	5	5
Sholing	4	4	4	4	4	4
Swaythling	7	7	7	7	7	7
Woolston	5	5	5	5	5	5

In [25]:
```
#plotting the most popular venue as per data received from API
pop_loc_venues = loc_venues.groupby('Neighborhood').count()
fig = plt.figure(figsize=(10,4))
colormap = cm.rainbow(np.linspace(0, 1, pop_loc_venues.shape[0]))
plt.title("Busiest Neighborhood of {}".format(loc))
ax = pop_loc_venues["Venue Category"].sort_values().plot.bar(color = colormap)
```



From the above chart we can clearly see that the bargate area of the location has the most footfalls. and is most visited by the people.

In [26]:
```
#Counting the number of uniques categories of venues
print('There are {} uniques categories.'.format(len(loc_venues['Venue Category'].unique())))
```
```
There are 74 uniques categories.
```

In [27]:
```
# Applying one hot encoding on the dataset
loc_onehot = pd.get_dummies(loc_venues[['Venue Category']], prefix="", prefix_sep="")
loc_onehot['Neighborhood'] = loc_venues['Neighborhood']
loc_onehot.head()
```

Out[27]:

	American Restaurant	Arcade	Asian Restaurant	Auto Workshop	BBQ Joint	Bar	Beach	Bookstore	Bowling Alley	Breakfast Spot	...	Shopping Mall	Ski Area	Sporting Goods Shop	Steakhouse
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0

5 rows × 75 columns

In [28]:
```
#checking the shape of the dataset
loc_onehot.shape
```

Out[28]:
```
(175, 75)
```

In [29]:
```
#grouping the dataset with respect to neighborhood
loc_grouped = loc_onehot.groupby('Neighborhood').mean().reset_index()
loc_grouped
```

Out[29]:

	Neighborhood	American Restaurant	Arcade	Asian Restaurant	Auto Workshop	BBQ Joint	Bar	Beach	Bookstore	Bowling Alley	...	Sandwich Place	Shopping Mall	Ski Area
0	Bargate	0.000000	0.0	0.019231	0.019231	0.019231	0.000000	0.0	0.038462	0.00	...	0.019231	0.019231	0.0
1	Bassett	0.000000	0.5	0.000000	0.000000	0.000000	0.500000	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
2	Bevois	0.045455	0.0	0.000000	0.000000	0.000000	0.181818	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
3	Bitterne	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
4	Bitterne Park	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
5	Coxford	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
6	Freemantle	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
7	Harefield	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
8	Millbrook	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.25	...	0.000000	0.000000	0.0
9	Peartree	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
10	Portswood	0.035714	0.0	0.000000	0.000000	0.000000	0.071429	0.0	0.000000	0.00	...	0.035714	0.000000	0.0
11	Redbridge	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
12	Shirley	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.000000	0.000000	0.2
13	Sholing	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.000000	0.000000	0.0
14	Swaythling	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.00	...	0.142857	0.000000	0.0
15	Woolston	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.2	0.000000	0.00	...	0.000000	0.000000	0.0

16 rows × 75 columns

In [30]:
```
#calculating the frequency of the venues as per the areas
num_top_venues = 5
for hood in loc_grouped['Neighborhood']:
    print("----"+hood+"----")
    temp = loc_grouped[loc_grouped['Neighborhood'] == hood].T.reset_index()
    temp.columns = ['venue','freq']
    temp = temp.iloc[1:]
    temp['freq'] = temp['freq'].astype(float)
    temp = temp.round({'freq': 2})
    print(temp.sort_values('freq', ascending=False).reset_index(drop=True).head(num_top_venues))
    print('\n')
```
```
----Bargate----
                   venue  freq
0     Italian Restaurant  0.08
1                  Hotel  0.06
2            Coffee Shop  0.06
3                    Pub  0.06
4  Portuguese Restaurant  0.04


----Bassett----
                  venue  freq
0                Arcade   0.5
1                   Bar   0.5
2            Kids Store   0.0
3                  Park   0.0
4  Outdoor Supply Store   0.0


----Bevois----
                 venue  freq
0                  Pub  0.23
1                  Bar  0.18
2   Mexican Restaurant  0.09
3  American Restaurant  0.05
4    Convenience Store  0.05


----Bitterne----
                  venue  freq
0         Grocery Store   0.4
1  Fast Food Restaurant   0.2
2           Pizza Place   0.2
3     Indian Restaurant   0.2
4   American Restaurant   0.0


----Bitterne Park----
                  venue  freq
0              Pharmacy   0.2
1             Gift Shop   0.1
2           Pizza Place   0.1
3  Fast Food Restaurant   0.1
4              Bus Stop   0.1


----Coxford----
                 venue  freq
0                 Café  0.50
1        Grocery Store  0.25
2            Newsagent  0.25
3  American Restaurant  0.00
4   Mexican Restaurant  0.00


----Freemantle----
               venue  freq
0                Pub   0.4
1  Convenience Store   0.2
2  Fish & Chips Shop   0.2
3      Grocery Store   0.2
4               Park   0.0


----Harefield----
                  venue  freq
0         Grocery Store  0.27
1  Fast Food Restaurant  0.18
2             Rock Club  0.09
3            Restaurant  0.09
4           Pizza Place  0.09


----Millbrook----
                  venue  freq
0         Grocery Store  0.25
1  Outdoor Supply Store  0.25
2        Discount Store  0.25
3         Bowling Alley  0.25
4   American Restaurant  0.00


----Peartree----
        venue  freq
0         Gym   0.2
1       Hotel   0.2
2       Diner   0.2
3  Playground   0.2
4         Pub   0.2


----Portswood----
           venue  freq
0  Grocery Store  0.11
1            Pub  0.11
2            Bar  0.07
3    Supermarket  0.07
4    Gelato Shop  0.07


----Redbridge----
                 venue  freq
0        Train Station  0.17
1           Playground  0.17
2            Gastropub  0.17
3                  Pub  0.17
4  Rental Car Location  0.17


----Shirley----
                  venue  freq
0                   Pub   0.4
1  Gym / Fitness Center   0.2
2              Ski Area   0.2
3        Hardware Store   0.2
4      Tapas Restaurant   0.0


----Sholing----
                venue  freq
0       Grocery Store  0.50
1  Photography Studio  0.25
2            Bus Stop  0.25
3  Mexican Restaurant  0.00
4                Park  0.00


----Swaythling----
                  venue  freq
0  Fast Food Restaurant  0.29
1         Grocery Store  0.29
2        Sandwich Place  0.14
3            Restaurant  0.14
4     Convenience Store  0.14


----Woolston----
                   venue  freq
0             Food Stand   0.2
1            Pizza Place   0.2
2          Grocery Store   0.2
3                  Beach   0.2
4  Entertainment Service   0.2
```

In [31]:
```
#defining a function which returns the sorted venues list
def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[1:]
    row_categories_sorted = row_categories.sort_values(ascending=False)

    return row_categories_sorted.index.values[0:num_top_venues]
```

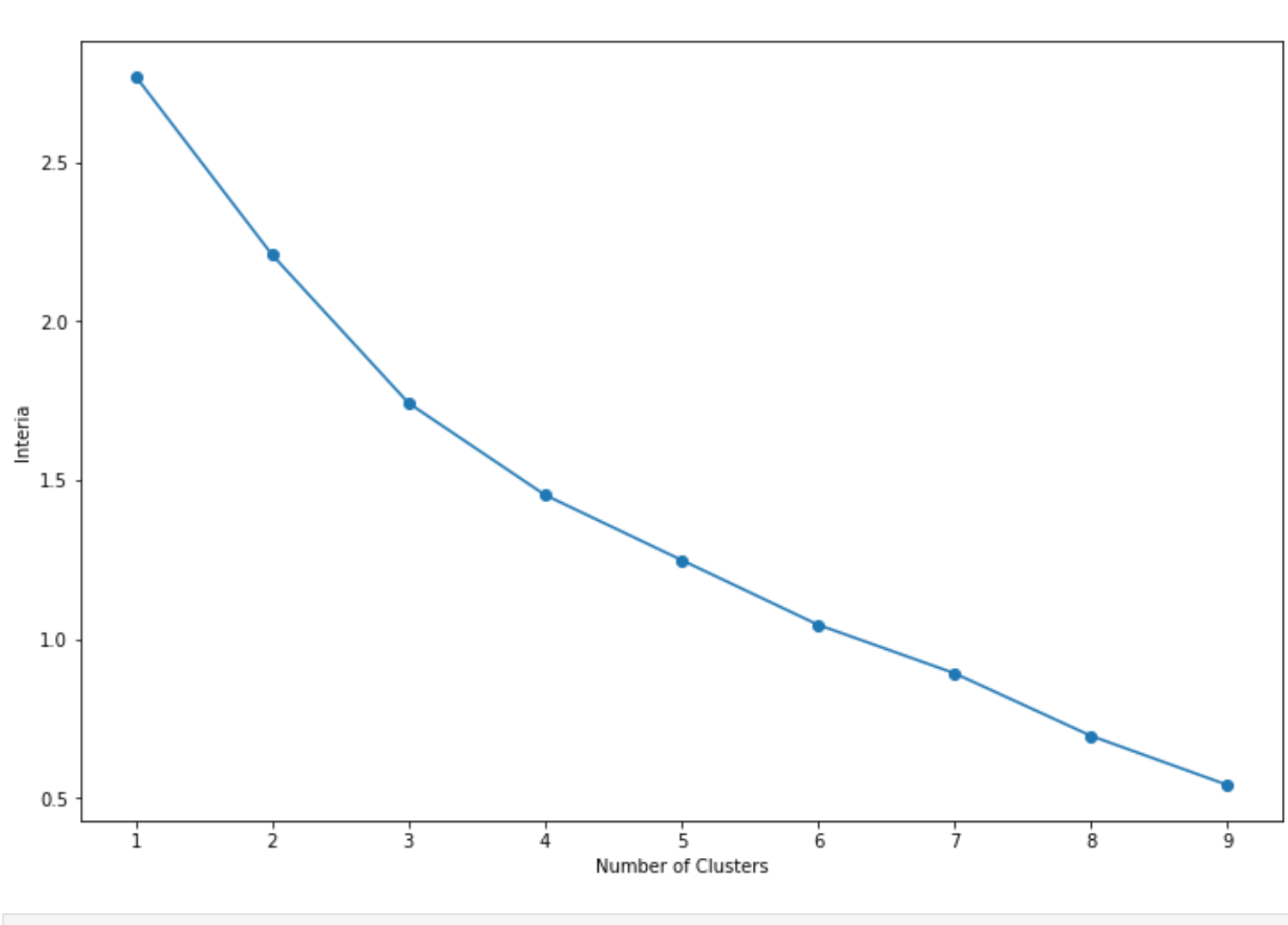
In [32]:
```
#creating a dataframe which shows top 10 most common venues
num_top_venues = 10
indicators = ['st', 'nd', 'rd']
columns = ['Neighborhood']
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
        columns.append('{}th Most Common Venue'.format(ind+1))
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['Neighborhood'] = loc_grouped['Neighborhood']
for ind in np.arange(loc_grouped.shape[0]):
    neighborhoods_venues_sorted.iloc[ind, 1:] = return_most_common_venues(loc_grouped.iloc[ind, :], num_top_ver

neighborhoods_venues_sorted.head()
```

Out[32]:

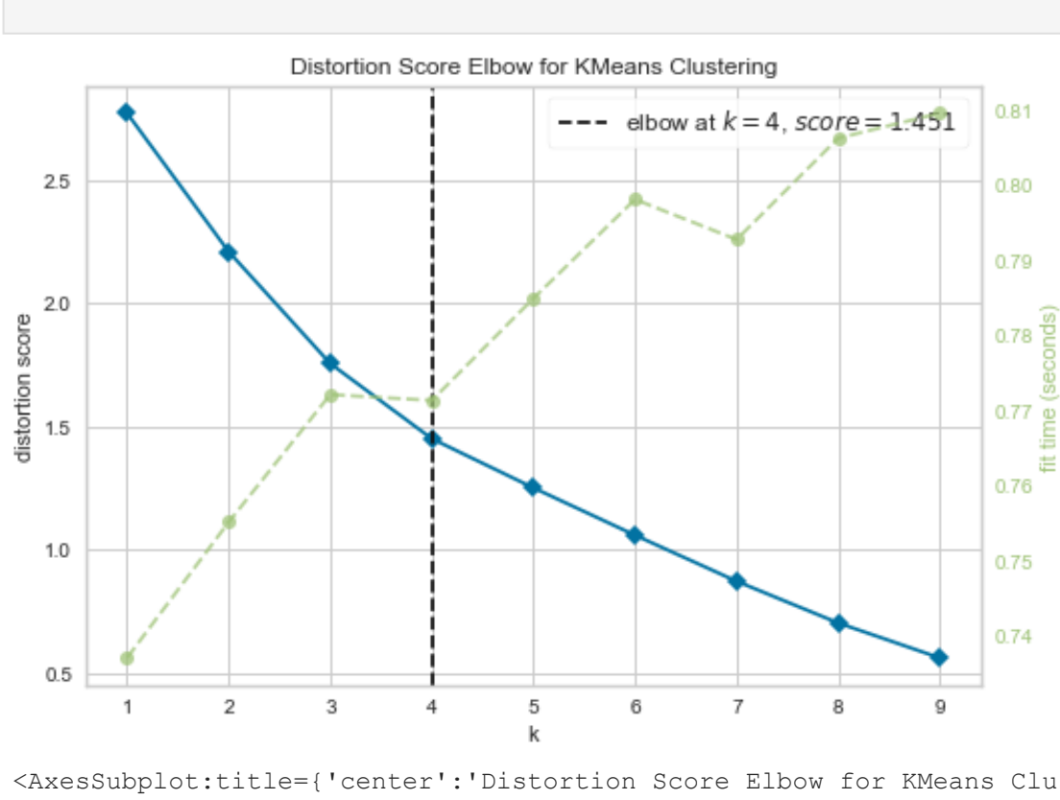
	Neighborhood	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th Most Common Venue	10th Most Common Venue
0	Bargate	Italian Restaurant	Hotel	Coffee Shop	Pub	Portuguese Restaurant	Bookstore	Furniture / Home Store	Historic Site	Clothing Store	Park
1	Bassett	Arcade	Bar	Kids Store	Park	Outdoor Supply Store	Nightclub	Newsagent	Movie Theater	Middle Eastern Restaurant	Mexican Restaurant
2	Bevois	Pub	Bar	Mexican Restaurant	American Restaurant	Convenience Store	Tapas Restaurant	Supermarket	Hookah Bar	Grocery Store	Greek Restaurant
3	Bitterne	Grocery Store	Fast Food Restaurant	Pizza Place	Indian Restaurant	American Restaurant	Outdoor Supply Store	Nightclub	Newsagent	Movie Theater	Middle Eastern Restaurant
4	Bitterne Park	Pharmacy	Gift Shop	Pizza Place	Fast Food Restaurant	Bus Stop	Park	Gym / Fitness Center	Supermarket	Grocery Store	Indian Restaurant

In [37]:
```
# elbow method for calculating the number of clusters

loc_grouped_clustering = loc_grouped.drop('Neighborhood', 1)

clus =[]
for cluster in range(1,10):
    modl = KMeans(n_clusters=cluster, init='k-means++')
    modl.fit(loc_grouped_clustering)
    clus.append(modl.inertia_)

temp = pd.DataFrame({'Cluster': range(1,10), 'cluster_list':clus})
plt.figure(figsize=(12,8))
plt.plot(temp['Cluster'],temp['cluster_list'], marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Interia')
```

Out[37]:
```
Text(0, 0.5, 'Interia')
```



In [40]:
```
#Visualizing the ELbow method with distortion score
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,10)).fit(loc_grouped_clustering)
visualizer.show()
```



Out[40]:
```
<AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion sc
ore'>
```

Setting the number of clusters as 4.

Applying KMeans Clustering

In [42]:
```
kclusters = 4
loc_grouped_clustering = loc_grouped.drop('Neighborhood', 1)
# applying model
kmeans = KMeans(n_clusters=kclusters, random_state=0).fit(loc_grouped_clustering)
# checking the clusters
kmeans.labels_
```

Out[42]:
```
array([0, 3, 0, 2, 0, 1, 0, 2, 2, 0, 0, 0, 0, 2, 2, 2])
```

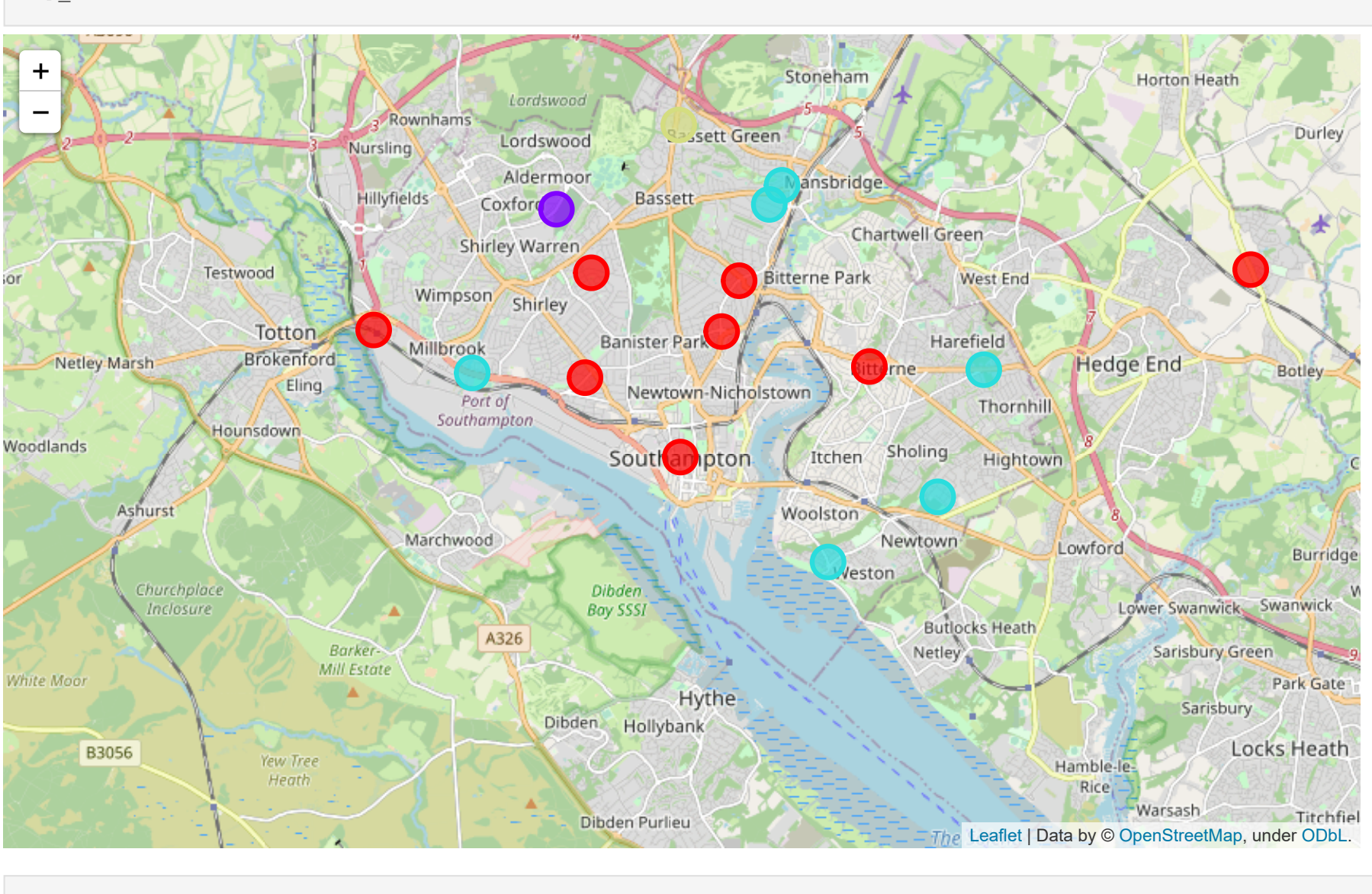
In [43]:
```
#adding the labels to the dataframe
neighborhoods_venues_sorted.insert(0, 'Cluster Labels', kmeans.labels_)
loc_merged = loc_neighborhoods
loc_merged = loc_merged.join(neighborhoods_venues_sorted.set_index('Neighborhood'), on='neighborhood')
loc_merged.head()
```

Out[43]:

	neighborhood	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th M Comm Ve
402	Bargate	50.902650	-1.404190	0	Italian Restaurant	Hotel	Coffee Shop	Pub	Portuguese Restaurant	Bookstore	Furniture / Home Store	Historic
403	Bassett	50.947819	-1.404591	3	Arcade	Bar	Kids Store	Park	Outdoor Supply Store	Nightclub	Newsagent	Mo The
404	Bevois	50.919690	-1.395334	0	Pub	Bar	Mexican Restaurant	American Restaurant	Convenience Store	Tapas Restaurant	Supermarket	Hookah
405	Bitterne	50.914540	-1.338460	2	Grocery Store	Fast Food Restaurant	Pizza Place	Indian Restaurant	American Restaurant	Outdoor Supply Store	Nightclub	Newsag
406	Bitterne Park	50.914840	-1.363278	0	Pharmacy	Gift Shop	Pizza Place	Fast Food Restaurant	Bus Stop	Park	Gym / Fitness Center	Superma

In [45]:
```
# create map with clustersing
map_clusters = folium.Map(location=[latlang[0], latlang[1]], zoom_start=12.3)

# set color scheme for the clusters
x = np.arange(kclusters)
ys = [i + x + (i*x)**2 for i in range(kclusters)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

markers_colors = []
for lat, lon, poi, cluster in zip(loc_merged['Latitude'], loc_merged['Longitude'], loc_merged['neighborhood'],
    label = folium.Popup(' Cluster ' + str(cluster + 1) + str(poi), parse_html=True)
    folium.CircleMarker(
        [lat, lon],
        radius=10,
        popup=label,
        color=rainbow[cluster-1],
        fill=True,
        fill_color=rainbow[cluster-1],
        fill_opacity=0.7).add_to(map_clusters)

map_clusters
```

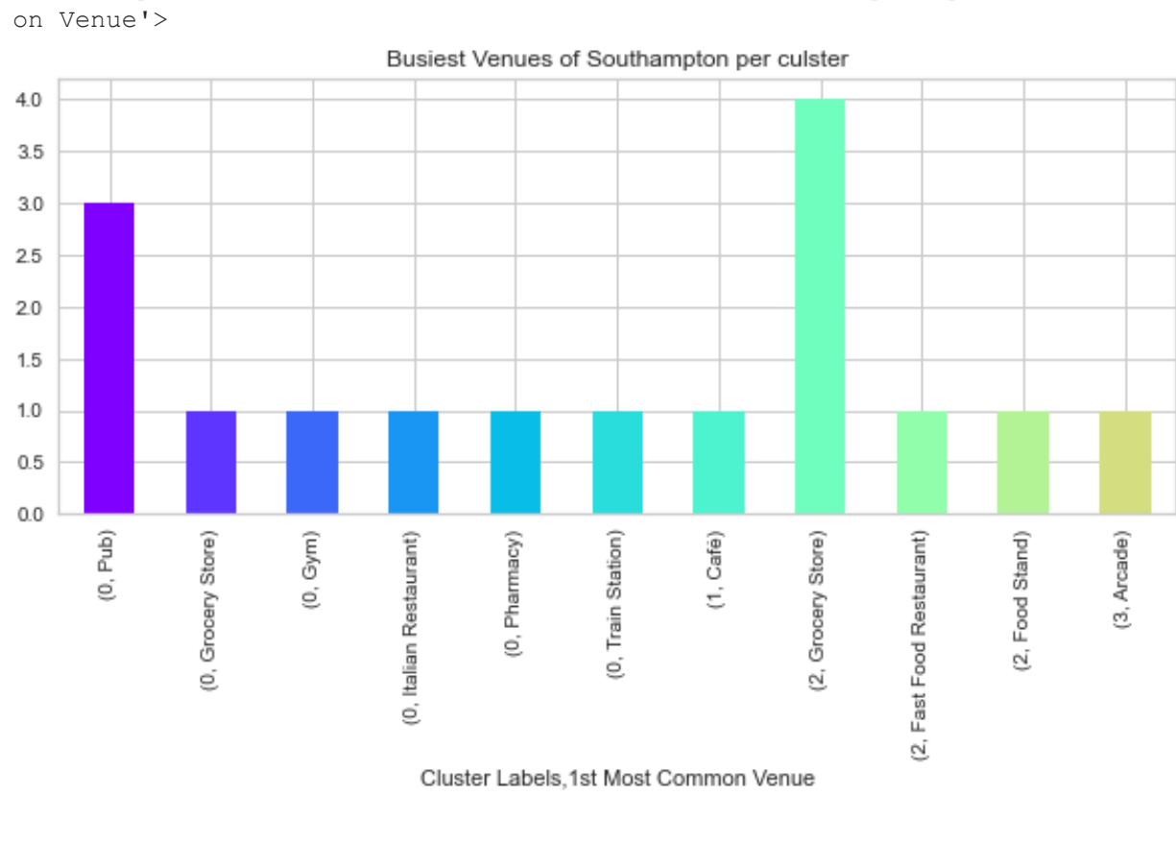
Out[45]:



+
−
Stoneham
Horton Heath
Lordswood
Rownhams
Nursling
Lordswood
Bassett Green
Durley
Aldermoor
Hillyfields
Coxford
Bassett
Swaythling
Shirley Warren
Chartwell Green
Testwood
Bitterne Park
West End
Wimpson
Shirley
Banister Park
Harefield
Totton
Netley Marsh
Brokenford
Millbrook
Bitterne
Hedge End
Botley
Eling
Port of Southampton
Newtown-Nicholstown
Thornhill
Hounsdown
Woodlands
Southampton
Itchen
Sholing
Hightown
Ashurst
Woolston
Newtown
Lowford
Marchwood
Weston
Burridge
Churchplace Inclosure
Dibden Bay SSSI
Butlocks Heath
Lower Swanwick
Swanwick
Netley
A326
Sarisbury Green
Barker-Mill Estate
White Moor
Park Gate
Hythe
Sarisbury
B3056
Dibden
Hollybank
Locks Heath
Yew Tree Heath
Hamble-le-Rice
Dibden Purlieu
Warsash
Leaflet | Data by © OpenStreetMap, under ODbL.

In [46]:
```
#plotting the bar chart of the busiest venues
tmp = loc_merged.groupby("Cluster Labels")["1st Most Common Venue"]
fig = plt.figure(figsize=(10,4))
colormap = cm.rainbow(np.linspace(0, 1, pop_loc_venues.shape[0]))
plt.title("Busiest Venues of {} per culster".format(loc))
tmp.value_counts().plot.bar(color = colormap)
```

Out[46]:
```
<AxesSubplot:title={'center':'Busiest Venues of Southampton per culster'}, xlabel='Cluster Labels,1st Most Comm
on Venue'>
```



Cluster No 1:

In [47]:
```
clus1 = loc_merged.loc[loc_merged['Cluster Labels'] == 0]
clus1.head()
```

Out[47]:

	neighborhood	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Com V
402	Bargate	50.90265	-1.404190	0	Italian Restaurant	Hotel	Coffee Shop	Pub	Portuguese Restaurant	Bookstore	Furniture / Home Store	Histori
404	Bevois	50.91969	-1.395334	0	Pub	Bar	Mexican Restaurant	American Restaurant	Convenience Store	Tapas Restaurant	Supermarket	Hooka
406	Bitterne Park	50.91484	-1.363278	0	Pharmacy	Gift Shop	Pizza Place	Fast Food Restaurant	Bus Stop	Park	Gym / Fitness Center	Superm
408	Freemantle	50.91338	-1.424640	0	Pub	Convenience Store	Fish & Chips Shop	Grocery Store	Park	Outdoor Supply Store	Nightclub	Newsa
411	Peartree	50.92822	-1.280820	0	Gym	Hotel	Diner	Playground	Pub	American Restaurant	Middle Eastern Restaurant	Out Supply

In [48]:
```
#making a temperary dataframe for plotting on map
dft_tmp_clus1 = pd.DataFrame(
    clus1.groupby("neighborhood")["1st Most Common Venue"].apply(
        lambda x: ','.join(x.unique())).reset_index())
dft_tmp_clus1.rename(columns={
    'neighborhood': 'neighborhood',
    '1st Most Common Venue': 'top_venues'
},
                     inplace=True)
dft_tmp_clus1["top_venues"].values
```

Out[48]:
```
array(['Italian Restaurant', 'Pub', 'Pharmacy', 'Pub', 'Gym',
       'Grocery Store', 'Train Station', 'Pub'], dtype=object)
```

We can see that the venues in Cluster 1 is majority is of Pubs.

Cluster No 2:

In [49]:
```
clus2 = loc_merged.loc[loc_merged['Cluster Labels'] == 1]
clus2.head()
```

Out[49]:

	neighborhood	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th Con \
407	Coxford	50.936443	-1.430981	1	Café	Grocery Store	Newsagent	American Restaurant	Mexican Restaurant	Park	Outdoor Supply Store	Nightclub	N Th

In [50]:
```
#creating a dataframe for plotting on map
dft_tmp_clus2 = pd.DataFrame(
    clus2.groupby("neighborhood")["1st Most Common Venue"].apply(
        lambda x: ','.join(x.unique())).reset_index())
dft_tmp_clus2.rename(columns={
    'neighborhood': 'neighborhood',
    '1st Most Common Venue': 'top_venues'
},
                     inplace=True)
dft_tmp_clus2["top_venues"].values
```

Out[50]:
```
array(['Café'], dtype=object)
```

Cluster no 2 basically consists of Cafes.

In [51]:
```
clus3 = loc_merged.loc[loc_merged['Cluster Labels'] == 2]
clus3.head()
```

Out[51]:

	neighborhood	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Con \
405	Bitterne	50.914540	-1.338460	2	Grocery Store	Fast Food Restaurant	Pizza Place	Indian Restaurant	American Restaurant	Outdoor Supply Store	Nightclub	News
409	Harefield	50.937075	-1.384875	2	Grocery Store	Fast Food Restaurant	Rock Club	Restaurant	Pizza Place	Playground	Convenience Store	Superm
410	Millbrook	50.914098	-1.449217	2	Grocery Store	Outdoor Supply Store	Discount Store	Bowling Alley	American Restaurant	Middle Eastern Restaurant	Park	Nigh
415	Sholing	50.897190	-1.348655	2	Grocery Store	Photography Studio	Bus Stop	Mexican Restaurant	Park	Outdoor Supply Store	Nightclub	News
416	Swaythling	50.939598	-1.382364	2	Fast Food Restaurant	Grocery Store	Sandwich Place	Restaurant	Convenience Store	Indian Restaurant	Italian Restaurant	Jap Resta

In [52]:
```
#creating a temporary dataframe to plot on map
dft_tmp_clus3 = pd.DataFrame(
    clus3.groupby("neighborhood")["1st Most Common Venue"].apply(
        lambda x: ','.join(x.unique())).reset_index())
dft_tmp_clus3.rename(columns={
    'neighborhood': 'neighborhood',
    '1st Most Common Venue': 'top_venues'
},
                     inplace=True)
dft_tmp_clus3["top_venues"].values
```

Out[52]:
```
array(['Grocery Store', 'Grocery Store', 'Grocery Store', 'Grocery Store',
       'Fast Food Restaurant', 'Food Stand'], dtype=object)
```

We can see that Cluster No 3 consists of Grocery stores , i.e. Supermarkets

In [53]:
```
clus4 = loc_merged.loc[loc_merged['Cluster Labels'] == 3]
clus4.head()
```

Out[53]:

	neighborhood	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th M Com Ve
403	Bassett	50.947819	-1.404591	3	Arcade	Bar	Kids Store	Park	Outdoor Supply Store	Nightclub	Newsagent	Movie Theater	Mi Ea Restau

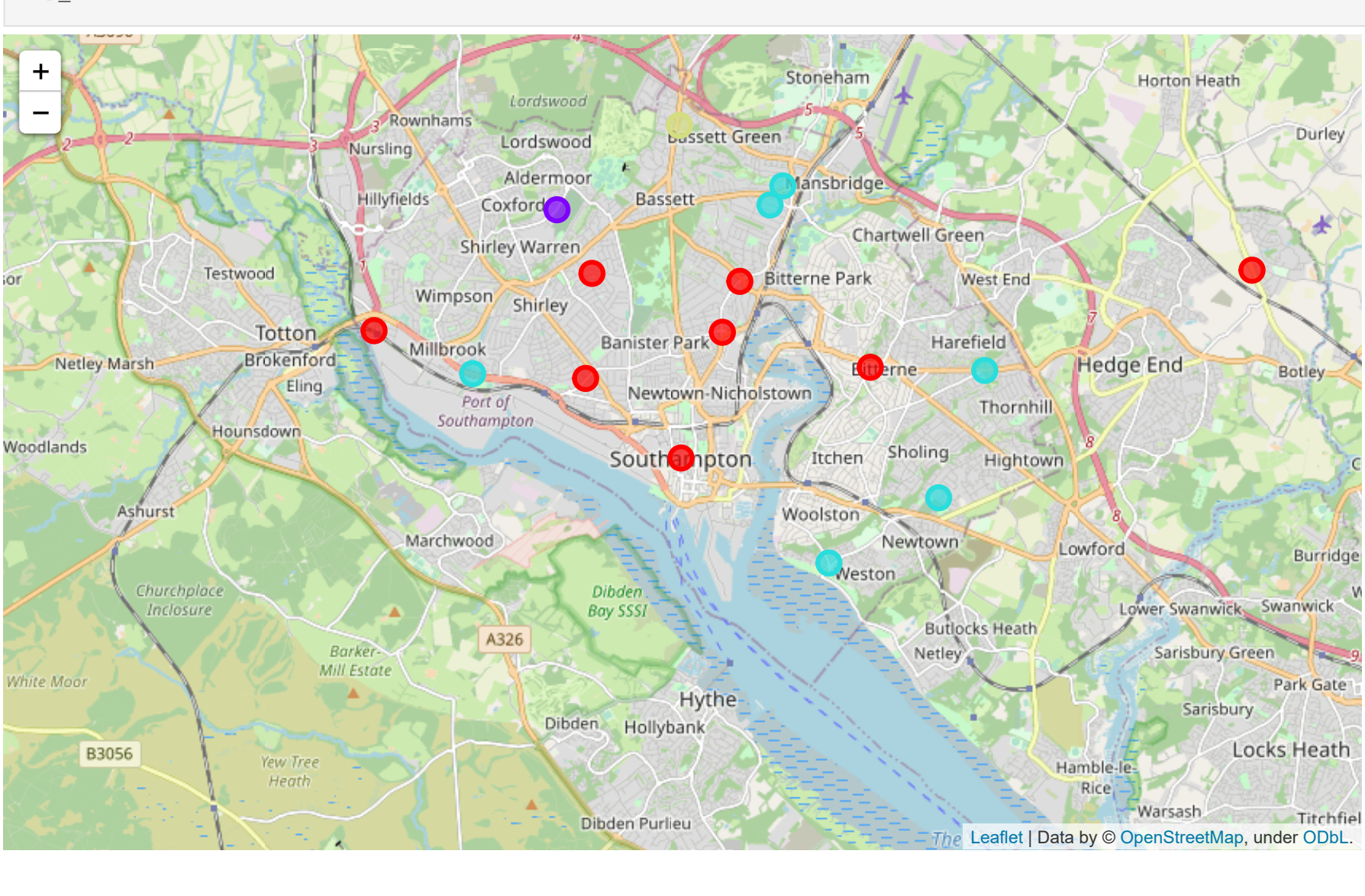
In [54]:
```
#creating a temporary dataframe to plot on map
dft_tmp_clus4 = pd.DataFrame(
    clus4.groupby("neighborhood")["1st Most Common Venue"].apply(
        lambda x: ','.join(x.unique())).reset_index())
dft_tmp_clus4.rename(columns={
    'neighborhood': 'neighborhood',
    '1st Most Common Venue': 'top_venues'
},
                     inplace=True)
dft_tmp_clus4["top_venues"].values
```

Out[54]:
```
array(['Arcade'], dtype=object)
```

Cluster No 4 consists of Arcade (games)

In [57]:
```
# create map to show the clusters
map_clusters = folium.Map(location=[latlang[0], latlang[1]], zoom_start=12)

# set color scheme for the clusters
x = np.arange(kclusters)
ys = [i + x + (i * x)**2 for i in range(kclusters)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

markers_colors = []
for lat, lon, poi, cluster, top_spot in zip(
        loc_merged['Latitude'], loc_merged['Longitude'],
        loc_merged['neighborhood'], loc_merged['Cluster Labels'],
        loc_merged['1st Most Common Venue']):
    # putting the extra options to explore in each cluster on the popups
    if cluster == 0:
        label = folium.Popup("<h6> <center> Pubs </center></h6> \
                              <h6><b> {}</b> : {}</h6> \
                           <h6>  : {} </h6> ".format(
            str(poi), str(top_spot),
            str(dft_tmp_clus1.loc[dft_tmp_clus1.neighborhood == neighborhood,
                                  "top_venues"].values)),
                              parse_html=False)
    elif cluster == 1:
        label = folium.Popup("<center>Museums </center>\
                              <h6><b> {}</b> : {}</h6> \
                           <h6>  : {} </h6> ".format(
            str(poi), str(top_spot),
            str(dft_tmp_clus2.loc[dft_tmp_clus2.neighborhood == neighborhood,
                                  "top_venues"].values)),
                              parse_html=False)

    elif cluster == 2:
        label = folium.Popup("<h6> <center> Bars </center> </h6> \
                              <h6><b> {}</b> : {}</h6> \
                           <h6>  : {} </h6> ".format(
            str(poi), str(top_spot),
            str(dft_tmp_clus3.loc[dft_tmp_clus3.neighborhood == neighborhood,
                                  "top_venues"].values)),
                              parse_html=False)

    elif cluster == 3:
        label = folium.Popup("<h6> <center> Market </center> </h6> \
                              <h6><b> {}</b> : {}</h6> \
                           <h6> : {} </h6> ".format(
            str(poi), str(top_spot),
            str(dft_tmp_clus4.loc[dft_tmp_clus4.neighborhood == neighborhood,
                                  "top_venues"].values)),
                              parse_html=False)


    folium.CircleMarker([lat, lon],
                        radius=7,
                        popup=label,
                        color=rainbow[cluster - 1],
                        fill=True,
                        fill_color=rainbow[cluster - 1],
                        fill_opacity=0.7).add_to(map_clusters)

map_clusters
```

Out[57]:


+
−
Stoneham
Horton Heath
Durley
Lordswood
Rownhams
Nursling
Lordswood
Bassett Green
Aldermoor
Hillyfields
Coxford
Bassett
Swaythling
Mansbridge
Chartwell Green
Shirley Warren
Testwood
Wimpson
Shirley
Bitterne Park
West End
Totton
Brokenford
Netley Marsh
Millbrook
Banister Park
Bitterne
Harefield
Hedge End
Botley
Eling
Port of Southampton
Newtown-Nicholstown
Thornhill
Woodlands
Hounsdown
Southampton
Itchen
Sholing
Hightown
Ashurst
Woolston
Lowford
Marchwood
Newtown
Burridge
Weston
Churchplace Inclosure
Dibden Bay SSSI
Lower Swanwick
Swanwick
Butlocks Heath
Netley
Sarisbury Green
A326
Barker-Mill Estate
Park Gate
White Moor
Hythe
Sarisbury
B3056
Dibden
Hollybank
Locks Heath
Yew Tree Heath
Hamble-le-Rice
Dibden Purlieu
Warsash
Titchfiel
Leaflet | Data by © OpenStreetMap, under ODbL.

Applying Machine Learning Models (Random Forest, XGBoost)

Data Source : https://www.gov.uk/guidance/about-the-price-paid-data#explanations-of-column-headers-in-the-ppd

Explanations of column headers in the PPD The data is published in columns in the order set out in the table, we do not supply column headers in the files.

- Transaction unique identifier: A reference number which is generated automatically recording each published sale. The number is unique and will change each time a sale is recorded.
- Price :Sale price stated on the transfer deed.
- Date of Transfer: Date when the sale was completed, as stated on the transfer deed.
- Postcode This is the postcode used at the time of the original transaction. Note that postcodes can be reallocated and these changes are not reflected in the Price Paid Dataset.
- Property Type D = Detached, S = Semi-Detached, T = Terraced, F = Flats/Maisonettes, O = Other

Note that:

- we only record the above categories to describe property type, we do not separately identify bungalows
- end-of-terrace properties are included in the Terraced category above
- 'Other' is only valid where the transaction relates to a property type that is not covered by existing values, for example where a property comprises more than one large parcel of land

- Old/New Indicates the age of the property and applies to all price paid transactions, residential and non-residential. Y = a newly built property, N = an established residential building

- Duration Relates to the tenure: F = Freehold, L= Leasehold etc.

Note that HM Land Registry does not record leases of 7 years or less in the Price Paid Dataset.

- PAON Primary Addressable Object Name. Typically the house number or name.
- SAON Secondary Addressable Object Name. Where a property has been divided into separate units (for example, flats), the PAON (above) will identify the building and a SAON will be specified that identifies the separate unit/flat.
- Street
- Locality
- Town/City
- District
- County
- PPD Category Type Indicates the type of Price Paid transaction. A = Standard Price Paid entry, includes single residential property sold for value. B = Additional Price Paid entry including transfers under a power of sale/repossessions, buy-to-lets (where they can be identified by a Mortgage), transfers to non-private individuals and sales where the property type is classed as 'Other'.

Note that category B does not separately identify the transaction types stated. HM Land Registry has been collecting information on Category A transactions from January 1995. Category B transactions were identified from October 2013.

- Record Status - monthly file only Indicates additions, changes and deletions to the records.(see guide below). A = Addition C = Change D = Delete

Note that where a transaction changes category type due to misallocation (as above) it will be deleted from the original category type and added to the correct category with a new transaction unique identifier. The date field - the date of the transfer deed lodged with us for registration.

In [124…

```python
#importing various libraries

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import requests
from io import StringIO
from statsmodels.graphics.factorplots import interaction_plot
import time
import statsmodels.api as sm
from statsmodels.formula.api import ols
import scipy.stats as stats
from scipy.stats import f_oneway
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
from sklearn.svm import SVR
import numpy as np
from xgboost import XGBRegressor
from sklearn import metrics
import joblib
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```python
#importing dataset for 2022
df_2022 = pd.read_excel(r'pp-2022.xlsx')
df_2022.head()
```

Out[3]:

	sno	Transaction unique identifier	Price	Date of Transfer	Postcode	Property Type	Old/New	Duration	PAON	SAON	Street	Locality	Town/City
0	1	{E073986B-94F0-2134-E053-6C04A8C0233B}	141000	2022-02-04	NG20 8XN	S	N	F	8	NaN	HEWETT STREET	WARSOP VALE	MANSFIELD
1	2	{E073986B-94F2-2134-E053-6C04A8C0233B}	90000	2022-02-22	NG6 0AX	T	N	F	16	NaN	BURNABY STREET	NaN	NOTTINGHAM
2	3	{E073986B-94F3-2134-E053-6C04A8C0233B}	250000	2022-02-04	NG24 1FR	T	N	F	RIVER VIEW	28	TRENT LANE	NaN	NEWARK
3	4	{E073986B-94F5-2134-E053-6C04A8C0233B}	160000	2022-04-25	NG17 2TR	S	N	F	41	NaN	PARKLAND VIEW	HUTHWAITE	SUTTON-IN-ASHFIELD
4	5	{E073986B-94F9-2134-E053-6C04A8C0233B}	315000	2022-02-28	NG9 4DP	S	N	F	17	NaN	CLIFFGROVE AVENUE	BEESTON	NOTTINGHAM

In [4]:

```python
#checking the shape
df_2022.shape
```

Out[4]:

```
(231624, 18)
```

In [5]:

```python
#checking shape of the dataset
df_2022.shape
```

Out[5]:

```
(231624, 18)
```

In [6]:

```python
#checking the last and first date of the dataframe
print("Initial date of Data",df_2022['Date of Transfer'].min())
print("Last date of Data",df_2022['Date of Transfer'].max())
```

```
Initial date of Data 2022-01-01 00:00:00
Last date of Data 2022-06-27 00:00:00
```

In [7]:

```python
#checking the top records
df_2022.head()
```

Out[7]:

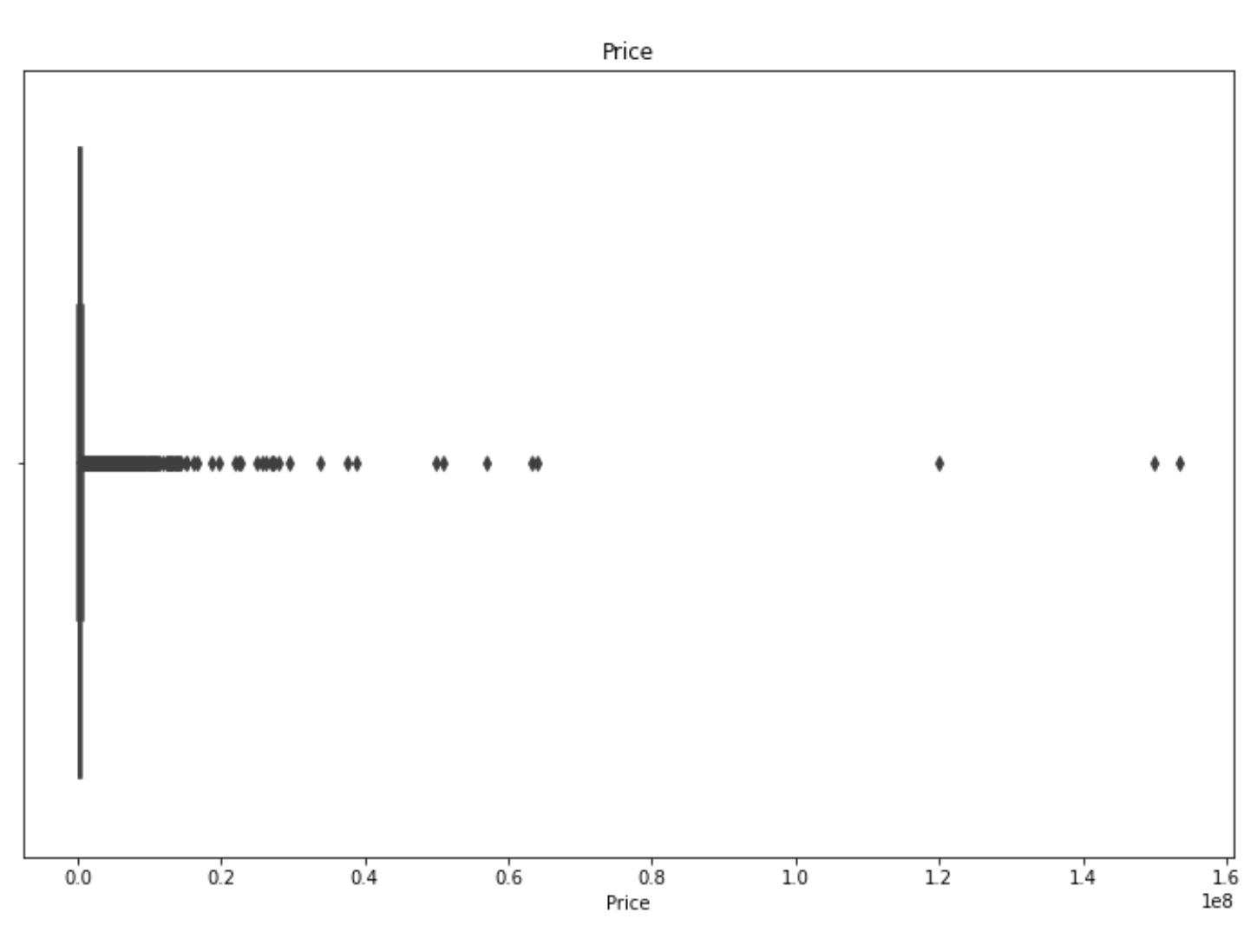
	sno	Transaction unique identifier	Price	Date of Transfer	Postcode	Property Type	Old/New	Duration	PAON	SAON	Street	Locality	Town/City
0	1	{E073986B-94F0-2134-E053-6C04A8C0233B}	141000	2022-02-04	NG20 8XN	S	N	F	8	NaN	HEWETT STREET	WARSOP VALE	MANSFIELD
1	2	{E073986B-94F2-2134-E053-6C04A8C0233B}	90000	2022-02-22	NG6 0AX	T	N	F	16	NaN	BURNABY STREET	NaN	NOTTINGHAM
2	3	{E073986B-94F3-2134-E053-6C04A8C0233B}	250000	2022-02-04	NG24 1FR	T	N	F	RIVER VIEW	28	TRENT LANE	NaN	NEWARK
3	4	{E073986B-94F5-2134-E053-6C04A8C0233B}	160000	2022-04-25	NG17 2TR	S	N	F	41	NaN	PARKLAND VIEW	HUTHWAITE	SUTTON-IN-ASHFIELD
4	5	{E073986B-94F9-2134-E053-6C04A8C0233B}	315000	2022-02-28	NG9 4DP	S	N	F	17	NaN	CLIFFGROVE AVENUE	BEESTON	NOTTINGHAM

In [8]:

```python
#plotting a boxplot for price
plt.figure(figsize=(12,8))
sns.boxplot(x = df_2022['Price'])
plt.title('Price')
```

Out[8]:

```
Text(0.5, 1.0, 'Price')
```



We can see various outliers, thus removing outliers

In [9]:

```python
#checking the max, min and median prices
df_2022.Price.max(), df_2022.Price.min(), df_2022.Price.median()
```

Out[9]:

```
(153450000, 100, 260000.0)
```

In [10]:

```python
def get_energy(PAON,Street,postcode):
    record=[]
    headers = {
        'Accept': 'text/csv',
        'Authorization': 'Basic YW5rdXNoYmFiYmFyMEBnbWFpbC5jb206M2E2ZDBmNTdlMWM1ZmViZTU0MWVkY2RjODlmMmE3NmMyN2
    }


    params = {
        'postcode': postcode,
    }
    try:
        response = requests.get('https://epc.opendatacommunities.org/api/v1/domestic/search', params=params, he
        df = pd.read_csv(StringIO(response.text))
        df['address1']=df['address1'].str.upper()
        df1 =df.copy()
        add ='{}, {}'.format(PAON,Street)
        df=df[df.address1==add]
        if len(df) ==0:
            add1 ='{} {}'.format(PAON,Street)
            df1=df1[df1.address1==add1]
            if len(df1)== 0:
                current_energy_rating ='0'
                construction_age_band =0
                total_floor_area =0
                flat_storey_count =0
                energy_tariff ='0'
                return (postcode,current_energy_rating,construction_age_band,total_floor_area,flat_storey_count

            else:
                pos =postcode
                current_energy_rating =df1['current-energy-rating'].values[-1]
                construction_age_band =df1['construction-age-band'].values[-1]
                total_floor_area =df1['total-floor-area'].values[-1]
                flat_storey_count =df1['flat-storey-count'].values[-1]
                energy_tariff =df1['energy-tariff'].values[-1]
                return (postcode,current_energy_rating,construction_age_band,total_floor_area,flat_storey_count
        else:
            current_energy_rating =df['current-energy-rating'].values[-1]
            construction_age_band =df['construction-age-band'].values[-1]
            total_floor_area =df['total-floor-area'].values[-1]
            flat_storey_count =df['flat-storey-count'].values[-1]
            energy_tariff =df['energy-tariff'].values[-1]

            return (postcode,current_energy_rating,construction_age_band,total_floor_area,flat_storey_count,ene
    except:
        pass
```

In [ ]:

```python
#getting the information from the API
lst_values=[]
for PAON,street,postcode in zip(df_2022['PAON'],df_2022['Street'],df_2022['Postcode']):
    tupvalues =get_energy(PAON=PAON,Street=street,postcode=postcode)
    lst_values.append(tupvalues)
```

In [ ]:

```python
#creating a dataframe
headers = ['Postcode1','current_energy_rating','construction_age_band','total_floor_area','flat_storey_count','
df_epc = pd.DataFrame(lst_values,columns=headers)
```

In [13]:

```python
#checking the head of dataset
df_epc.head()
```

Out[13]:

	Postcode1	current_energy_rating	construction_age_band	total_floor_area	flat_storey_count	energy_tariff
0	NG20 8XN	C	NO DATA!	85.0	NaN	standard tariff
1	NG6 0AX	E	England and Wales: 1900-1929	79.0	NaN	dual
2	NG24 1FR	0	0	0.0	0.0	0
3	NG17 2TR	0	0	0.0	0.0	0
4	NG9 4DP	D	England and Wales: 1950-1966	75.0	NaN	Single

In [171…

```python
len(df_epc)
```

Out[171…

```
231624
```

In [11]:

```python
df_epc = pd.read_excel(r'EPC.xlsx')
df_epc.drop(['Unnamed: 0'],axis=1,inplace=True)
df_epc.head()
```

Out[11]:

	Postcode1	current_energy_rating	construction_age_band	total_floor_area	flat_storey_count	energy_tariff
0	NG20 8XN	C	NO DATA!	85.0	NaN	standard tariff
1	NG6 0AX	E	England and Wales: 1900-1929	79.0	NaN	dual
2	NG24 1FR	0	0	0.0	0.0	0
3	NG17 2TR	0	0	0.0	0.0	0
4	NG9 4DP	D	England and Wales: 1950-1966	75.0	NaN	Single

In [ ]:

```python
#saving the file into system
df_epc.to_excel('EPC.xlsx')
```

In [12]:

```python
#merging both dataframes
df_final = pd.concat([df_2022, df_epc], axis=1)
```

In [13]:

```python
df_final
```

Out[13]:

	sno	Transaction unique identifier	Price	Date of Transfer	Postcode	Property Type	Old/New	Duration	PAON	SAON	...	County
0	1	{E073986B-94F0-2134-E053-6C04A8C0233B}	141000	2022-02-04	NG20 8XN	S	N	F	8	NaN	...	NOTTINGHAMSHIRE
1	2	{E073986B-94F2-2134-E053-6C04A8C0233B}	90000	2022-02-22	NG6 0AX	T	N	F	16	NaN	...	CITY OF NOTTINGHAM
2	3	{E073986B-94F3-2134-E053-6C04A8C0233B}	250000	2022-02-04	NG24 1FR	T	N	F	RIVER VIEW	28	...	NOTTINGHAMSHIRE
3	4	{E073986B-94F5-2134-E053-6C04A8C0233B}	160000	2022-04-25	NG17 2TR	S	N	F	41	NaN	...	NOTTINGHAMSHIRE
4	5	{E073986B-94F9-2134-E053-6C04A8C0233B}	315000	2022-02-28	NG9 4DP	S	N	F	17	NaN	...	NOTTINGHAMSHIRE
...	...	...	...	...	...	...	...	...	...	...	...	...
231619	231620	{E2D14905-6FCB-4C2D-E053-6B04A8C0422B}	240110	2022-04-29	GL51 6UJ	T	N	F	10	NaN	...	GLOUCESTERSHIRE
231620	231621	{E2D14905-6FCC-4C2D-E053-6B04A8C0422B}	695000	2022-05-25	BS36 2EW	D	N	F	239	NaN	...	SOUTH GLOUCESTERSHIRE
231621	231622	{E2D14905-6FCE-4C2D-E053-6B04A8C0422B}	637500	2022-06-06	GL52 3DU	D	N	F	60	NaN	...	GLOUCESTERSHIRE
231622	231623	{E2D14905-6FD0-4C2D-E053-6B04A8C0422B}	250000	2022-06-10	GL50 4EP	T	N	F	3	NaN	...	GLOUCESTERSHIRE
231623	231624	{E2D14905-6FD1-4C2D-E053-6B04A8C0422B}	470000	2022-05-25	GL11 4BZ	D	N	F	3	NaN	...	GLOUCESTERSHIRE

231624 rows × 24 columns

In [14]:

```python
#checking the null values
df_final.isna().sum()
```

Out[14]:

```
sno                                      0
Transaction unique identifier            0
Price                                    0
Date of Transfer                         0
Postcode                               293
Property Type                            0
Old/New                                  0
Duration                                 0
PAON                                     0
SAON                                205015
Street                                3565
Locality                            147801
Town/City                                0
District                                 0
County                                   0
PPD Category Type                        0
Record Status - monthly file only        0
Country                                  0
Postcode1                             1668
current_energy_rating                 1668
construction_age_band                 2378
total_floor_area                      1668
flat_storey_count                   171047
energy_tariff                       110156
dtype: int64
```

In [15]:

```python
#dropping non required columns and columns with maximum null values
print("Length of dataset before dropping null values",len(df_final))
df_final.drop(['sno','Transaction unique identifier','PAON','SAON','Street','Locality','Record Status - monthly
print("Length of dataset after dropping null values",len(df_final))
```

```
Length of dataset before dropping null values 231624
Length of dataset after dropping null values 231624
```

In [16]:

```python
#top 5 rows
df_final.head()
```

Out[16]:

	Price	Date of Transfer	Postcode	Property Type	Old/New	Duration	Town/City	District	County	PPD Category Type	Country
0	141000	2022-02-04	NG20 8XN	S	N	F	MANSFIELD	MANSFIELD	NOTTINGHAMSHIRE	A	England
1	90000	2022-02-22	NG6 0AX	T	N	F	NOTTINGHAM	CITY OF NOTTINGHAM	CITY OF NOTTINGHAM	A	England
2	250000	2022-02-04	NG24 1FR	T	N	F	NEWARK	NEWARK AND SHERWOOD	NOTTINGHAMSHIRE	A	England
3	160000	2022-04-25	NG17 2TR	S	N	F	SUTTON-IN-ASHFIELD	ASHFIELD	NOTTINGHAMSHIRE	A	England
4	315000	2022-02-28	NG9 4DP	S	N	F	NOTTINGHAM	BROXTOWE	NOTTINGHAMSHIRE	A	England

In [17]:

```python
##dropping duplicates
print("Length of dataframe before dropping duplicates",len(df_final))
df_final.drop_duplicates(inplace=True)
print("Length of dataframe after dropping duplicates",len(df_final))
```

```
Length of dataframe before dropping duplicates 231624
Length of dataframe after dropping duplicates 231240
```

In [18]:

```python
#dropping the rows where postcode mismatches
print("Number of rows where postcode mismatches: ",len(df_final[df_final.Postcode!=df_final.Postcode1]))
print("Length of dataframe before dropping postcodes",len(df_final))
df_final =df_final[df_final.Postcode==df_final.Postcode1]
print("Length of dataframe after dropping postcodes",len(df_final))
```

```
Number of rows where postcode mismatches:  1620
Length of dataframe before dropping postcodes 231240
Length of dataframe after dropping postcodes 229620
```

In [19]:

```python
#checking the null values
df_final.isna().sum()
```

Out[19]:

```
Price                         0
Date of Transfer              0
Postcode                      0
Property Type                 0
Old/New                       0
Duration                      0
Town/City                     0
District                      0
County                        0
PPD Category Type             0
Country                       0
Postcode1                     0
current_energy_rating         0
construction_age_band       710
total_floor_area              0
energy_tariff            108403
dtype: int64
```

In [20]:

```python
#filling the nan values with unknown
df_final['construction_age_band']=df_final['construction_age_band'].replace(np.nan, "Unknown")
df_final['energy_tariff']=df_final['energy_tariff'].replace(np.nan,"Unknown")
```

In [21]:

```python
#checking null values
df_final.isna().sum()
```

Out[21]:

```
Price                    0
Date of Transfer         0
Postcode                 0
Property Type            0
Old/New                  0
Duration                 0
Town/City                0
District                 0
County                   0
PPD Category Type        0
Country                  0
Postcode1                0
current_energy_rating    0
construction_age_band    0
total_floor_area         0
energy_tariff            0
dtype: int64
```

We have finally got rid of null values

In [22]:

```python
#checking the unique postcodes available
df_final.Postcode.nunique()
```

Out[22]:

```
193081
```

In [23]:

```python
#splitting the postcode into postcode and subcode
df_final[['areacode','subcode']]=df_final.Postcode.str.split(' ',expand=True)
df_final.head()
```

Out[23]:

	Price	Date of Transfer	Postcode	Property Type	Old/New	Duration	Town/City	District	County	PPD Category Type	Country
0	141000	2022-02-04	NG20 8XN	S	N	F	MANSFIELD	MANSFIELD	NOTTINGHAMSHIRE	A	England
1	90000	2022-02-22	NG6 0AX	T	N	F	NOTTINGHAM	CITY OF NOTTINGHAM	CITY OF NOTTINGHAM	A	England
2	250000	2022-02-04	NG24 1FR	T	N	F	NEWARK	NEWARK AND SHERWOOD	NOTTINGHAMSHIRE	A	England
3	160000	2022-04-25	NG17 2TR	S	N	F	SUTTON-IN-ASHFIELD	ASHFIELD	NOTTINGHAMSHIRE	A	England
4	315000	2022-02-28	NG9 4DP	S	N	F	NOTTINGHAM	BROXTOWE	NOTTINGHAMSHIRE	A	England

In [24]:

```python
#dropping the postcode, postcode1
df_final.drop(['Postcode','Postcode1','subcode'],axis=1,inplace=True)
df_final.head()
```

Out[24]:

	Price	Date of Transfer	Property Type	Old/New	Duration	Town/City	District	County	PPD Category Type	Country
0	141000	2022-02-04	S	N	F	MANSFIELD	MANSFIELD	NOTTINGHAMSHIRE	A	England
1	90000	2022-02-22	T	N	F	NOTTINGHAM	CITY OF NOTTINGHAM	CITY OF NOTTINGHAM	A	England
2	250000	2022-02-04	T	N	F	NEWARK	NEWARK AND SHERWOOD	NOTTINGHAMSHIRE	A	England
3	160000	2022-04-25	S	N	F	SUTTON-IN-ASHFIELD	ASHFIELD	NOTTINGHAMSHIRE	A	England
4	315000	2022-02-28	S	N	F	NOTTINGHAM	BROXTOWE	NOTTINGHAMSHIRE	A	England

In [25]:

```python
#dropping the values of construction_age_band where value is no data!

print("Number of rows with no data! in construction_age_band: ",len(df_final[df_final.construction_age_band=='N
print("Length of dataframe before dropping specific values in construction_age_band:",len(df_final))
df_final =df_final[df_final.construction_age_band != 'NO DATA!']
print("Length of dataframe after dropping specific values in construction_age_band:",len(df_final))
```

```
Number of rows with no data! in construction_age_band:  8594
Length of dataframe before dropping specific values in construction_age_band: 229620
Length of dataframe after dropping specific values in construction_age_band: 221026
```

In [26]:

```python
#checking values in current_energy_rating column
df_final.current_energy_rating.value_counts()
```

Out[26]:

```
D    81935
0    58622
C    42142
E    28216
F     5854
B     2502
G     1671
A       84
Name: current_energy_rating, dtype: int64
```

In [27]:

```python
#checking values in current_energy_rating column
df_final.energy_tariff.value_counts()
```

Out[27]:

```
Unknown             113761
Single               61784
0                    32521
dual                  8606
off-peak 7 hour       2642
standard tariff       1084
off-peak 10 hour       533
dual (24 hour)          46
NO DATA!                34
off-peak 18 hour         7
24 hour                  6
5                        2
Name: energy_tariff, dtype: int64
```

In [28]:

```python
#mapping the no data to unknown in energy ratings columns
df_final.loc[df_final["energy_tariff"] == "NO DATA!", "energy_tariff"] = 'Unknown'
df_final.loc[df_final["energy_tariff"] == '5', "energy_tariff"] = 'Unknown'
df_final.loc[df_final["energy_tariff"] == '0', "energy_tariff"] = 'Unknown'
```

In [29]:

```python
#checking values in current_energy_rating column
df_final.energy_tariff.value_counts()
```

Out[29]:

```
Unknown             146318
Single               61784
dual                  8606
off-peak 7 hour       2642
standard tariff       1084
off-peak 10 hour       533
dual (24 hour)          46
off-peak 18 hour         7
24 hour                  6
Name: energy_tariff, dtype: int64
```

In [30]:

```python
#checking the value counts of Property Type column
df_final['Property Type'].value_counts()
```

Out[30]:

```
T    69994
S    61657
D    44649
F    38912
O     5814
Name: Property Type, dtype: int64
```

In [31]:

```python
#checking the value counts of Old/New column
df_final['Old/New'].value_counts()
```

Out[31]:

```
N    219689
Y      1337
Name: Old/New, dtype: int64
```

In [32]:

```python
#checking the value counts of Duration column
df_final['Duration'].value_counts()
```

Out[32]:

```
F    171343
L     49683
Name: Duration, dtype: int64
```

In [33]:

```python
#checking the count of Town/City columns
print("number of unique values :",df_final['Town/City'].nunique())
df_final['Town/City'].value_counts()
```

Out[33]:

```
number of unique values : 1141
LONDON                 14427
MANCHESTER              3493
BIRMINGHAM              3316
BRISTOL                 3156
NOTTINGHAM              3151
                        ...
WESTBURY-ON-SEVERN         1
ARTHOG                     1
LYDBURY NORTH              1
NEWCASTLETON               1
CONISTON                   1
Name: Town/City, Length: 1141, dtype: int64
```

In [34]:

```python
#checking the count of District columns
print("number of unique values :",df_final['District'].nunique())
df_final['District'].value_counts()
```

Out[34]:

```
number of unique values : 331
BIRMINGHAM            3273
LEEDS                 3014
CORNWALL              2355
COUNTY DURHAM         2276
BRADFORD              2089
                      ...
OADBY AND WIGSTON      189
MELTON                 171
RUTLAND                138
CITY OF LONDON          46
ISLES OF SCILLY          5
Name: District, Length: 331, dtype: int64
```

In [35]:

```python
#checking the count of County columns
print("number of unique values :",df_final['County'].nunique())
df_final['County'].value_counts()
```

Out[35]:

```
number of unique values : 112
GREATER LONDON        24835
GREATER MANCHESTER    10054
WEST MIDLANDS          9113
WEST YORKSHIRE         8926
KENT                   6286
                      ...
CEREDIGION              303
BLAENAU GWENT           291
MERTHYR TYDFIL          263
RUTLAND                 138
ISLES OF SCILLY           5
Name: County, Length: 112, dtype: int64
```

In [36]:

```python
#checking the count of PPD Category Type columns
print("number of unique values :",df_final['PPD\xa0Category Type'].nunique())
df_final['PPD\xa0Category Type'].value_counts()
```

Out[36]:

```
number of unique values : 2
A    201171
B     19855
Name: PPD Category Type, dtype: int64
```

In [37]:

```python
#checking the count of current_energy_rating columns
print("number of unique values :",df_final['current_energy_rating'].nunique())
df_final['current_energy_rating'].value_counts()
```

Out[37]:

```
number of unique values : 8
D    81935
0    58622
C    42142
E    28216
F     5854
B     2502
G     1671
A       84
Name: current_energy_rating, dtype: int64
```

In [38]:

```python
#checking the count of construction_age_band columns
print("number of unique values :",df_final['construction_age_band'].nunique())
df_final['construction_age_band'].value_counts()
```

Out[38]:

```
number of unique values : 44
0                                   58622
England and Wales: 1950-1966       27642
England and Wales: 1900-1929       27417
England and Wales: 1930-1949       25643
England and Wales: 1967-1975       20057
England and Wales: before 1900     15185
England and Wales: 1983-1990       11165
England and Wales: 1976-1982        9726
England and Wales: 1996-2002        8356
England and Wales: 1991-1995        6069
England and Wales: 2003-2006        5933
England and Wales: 2007-2011        1807
England and Wales: 2007 onwards     1601
Unknown                              710
2021                                 207
2021                                 129
England and Wales: 2012 onwards      122
INVALID!                              94
2020                                  79
2022                                  78
2019                                  77
2018                                  58
2020                                  58
2022                                  54
2018                                  26
2019                                  17
2017                                  17
2017                                  15
2016                                  13
2013                                  10
2014                                   8
2016                                   7
2014                                   6
2015                                   4
2010                                   3
2012                                   2
2008                                   2
1950                                   1
2013                                   1
1930                                   1
1900                                   1
2015                                   1
2009                                   1
2010                                   1
Name: construction_age_band, dtype: int64
```

In [39]:

```python
#mapping the no data to unknown in construction_age_band columns
df_final.loc[df_final["construction_age_band"] == "INVALID!", "construction_age_band"] = 'Unknown'
df_final.loc[df_final["construction_age_band"] == 0, "construction_age_band"] = 'Unknown'
```

In [40]:

```python
#checking the count of construction_age_band columns
print("number of unique values :",df_final['construction_age_band'].nunique())
df_final['construction_age_band'].value_counts()
```

Out[40]:

```
number of unique values : 42
Unknown                             59426
England and Wales: 1950-1966       27642
England and Wales: 1900-1929       27417
England and Wales: 1930-1949       25643
England and Wales: 1967-1975       20057
England and Wales: before 1900     15185
England and Wales: 1983-1990       11165
England and Wales: 1976-1982        9726
England and Wales: 1996-2002        8356
England and Wales: 1991-1995        6069
England and Wales: 2003-2006        5933
England and Wales: 2007-2011        1807
England and Wales: 2007 onwards     1601
2021                                 207
2021                                 129
England and Wales: 2012 onwards      122
2020                                  79
2022                                  78
2019                                  77
2018                                  58
2020                                  58
2022                                  54
2018                                  26
2019                                  17
2017                                  17
2017                                  15
2016                                  13
2013                                  10
2014                                   8
2016                                   7
2014                                   6
2015                                   4
2010                                   3
2012                                   2
2008                                   2
1950                                   1
2013                                   1
1930                                   1
1900                                   1
2015                                   1
2009                                   1
2010                                   1
Name: construction_age_band, dtype: int64
```

In [41]:

```python
#checking the count of total_floor_area columns
print("number of unique values :",df_final['total_floor_area'].nunique())
df_final['total_floor_area'].value_counts()
```

Out[41]:

```
number of unique values : 10098
0.00      58630
83.00      2645
84.00      2617
78.00      2610
86.00      2573
          ...
129.50        1
86.97         1
207.81        1
56.01         1
123.77        1
Name: total_floor_area, Length: 10098, dtype: int64
```

In [42]:

```python
#checking the count of energy_tariff columns
print("number of unique values :",df_final['energy_tariff'].nunique())
df_final['energy_tariff'].value_counts()
```

Out[42]:

```
number of unique values : 9
Unknown             146318
Single               61784
dual                  8606
off-peak 7 hour       2642
standard tariff       1084
off-peak 10 hour       533
dual (24 hour)          46
off-peak 18 hour         7
24 hour                  6
Name: energy_tariff, dtype: int64
```

In [43]:
```
#checking the count of areacode columns
print("number of unique values :",df_final['areacode'].nunique())
df_final['areacode'].value_counts()
```

```
number of unique values : 2265
```

Out[43]:
```
CR0     489
BN3     430
BN1     387
LE2     385
NG5     383
       ...
WC2N      1
MK1       1
B4        1
EC3R      1
LA21      1
Name: areacode, Length: 2265, dtype: int64
```

In [44]:
```
df_final[['name','construction_age_band']]=df_final.construction_age_band.str.split(':',expand=True)
```

In [45]:
```
df_final['construction_age_band'].value_counts()
```

Out[45]:
```
 1950-1966       27642
 1900-1929       27417
 1930-1949       25643
 1967-1975       20057
 before 1900     15185
 1983-1990       11165
 1976-1982        9726
 1996-2002        8356
 1991-1995        6069
 2003-2006        5933
 2007-2011        1807
 2007 onwards     1601
 2012 onwards      122
Name: construction_age_band, dtype: int64
```

In [46]:
```
#dropping name column
df_final.drop('name',axis=1,inplace=True)
print("Dropped column")
```

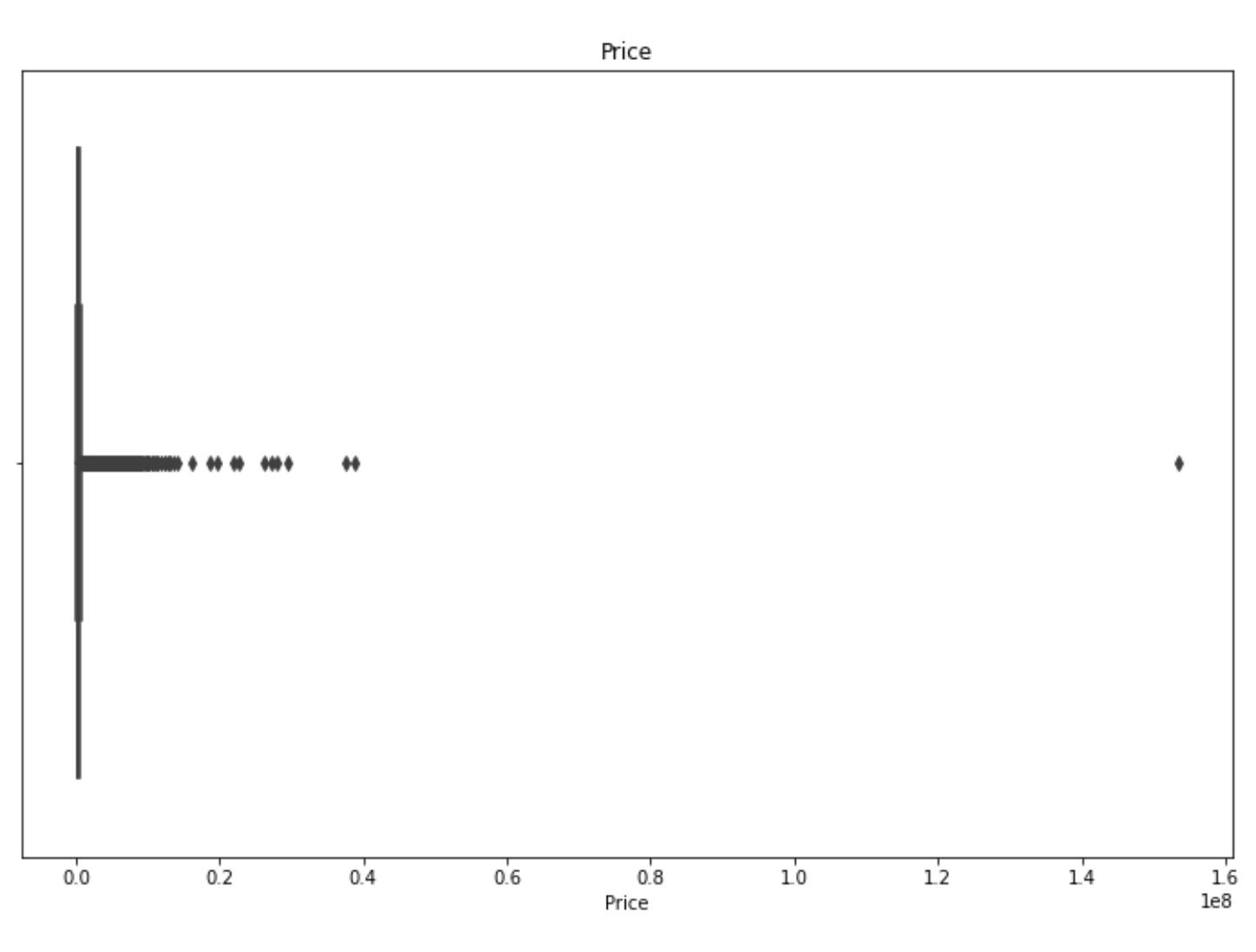
```
Dropped column
```

In [47]:
```
# checking the datatypes of dataset
df_final.dtypes
```

Out[47]:
```
Price                               int64
Date of Transfer           datetime64[ns]
Property Type                      object
Old/New                            object
Duration                           object
Town/City                          object
District                           object
County                             object
PPD Category Type                  object
Country                            object
current_energy_rating              object
construction_age_band              object
total_floor_area                  float64
energy_tariff                      object
areacode                           object
dtype: object
```

In [48]:
```
#plotting a boxplot for price
plt.figure(figsize=(12,8))
sns.boxplot(x = df_final['Price'])
plt.title('Price')
```

Out[48]:
```
Text(0.5, 1.0, 'Price')
```



We can see there are outliers, thus dropping the outliers

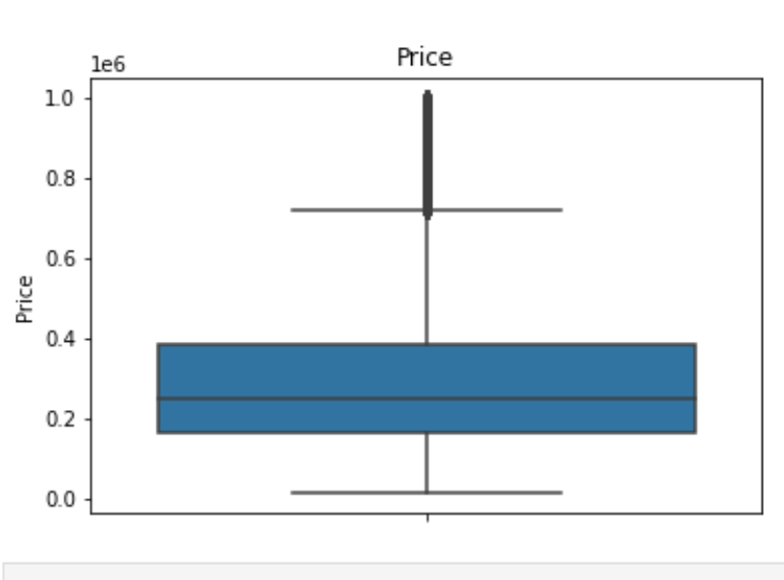
In [49]:
```
#checking the max, min and median prices
df_final.Price.max(), df_final.Price.min(), df_final.Price.median()
```

Out[49]:
```
(153450000, 100, 260000.0)
```

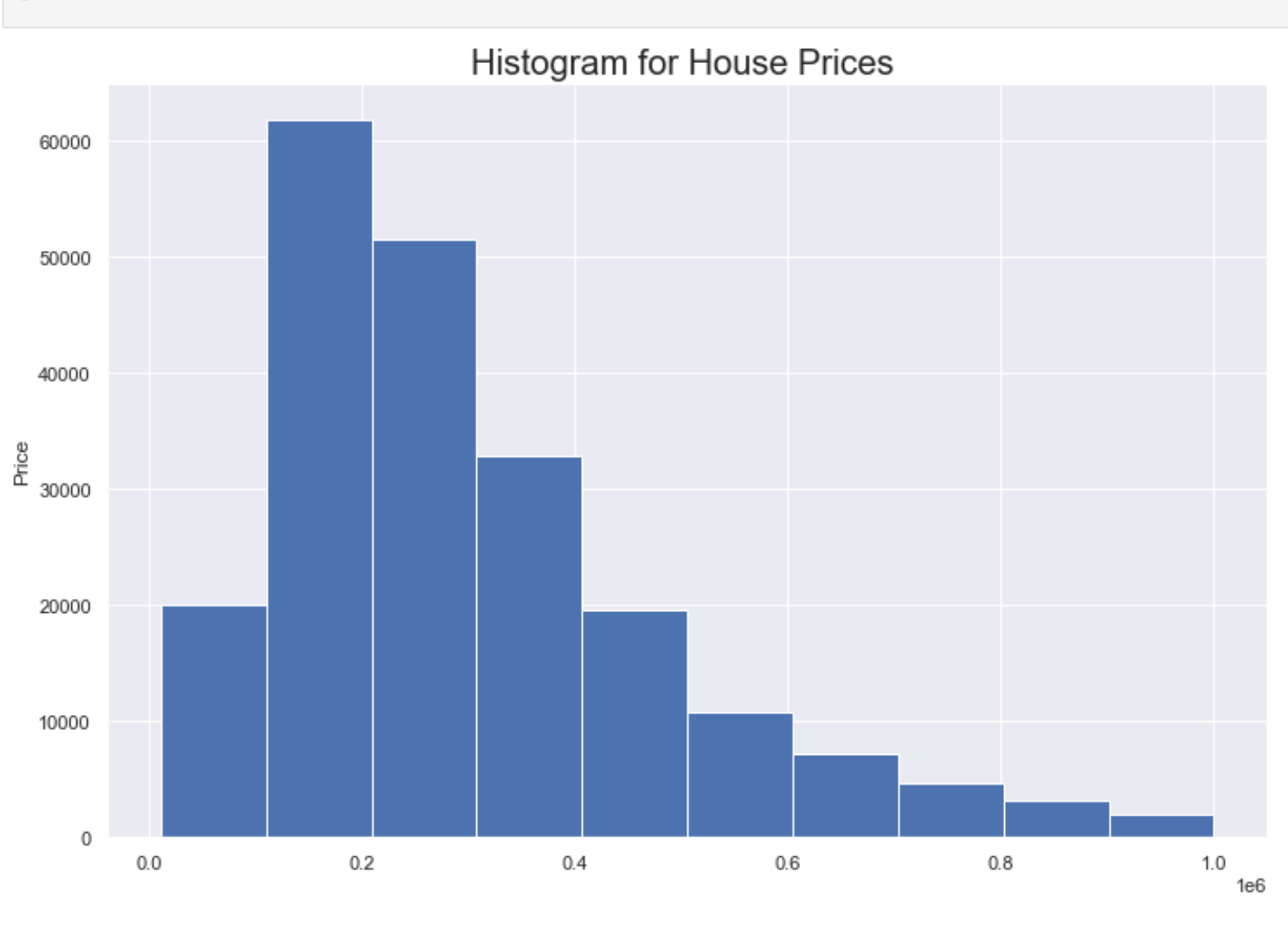
In [50]:
```
print("Length of dataset before dropping outliers" ,len(df_final))
df_final = df_final.loc[(df_final['Price'] < 1000000) & (df_final['Price'] > 10000)]
print("Length of dataset after dropping outliers" ,len(df_final))
sns.boxplot(y = df_final['Price'])
plt.title('Price')
```

```
Length of dataset before dropping outliers 221026
Length of dataset after dropping outliers 213440
```

Out[50]:
```
Text(0.5, 1.0, 'Price')
```

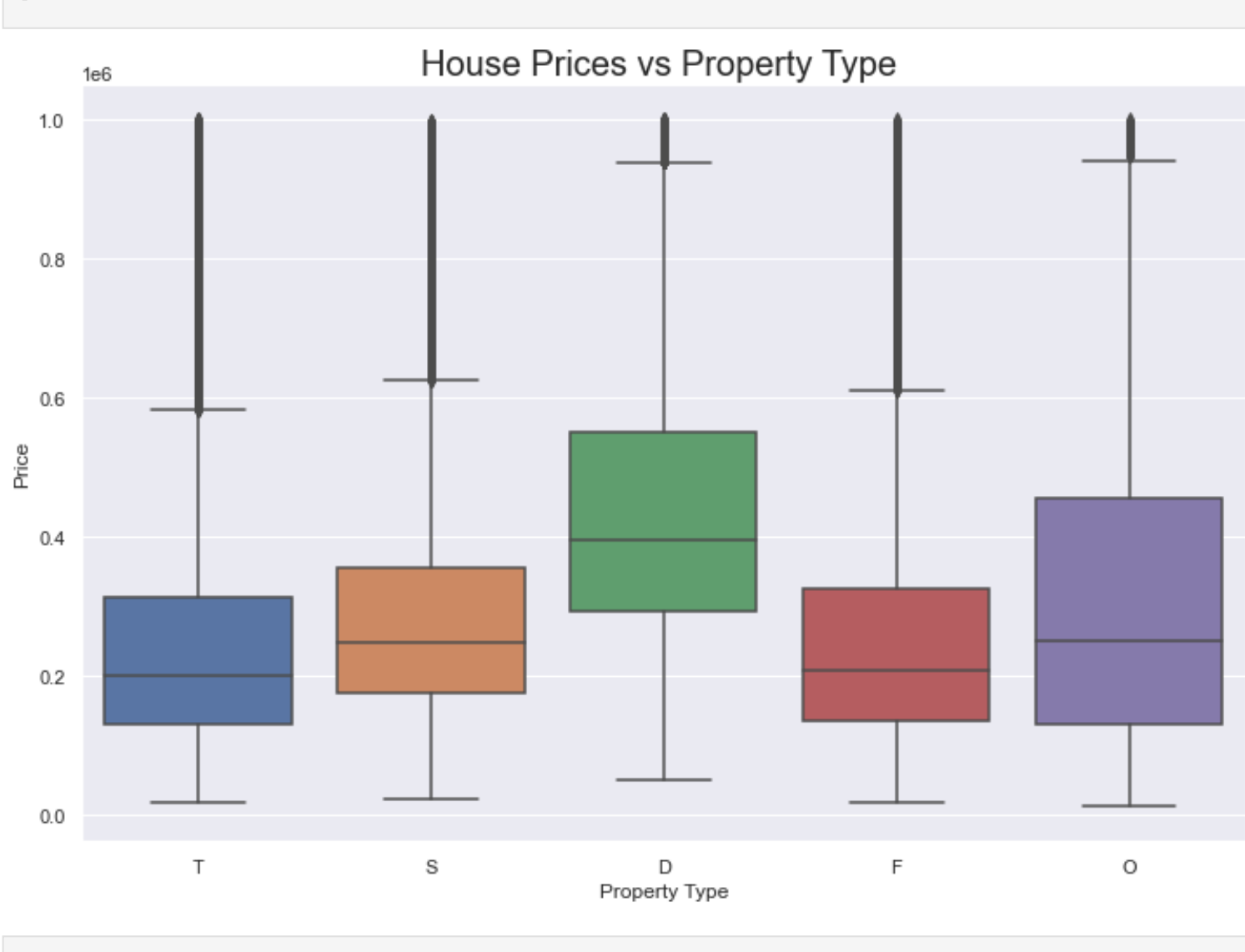


In [51]:
```
#plotting the histogram for house prices
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
df_final['Price'].hist()
plt.title("Histogram for House Prices ",fontdict={'fontsize':20})
plt.ylabel('Price')
plt.show()
```

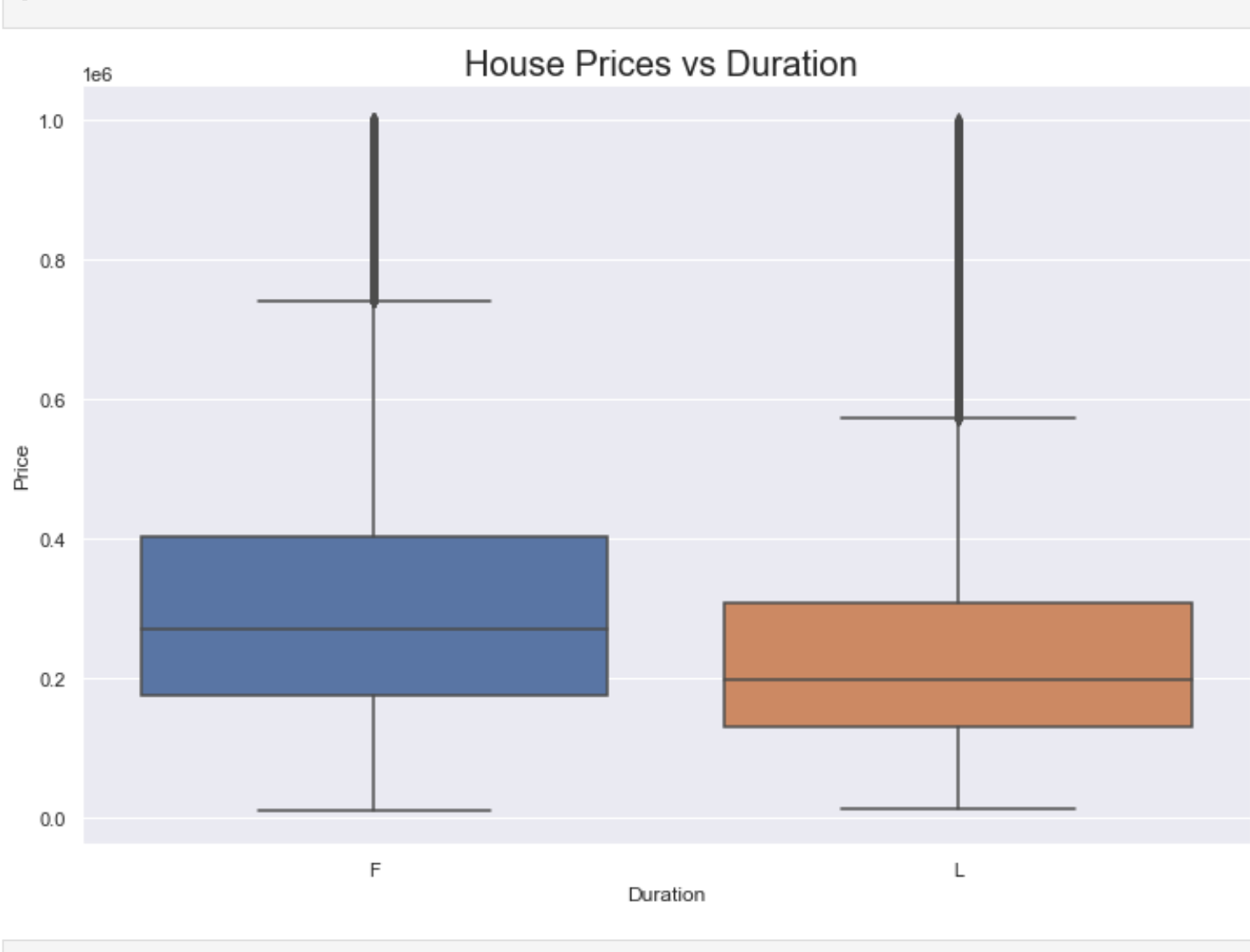


We can see that the prices postive skewed(right-skewed distribution)

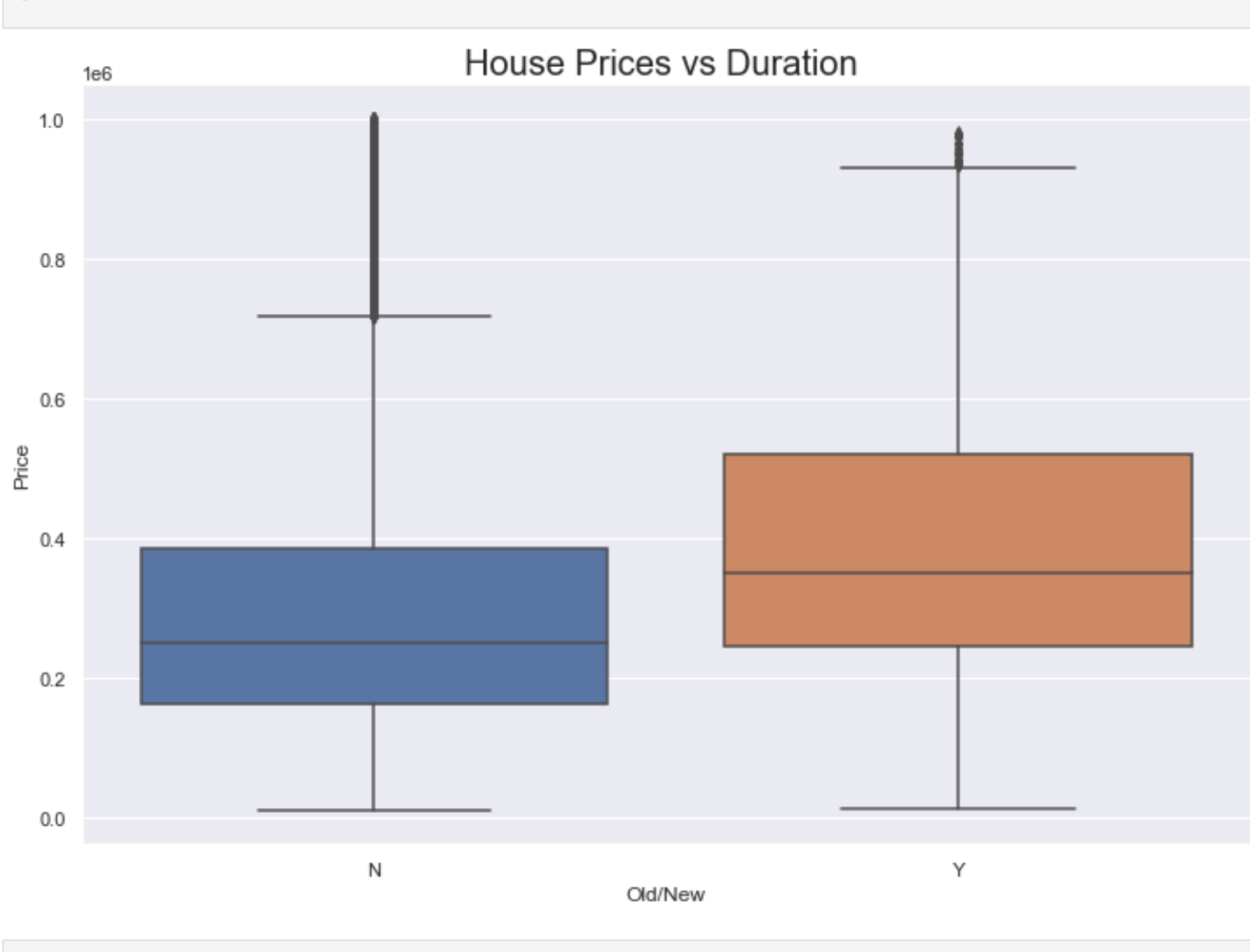
In [52]:
```
#plotting the prices with type of property

sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.boxplot(data=df_final ,y ='Price',x='Property Type')
plt.title("House Prices vs Property Type ",fontdict={'fontsize':20})
plt.show()
```



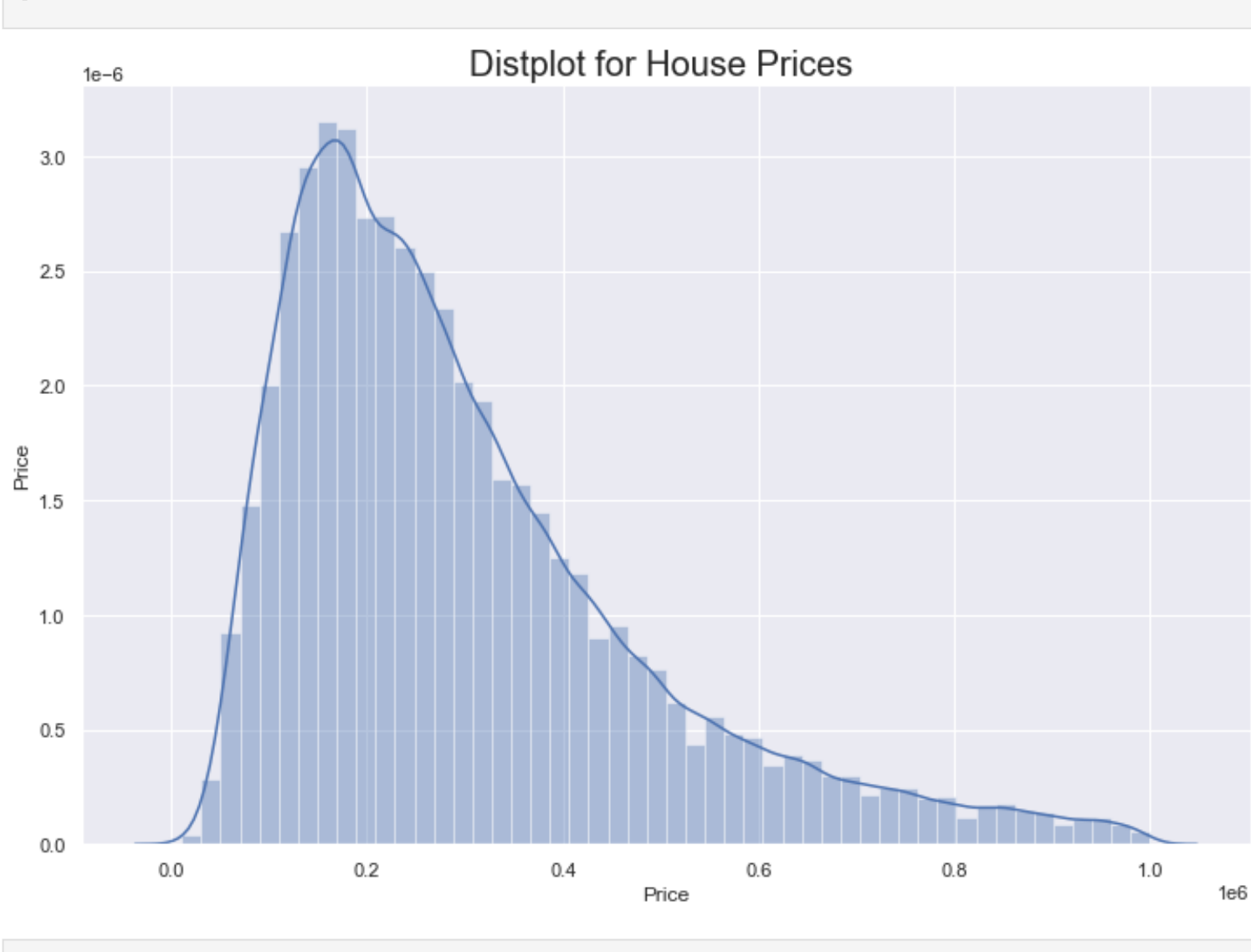
In [53]:
```
#plotting the prices with Duration
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.boxplot(data=df_final ,y ='Price',x='Duration')
plt.title("House Prices vs Duration ",fontdict={'fontsize':20})
plt.show()
```



In [54]:
```
#plotting the prices with Duration
sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.boxplot(data=df_final ,y ='Price',x='Old/New')
plt.title("House Prices vs Duration ",fontdict={'fontsize':20})
plt.show()
```

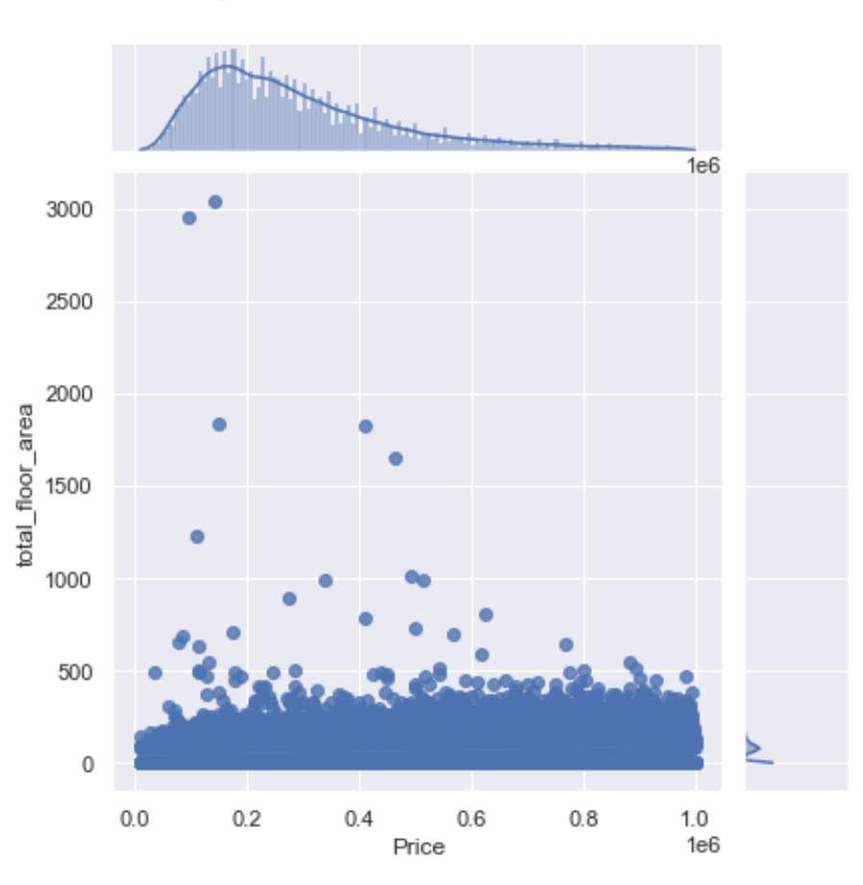


In [55]:
```
#creating a distplot for price

sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.distplot(df_final['Price'])
plt.title("Distplot for House Prices ",fontdict={'fontsize':20})
plt.ylabel('Price')
plt.show()
```

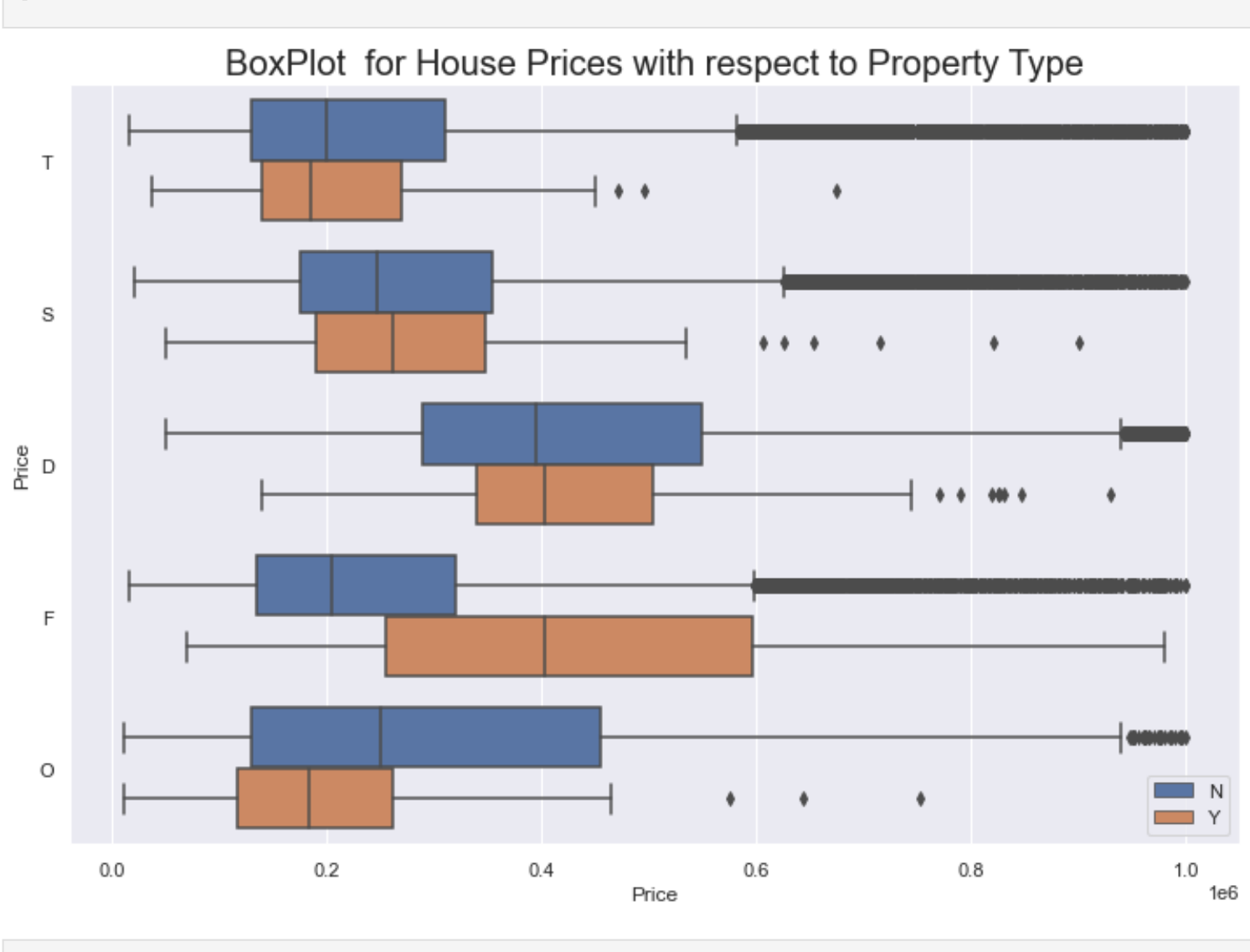


In [56]:
```
#jointplot allows to study relationship between 2 features - price and square footage

sns.jointplot(x='Price',y='total_floor_area',data=df_final,kind='reg')
```

Out[56]:
```
<seaborn.axisgrid.JointGrid at 0x2995ee4cca0>
```



In [57]:
```
#use data based on some categorical feature use 'hue' parameter

sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.boxplot(x='Price',y='Property Type',hue='Old/New',data=df_final)
plt.title("BoxPlot  for House Prices with respect to Property Type",fontdict={'fontsize':20})
plt.ylabel('Price')
plt.legend()
plt.show()
```

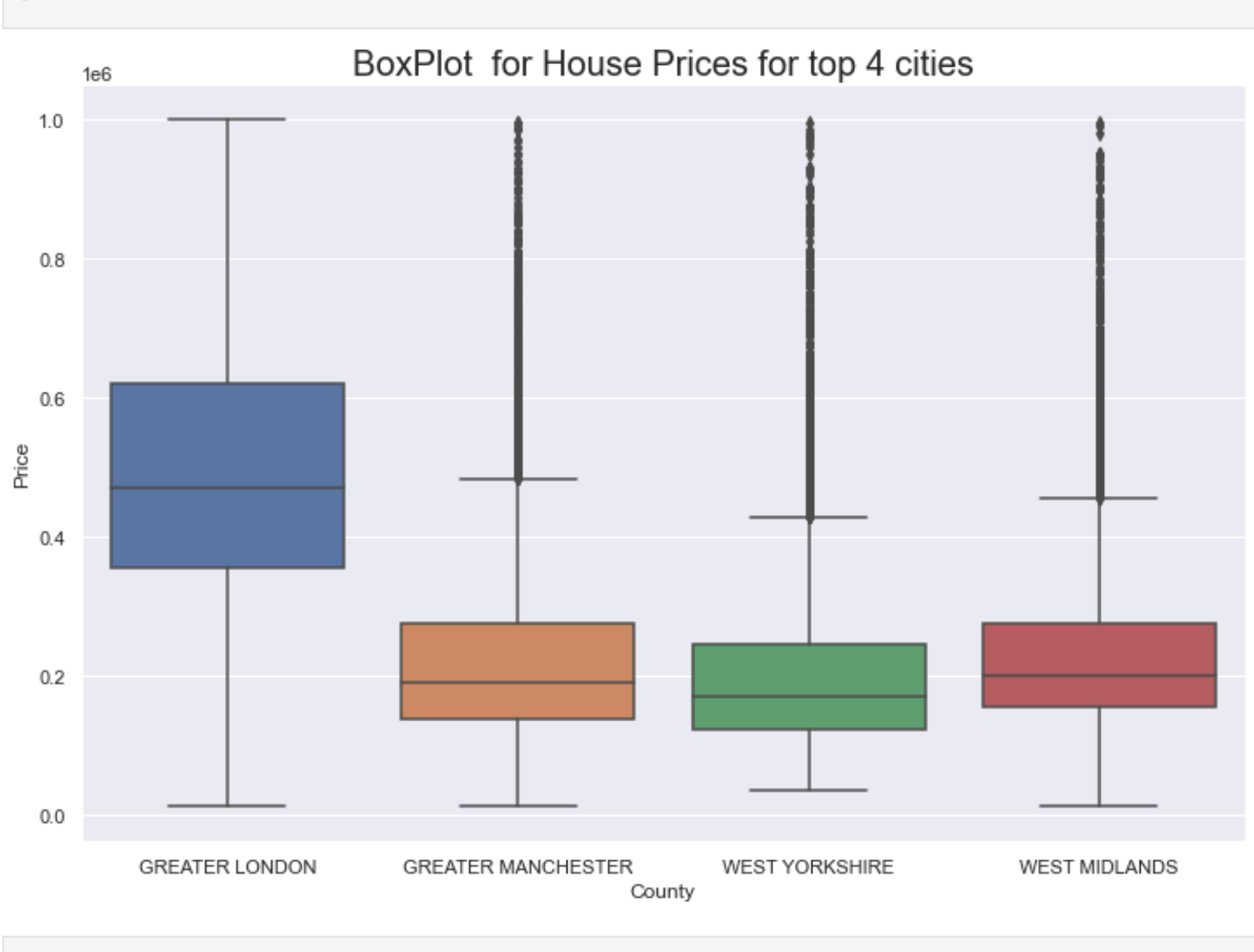


In [58]:
```
#checking the different counties data
df_final['County'].value_counts()
```

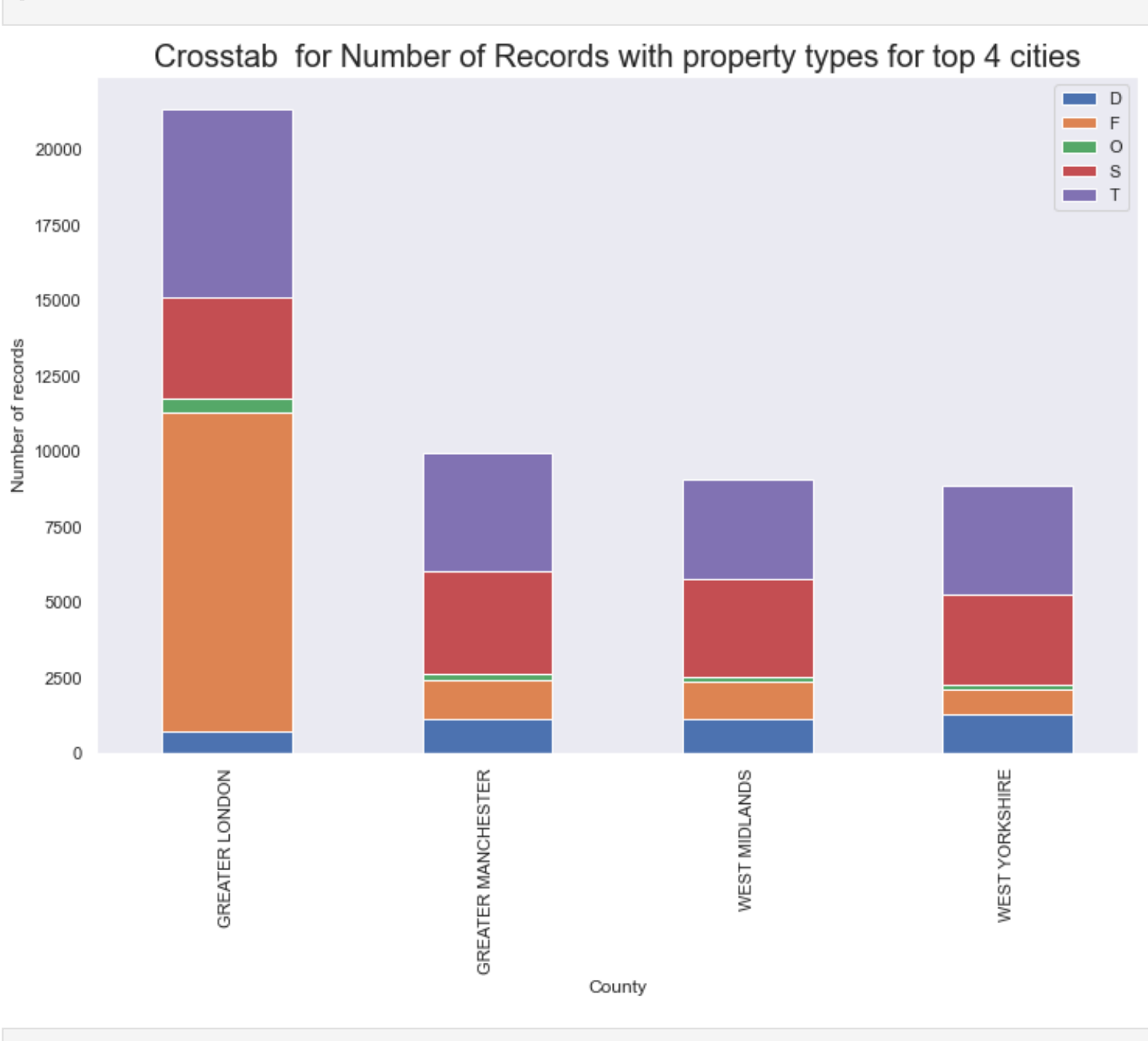
Out[58]:
```
GREATER LONDON        21355
GREATER MANCHESTER     9959
WEST MIDLANDS          9062
WEST YORKSHIRE         8873
KENT                   6093
                      ...
CEREDIGION              300
BLAENAU GWENT           291
MERTHYR TYDFIL          263
RUTLAND                 137
ISLES OF SCILLY           5
Name: County, Length: 112, dtype: int64
```

In [59]:
```
#creating a temp dataframe with top 4 records city
temp1=df_final[df_final['County']=='GREATER LONDON']
temp2=df_final[df_final['County']=='GREATER MANCHESTER']
temp3=df_final[df_final['County']=='WEST YORKSHIRE']
temp4=df_final[df_final['County']=='WEST MIDLANDS']
temp = pd.concat([temp1,temp2,temp3,temp4])
```

In [60]:
```
#comparing top 4 records cities

sns.set_theme(style="darkgrid")
plt.figure(figsize=(12,8))
sns.boxplot(y='Price',x='County',data=temp)
plt.title("BoxPlot  for House Prices for top 4 cities",fontdict={'fontsize':20})
plt.ylabel('Price')
plt.show()
```



In [61]:
```
#creating a crosstab for cities with property type
sns.set_theme(style="darkgrid")
temp1 = pd.crosstab(temp['County'], temp['Property Type'])
temp1.plot(kind = 'bar', stacked = True, grid = False,figsize=(12,8))
plt.title("Crosstab  for Number of Records with property types for top 4 cities",fontdict={'fontsize':20})
plt.ylabel('Number of records')
plt.legend()
plt.show()
```

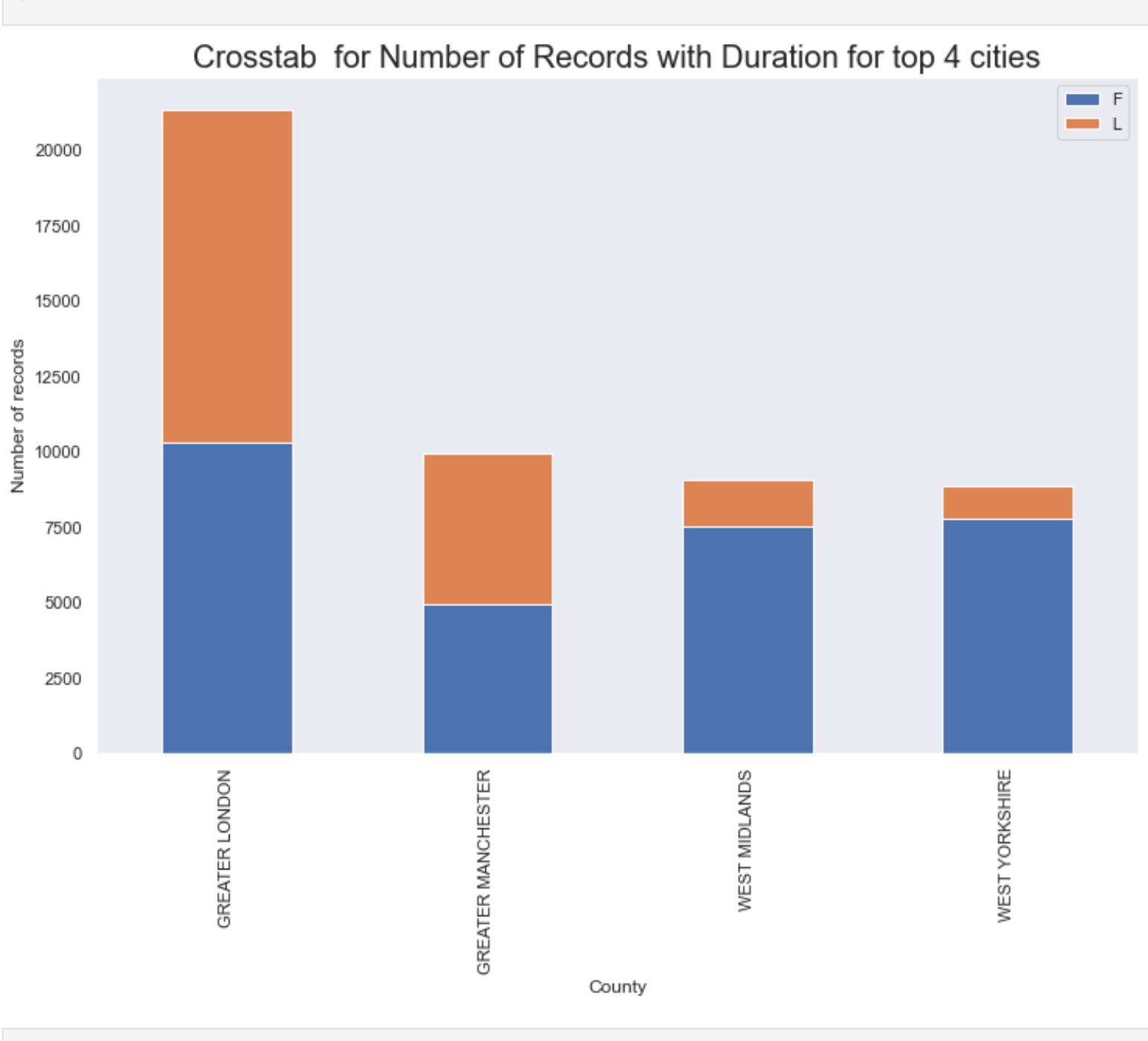


In [62]:
```
#crosstab for property type with county for top 4 cities
temp1
```

Out[62]:

Property Type	D	F	O	S	T
County					
GREATER LONDON	687	10598	437	3373	6260
GREATER MANCHESTER	1128	1263	223	3416	3929
WEST MIDLANDS	1141	1223	144	3230	3324
WEST YORKSHIRE	1299	790	163	3009	3612

In [63]:
```
#creating a crosstab for cities with property type
sns.set_theme(style="darkgrid")
temp1 = pd.crosstab(temp['County'], temp['Duration'])
temp1.plot(kind = 'bar', stacked = True, grid = False,figsize=(12,8))
plt.title("Crosstab  for Number of Records with Duration for top 4 cities",fontdict={'fontsize':20})
plt.ylabel('Number of records')
plt.legend()
plt.show()
```

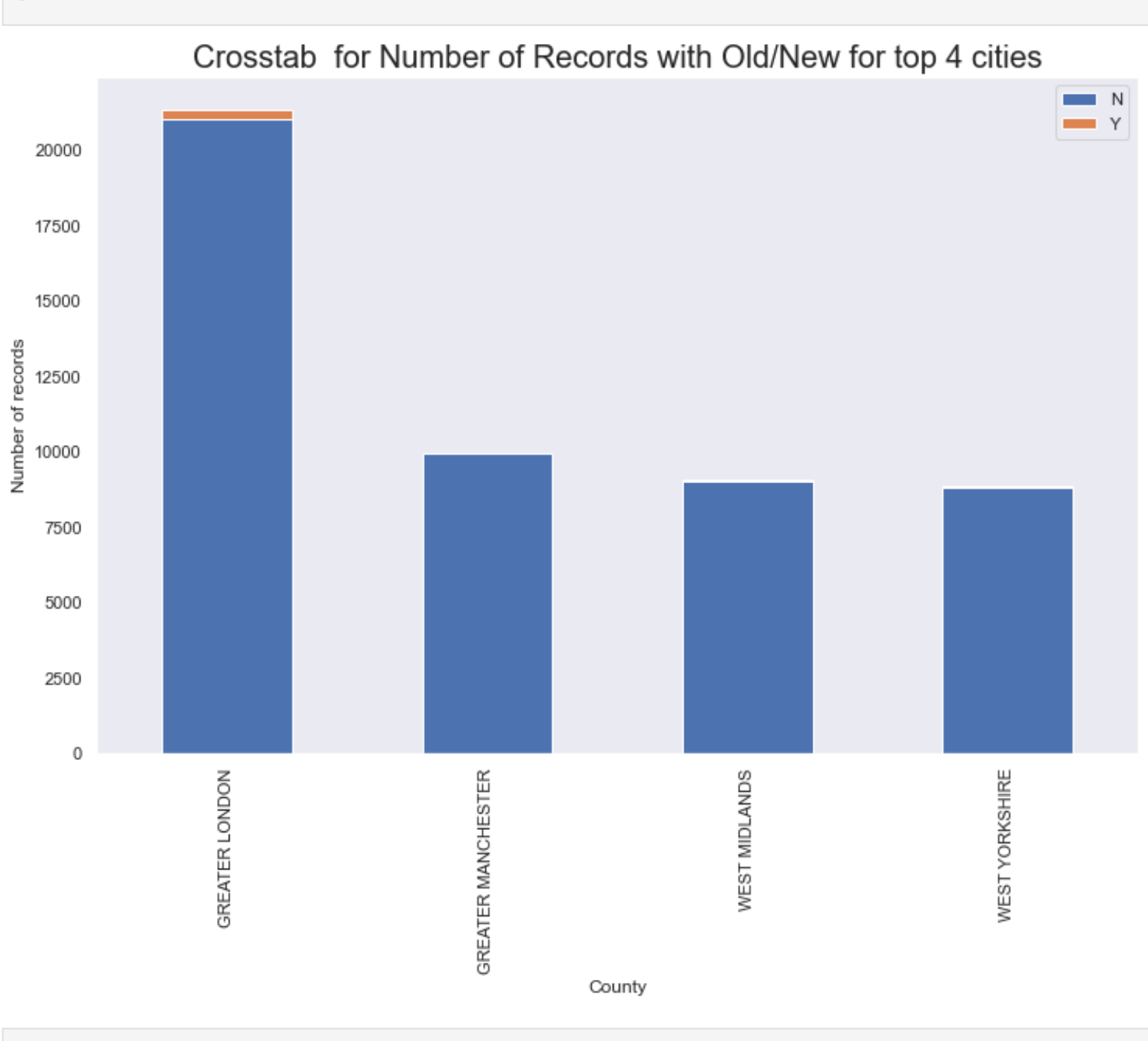


In [64]:
```
#crosstab for property type with county for top 4 cities
temp1
```

Out[64]:

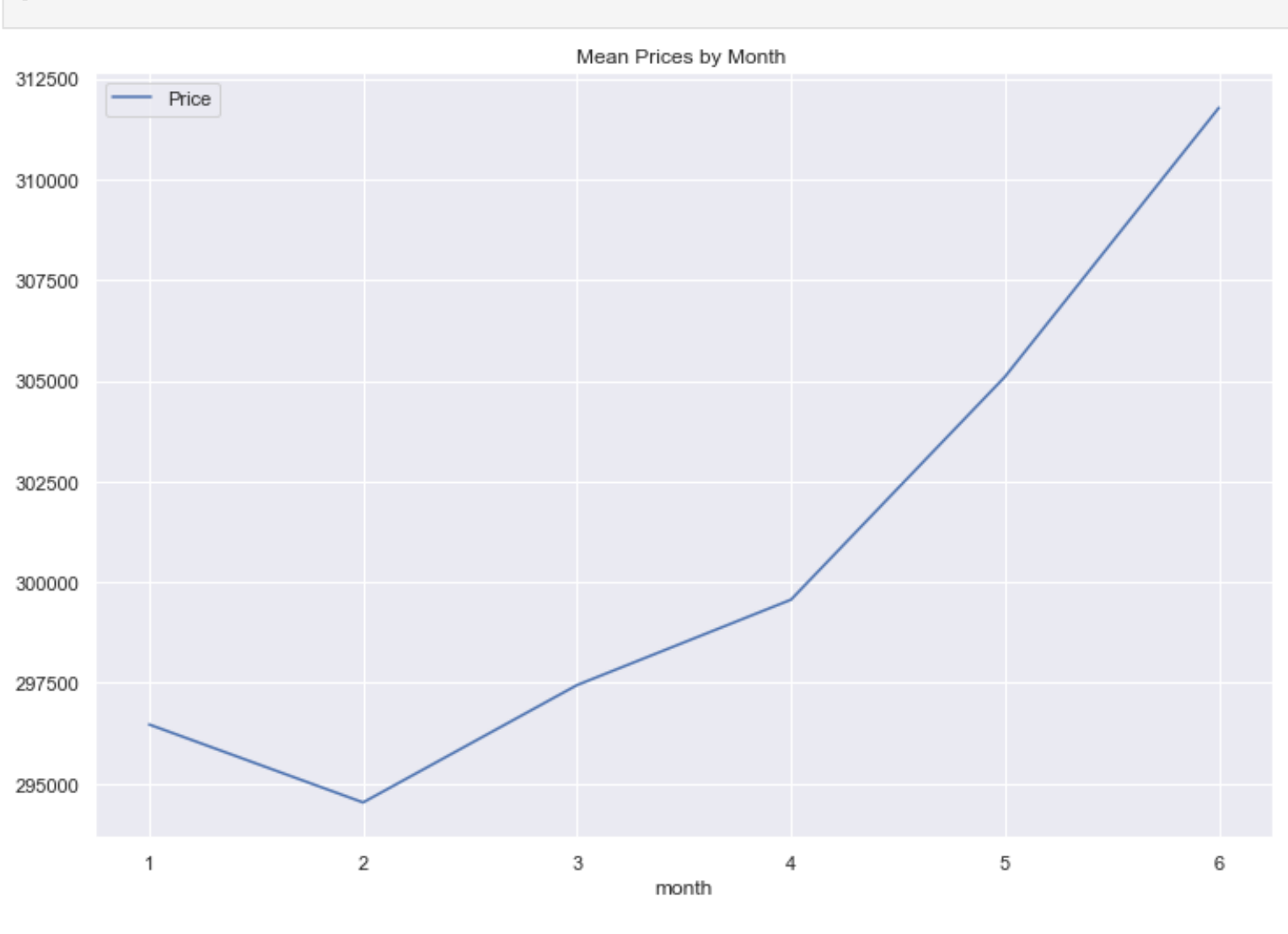
Duration	F	L
County		
GREATER LONDON	10297	11058
GREATER MANCHESTER	4927	5032
WEST MIDLANDS	7530	1532
WEST YORKSHIRE	7770	1103

In [65]:
```
#creating a crosstab for cities with property type
sns.set_theme(style="darkgrid")
temp1 = pd.crosstab(temp['County'], temp['Old/New'])
temp1.plot(kind = 'bar', stacked = True, grid = False,figsize=(12,8))
plt.title("Crosstab  for Number of Records with Old/New for top 4 cities",fontdict={'fontsize':20})
plt.ylabel('Number of records')
plt.legend()
plt.show()
```



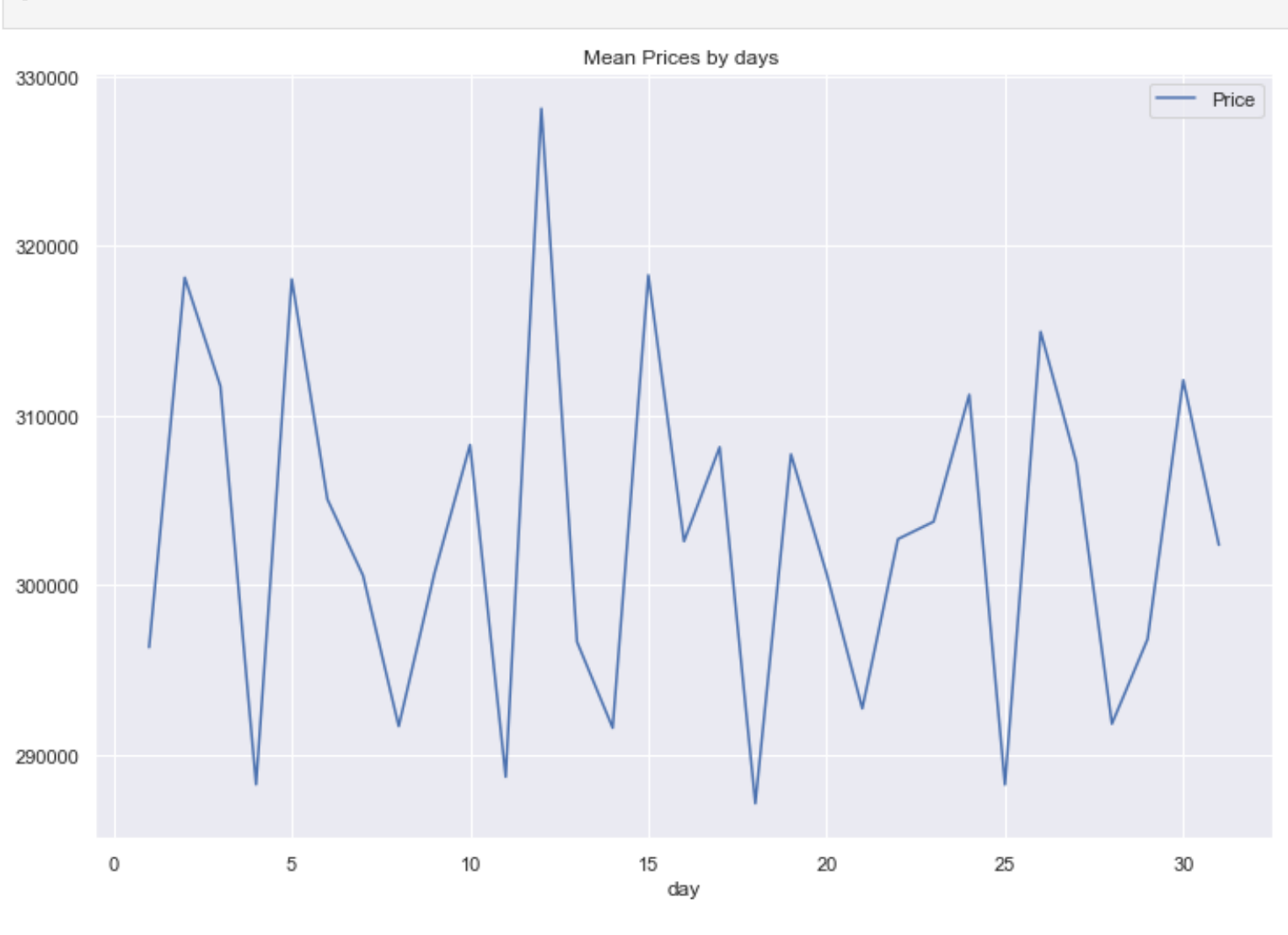
In [66]:
```
#spliting the date
df_final['day'] = df_final['Date of Transfer'].dt.day
df_final['month'] = df_final['Date of Transfer'].dt.month
df_final['year'] = df_final['Date of Transfer'].dt.year
```

In [67]:
```
#grouping the prices by month
temp_month=df_final.groupby(['month'])['Price'].mean().reset_index()
temp_month.set_index('month', inplace=True)
temp_month.plot(figsize=(12,8))
plt.title("Mean Prices by Month ")
plt.show()
```



We can see from the above plot that June month has the Highest prices , while Feb has the lowest prices.

In [68]:
```
#grouping the prices by day
temp_month=df_final.groupby(['day'])['Price'].mean().reset_index()
temp_month.set_index('day', inplace=True)
temp_month.plot(figsize=(12,8))
plt.title("Mean Prices by days ")
plt.show()
```



We can see from the above chart that the mid days of the months are the highest Prices.

In [69]:
```
#checking the values for town/city column
df_final['Town/City'].value_counts()
```

Out[69]:
```
LONDON            11580
MANCHESTER         3470
BIRMINGHAM         3296
NOTTINGHAM         3131
BRISTOL            3069
                  ...
BAMBURGH              1
ARTHOG                1
LYDBURY NORTH         1
NEWCASTLETON          1
CONISTON              1
Name: Town/City, Length: 1141, dtype: int64
```

In [70]:
```
#creating a list of cities
citieslist =list(df_final['County'].unique())
```

In [71]:
```
#uploading the dataset of population and converting the county into rank as per population
pop = pd.read_csv(r'city.csv',index_col=False)
pop.head()
```

Out[71]:

	Rank	city	population
0	113	GREATER LONDON	8799800
1	112	WEST MIDLANDS	5950800
2	111	GREATER MANCHESTER	2867800
3	110	WEST YORKSHIRE	2351600
4	109	KENT	1576100

In [72]:
```
#ranking has been given as reverse , as it would give more weightage to the cities with higher population
map_dict =dict(zip(pop.city,pop.Rank))
df_final['county_rank'] = df_final['County'].map(map_dict)
```

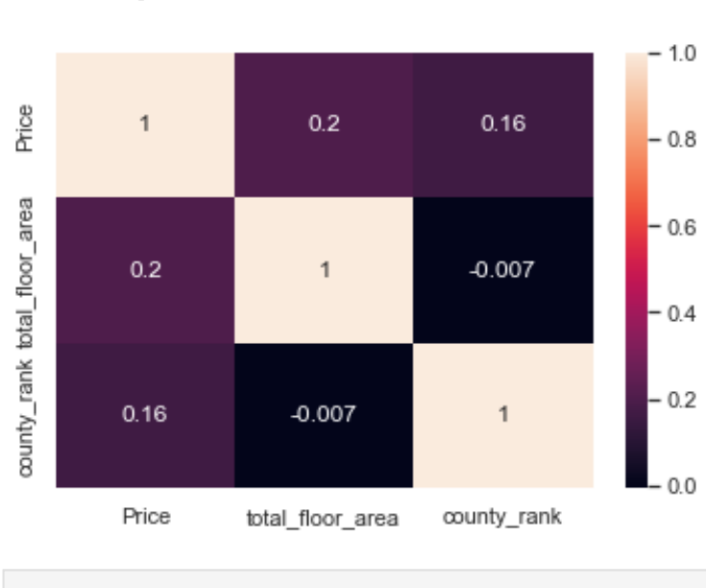
In [73]:
```
df_final.head()
```

Out[73]:

	Price	Date of Transfer	Property Type	Old/New	Duration	Town/City	District	County	PPD Category Type	Country	current_energy_ra
1	90000	2022-02-22	T	N	F	NOTTINGHAM	CITY OF NOTTINGHAM	CITY OF NOTTINGHAM	A	England	
2	250000	2022-02-04	T	N	F	NEWARK	NEWARK AND SHERWOOD	NOTTINGHAMSHIRE	A	England	
3	160000	2022-04-25	S	N	F	SUTTON-IN-ASHFIELD	ASHFIELD	NOTTINGHAMSHIRE	A	England	
4	315000	2022-02-28	S	N	F	NOTTINGHAM	BROXTOWE	NOTTINGHAMSHIRE	A	England	
6	115000	2022-04-29	T	N	F	RETFORD	BASSETLAW	NOTTINGHAMSHIRE	A	England	

In [74]:
```
#find the correlation between numerical variables
sns.heatmap(df_final[['Price','total_floor_area','county_rank']].corr(),annot=True)
```

Out[74]:
```
<AxesSubplot:>
```



In [75]:
```
df_final.dtypes
```

Out[75]:
```
Price                               int64
Date of Transfer           datetime64[ns]
Property Type                      object
Old/New                            object
Duration                           object
Town/City                          object
District                           object
County                             object
PPD Category Type                  object
Country                            object
current_energy_rating              object
construction_age_band              object
total_floor_area                  float64
energy_tariff                      object
areacode                           object
day                                 int64
month                               int64
year                                int64
county_rank                         int64
dtype: object
```

In [76]:
```
lbl_cols = ['Duration','Old/New','PPD\xa0Category Type']
df_final[lbl_cols] = df_final[lbl_cols].apply(LabelEncoder().fit_transform)
```

In [77]:
```
#calculating the pount biserial relationship between price and duration
stats.pointbiserialr(df_final['Duration'], df_final['Price'])
```

Out[77]:
```
PointbiserialrResult(correlation=-0.15758437984589602, pvalue=0.0)
```

Since the correlation coefficient is negative, this indicates that when the variable Duration takes on the value “1” that the variable y tends to take on lower values compared to when the variable Duration takes on the value “0.”

Since the p-value of this correlation is less than .05, this correlation is statistically significant.

In [78]:
```
#calculating the pount biserial relationship between price and Old/New
stats.pointbiserialr(df_final['Old/New'], df_final['Price'])
```

Out[78]:
```
PointbiserialrResult(correlation=0.03814508614481268, pvalue=1.4709251984730342e-69)
```

Since the correlation coefficient is positive, this indicates that when the variable old/new takes on the value “1” that the variable y tends to take on higher values compared to when the variable Duration takes on the value “0.”

Since the p-value of this correlation is less than .05, this correlation is statistically significant.

In [79]:
```
#calculating the pount biserial relationship between price and PPD category type
stats.pointbiserialr(df_final['PPD\xa0Category Type'], df_final['Price'])
```

Out[79]:
```
PointbiserialrResult(correlation=-0.09294680132913051, pvalue=0.0)
```

Since the correlation coefficient is negative, this indicates that when the variable PPD category type takes on the value “1” that the variable y tends to take on lower values compared to when the variable Duration takes on the value “0.”

Since the p-value of this correlation is less than .05, this correlation is statistically significant.

Using ANOVA tests

Null hypothesis(H0): The variables are not correlated with each other
Alternate hypothesis (HA) : The variables are correlated withe ach other

P-value: The probability of Null hypothesis being true
Accept Null hypothesis if P-value>0.05. Means variables are NOT correlated
Reject Null hypothesis if P-value<0.05. Means variables are correlated

In [80]:
```
#finding the correlation between categorical variables and continous variable using ANOVA tests
# creating a category list for property type
cat_group=df_final.groupby('Property Type')['Price'].apply(list)
```

In [81]:
```
cat_group
```

Out[81]:
```
Property Type
D    [95000, 430000, 342500, 399950, 299950, 340000...
F    [42500, 80000, 260000, 202500, 272500, 98500, ...
O    [940000, 425000, 155000, 435000, 315000, 47112...
S    [160000, 315000, 142000, 180000, 147500, 11500...
T    [90000, 250000, 115000, 145000, 255000, 120000...
Name: Price, dtype: object
```

In [82]:
```
AnovaResults = f_oneway(*cat_group)
print('P-Value for Anova is: ', AnovaResults[1])
```

```
P-Value for Anova is:  0.0
```

P-value is zero, hence, we reject H0.Thus, Property type are correlated with Price

In [83]:
```
#finding the correlation between categorical variables and continous variable using ANOVA tests
# creating a category list for current_energy_rating
cat_group_1=df_final.groupby('current_energy_rating')['Price'].apply(list)
```

In [84]:
```
cat_group_1
```

Out[84]:
```
current_energy_rating
0    [250000, 160000, 95000, 42500, 312000, 228000,...
A    [398000, 315000, 326000, 545000, 314000, 49000...
B    [235000, 340000, 163500, 200000, 362500, 29500...
C    [142000, 180000, 145000, 160000, 185000, 16000...
D    [315000, 255000, 147500, 115000, 430000, 26200...
E    [90000, 120000, 410000, 278000, 260000, 220000...
F    [163000, 305000, 120000, 428000, 285000, 18495...
G    [115000, 185000, 170000, 235000, 430000, 13750...
Name: Price, dtype: object
```

In [85]:
```
AnovaResults = f_oneway(*cat_group_1)
print('P-Value for Anova is: ', AnovaResults[1])
```

```
P-Value for Anova is:  1.2195400678904125e-233
```

P-value is zero, hence, we reject H0.Thus, Energy Rating are correlated with Price

In [86]:
```
#finding the correlation between categorical variables and continous variable using ANOVA tests
# creating a category list for construction_age_band
cat_group_2=df_final.groupby('construction_age_band')['Price'].apply(list)
```

In [87]:
```
cat_group_2
```

Out[87]:
```
construction_age_band
 1900-1929       [90000, 115000, 255000, 120000, 262000, 245000...
 1930-1949       [160000, 175000, 128000, 365000, 220000, 17350...
 1950-1966       [315000, 115000, 430000, 133000, 170000, 41500...
 1967-1975       [180000, 160000, 227000, 220000, 287500, 26500...
 1976-1982       [147500, 333000, 425000, 165000, 260000, 24000...
 1983-1990       [98500, 220000, 158000, 415000, 218000, 280000...
 1991-1995       [145000, 541000, 195000, 228000, 235000, 70000...
 1996-2002       [340000, 360000, 270000, 380000, 445000, 55000...
 2003-2006       [295000, 350000, 250000, 220000, 335000, 13100...
 2007 onwards    [114000, 285000, 380000, 405001, 285000, 31500...
 2007-2011       [142000, 272000, 360000, 240000, 499999, 33000...
 2012 onwards    [185000, 515000, 425000, 219950, 225000, 14000...
 before 1900     [410000, 427500, 260000, 295000, 202500, 40000...
Name: Price, dtype: object
```

In [88]:
```
AnovaResults = f_oneway(*cat_group_2)
print('P-Value for Anova is: ', AnovaResults[1])
```

```
P-Value for Anova is:  0.0
```

P-value is zero, hence, we reject H0.Thus, construction_age_band are correlated with Price

In [89]:
```
#finding the correlation between categorical variables and continous variable using ANOVA tests
# creating a category list for current_energy_rating
cat_group_3=df_final.groupby('energy_tariff')['Price'].apply(list)
```

In [90]:
```
cat_group_3
```

Out[90]:
```
energy_tariff
24 hour                 [165000, 232000, 640000, 720000, 83000, 84000]
Single               [315000, 115000, 142000, 180000, 145000, 25500...
Unknown              [250000, 160000, 95000, 430000, 42500, 312000,...
dual                 [90000, 287500, 295000, 245000, 98500, 220000,...
dual (24 hour)       [330000, 246000, 265000, 265000, 117000, 10000...
off-peak 10 hour     [427500, 460000, 225000, 600000, 300000, 69500...
off-peak 18 hour     [775000, 337500, 170000, 495000, 215000, 48500...
off-peak 7 hour      [170000, 484999, 285000, 425000, 375000, 18400...
standard tariff      [158000, 218000, 235000, 725000, 340000, 16000...
Name: Price, dtype: object
```

In [91]:
```
AnovaResults = f_oneway(*cat_group_3)
print('P-Value for Anova is: ', AnovaResults[1])
```

```
P-Value for Anova is:  3.2217334594631206e-143
```

P-value is zero, hence, we reject H0.Thus,energy_tariff are correlated with Price

Applying different machine learning models

In [93]:
```
#dropping the country column
df_final.drop("Country",axis=1,inplace=True)
```

In [94]:
```
#one hot encoding
dummy_df = pd.get_dummies(df_final, prefix={'Property Type':'Property_Type',
                                            'District':'District',
                                                  'Town/City':'Town/City',
                                            'County':'County',
                                            'current_energy_rating':'current_energy_rating',
                                            'construction_age_band':'construction_age_band',
                                            'construction_age_band':'construction_age_band',
                                           'energy_tariff':'energy_tariff',
                                           'areacode':'areacode'})
```

In [95]:
```
#checking the top 5 rows
dummy_df.head()
```

Out[95]:

	Price	Date of Transfer	Old/New	Duration	PPD Category Type	total_floor_area	day	month	year	county_rank	...	areacode_YO32	areacode_YO41
1	90000	2022-02-22	0	0	0	79.0	22	2	2022	67	...	0	0
2	250000	2022-02-04	0	0	0	0.0	4	2	2022	97	...	0	0
3	160000	2022-04-25	0	0	0	0.0	25	4	2022	97	...	0	0
4	315000	2022-02-28	0	0	0	75.0	28	2	2022	97	...	0	0
6	115000	2022-04-29	0	0	0	80.0	29	4	2022	97	...	0	0

5 rows × 3889 columns

In [96]:
```
#splitting the data into dependent and independent data
X = dummy_df.drop(['Price','Date of Transfer','day','month','year'],axis=1)
y =dummy_df['Price']
```

In [97]:
```
#printing the shape
print("Shape of X :",X.shape)
print("Shape of y :",y.shape)
```

```
Shape of X : (213440, 3884)
Shape of y : (213440,)
```

In [98]:
```
#scaling the dataset
sc_X = MinMaxScaler()
sc_y = MinMaxScaler()
X_scaled = sc_X.fit_transform(X)
y_scaled = sc_y.fit_transform(df_final[['Price']])
```

In [99]:
```
#splitting into train and test split
X_train,X_test,y_train,y_test = train_test_split(X_scaled,y_scaled,test_size=0.2)
```

In [100…
```
#checking the shape
print("Length of X train :",len(X_train))
print("Length of X test :",len(X_test))
print("Length of y train :",len(y_train))
print("Length of y test :",len(y_test))
```

```
Length of X train : 170752
Length of X test : 42688
Length of y train : 170752
Length of y test : 42688
```

Random Forest Regressor

In [101…
```
#applying random forest regressor
randomfr = RandomForestRegressor()
randomfr.fit(X_train, y_train)
```

Out[101…
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [102…
```
#predicting the test data
y_pred = randomfr.predict(X_test)
```

In [103…
```
#checking the accuracy
MAE_RF =metrics.mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error:', MAE_RF)
MSE_RF =metrics.mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', MSE_RF)
RMSE_RF=np.sqrt(metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', RMSE_RF)
ACC_RF =metrics.r2_score(y_test,y_pred)
print('Accuracy Score :',ACC_RF)
```

```
Mean Absolute Error: 0.06218586146351945
Mean Squared Error: 0.009524752364211374
Root Mean Squared Error: 0.09759483779489248
Accuracy Score : 0.7299776677246597
```

In [104…
```
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
random_search = HalvingRandomSearchCV(rf,param_grid, n_jobs=-1, factor=3)
```

In [105…
```
# Fit the grid search to the data
random_search.fit(X_train, y_train)
```

Out[105…
```
▸       HalvingRandomSearchCV
▸ estimator: RandomForestRegressor
      ▸ RandomForestRegressor
```

In [106…
```
#finding the best parameters
random_search.best_params_
```

Out[106…
```
{'n_estimators': 300,
 'min_samples_split': 12,
 'min_samples_leaf': 5,
 'max_features': 3,
 'max_depth': 110,
 'bootstrap': True}
```

Best Parameters :{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 3, 'max_depth': 110, 'bootstrap': True}

In [107…
```
#using random forest with best parameters
randomfr1 = RandomForestRegressor(bootstrap= True, max_depth= 110, max_features= 3,min_samples_leaf= 5,min_samp
randomfr1.fit(X_train, y_train)
```

Out[107…
```
▾                     RandomForestRegressor
RandomForestRegressor(max_depth=110, max_features=3, min_samples_leaf=5,
                      min_samples_split=12, n_estimators=300)
```

In [108…
```
#predicting the test data
y_pred1 = randomfr1.predict(X_test)
```

In [109…
```
#checking the accuracy
MAE_RF1 =metrics.mean_absolute_error(y_test, y_pred1)
print('Mean Absolute Error:', MAE_RF1)
MSE_RF1 =metrics.mean_squared_error(y_test, y_pred1)
print('Mean Squared Error:', MSE_RF1)
RMSE_RF1=np.sqrt(metrics.mean_squared_error(y_test, y_pred1))
print('Root Mean Squared Error:', RMSE_RF1)
ACC_RF1 =metrics.r2_score(y_test,y_pred1)
print('Accuracy Score :',ACC_RF1)
```

```
Mean Absolute Error: 0.12657476102724735
Mean Squared Error: 0.027794655718243414
Root Mean Squared Error: 0.16671729279904773
Accuracy Score : 0.2120343422228571
```

XGBoost

In [110…
```
#applying xgboost model
XG_model = XGBRegressor()
```

In [111…
```
#fitting the dataset
XG_model.fit(X_train, y_train)
```

Out[111…
```
▾                          XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=8,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

In [112…
```
#predicting the x test
y_pred2 = XG_model.predict(X_test)
```

In [113…
```
#checking the accuracy
MAE_SVR =metrics.mean_absolute_error(y_test, y_pred2)
print('Mean Absolute Error:', MAE_SVR)
MSE_SVR =metrics.mean_squared_error(y_test, y_pred2)
print('Mean Squared Error:', MSE_SVR)
RMSE_SVR=np.sqrt(metrics.mean_squared_error(y_test, y_pred2))
print('Root Mean Squared Error:', RMSE_SVR)
ACC_SVR =metrics.r2_score(y_test,y_pred2)
print('Accuracy Score :',ACC_SVR)
```

```
Mean Absolute Error: 0.0647268204437298
Mean Squared Error: 0.00938955641592986
Root Mean Squared Error: 0.09689972350801554
Accuracy Score : 0.7338104104431284
```

In [114…
```
#hyper parameters
hyperparameter_grid = {
    'n_estimators': [100, 500, 900, 1100, 1500],
    'max_depth': [2, 3, 5, 10, 15],
    'learning_rate': [0.05, 0.1, 0.15, 0.20],
    'min_child_weight': [1, 2, 3, 4]
    }
```

In [115…
```
#using hyper parameter tunning
random_search1 = HalvingRandomSearchCV(XG_model, hyperparameter_grid, n_jobs=-1, factor=3)
random_search1.fit(X_train, y_train)
```

Out[115…
```
▸  HalvingRandomSearchCV
▸ estimator: XGBRegressor
      ▸ XGBRegressor
```

In [116…
```
#printing the best parameters
random_search1.best_params_
```

Out[116…
```
{'n_estimators': 1100,
 'min_child_weight': 3,
 'max_depth': 10,
 'learning_rate': 0.1}
```

In [126…
```
#using best parameters on XGBoost
XG_model1 = XGBRegressor(n_estimators= 1100,min_child_weight= 3,max_depth=10,learning_rate=0.1)
```

In [127…
```
#model fitting
XG_model1.fit(X_train, y_train)
```

Out[127…
```
▾                          XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.1, max_delta_step=0,
             max_depth=10, min_child_weight=3, missing=nan,
             monotone_constraints='()', n_estimators=1100, n_jobs=8,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

In [128…
```
#predicting the x test
y_pred3 = XG_model1.predict(X_test)
```

In [129…
```
#checking the accuracy
MAE_XG1 =metrics.mean_absolute_error(y_test, y_pred3)
print('Mean Absolute Error:', MAE_XG1)
MSE_XG1 =metrics.mean_squared_error(y_test, y_pred3)
print('Mean Squared Error:', MSE_XG1)
RMSE_XG1=np.sqrt(metrics.mean_squared_error(y_test, y_pred3))
print('Root Mean Squared Error:', RMSE_XG1)
ACC_XG1 =metrics.r2_score(y_test,y_pred3)
print('Accuracy Score :',ACC_XG1)
```

```
Mean Absolute Error: 0.05886637033859404
Mean Squared Error: 0.008250705609878145
Root Mean Squared Error: 0.09083339479441548
Accuracy Score : 0.7660963050265094
```

In [121…
```
#saving the dataset
df_final.to_excel(r'final_data.xlsx')
```

In [125…
```
#saving XGBoost model
filename = 'xgboost_model.sav'
joblib.dump(XG_model1, filename)
```

Out[125…
```
['xgboost_model.sav']
```

In [ ]:

In [131…
```
#copying the original data
newdf = df_final.copy()
```

In [133…
```
#dropping non required fields such as district , county because we have area code in the columns which would re
newdf.drop(['Date of Transfer','District','County','day','Town/City','month','year'],axis=1,inplace=True)
newdf.head()
```

Out[133…

	Price	Property Type	Old/New	Duration	PPD Category Type	current_energy_rating	construction_age_band	total_floor_area	energy_tariff	areacode
1	90000	T	0	0	0	E	1900-1929	79.0	dual	NG6
2	250000	T	0	0	0	0	None	0.0	Unknown	NG24
3	160000	S	0	0	0	0	None	0.0	Unknown	NG17
4	315000	S	0	0	0	D	1950-1966	75.0	Single	NG9
6	115000	T	0	0	0	G	1900-1929	80.0	Single	DN22

In [138…
```
#checking the length
len(newdf)
```

Out[138…
```
213440
```

In [139…
```
#dropping null values
newdf.dropna(inplace=True)
```

In [141…
```
#one hot encoding
dummy_df1 = pd.get_dummies(newdf, prefix={'Property Type':'Property_Type',
                                          'current_energy_rating':'current_energy_rating',
                                          'construction_age_band':'construction_age_band',
                                          'construction_age_band':'construction_age_band',
                                         'energy_tariff':'energy_tariff',
                                         'areacode':'areacode'})
```

In [143…
```
#splitting into X and Y
X1 = dummy_df1.drop(['Price'],axis=1)
y1 =dummy_df1['Price']
```

In [144…
```
#splitting into train and test split
new_X_train,new_X_test1,new_y_train,new_y_test = train_test_split(X1,y1,test_size=0.2)
```

In [145…
```
#applying XG boost model
new_XG_model = XGBRegressor()
```

In [146…
```
#fitting the model
new_XG_model.fit(new_X_train, new_y_train)
```

Out[146…
```
▾                          XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=8,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

In [148…
```
#predicting the X test
new_y_pred = new_XG_model.predict(new_X_test1)
```

In [149…
```
#checking the accuracy
new_MAE_XG =metrics.mean_absolute_error(new_y_test, new_y_pred)
print('Mean Absolute Error:', new_MAE_XG)
new_MSE_XG =metrics.mean_squared_error(new_y_test, new_y_pred)
print('Mean Squared Error:', new_MSE_XG)
new_RMSE_XG=np.sqrt(metrics.mean_squared_error(new_y_test, new_y_pred))
print('Root Mean Squared Error:', new_RMSE_XG)
new_ACC_XG =metrics.r2_score(new_y_test,new_y_pred)
print('Accuracy Score :',new_ACC_XG)
```

```
Mean Absolute Error: 58347.17076307548
Mean Squared Error: 6915789336.875405
Root Mean Squared Error: 83161.22496016641
Accuracy Score : 0.7825763983704952
```

In [150…
```
#hyper parameters
hyperparameter_grid = {
    'n_estimators': [100, 500, 900, 1100, 1500],
    'max_depth': [2, 3, 5, 10, 15],
    'learning_rate': [0.05, 0.1, 0.15, 0.20],
    'min_child_weight': [1, 2, 3, 4]
    }
```

In [151…
```
#applying hyper parameter tunning
new_random_search = HalvingRandomSearchCV(new_XG_model, hyperparameter_grid, n_jobs=-1)
new_random_search.fit(new_X_train, new_y_train)
```

Out[151…
```
▸  HalvingRandomSearchCV
▸ estimator: XGBRegressor
      ▸ XGBRegressor
```

In [152…
```
#checking best parameters
new_random_search.best_params_
```

Out[152…
```
{'n_estimators': 1500,
 'min_child_weight': 1,
 'max_depth': 2,
 'learning_rate': 0.05}
```

In [153…
```
#applying model with best parameters
new_XG_model1 = XGBRegressor(n_estimators=1500,min_child_weight=1,max_depth= 2,learning_rate= 0.05)
```

In [154…
```
#fitting the model
new_XG_model1.fit(new_X_train, new_y_train)
```

Out[154…
```
▾                          XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.05, max_delta_step=0,
             max_depth=2, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=1500, n_jobs=8,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

In [155…
```
#predict the test data
new_y_pred1 = new_XG_model1.predict(new_X_test1)
```

In [156…
```
#checking the accuracy
new_MAE_XG1 =metrics.mean_absolute_error(new_y_test, new_y_pred1)
print('Mean Absolute Error:', new_MAE_XG1)
new_MSE_XG1 =metrics.mean_squared_error(new_y_test, new_y_pred1)
print('Mean Squared Error:', new_MSE_XG1)
new_RMSE_XG1=np.sqrt(metrics.mean_squared_error(new_y_test, new_y_pred1))
print('Root Mean Squared Error:', new_RMSE_XG1)
new_ACC_XG1 =metrics.r2_score(new_y_test,new_y_pred1)
print('Accuracy Score :',new_ACC_XG1)
```

```
Mean Absolute Error: 69493.09799306658
Mean Squared Error: 9014938196.038923
Root Mean Squared Error: 94947.02836865892
Accuracy Score : 0.7165818338914673
```

In [157…
```
#saving XGBoost model
filename = 'xgboost_model1.sav'
joblib.dump(new_XG_model, filename)
```

Out[157…
```
['xgboost_model1.sav']
```

Code to Apply Sequential Neural Network for House Price Prediction

In [1]:

```
#importing various libraries
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense
from kerastuner.tuners import RandomSearch
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from keras import optimizers
from sklearn import metrics
import numpy as np
from keras.models import Sequential
from keras.callbacks import EarlyStopping
```

```
C:\Users\hp\AppData\Local\Temp/ipykernel_38820/3656945953.py:6: DeprecationWarning: `import kerastuner` is depr
ecated, please use `import keras_tuner`.
  from kerastuner.tuners import RandomSearch
```

In [4]:

```
#reading the dataset
df = pd.read_excel(r'final_data.xlsx')
df.drop('Unnamed: 0',axis=1,inplace=True)
df.head()
```

Out[4]:

	Price	Date of Transfer	Property Type	Old/New	Duration	Town/City	District	County	PPD Category Type	current_energy_rating	cons
0	90000	2022-02-22	T	0	0	NOTTINGHAM	CITY OF NOTTINGHAM	CITY OF NOTTINGHAM	0	E	
1	250000	2022-02-04	T	0	0	NEWARK	NEWARK AND SHERWOOD	NOTTINGHAMSHIRE	0	0	
2	160000	2022-04-25	S	0	0	SUTTON-IN-ASHFIELD	ASHFIELD	NOTTINGHAMSHIRE	0	0	
3	315000	2022-02-28	S	0	0	NOTTINGHAM	BROXTOWE	NOTTINGHAMSHIRE	0	D	
4	115000	2022-04-29	T	0	0	RETFORD	BASSETLAW	NOTTINGHAMSHIRE	0	G	

In [5]:

```
#one hot encoding
dummy_df = pd.get_dummies(df, prefix={'Property Type':'Property_Type',
                                      'District':'District',
                                            'Town/City':'Town/City',
                                      'County':'County',
                                      'current_energy_rating':'current_energy_rating',
                                      'construction_age_band':'construction_age_band',
                                      'construction_age_band':'construction_age_band',
                                     'energy_tariff':'energy_tariff',
                                     'areacode':'areacode'})
```

In [6]:

```
#viewing top 5 rows
dummy_df.head()
```

Out[6]:

	Price	Date of Transfer	Old/New	Duration	PPD Category Type	total_floor_area	day	month	year	county_rank	...	areacode_YO32	areacode_YO41
0	90000	2022-02-22	0	0	0	79.0	22	2	2022	67	...	0	0
1	250000	2022-02-04	0	0	0	0.0	4	2	2022	97	...	0	0
2	160000	2022-04-25	0	0	0	0.0	25	4	2022	97	...	0	0
3	315000	2022-02-28	0	0	0	75.0	28	2	2022	97	...	0	0
4	115000	2022-04-29	0	0	0	80.0	29	4	2022	97	...	0	0

5 rows × 3891 columns

In [7]:

```
#splitting the dataset into dependent and independent variables
X = dummy_df.drop(['Price','Date of Transfer','day','month','year'],axis=1)
y =dummy_df['Price']
```

In [8]:

```
#printing the shapes
print("Shape of X :",X.shape)
print("Shape of y :",y.shape)
```

```
Shape of X : (213440, 3886)
Shape of y : (213440,)
```

In [9]:

```
#scaling the dataset
sc_X = MinMaxScaler()
sc_y = MinMaxScaler()
X_scaled = sc_X.fit_transform(X)
y_scaled = sc_y.fit_transform(df[['Price']])
```

In [10]:

```
#splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.2, random_state=0)
```

In [11]:

```
#function to return the layers of model
def seq_model(parameters):
    model = keras.Sequential()
    for i in range(parameters.Int('num_layers', 2, 6)):
        model.add(layers.Dense(units=parameters.Int('units_' + str(i),
                                            min_value=32,
                                            max_value=256,
                                            step=32),
                               activation='relu'))
    model.add(layers.Dense(1, activation='linear'))
    model.compile(
        optimizer=keras.optimizers.Adam(
            parameters.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
        loss='mean_absolute_error',
        metrics=['mean_absolute_error'])
    return model
```

In [12]:

```
#using random search to find optimized number of layers of neural network
keras_tuner = RandomSearch(seq_model,objective='val_mean_absolute_error',max_trials=5,executions_per_trial=3,
                          directory='network',project_name='House price prediction')
```

```
INFO:tensorflow:Reloading Oracle from existing project network\House price prediction\oracle.json
INFO:tensorflow:Reloading Tuner from network\House price prediction\tuner0.json
```

In [13]:

```
#printing the summary of randomsearch
keras_tuner.search_space_summary()
```

```
Search space summary
Default search space size: 11
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 10, 'step': 1, 'sampling': None}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
learning_rate (Choice)
{'default': 0.01, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
units_2 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
units_3 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
units_4 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
units_5 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
units_6 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
units_7 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
units_8 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
```

In [14]:

```
#training the data to find optimum parameters
keras_tuner.search(X_train, y_train,epochs=30,validation_data=(X_test, y_test))
```

```
INFO:tensorflow:Oracle triggered exit
```

In [15]:

```
#printing the summary of results
keras_tuner.results_summary()
```

```
Results summary
Results in network\House price prediction
Showing 10 best trials
<keras_tuner.engine.objective.Objective object at 0x000002088B576040>
Trial summary
Hyperparameters:
num_layers: 9
units_0: 224
units_1: 160
learning_rate: 0.01
units_2: 64
units_3: 64
units_4: 96
units_5: 128
units_6: 128
units_7: 192
units_8: 32
Score: 57607.6328125
Trial summary
Hyperparameters:
num_layers: 8
units_0: 32
units_1: 32
learning_rate: 0.01
units_2: 32
units_3: 32
units_4: 32
units_5: 32
units_6: 32
units_7: 32
Score: 57986.604166666664
Trial summary
Hyperparameters:
num_layers: 3
units_0: 128
units_1: 32
learning_rate: 0.001
units_2: 192
units_3: 224
units_4: 128
units_5: 224
units_6: 224
units_7: 64
units_8: 96
Score: 58123.764322916664
Trial summary
Hyperparameters:
num_layers: 6
units_0: 64
units_1: 32
learning_rate: 0.0001
units_2: 192
units_3: 256
units_4: 192
units_5: 64
units_6: 256
units_7: 64
Score: 58733.580729166664
Trial summary
Hyperparameters:
num_layers: 3
units_0: 64
units_1: 96
learning_rate: 0.0001
units_2: 64
units_3: 192
units_4: 160
units_5: 96
units_6: 192
units_7: 128
units_8: 256
Score: 95229.078125
```

In [16]:

```
#saving the optimum parameters into variable
best_params = keras_tuner.get_best_hyperparameters(num_trials=1)[0]
```

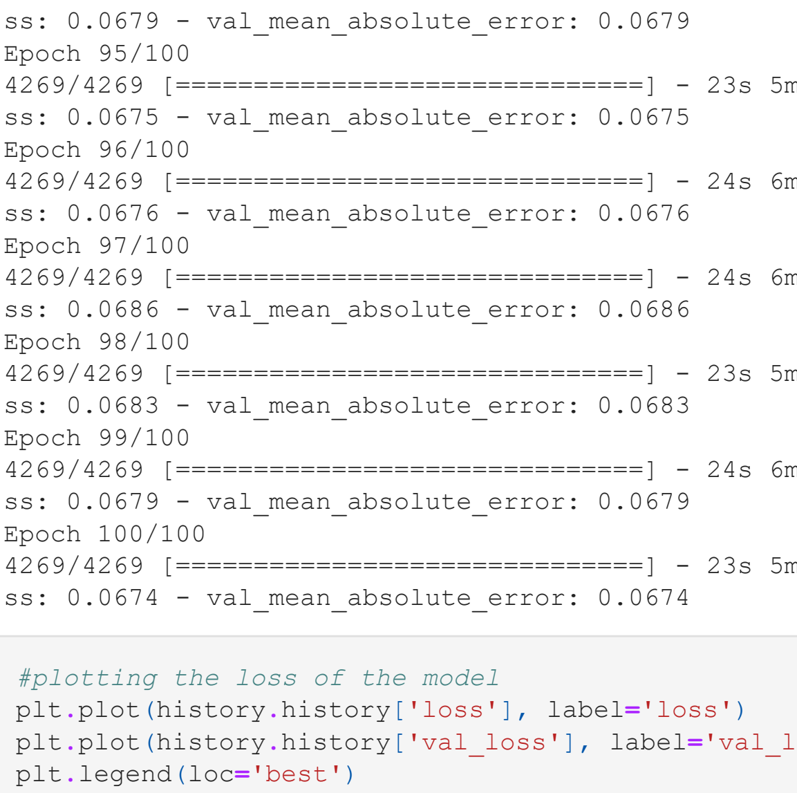
In [17]:

```
#building model with optimal parameters returned by random seaRch
model = keras_tuner.hypermodel.build(best_params)
history = model.fit(X_train, y_train, epochs=100,validation_data=(X_test, y_test),validation_split=0.2)
```

```
Epoch 1/100
4269/4269 [==============================] - 30s 7ms/step - loss: 0.0787 - mean_absolute_error: 0.0787 - val_lo
ss: 0.0714 - val_mean_absolute_error: 0.0714
Epoch 2/100
4269/4269 [==============================] - 25s 6ms/step - loss: 0.0706 - mean_absolute_error: 0.0706 - val_lo
ss: 0.0682 - val_mean_absolute_error: 0.0682
Epoch 3/100
4269/4269 [==============================] - 27s 6ms/step - loss: 0.0711 - mean_absolute_error: 0.0711 - val_lo
ss: 0.0704 - val_mean_absolute_error: 0.0704
Epoch 4/100
4269/4269 [==============================] - 26s 6ms/step - loss: 0.0669 - mean_absolute_error: 0.0669 - val_lo
ss: 0.0659 - val_mean_absolute_error: 0.0659
Epoch 5/100
4269/4269 [==============================] - 26s 6ms/step - loss: 0.0641 - mean_absolute_error: 0.0641 - val_lo
ss: 0.0700 - val_mean_absolute_error: 0.0700
Epoch 6/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0622 - mean_absolute_error: 0.0622 - val_lo
ss: 0.0643 - val_mean_absolute_error: 0.0643
Epoch 7/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0602 - mean_absolute_error: 0.0602 - val_lo
ss: 0.0670 - val_mean_absolute_error: 0.0670
Epoch 8/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0600 - mean_absolute_error: 0.0600 - val_lo
ss: 0.0664 - val_mean_absolute_error: 0.0664
Epoch 9/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0580 - mean_absolute_error: 0.0580 - val_lo
ss: 0.0680 - val_mean_absolute_error: 0.0680
Epoch 10/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0565 - mean_absolute_error: 0.0565 - val_lo
ss: 0.0722 - val_mean_absolute_error: 0.0722
Epoch 11/100
4269/4269 [==============================] - 25s 6ms/step - loss: 0.0564 - mean_absolute_error: 0.0564 - val_lo
ss: 0.0673 - val_mean_absolute_error: 0.0673
Epoch 12/100
4269/4269 [==============================] - 26s 6ms/step - loss: 0.0555 - mean_absolute_error: 0.0555 - val_lo
ss: 0.0636 - val_mean_absolute_error: 0.0636
Epoch 13/100
4269/4269 [==============================] - 25s 6ms/step - loss: 0.0537 - mean_absolute_error: 0.0537 - val_lo
ss: 0.0639 - val_mean_absolute_error: 0.0639
Epoch 14/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0531 - mean_absolute_error: 0.0531 - val_lo
ss: 0.0636 - val_mean_absolute_error: 0.0636
Epoch 15/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0528 - mean_absolute_error: 0.0528 - val_lo
ss: 0.0643 - val_mean_absolute_error: 0.0643
Epoch 16/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0520 - mean_absolute_error: 0.0520 - val_lo
ss: 0.0654 - val_mean_absolute_error: 0.0654
Epoch 17/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0514 - mean_absolute_error: 0.0514 - val_lo
ss: 0.0641 - val_mean_absolute_error: 0.0641
Epoch 18/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0501 - mean_absolute_error: 0.0501 - val_lo
ss: 0.0645 - val_mean_absolute_error: 0.0645
Epoch 19/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0496 - mean_absolute_error: 0.0496 - val_lo
ss: 0.0643 - val_mean_absolute_error: 0.0643
Epoch 20/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0490 - mean_absolute_error: 0.0490 - val_lo
ss: 0.0646 - val_mean_absolute_error: 0.0646
Epoch 21/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0483 - mean_absolute_error: 0.0483 - val_lo
ss: 0.0643 - val_mean_absolute_error: 0.0643
Epoch 22/100
4269/4269 [==============================] - 25s 6ms/step - loss: 0.0478 - mean_absolute_error: 0.0478 - val_lo
ss: 0.0650 - val_mean_absolute_error: 0.0650
Epoch 23/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0476 - mean_absolute_error: 0.0476 - val_lo
ss: 0.0665 - val_mean_absolute_error: 0.0665
Epoch 24/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0470 - mean_absolute_error: 0.0470 - val_lo
ss: 0.0662 - val_mean_absolute_error: 0.0662
Epoch 25/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0466 - mean_absolute_error: 0.0466 - val_lo
ss: 0.0643 - val_mean_absolute_error: 0.0643
Epoch 26/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0464 - mean_absolute_error: 0.0464 - val_lo
ss: 0.0655 - val_mean_absolute_error: 0.0655
Epoch 27/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0460 - mean_absolute_error: 0.0460 - val_lo
ss: 0.0664 - val_mean_absolute_error: 0.0664
Epoch 28/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0459 - mean_absolute_error: 0.0459 - val_lo
ss: 0.0673 - val_mean_absolute_error: 0.0673
Epoch 29/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0454 - mean_absolute_error: 0.0454 - val_lo
ss: 0.0662 - val_mean_absolute_error: 0.0662
Epoch 30/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0454 - mean_absolute_error: 0.0454 - val_lo
ss: 0.0694 - val_mean_absolute_error: 0.0694
Epoch 31/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0456 - mean_absolute_error: 0.0456 - val_lo
ss: 0.0661 - val_mean_absolute_error: 0.0661
Epoch 32/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0455 - mean_absolute_error: 0.0455 - val_lo
ss: 0.0655 - val_mean_absolute_error: 0.0655
Epoch 33/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0463 - mean_absolute_error: 0.0463 - val_lo
ss: 0.0796 - val_mean_absolute_error: 0.0796
Epoch 34/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0458 - mean_absolute_error: 0.0458 - val_lo
ss: 0.0709 - val_mean_absolute_error: 0.0709
Epoch 35/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0450 - mean_absolute_error: 0.0450 - val_lo
ss: 0.0680 - val_mean_absolute_error: 0.0680
Epoch 36/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0441 - mean_absolute_error: 0.0441 - val_lo
ss: 0.0725 - val_mean_absolute_error: 0.0725
Epoch 37/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0435 - mean_absolute_error: 0.0435 - val_lo
ss: 0.0657 - val_mean_absolute_error: 0.0657
Epoch 38/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0431 - mean_absolute_error: 0.0431 - val_lo
ss: 0.0657 - val_mean_absolute_error: 0.0657
Epoch 39/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0427 - mean_absolute_error: 0.0427 - val_lo
ss: 0.0658 - val_mean_absolute_error: 0.0658
Epoch 40/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0430 - mean_absolute_error: 0.0430 - val_lo
ss: 0.0652 - val_mean_absolute_error: 0.0652
Epoch 41/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0448 - mean_absolute_error: 0.0448 - val_lo
ss: 0.0664 - val_mean_absolute_error: 0.0664
Epoch 42/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0425 - mean_absolute_error: 0.0425 - val_lo
ss: 0.0680 - val_mean_absolute_error: 0.0680
Epoch 43/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0426 - mean_absolute_error: 0.0426 - val_lo
ss: 0.0663 - val_mean_absolute_error: 0.0663
Epoch 44/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0420 - mean_absolute_error: 0.0420 - val_lo
ss: 0.0658 - val_mean_absolute_error: 0.0658
Epoch 45/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0417 - mean_absolute_error: 0.0417 - val_lo
ss: 0.0689 - val_mean_absolute_error: 0.0689
Epoch 46/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0422 - mean_absolute_error: 0.0422 - val_lo
ss: 0.0669 - val_mean_absolute_error: 0.0669
Epoch 47/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0417 - mean_absolute_error: 0.0417 - val_lo
ss: 0.0685 - val_mean_absolute_error: 0.0685
Epoch 48/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0413 - mean_absolute_error: 0.0413 - val_lo
ss: 0.0683 - val_mean_absolute_error: 0.0683
Epoch 49/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0415 - mean_absolute_error: 0.0415 - val_lo
ss: 0.0688 - val_mean_absolute_error: 0.0688
Epoch 50/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0417 - mean_absolute_error: 0.0417 - val_lo
ss: 0.0670 - val_mean_absolute_error: 0.0670
Epoch 51/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0408 - mean_absolute_error: 0.0408 - val_lo
ss: 0.0677 - val_mean_absolute_error: 0.0677
Epoch 52/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0407 - mean_absolute_error: 0.0407 - val_lo
ss: 0.0663 - val_mean_absolute_error: 0.0663
Epoch 53/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0410 - mean_absolute_error: 0.0410 - val_lo
ss: 0.0663 - val_mean_absolute_error: 0.0663
Epoch 54/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0408 - mean_absolute_error: 0.0408 - val_lo
ss: 0.0698 - val_mean_absolute_error: 0.0698
Epoch 55/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0414 - mean_absolute_error: 0.0414 - val_lo
ss: 0.0661 - val_mean_absolute_error: 0.0661
Epoch 56/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0414 - mean_absolute_error: 0.0414 - val_lo
ss: 0.0669 - val_mean_absolute_error: 0.0669
Epoch 57/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0411 - mean_absolute_error: 0.0411 - val_lo
ss: 0.0670 - val_mean_absolute_error: 0.0670
Epoch 58/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0406 - mean_absolute_error: 0.0406 - val_lo
ss: 0.0671 - val_mean_absolute_error: 0.0671
Epoch 59/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0407 - mean_absolute_error: 0.0407 - val_lo
ss: 0.0679 - val_mean_absolute_error: 0.0679
Epoch 60/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0410 - mean_absolute_error: 0.0410 - val_lo
ss: 0.0668 - val_mean_absolute_error: 0.0668
Epoch 61/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0408 - mean_absolute_error: 0.0408 - val_lo
ss: 0.0725 - val_mean_absolute_error: 0.0725
Epoch 62/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0410 - mean_absolute_error: 0.0410 - val_lo
ss: 0.0676 - val_mean_absolute_error: 0.0676
Epoch 63/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0411 - mean_absolute_error: 0.0411 - val_lo
ss: 0.0672 - val_mean_absolute_error: 0.0672
Epoch 64/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0420 - mean_absolute_error: 0.0420 - val_lo
ss: 0.0687 - val_mean_absolute_error: 0.0687
Epoch 65/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0419 - mean_absolute_error: 0.0419 - val_lo
ss: 0.0687 - val_mean_absolute_error: 0.0687
Epoch 66/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0410 - mean_absolute_error: 0.0410 - val_lo
ss: 0.0710 - val_mean_absolute_error: 0.0710
Epoch 67/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0408 - mean_absolute_error: 0.0408 - val_lo
ss: 0.0680 - val_mean_absolute_error: 0.0680
Epoch 68/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0407 - mean_absolute_error: 0.0407 - val_lo
ss: 0.0678 - val_mean_absolute_error: 0.0678
Epoch 69/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0413 - mean_absolute_error: 0.0413 - val_lo
ss: 0.0696 - val_mean_absolute_error: 0.0696
Epoch 70/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0420 - mean_absolute_error: 0.0420 - val_lo
ss: 0.0687 - val_mean_absolute_error: 0.0687
Epoch 71/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0417 - mean_absolute_error: 0.0417 - val_lo
ss: 0.0680 - val_mean_absolute_error: 0.0680
Epoch 72/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0421 - mean_absolute_error: 0.0421 - val_lo
ss: 0.0697 - val_mean_absolute_error: 0.0697
Epoch 73/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0413 - mean_absolute_error: 0.0413 - val_lo
ss: 0.0678 - val_mean_absolute_error: 0.0678
Epoch 74/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0397 - mean_absolute_error: 0.0397 - val_lo
ss: 0.0678 - val_mean_absolute_error: 0.0678
Epoch 75/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0392 - mean_absolute_error: 0.0392 - val_lo
ss: 0.0675 - val_mean_absolute_error: 0.0675
Epoch 76/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0393 - mean_absolute_error: 0.0393 - val_lo
ss: 0.0672 - val_mean_absolute_error: 0.0672
Epoch 77/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0387 - mean_absolute_error: 0.0387 - val_lo
ss: 0.0668 - val_mean_absolute_error: 0.0668
Epoch 78/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0386 - mean_absolute_error: 0.0386 - val_lo
ss: 0.0693 - val_mean_absolute_error: 0.0693
Epoch 79/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0387 - mean_absolute_error: 0.0387 - val_lo
ss: 0.0689 - val_mean_absolute_error: 0.0689
Epoch 80/100
4269/4269 [==============================] - 25s 6ms/step - loss: 0.0386 - mean_absolute_error: 0.0386 - val_lo
ss: 0.0683 - val_mean_absolute_error: 0.0683
Epoch 81/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0387 - mean_absolute_error: 0.0387 - val_lo
ss: 0.0668 - val_mean_absolute_error: 0.0668
Epoch 82/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0381 - mean_absolute_error: 0.0381 - val_lo
ss: 0.0679 - val_mean_absolute_error: 0.0679
Epoch 83/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0383 - mean_absolute_error: 0.0383 - val_lo
ss: 0.0709 - val_mean_absolute_error: 0.0709
Epoch 84/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0380 - mean_absolute_error: 0.0380 - val_lo
ss: 0.0682 - val_mean_absolute_error: 0.0682
Epoch 85/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0378 - mean_absolute_error: 0.0378 - val_lo
ss: 0.0670 - val_mean_absolute_error: 0.0670
Epoch 86/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0378 - mean_absolute_error: 0.0378 - val_lo
ss: 0.0699 - val_mean_absolute_error: 0.0699
Epoch 87/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0376 - mean_absolute_error: 0.0376 - val_lo
ss: 0.0704 - val_mean_absolute_error: 0.0704
Epoch 88/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0376 - mean_absolute_error: 0.0376 - val_lo
ss: 0.0672 - val_mean_absolute_error: 0.0672
Epoch 89/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0377 - mean_absolute_error: 0.0377 - val_lo
ss: 0.0669 - val_mean_absolute_error: 0.0669
Epoch 90/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0375 - mean_absolute_error: 0.0375 - val_lo
ss: 0.0678 - val_mean_absolute_error: 0.0678
Epoch 91/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0375 - mean_absolute_error: 0.0375 - val_lo
ss: 0.0668 - val_mean_absolute_error: 0.0668
Epoch 92/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0373 - mean_absolute_error: 0.0373 - val_lo
ss: 0.0669 - val_mean_absolute_error: 0.0669
Epoch 93/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0372 - mean_absolute_error: 0.0372 - val_lo
ss: 0.0675 - val_mean_absolute_error: 0.0675
Epoch 94/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0379 - mean_absolute_error: 0.0379 - val_lo
ss: 0.0679 - val_mean_absolute_error: 0.0679
Epoch 95/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0378 - mean_absolute_error: 0.0378 - val_lo
ss: 0.0675 - val_mean_absolute_error: 0.0675
Epoch 96/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0375 - mean_absolute_error: 0.0375 - val_lo
ss: 0.0676 - val_mean_absolute_error: 0.0676
Epoch 97/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0381 - mean_absolute_error: 0.0381 - val_lo
ss: 0.0686 - val_mean_absolute_error: 0.0686
Epoch 98/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0379 - mean_absolute_error: 0.0379 - val_lo
ss: 0.0683 - val_mean_absolute_error: 0.0683
Epoch 99/100
4269/4269 [==============================] - 24s 6ms/step - loss: 0.0380 - mean_absolute_error: 0.0380 - val_lo
ss: 0.0679 - val_mean_absolute_error: 0.0679
Epoch 100/100
4269/4269 [==============================] - 23s 5ms/step - loss: 0.0382 - mean_absolute_error: 0.0382 - val_lo
ss: 0.0674 - val_mean_absolute_error: 0.0674
```

In [18]:

```
#plotting the loss of the model
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend(loc='best')
plt.title('Training Model')
plt.xlabel('Epoches')
plt.ylabel('Loss')
plt.show()
```



In [19]:

```
#predicting the test data
y_pred = model.predict(X_test)
```

In [21]:

```
#checking the accuracy
MAE_SNN =metrics.mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error:', MAE_SNN)
MSE_SNN =metrics.mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', MSE_SNN)
RMSE_SNN=np.sqrt(metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', RMSE_SNN)
ACC_SNN =metrics.r2_score(y_test,y_pred)
print('Accuracy Score :',ACC_SNN)
```

```
Mean Absolute Error: 0.0670499007014355
Mean Squared Error: 0.010840937952645274
Root Mean Squared Error: 0.10411982497413869
Accuracy Score : 0.6873221273228846
```

Predicting the House Price Index using Machine Learning Model

Using LSTM Model

In [1]:

```python
#importing various libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import LSTM
import math
from sklearn.metrics import mean_squared_error
```

In [2]:

```python
#importing the dataset
df = pd.read_csv(r'indices_dataset/Indices-2022-05_ONS.csv')
df.head()
```

Out[2]:

	Date	Region_Name	Area_Code	Index
0	1968-04-01	England	E92000001	1.680067
1	1968-04-01	Scotland	S92000003	2.108087
2	1968-04-01	Northern Ireland	N92000001	3.300420
3	1968-04-01	Wales	W92000004	2.119327
4	1968-04-01	West Midlands Region	E12000005	2.097808

In [3]:

```python
# Since there are different regions , thus selecting region as UK
df = df[df.Region_Name=='United Kingdom']
df.head()
```

Out[3]:

	Date	Region_Name	Area_Code	Index
9	1968-04-01	United Kingdom	K02000001	1.885299
19	1968-05-01	United Kingdom	K02000001	1.885299
29	1968-06-01	United Kingdom	K02000001	1.885299
39	1968-07-01	United Kingdom	K02000001	1.949207
49	1968-08-01	United Kingdom	K02000001	1.949207

In [4]:

```python
#Dropping the area code column and region name as it is not required
df.drop(['Area_Code','Region_Name'],axis=1,inplace=True)
df.head()
```

Out[4]:

	Date	Index
9	1968-04-01	1.885299
19	1968-05-01	1.885299
29	1968-06-01	1.885299
39	1968-07-01	1.949207
49	1968-08-01	1.949207

In [5]:

```python
#checking the datatypes
df.dtypes
```

Out[5]:

```
Date       object
Index     float64
dtype: object
```

In [6]:

```python
#checking the length of records
df.shape
```

Out[6]:

```
(650, 2)
```

In [7]:

```python
#checking the to and from dates :
print("Starting of the dataset :",df.Date.min())
print("Ending of the dataset :",df.Date.max())
```

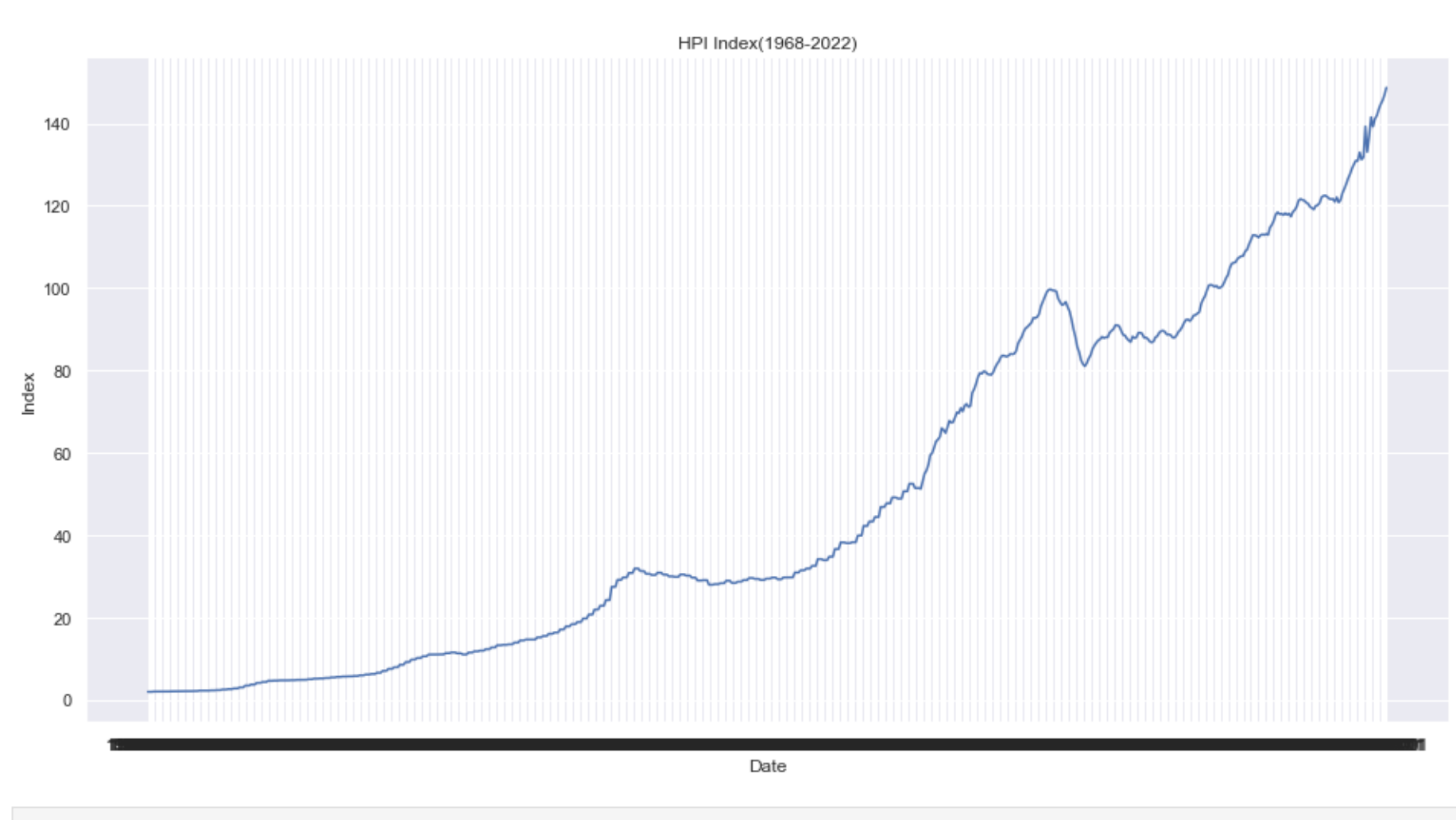
```
Starting of the dataset : 1968-04-01
Ending of the dataset : 2022-05-01
```

In [8]:

```python
#plotting the data series

sns.set(rc={'figure.figsize':(16,8)})
sns.lineplot(data=df , x='Date',y='Index')
plt.title("HPI Index(1968-2022)")
```

Out[8]:

```
Text(0.5, 1.0, 'HPI Index(1968-2022)')
```



In [9]:

```python
#only saving the index values of HPI
df =df['Index']
```

In [10]:

```python
#viewing the values
df
```

Out[10]:

```
9           1.885299
19          1.885299
29          1.885299
39          1.949207
49          1.949207
              ...
135249    143.422582
135670    144.646684
136091    145.556596
136512    146.898652
136933    148.688246
Name: Index, Length: 650, dtype: float64
```

In [11]:

```python
#scaling down the magnitude of index
scaler = MinMaxScaler(feature_range=(0,1))
df = scaler.fit_transform(np.array(df).reshape(-1,1))
```

In [12]:

```python
#checking the shape
df.shape
```

Out[12]:

```
(650, 1)
```

In [13]:

```python
#splitting dataset into train and test split
size =int(len(df)*0.65)
test_size =len(df)-size
train_data,test_data = df[0:size,:],df[size:len(df),:]
```

In [14]:

```python
#checking length of train and test data
print("length of training data", len(train_data))
print("length of test data", len(test_data))
```

```
length of training data 422
length of test data 228
```

In [15]:

```python
# create a function for dataset matrix
def mat_data(data, step=1):
    X = []
    Y = []
    for i in range(len(data)-step-1):
        a = data[i:(i+step), 0]
        X.append(a)
        Y.append(data[i + step, 0])
    return np.array(X), np.array(Y)
```

In [16]:

```python
#diving the train and test data into time step of 30
time_step = 30
X_train, y_train = mat_data(train_data, time_step)
X_test, y_test = mat_data(test_data, time_step)
```

In [17]:

```python
#checking the values of x train
print(X_train)
```

```
[[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 2.03156249e-03
  2.03156249e-03 2.03156249e-03]
 [0.00000000e+00 0.00000000e+00 4.35334823e-04 ... 2.03156249e-03
  2.03156249e-03 2.03156249e-03]
 [0.00000000e+00 4.35334823e-04 4.35334823e-04 ... 2.03156249e-03
  2.03156249e-03 2.03156249e-03]
 ...
 [3.11957442e-01 3.11957442e-01 3.21647174e-01 ... 4.21954311e-01
  4.36411391e-01 4.33158548e-01]
 [3.11957442e-01 3.21647174e-01 3.21647174e-01 ... 4.36411391e-01
  4.33158548e-01 4.28459997e-01]
 [3.21647174e-01 3.21647174e-01 3.21647174e-01 ... 4.33158548e-01
  4.28459997e-01 4.37857099e-01]]
```

In [18]:

```python
#checking the values of y train
print(X_test)
```

```
[[0.44653135 0.45448274 0.46279556 ... 0.55586832 0.55473012 0.55641929]
 [0.45448274 0.46279556 0.46134985 ... 0.55473012 0.55641929 0.55953338]
 [0.46279556 0.46134985 0.46966268 ... 0.55641929 0.55953338 0.5586793 ]
 ...
 [0.82090066 0.82150806 0.81930461 ... 0.94848795 0.95370845 0.96413108]
 [0.82150806 0.81930461 0.81636438 ... 0.95370845 0.96413108 0.97246947]
 [0.81930461 0.81636438 0.81527782 ... 0.96413108 0.97246947 0.97866766]]
```

In [19]:

```python
#checking the shape of train and test data
print('Shape of X train',X_train.shape)
print('Shape of X test',X_test.shape)
print('Shape of y train',y_train.shape)
print('Shape of y test',y_test.shape)
```

```
Shape of X train (391, 30)
Shape of X test (197, 30)
Shape of y train (391,)
Shape of y test (197,)
```

In [20]:

```python
#reshape the dataset for model building
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
```

In [22]:

```python
#LSTM model building
model_LSTM=Sequential()
model_LSTM.add(LSTM(64,return_sequences=True,input_shape=(30,1)))
model_LSTM.add(Dropout(0.2))
model_LSTM.add(LSTM(32))
model_LSTM.add(Dense(1))
model_LSTM.compile(loss='mean_squared_error',optimizer='adam')
```

In [24]:

```python
#checking the model summary
model_LSTM.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            16896

 dropout (Dropout)           (None, 30, 64)            0

 lstm_1 (LSTM)               (None, 32)                12416

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 29,345
Trainable params: 29,345
Non-trainable params: 0
_________________________________________________________________
```

In [30]:

```python
#fitting the model
model_LSTM.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100,batch_size=64,verbose=1)
```

```
Epoch 1/100
7/7 [==============================] - 1s 96ms/step - loss: 7.4352e-05 - val_loss: 0.0035
Epoch 2/100
7/7 [==============================] - 0s 62ms/step - loss: 6.3077e-05 - val_loss: 0.0035
Epoch 3/100
7/7 [==============================] - 0s 67ms/step - loss: 6.0944e-05 - val_loss: 0.0035
Epoch 4/100
7/7 [==============================] - 0s 63ms/step - loss: 5.4397e-05 - val_loss: 0.0023
Epoch 5/100
7/7 [==============================] - 1s 84ms/step - loss: 5.1867e-05 - val_loss: 0.0039
Epoch 6/100
7/7 [==============================] - 1s 85ms/step - loss: 5.5977e-05 - val_loss: 0.0036
Epoch 7/100
7/7 [==============================] - 1s 87ms/step - loss: 6.1370e-05 - val_loss: 0.0020
Epoch 8/100
7/7 [==============================] - 0s 71ms/step - loss: 5.4096e-05 - val_loss: 0.0055
Epoch 9/100
7/7 [==============================] - 1s 99ms/step - loss: 6.1217e-05 - val_loss: 0.0042
Epoch 10/100
7/7 [==============================] - 0s 65ms/step - loss: 5.7178e-05 - val_loss: 9.5469e-04
Epoch 11/100
7/7 [==============================] - 0s 65ms/step - loss: 7.6840e-05 - val_loss: 0.0035
Epoch 12/100
7/7 [==============================] - 0s 68ms/step - loss: 6.9656e-05 - val_loss: 0.0053
Epoch 13/100
7/7 [==============================] - 1s 73ms/step - loss: 6.1387e-05 - val_loss: 0.0015
Epoch 14/100
7/7 [==============================] - 1s 87ms/step - loss: 5.8458e-05 - val_loss: 0.0037
Epoch 15/100
7/7 [==============================] - 1s 77ms/step - loss: 5.8463e-05 - val_loss: 0.0025
Epoch 16/100
7/7 [==============================] - 0s 69ms/step - loss: 6.0075e-05 - val_loss: 0.0031
Epoch 17/100
7/7 [==============================] - 1s 67ms/step - loss: 5.2380e-05 - val_loss: 0.0085
Epoch 18/100
7/7 [==============================] - 0s 66ms/step - loss: 8.3721e-05 - val_loss: 0.0023
Epoch 19/100
7/7 [==============================] - 1s 73ms/step - loss: 7.6698e-05 - val_loss: 0.0013
Epoch 20/100
7/7 [==============================] - 0s 69ms/step - loss: 1.2203e-04 - val_loss: 0.0043
Epoch 21/100
7/7 [==============================] - 0s 66ms/step - loss: 7.0012e-05 - val_loss: 0.0027
Epoch 22/100
7/7 [==============================] - 0s 68ms/step - loss: 6.0327e-05 - val_loss: 0.0022
Epoch 23/100
7/7 [==============================] - 1s 72ms/step - loss: 6.0236e-05 - val_loss: 0.0063
Epoch 24/100
7/7 [==============================] - 0s 72ms/step - loss: 6.6795e-05 - val_loss: 0.0054
Epoch 25/100
7/7 [==============================] - 0s 65ms/step - loss: 6.4809e-05 - val_loss: 0.0015
Epoch 26/100
7/7 [==============================] - 1s 78ms/step - loss: 5.7350e-05 - val_loss: 0.0046
Epoch 27/100
7/7 [==============================] - 0s 65ms/step - loss: 6.1886e-05 - val_loss: 0.0028
Epoch 28/100
7/7 [==============================] - 1s 85ms/step - loss: 6.8587e-05 - val_loss: 0.0026
Epoch 29/100
7/7 [==============================] - 0s 67ms/step - loss: 5.7491e-05 - val_loss: 0.0032
Epoch 30/100
7/7 [==============================] - 0s 70ms/step - loss: 6.2797e-05 - val_loss: 0.0010
Epoch 31/100
7/7 [==============================] - 1s 73ms/step - loss: 6.2969e-05 - val_loss: 0.0024
Epoch 32/100
7/7 [==============================] - 0s 69ms/step - loss: 7.3474e-05 - val_loss: 0.0022
Epoch 33/100
7/7 [==============================] - 0s 68ms/step - loss: 7.3155e-05 - val_loss: 0.0022
Epoch 34/100
7/7 [==============================] - 0s 71ms/step - loss: 6.1248e-05 - val_loss: 0.0023
Epoch 35/100
7/7 [==============================] - 0s 71ms/step - loss: 5.1920e-05 - val_loss: 0.0017
Epoch 36/100
7/7 [==============================] - 1s 89ms/step - loss: 4.4118e-05 - val_loss: 0.0027
Epoch 37/100
7/7 [==============================] - 0s 55ms/step - loss: 6.8842e-05 - val_loss: 0.0011
Epoch 38/100
7/7 [==============================] - 0s 60ms/step - loss: 6.9882e-05 - val_loss: 4.2578e-04
Epoch 39/100
7/7 [==============================] - 0s 56ms/step - loss: 7.9708e-05 - val_loss: 0.0017
Epoch 40/100
7/7 [==============================] - 0s 54ms/step - loss: 5.9638e-05 - val_loss: 0.0038
Epoch 41/100
7/7 [==============================] - 0s 55ms/step - loss: 7.6531e-05 - val_loss: 0.0020
Epoch 42/100
7/7 [==============================] - 0s 67ms/step - loss: 5.5602e-05 - val_loss: 0.0027
Epoch 43/100
7/7 [==============================] - 0s 64ms/step - loss: 5.9910e-05 - val_loss: 0.0014
Epoch 44/100
7/7 [==============================] - 0s 69ms/step - loss: 5.3277e-05 - val_loss: 0.0010
Epoch 45/100
7/7 [==============================] - 0s 66ms/step - loss: 5.1738e-05 - val_loss: 0.0031
Epoch 46/100
7/7 [==============================] - 0s 62ms/step - loss: 4.8882e-05 - val_loss: 0.0014
Epoch 47/100
7/7 [==============================] - 0s 65ms/step - loss: 5.1974e-05 - val_loss: 0.0015
Epoch 48/100
7/7 [==============================] - 0s 67ms/step - loss: 4.7365e-05 - val_loss: 0.0017
Epoch 49/100
7/7 [==============================] - 0s 66ms/step - loss: 6.5271e-05 - val_loss: 0.0011
Epoch 50/100
7/7 [==============================] - 1s 116ms/step - loss: 5.4255e-05 - val_loss: 0.0022
Epoch 51/100
7/7 [==============================] - 1s 89ms/step - loss: 4.4477e-05 - val_loss: 0.0020
Epoch 52/100
7/7 [==============================] - 0s 65ms/step - loss: 6.2924e-05 - val_loss: 0.0029
Epoch 53/100
7/7 [==============================] - 0s 65ms/step - loss: 7.5896e-05 - val_loss: 0.0010
Epoch 54/100
7/7 [==============================] - 0s 71ms/step - loss: 7.0213e-05 - val_loss: 0.0017
Epoch 55/100
7/7 [==============================] - 0s 69ms/step - loss: 4.3784e-05 - val_loss: 0.0040
Epoch 56/100
7/7 [==============================] - 0s 70ms/step - loss: 5.5080e-05 - val_loss: 0.0013
Epoch 57/100
7/7 [==============================] - 1s 115ms/step - loss: 5.6540e-05 - val_loss: 0.0046
Epoch 58/100
7/7 [==============================] - 1s 89ms/step - loss: 5.5590e-05 - val_loss: 0.0017
Epoch 59/100
7/7 [==============================] - 0s 63ms/step - loss: 6.1450e-05 - val_loss: 0.0012
Epoch 60/100
7/7 [==============================] - 0s 54ms/step - loss: 4.8183e-05 - val_loss: 0.0035
Epoch 61/100
7/7 [==============================] - 0s 68ms/step - loss: 4.5041e-05 - val_loss: 0.0013
Epoch 62/100
7/7 [==============================] - 0s 55ms/step - loss: 5.4074e-05 - val_loss: 0.0030
Epoch 63/100
7/7 [==============================] - 0s 65ms/step - loss: 5.3017e-05 - val_loss: 0.0027
Epoch 64/100
7/7 [==============================] - 0s 67ms/step - loss: 5.4609e-05 - val_loss: 0.0027e
Epoch 65/100
7/7 [==============================] - 0s 56ms/step - loss: 6.5653e-05 - val_loss: 0.0062
Epoch 66/100
7/7 [==============================] - 0s 69ms/step - loss: 5.9848e-05 - val_loss: 0.0022
Epoch 67/100
7/7 [==============================] - 1s 73ms/step - loss: 5.1254e-05 - val_loss: 0.0036
Epoch 68/100
7/7 [==============================] - 1s 74ms/step - loss: 5.1232e-05 - val_loss: 0.0043
Epoch 69/100
7/7 [==============================] - 0s 67ms/step - loss: 5.1521e-05 - val_loss: 0.0029
Epoch 70/100
7/7 [==============================] - 0s 66ms/step - loss: 4.4319e-05 - val_loss: 0.0019
Epoch 71/100
7/7 [==============================] - 0s 61ms/step - loss: 7.0545e-05 - val_loss: 0.0031
Epoch 72/100
7/7 [==============================] - 0s 69ms/step - loss: 6.3606e-05 - val_loss: 0.0024
Epoch 73/100
7/7 [==============================] - 0s 68ms/step - loss: 7.1248e-05 - val_loss: 0.0022
Epoch 74/100
7/7 [==============================] - 1s 80ms/step - loss: 6.2988e-05 - val_loss: 0.0052
Epoch 75/100
7/7 [==============================] - 0s 66ms/step - loss: 7.1046e-05 - val_loss: 0.0019
Epoch 76/100
7/7 [==============================] - 0s 67ms/step - loss: 6.4913e-05 - val_loss: 0.0023
Epoch 77/100
7/7 [==============================] - 0s 69ms/step - loss: 5.6634e-05 - val_loss: 0.0027
Epoch 78/100
7/7 [==============================] - 0s 66ms/step - loss: 6.5549e-05 - val_loss: 0.0014
Epoch 79/100
7/7 [==============================] - 0s 66ms/step - loss: 6.2490e-05 - val_loss: 0.0016e-
Epoch 80/100
7/7 [==============================] - 0s 55ms/step - loss: 4.2525e-05 - val_loss: 0.0031
Epoch 81/100
7/7 [==============================] - 0s 57ms/step - loss: 4.5895e-05 - val_loss: 0.0018
Epoch 82/100
7/7 [==============================] - 0s 60ms/step - loss: 4.9728e-05 - val_loss: 0.0026
Epoch 83/100
7/7 [==============================] - 0s 57ms/step - loss: 5.1288e-05 - val_loss: 0.0016
Epoch 84/100
7/7 [==============================] - 0s 63ms/step - loss: 4.8391e-05 - val_loss: 0.0033
Epoch 85/100
7/7 [==============================] - 0s 66ms/step - loss: 5.1617e-05 - val_loss: 0.0032
Epoch 86/100
7/7 [==============================] - 0s 67ms/step - loss: 5.5168e-05 - val_loss: 0.0024
Epoch 87/100
7/7 [==============================] - 0s 63ms/step - loss: 4.1906e-05 - val_loss: 0.0019
Epoch 88/100
7/7 [==============================] - 0s 72ms/step - loss: 5.4777e-05 - val_loss: 0.0025
Epoch 89/100
7/7 [==============================] - 0s 67ms/step - loss: 4.8762e-05 - val_loss: 0.0032
Epoch 90/100
7/7 [==============================] - 0s 72ms/step - loss: 4.4579e-05 - val_loss: 0.0011
Epoch 91/100
7/7 [==============================] - 0s 67ms/step - loss: 5.4594e-05 - val_loss: 0.0018
Epoch 92/100
7/7 [==============================] - 0s 69ms/step - loss: 4.8852e-05 - val_loss: 0.0033
Epoch 93/100
7/7 [==============================] - 0s 65ms/step - loss: 4.4598e-05 - val_loss: 0.0020
Epoch 94/100
7/7 [==============================] - 0s 69ms/step - loss: 4.3907e-05 - val_loss: 0.0025
Epoch 95/100
7/7 [==============================] - 1s 98ms/step - loss: 5.5571e-05 - val_loss: 0.0030
Epoch 96/100
7/7 [==============================] - 0s 55ms/step - loss: 7.0220e-05 - val_loss: 9.7533e-04
Epoch 97/100
7/7 [==============================] - 0s 57ms/step - loss: 6.8708e-05 - val_loss: 0.0011
Epoch 98/100
7/7 [==============================] - 0s 65ms/step - loss: 7.7052e-05 - val_loss: 0.0023
Epoch 99/100
7/7 [==============================] - 0s 67ms/step - loss: 5.1925e-05 - val_loss: 0.0015
Epoch 100/100
7/7 [==============================] - 0s 68ms/step - loss: 4.9179e-05 - val_loss: 0.0032
```

Out[30]:

```
<keras.callbacks.History at 0x2441f486df0>
```

In [31]:

```python
#model prediction
train_predict=model_LSTM.predict(X_train)
test_predict=model_LSTM.predict(X_test)
```

In [32]:

```python
##Transferring the values  to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

In [33]:

```python
#calculating the error rate for training data
math.sqrt(mean_squared_error(y_train,train_predict))
```

Out[33]:

```
26.935864344145358
```

In [34]:

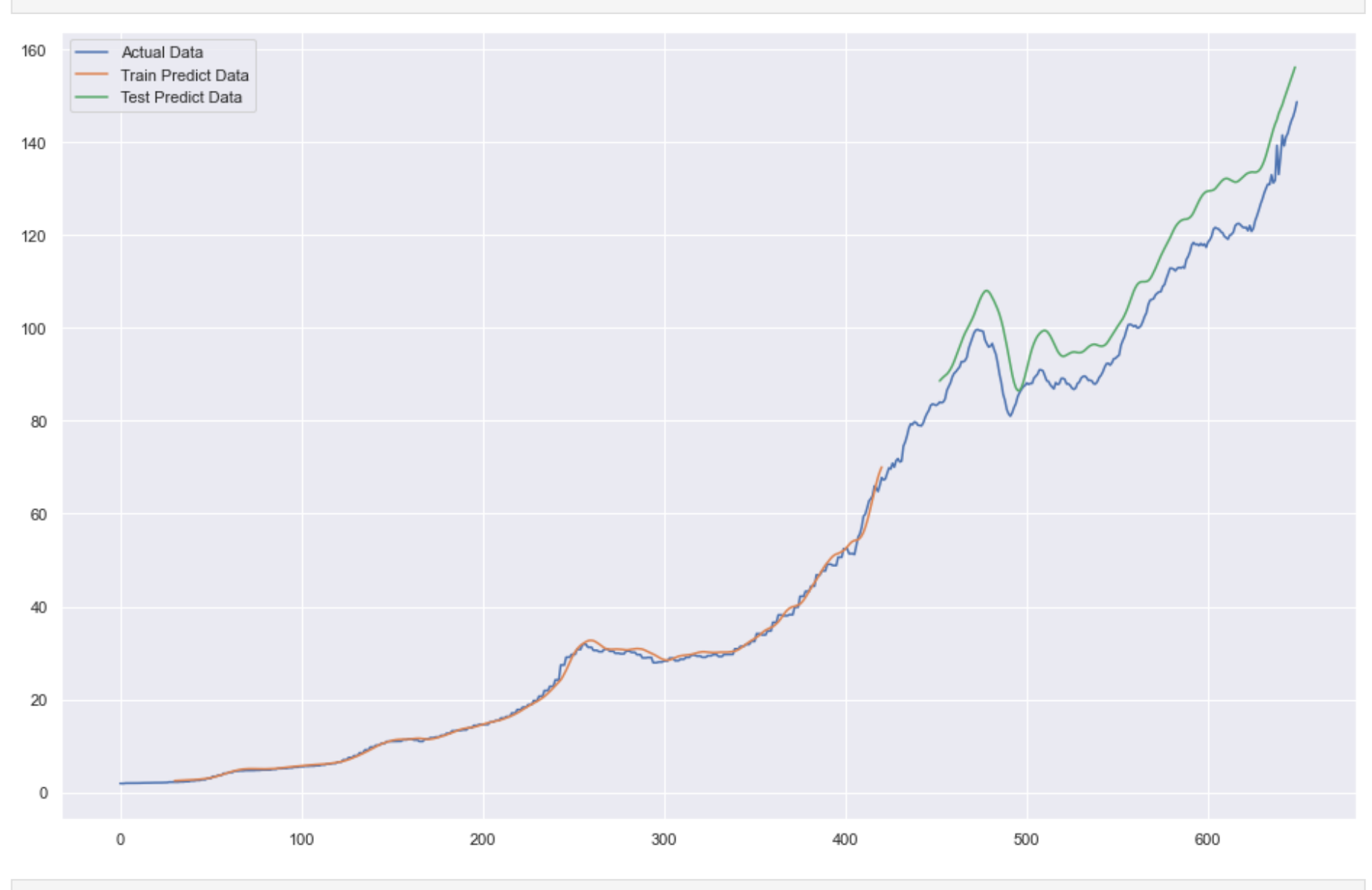
```python
#calculating the error rate of test data
math.sqrt(mean_squared_error(y_test,test_predict))
```

Out[34]:

```
112.37433070307412
```

In [35]:

```python
#plotting the train and test data with predicted data

look_back=30
trainPredictPlot = np.empty_like(df)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = np.empty_like(df)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df)-1, :] = test_predict
# plot baseline and predictions
plt.figure(figsize=(16,10))
plt.plot(scaler.inverse_transform(df), label ='Actual Data')
plt.plot(trainPredictPlot, label='Train Predict Data')
plt.plot(testPredictPlot,label='Test Predict Data')
plt.legend()
plt.show()
```



In [36]:

```python
#checking the length of test data
len(test_data)
```

Out[36]:

```
228
```

In [37]:

```python
#picking the last 30 data items from test data to predict next
x_input=test_data[198:].reshape(1,-1)
x_input.shape
```

Out[37]:

```
(1, 30)
```

In [38]:

```python
#creating a temporary list
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

In [41]:

```python
#creating a loop which predicts the next 10 months
lst_output=[]
n_steps=30
i=0
while(i<10):
    if(len(temp_input)>30):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} Month input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model_LSTM.predict(x_input, verbose=0)
        print("{} Month output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model_LSTM.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)
```

```
[1.0680151]
31
1 Month input [0.8158052  0.81105029 0.8184629  0.81000977 0.81426109 0.82567772
 0.83276479 0.84102041 0.85010828 0.8573754  0.86626959 0.87312353
 0.87921897 0.87884944 0.89285238 0.88107928 0.88492283 0.9359855
 0.89360627 0.91972175 0.95113835 0.93553713 0.94848795 0.95370845
 0.96413108 0.97246947 0.97866766 0.98780955 1.         1.0680151 ]
1 Month output [[1.0828885]]
2 Month input [0.81105029 0.8184629  0.81000977 0.81426109 0.82567772 0.83276479
 0.84102041 0.85010828 0.8573754  0.86626959 0.87312353 0.87921897
 0.87884944 0.89285238 0.88107928 0.88492283 0.9359855  0.89360627
 0.91972175 0.95113835 0.93553713 0.94848795 0.95370845 0.96413108
 0.97246947 0.97866766 0.98780955 1.         1.0680151  1.08288848]
2 Month output [[1.1018721]]
3 Month input [0.8184629  0.81000977 0.81426109 0.82567772 0.83276479 0.84102041
 0.85010828 0.8573754  0.86626959 0.87312353 0.87921897 0.87884944
 0.89285238 0.88107928 0.88492283 0.9359855  0.89360627 0.91972175
 0.95113835 0.93553713 0.94848795 0.95370845 0.96413108 0.97246947
 0.97866766 0.98780955 1.         1.0680151  1.08288848 1.10187209]
3 Month output [[1.1241963]]
4 Month input [0.81000977 0.81426109 0.82567772 0.83276479 0.84102041 0.85010828
 0.8573754  0.86626959 0.87312353 0.87921897 0.87884944 0.89285238
 0.88107928 0.88492283 0.9359855  0.89360627 0.91972175 0.95113835
 0.93553713 0.94848795 0.95370845 0.96413108 0.97246947 0.97866766
 0.98780955 1.         1.0680151  1.08288848 1.10187209 1.12419629]
4 Month output [[1.1491104]]
5 Month input [0.81426109 0.82567772 0.83276479 0.84102041 0.85010828 0.8573754
 0.86626959 0.87312353 0.87921897 0.87884944 0.89285238 0.88107928
 0.88492283 0.9359855  0.89360627 0.91972175 0.95113835 0.93553713
 0.94848795 0.95370845 0.96413108 0.97246947 0.97866766 0.98780955
 1.         1.0680151  1.08288848 1.10187209 1.12419629 1.14911044]
5 Month output [[1.1761914]]
6 Month input [0.82567772 0.83276479 0.84102041 0.85010828 0.8573754  0.86626959
 0.87312353 0.87921897 0.87884944 0.89285238 0.88107928 0.88492283
 0.9359855  0.89360627 0.91972175 0.95113835 0.93553713 0.94848795
 0.95370845 0.96413108 0.97246947 0.97866766 0.98780955 1.
 1.0680151  1.08288848 1.10187209 1.12419629 1.14911044 1.17619145]
6 Month output [[1.2048291]]
7 Month input [0.83276479 0.84102041 0.85010828 0.8573754  0.86626959 0.87312353
 0.87921897 0.87884944 0.89285238 0.88107928 0.88492283 0.9359855
 0.89360627 0.91972175 0.95113835 0.93553713 0.94848795 0.95370845
 0.96413108 0.97246947 0.97866766 0.98780955 1.         1.0680151
 1.08288848 1.10187209 1.12419629 1.14911044 1.17619145 1.2048291 ]
7 Month output [[1.2345588]]
8 Month input [0.84102041 0.85010828 0.8573754  0.86626959 0.87312353 0.87921897
 0.87884944 0.89285238 0.88107928 0.88492283 0.9359855  0.89360627
 0.91972175 0.95113835 0.93553713 0.94848795 0.95370845 0.96413108
 0.97246947 0.97866766 0.98780955 1.         1.0680151  1.08288848
 1.10187209 1.12419629 1.14911044 1.17619145 1.2048291  1.23455882]
8 Month output [[1.2650573]]
9 Month input [0.85010828 0.8573754  0.86626959 0.87312353 0.87921897 0.87884944
 0.89285238 0.88107928 0.88492283 0.9359855  0.89360627 0.91972175
 0.95113835 0.93553713 0.94848795 0.95370845 0.96413108 0.97246947
 0.97866766 0.98780955 1.         1.0680151  1.08288848 1.10187209
 1.12419629 1.14911044 1.17619145 1.2048291  1.23455882 1.26505733]
9 Month output [[1.2959816]]
[[1.0680150985717773], [1.0828884840011597], [1.1018720865249634], [1.1241962909698486], [1.1491104364395142],
[1.1761914491653442], [1.2048290967941284], [1.2345588207244873], [1.2650573253631592], [1.2959816455841064]]
```

In [42]:

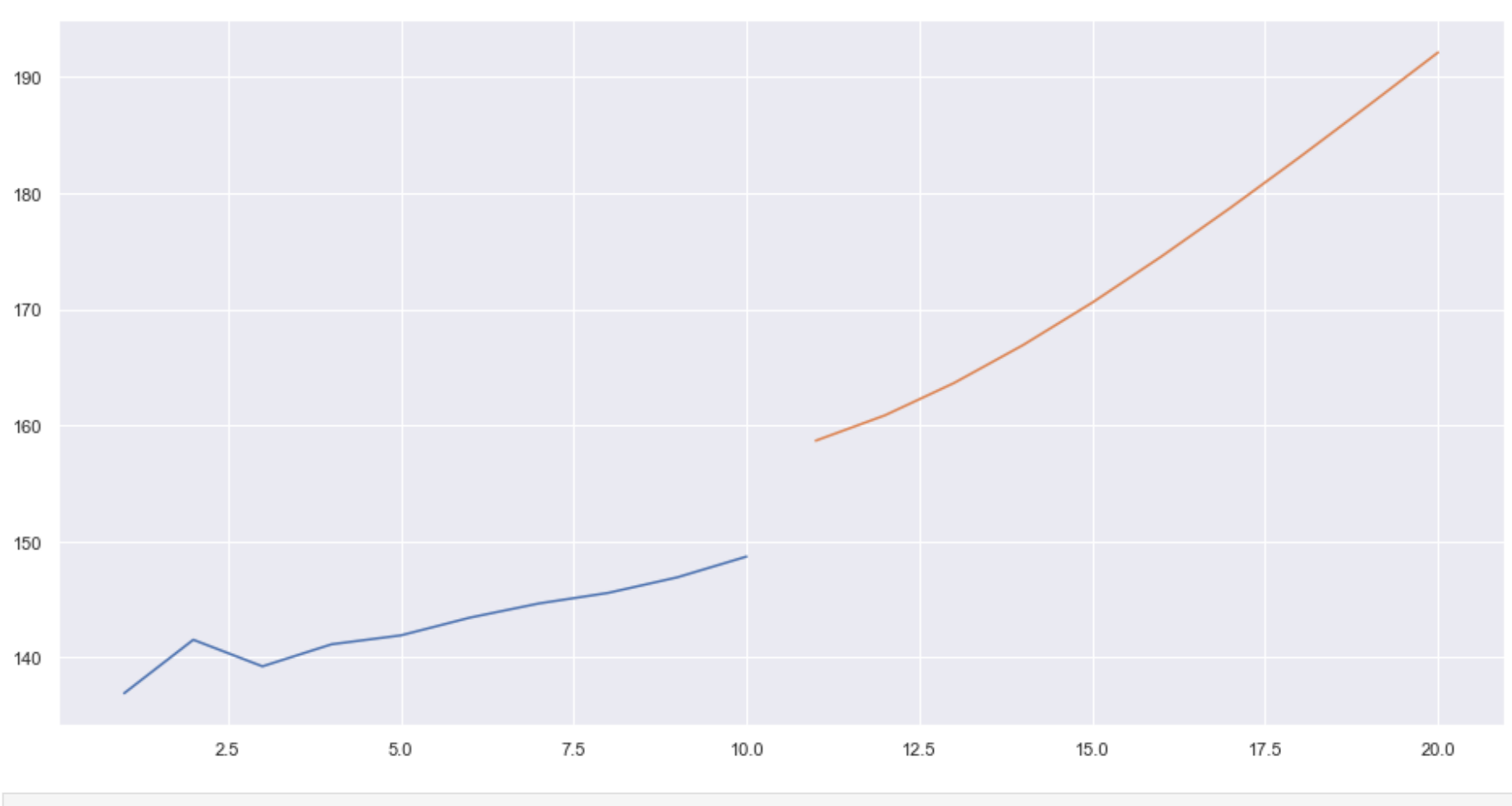
```python
#creating variables for evenly ranged values
day_new=np.arange(1,11)
day_pred=np.arange(11,21)
```

In [43]:

```python
plt.plot(day_new,scaler.inverse_transform(df[640:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
```

Out[43]:

```
[<matplotlib.lines.Line2D at 0x2442710b220>]
```

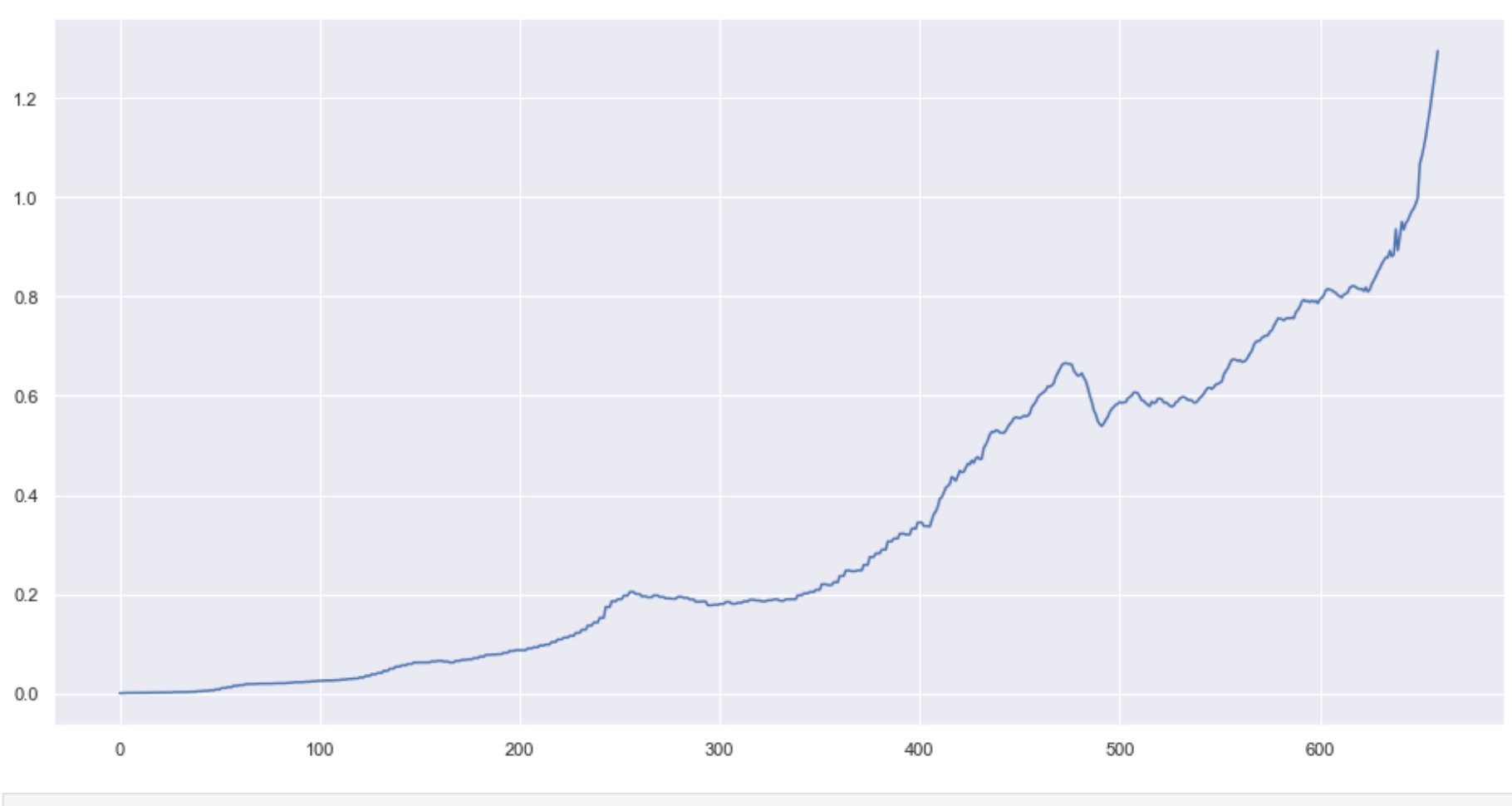


In [44]:

```python
#adding the predicted values to the previous ones and plotting it
df_ex=df.tolist()
df_ex.extend(lst_output)
plt.plot(df_ex[:680])
```

Out[44]:

```
[<matplotlib.lines.Line2D at 0x244271e7a60>]
```



In [45]:

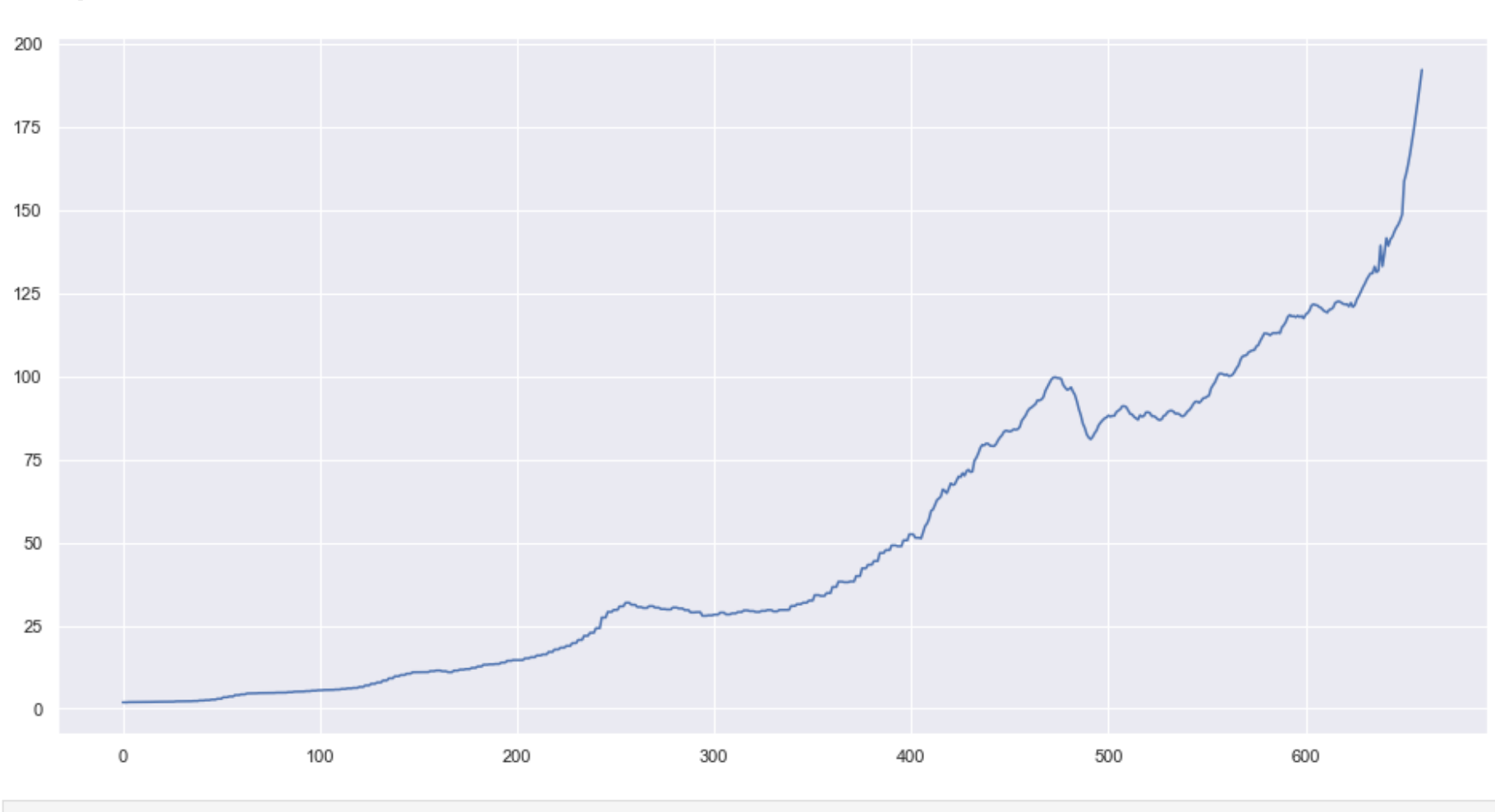
```python
#transforming the data
df_ex=scaler.inverse_transform(df_ex).tolist()
```

In [46]:

```python
#plotting the total
plt.plot(df_ex)
```

Out[46]:

```
[<matplotlib.lines.Line2D at 0x2442744ce50>]
```



In [47]:

```python
df_ex
```

```
Out[47]: [[1.8852988130000001],
 [1.8852988130000001],
 [1.8852988130000001],
 [1.949207248],
 [1.949207248],
 [1.949207248],
 [1.938555842],
 [1.938555842],
 [1.938555842],
 [1.970510059],
 [1.970510059],
 [1.970510059],
 [2.002464276],
 [2.002464276],
 [2.002464276],
 [2.023767088],
 [2.023767088],
 [2.023767088],
 [2.045069899],
 [2.045069899],
 [2.045069899],
 [2.055721305],
 [2.055721305],
 [2.055721305],
 [2.087675522],
 [2.087675522],
 [2.087675522],
 [2.183538174],
 [2.183538174],
 [2.183538174],
 [2.183538174],
 [2.183538174],
 [2.183538174],
 [2.236795202],
 [2.236795202],
 [2.236795202],
 [2.300703637],
 [2.300703637],
 [2.300703637],
 [2.439171911],
 [2.439171911],
 [2.439171911],
 [2.545685968],
 [2.545685968],
 [2.545685968],
 [2.705457054],
 [2.705457054],
 [2.705457054],
 [2.961090792],
 [2.961090792],
 [2.961090792],
 [3.408449832],
 [3.408449832],
 [3.408449832],
 [3.653432164],
 [3.653432164],
 [3.653432164],
 [4.068836987],
 [4.068836987],
 [4.068836987],
 [4.271213696],
 [4.271213696],
 [4.271213696],
 [4.5588016499999995],
 [4.5588016499999995],
 [4.5588016499999995],
 [4.6120586789999996],
 [4.6120586789999996],
 [4.6120586789999996],
 [4.675967113],
 [4.675967113],
 [4.675967113],
 [4.686618519],
 [4.686618519],
 [4.686618519],
 [4.729224142],
 [4.729224142],
 [4.729224142],
 [4.793132576],
 [4.793132576],
 [4.793132576],
 [4.814435388000001],
 [4.814435388000001],
 [4.814435388000001],
 [4.942252256],
 [4.942252256],
 [4.942252256],
 [5.091371937000001],
 [5.091371937000001],
 [5.091371937000001],
 [5.144628965],
 [5.144628965],
 [5.144628965],
 [5.251143022],
 [5.251143022],
 [5.251143022],
 [5.389611297],
 [5.389611297],
 [5.389611297],
 [5.517428166000001],
 [5.517428166000001],
 [5.517428166000001],
 [5.602639411],
 [5.602639411],
 [5.602639411],
 [5.645245034],
 [5.645245034],
 [5.645245034],
 [5.73045628],
 [5.73045628],
 [5.73045628],
 [5.900878772],
 [5.900878772],
 [5.900878772],
 [6.081952668999999],
 [6.081952668999999],
 [6.081952668999999],
 [6.220420943],
 [6.220420943],
 [6.220420943],
 [6.518660304],
 [6.518660304],
 [6.518660304],
 [6.97667075],
 [6.97667075],
 [6.97667075],
 [7.466635413],
 [7.466635413],
 [7.466635413],
 [7.860737425],
 [7.860737425],
 [7.860737425],
 [8.467867551],
 [8.467867551],
 [8.467867551],
 [9.117603301],
 [9.117603301],
 [9.117603301],
 [9.724733427000002],
 [9.724733427000002],
 [9.724733427000002],
 [10.10818403],
 [10.10818403],
 [10.10818403],
 [10.512937449999999],
 [10.512937449999999],
 [10.512937449999999],
 [10.93899368],
 [10.93899368],
 [10.93899368],
 [10.96029649],
 [10.96029649],
 [10.96029649],
 [10.981599299999997],
 [10.981599299999997],
 [10.981599299999997],
 [11.279838659999998],
 [11.279838659999998],
 [11.279838659999998],
 [11.46091256],
 [11.46091256],
 [11.46091256],
 [11.23723304],
 [11.23723304],
 [11.23723304],
 [10.96029649],
 [10.96029649],
 [10.96029649],
 [11.439609750000002],
 [11.439609750000002],
 [11.439609750000002],
 [11.727197700000001],
 [11.727197700000001],
 [11.727197700000001],
 [11.89762019],
 [11.89762019],
 [11.89762019],
 [12.24911658],
 [12.24911658],
 [12.24911658],
 [12.67517281],
 [12.67517281],
 [12.67517281],
 [13.2077431],
 [13.2077431],
 [13.2077431],
 [13.31425716],
 [13.31425716],
 [13.31425716],
 [13.420771209999998],
 [13.420771209999998],
 [13.420771209999998],
 [13.84682744],
 [13.84682744],
 [13.84682744],
 [14.37939773],
 [14.37939773],
 [14.37939773],
 [14.59242584],
 [14.59242584],
 [14.59242584],
 [14.59242584],
 [14.59242584],
 [14.59242584],
 [15.124996130000001],
 [15.124996130000001],
 [15.124996130000001],
 [15.4445383],
 [15.4445383],
 [15.4445383],
 [15.97710859],
 [15.97710859],
 [15.97710859],
 [16.29665076],
 [16.29665076],
 [16.29665076],
 [17.04224916],
 [17.04224916],
 [17.04224916],
 [17.78784756],
 [17.78784756],
 [17.78784756],
 [18.32041785],
 [18.32041785],
 [18.32041785],
 [18.85298813],
 [18.85298813],
 [18.85298813],
 [19.70510059],
 [19.70510059],
 [19.70510059],
 [20.66372711],
 [20.66372711],
 [20.66372711],
 [21.88749847],
 [21.88749847],
 [21.88749847],
 [22.80585505],
 [22.80585505],
 [22.80585505],
 [24.18338992],
 [24.18338992],
 [24.18338992],
 [27.39763795],
 [27.39763795],
 [27.39763795],
 [29.08129168],
 [29.08129168],
 [29.08129168],
 [29.6935294],
 [29.6935294],
 [29.6935294],
 [30.76494541],
 [30.76494541],
 [30.76494541],
 [31.83636141],
 [31.83636141],
 [31.83636141],
 [31.22412369],
 [31.22412369],
 [31.22412369],
 [30.55084533],
 [30.55084533],
 [30.55084533],
 [30.2761624],
 [30.2761624],
 [30.2761624],
 [30.825528249999998],
 [30.825528249999998],
 [30.825528249999998],
 [30.36772338],
 [30.36772338],
 [30.36772338],
 [29.94043883],
 [29.94043883],
 [29.94043883],
 [29.81835753],
 [29.81835753],
 [29.81835753],
 [30.398243700000002],
 [30.398243700000002],
 [30.398243700000002],
 [30.12356078],
 [30.12356078],
 [30.12356078],
 [29.63523558],
 [29.63523558],
 [29.63523558],
 [28.9332681],
 [28.9332681],
 [28.9332681],
 [29.01861697],
 [29.01861697],
 [29.01861697],
 [27.90908162],
 [27.90908162],
 [27.90908162],
 [28.05132974],
 [28.05132974],
 [28.05132974],
 [28.27892674],
 [28.27892674],
 [28.27892674],
 [28.876368850000002],
 [28.876368850000002],
 [28.876368850000002],
 [28.33582599],
 [28.33582599],
 [28.33582599],
 [28.64877186],
 [28.64877186],
 [28.64877186],
 [29.0470666],
 [29.0470666],
 [29.0470666],
 [29.53071021],
 [29.53071021],
 [29.53071021],
 [29.33156284],
 [29.33156284],
 [29.33156284],
 [29.07551622],
 [29.07551622],
 [29.07551622],
 [29.384620899999997],
 [29.384620899999997],
 [29.384620899999997],
 [29.644508709999997],
 [29.644508709999997],
 [29.644508709999997],
 [29.24621397],
 [29.24621397],
 [29.24621397],
 [29.67295834],
 [29.67295834],
 [29.67295834],
 [29.70140796],
 [29.70140796],
 [29.70140796],
 [30.86784257],
 [30.86784257],
 [30.86784257],
 [31.40838543],
 [31.40838543],
 [31.40838543],
 [31.835129799999997],
 [31.835129799999997],
 [31.835129799999997],
 [32.48947116],
 [32.48947116],
 [32.48947116],
 [34.13954938],
 [34.13954938],
 [34.13954938],
 [33.88350276],
 [33.88350276],
 [33.88350276],
 [34.7369915],
 [34.7369915],
 [34.7369915],
 [36.58621709],
 [36.58621709],
 [36.58621709],
 [38.17939606],
 [38.17939606],
 [38.17939606],
 [38.00869831],
 [38.00869831],
 [38.00869831],
 [38.23629531],
 [38.23629531],
 [38.23629531],
 [39.8579239],
 [39.8579239],
 [39.8579239],
 [42.19079311],
 [42.19079311],
 [42.19079311],
 [43.27187884],
 [43.27187884],
 [43.27187884],
 [44.3814142],
 [44.3814142],
 [44.3814142],
 [46.79963228],
 [46.79963228],
 [46.79963228],
 [47.68157064],
 [47.68157064],
 [47.68157064],
 [49.10405185999999],
 [49.10405185999999],
 [49.10405185999999],
 [48.84800524],
 [48.84800524],
 [48.84800524],
 [50.611881960000005],
 [50.611881960000005],
 [50.611881960000005],
 [52.43265792],
 [52.43265792],
 [52.43265792],
 [51.38002182],
 [51.38002182],
 [51.38002182],
 [51.20150042],
 [53.05854966],
 [54.9155989],
 [55.6584186],
 [57.09099944],
 [59.42557562],
 [59.90310257],
 [61.22956631],
 [62.7152057],
 [63.19273265],
 [63.82943524],
 [65.95177723],
 [65.47425028],
 [64.78448914],
 [66.16401143],
 [67.75576792],
 [67.27824097],
 [67.43741662],
 [68.60470471],
 [69.82505136],
 [69.61281716],
 [70.8331638],
 [70.03728555],
 [71.41680785],
 [71.84127624],
 [71.1515151],
 [71.31069075],
 [74.54726228],
 [75.44925762],
 [76.77572136],
 [78.4205364],
 [79.32253175],
 [79.16335610000002],
 [79.74700014],
 [79.64088304],
 [79.05723900000001],
 [79.00414733],
 [78.92821617],
 [79.5757433],
 [80.70720473000002],
 [81.57373712],
 [82.22123063],
 [83.27997671],
 [83.61857602],
 [83.4884067],
 [83.3213147],
 [83.56929005],
 [84.02644839],
 [83.90106635],
 [84.03811568],
 [84.7196006],
 [86.56111003],
 [87.38144064],
 [88.20926226],
 [89.47865094],
 [90.22983183],
 [90.59768337],
 [91.12833682],
 [91.59754895],
 [92.73809927],
 [92.7060532],
 [92.9699346],
 [93.6912864],
 [95.58267348],
 [96.67766106],
 [97.73591554],
 [98.96472591],
 [99.53883743],
 [99.66799121000001],
 [99.4355989],
 [99.38335173],
 [99.22825691],
 [97.43880398],
 [96.60122108],
 [95.89847146],
 [96.0574324],
 [96.64460261],
 [95.3665921],
 [94.3252631],
 [92.35697046],
 [90.03203546],
 [88.20557481],
 [85.72469566],
 [84.41731915],
 [82.46609398],
 [81.51335523],
 [81.00722929],
 [81.74138369],
 [82.87022818],
 [83.68663124],
 [85.18780297],
 [86.02804737],
 [86.70381405],
 [87.28658117],
 [87.58094362],
 [88.15549102000001],
 [87.83415287],
 [88.05378017999999],
 [88.04867624],
 [89.17627719],
 [89.60532026],
 [90.04725361],
 [90.95909915],
 [90.95403852],
 [90.72042788],
 [89.70958658999999],
 [88.64406248999998],
 [88.48125218000001],
 [87.74567853],
 [87.28147147000001],
 [86.87939315],
 [88.22705682],
 [87.83456424],
 [87.98309446],
 [89.09127032],
 [89.14487428],
 [88.92226247],
 [87.94127174000002],
 [87.9851811],
 [87.61326309999998],
 [87.01572809000001],
 [86.77897713],
 [87.03610237],
 [88.03612531],
 [88.32005117],
 [89.18743457000001],
 [89.52941305],
 [89.63492853],
 [89.31643871],
 [88.68411942],
 [88.75635758000001],
 [88.55474342],
 [87.96352364],
 [87.94614511],
 [88.47010538999999],
 [89.33754346],
 [89.80872879],
 [90.5538877],
 [91.57019948],
 [92.29924408],
 [92.35983956],
 [91.98251248],
 [92.49292028],
 [93.3420886],
 [93.45304911],
 [93.84076857],
 [94.16353809],
 [96.25871421999999],
 [97.2787639],
 [98.11845072],
 [99.49876176],
 [100.6646094],
 [100.7726233],
 [100.6239445],
 [100.285454],
 [100.5267755],
 [100.0],
 [100.085049],
 [100.4575155],
 [101.3425525],
 [102.43788],
 [103.2190061],
 [104.9703002],
 [105.93104479999998],
 [106.1488952],
 [106.293447],
 [107.1110571],
 [107.476627],
 [107.7619676],
 [107.8097556],
 [108.9173311],
 [109.3243607],
 [110.5984228],
 [111.6547846],
 [112.829855],
 [112.8392873],
 [112.6668774],
 [112.2948097],
 [112.8225539],
 [113.0254097],
 [112.8904772],
 [113.12906999999998],
 [112.8872191],
 [114.6732025],
 [115.3613565],
 [116.3472825],
 [117.8607536],
 [118.3952151],
 [117.9530007],
 [118.0564838],
 [117.7209981],
 [118.1811274],
 [117.7687697],
 [118.0768678],
 [117.3640934],
 [118.4854363],
 [118.9698875],
 [119.7678545],
 [121.2530285],
 [121.62576919999998],
 [121.39318760000002],
 [121.2657284],
 [120.74810480000001],
 [120.48846380000002],
 [119.74644669999999],
 [119.4443311],
 [119.1115474],
 [119.9745207],
 [120.1380376],
 [120.6561853],
 [122.0036281],
 [122.3959345],
 [122.4851029],
 [122.1616307],
 [121.7299961],
 [121.57048530000002],
 [121.647906],
 [120.9498714],
 [122.0380654],
 [120.7971199],
 [121.4212261],
 [123.0972213],
 [124.1376241],
 [125.3495738],
 [126.6836999],
 [127.7505349],
 [129.0562277],
 [130.0624068],
 [130.9572342],
 [130.9029866],
 [132.9586598],
 [131.2303333],
 [131.7945781],
 [139.2907281],
 [133.0693331],
 [136.9031629],
 [141.5152113],
 [139.2249071],
 [141.1261255],
 [141.8925094],
 [143.4225821],
 [144.6466837],
 [145.5565956],
 [146.8986518],
 [148.688246],
 [158.6730629235512],
 [160.8565197392327],
 [163.64336853795368],
 [166.92062754406797],
 [170.5780975256605],
 [174.55367000662102],
 [178.75776107902934],
 [183.1221721710619],
 [187.5994425368153],
 [192.13922388500492]]
```

```
In [49]: #save model
         model_LSTM.save("HPI_model.H5")
```

```
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditio
nal_losses, lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses, lstm_cell_layer_ca
ll_fn while saving (showing 5 of 10). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: HPI_model.H5\assets
INFO:tensorflow:Assets written to: HPI_model.H5\assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x000002447E7A8520> has the same name 'LSTMCell' as a b
uilt-in Keras object. Consider renaming <class 'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts whe
n loading with `tf.keras.models.load_model`. If renaming is not possible, pass the object in the `custom_object
s` parameter of the load function.
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x000002441A66D910> has the same name 'LSTMCell' as a b
uilt-in Keras object. Consider renaming <class 'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts whe
n loading with `tf.keras.models.load_model`. If renaming is not possible, pass the object in the `custom_object
s` parameter of the load function.
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Helps GUI in clustering the venues and returning the most visited venues

In [51]:

```
#importing libraries
import requests # library to handle requests
import pandas as pd # library for data analsysis
import numpy as np # library to handle data in a vectorized manner
import random # library for random number generation
from geopy.geocoders import Nominatim # module to convert an address into latitude and longitude values
# tranforming json file into a pandas dataframe library
import matplotlib.pyplot as plt # plotting library
# backend for rendering plots within the browser
%matplotlib inline
import matplotlib.cm as cm
import matplotlib.colors as colors
from sklearn.cluster import KMeans
import folium # plotting library
import geocoder

#function to get the co-ordinates of the location
def get_location(location):
    #getting the co-ordinates of the location
    geolocator = Nominatim(user_agent="uk")
    address = str(location +", United Kingdom")
    location = geocoder.arcgis(address)
    latlang = location.latlng
    return latlang

#function to import data of neighborhoods
def import_dataset(location):
    loc_neighborhoods = pd.read_excel(r'top25.xlsx',sheet_name='Sheet2')
    loc_neighborhoods=loc_neighborhoods[loc_neighborhoods.city==location]
    loc_neighborhoods.drop('city',inplace=True,axis=1)
    return loc_neighborhoods

#function to get co-ordinates of the neighborhoods
def getNeighborhoodLatLong(Neighborhoods,loc):
    #Neighborhood with lat , long
    latlang = []
    # defining the user agent for geopy
    geolocator = Nominatim(user_agent="uk_neighborhoods")
    # for every unique  of Neighborhood  get the lattitude and longitude
    for  neigh in Neighborhoods:
        address = str(neigh +", {}".format(loc))
        location = geocoder.arcgis(address)
        latlang.append(
        location.latlng
        )

    return(latlang)

#function to add coordinates to the dataframe
def add_coordinates(loc_neigh,loc_neighborhoods):
    latitude= []
    longitude = []
    for lat, lng in loc_neigh:
        latitude.append(lat)
        longitude.append(lng)
    loc_neighborhoods["Latitude"] = latitude
    loc_neighborhoods["Longitude"] = longitude
    return loc_neighborhoods

#function to call API to get most visitied venues
def api_call(names, latitudes, longitudes, radius=500):
    CLIENT_ID = 'WXER4KXQTW0IHEGA0HLO2EMDOOWEZKAX4YFQLRDMZ2TSBRVM'  # your Foursquare ID
    CLIENT_SECRET = '4NMKXNOKJBU4FR312SHLECHR1WSY0ONMQP2JXUYOZ0TWKTJG'  # your Foursquare Secret
    VERSION = '20210118'  # Foursquare API version
    LIMIT = 1000
    venues_list = []
    for name, lat, lng in zip(names, latitudes, longitudes):
        # create the API request URL
        url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius
            CLIENT_ID, CLIENT_SECRET, VERSION, lat, lng, radius, LIMIT)
        # make the GET request
        results = requests.get(url).json()["response"]['groups'][0]['items']
        # return only relevant information for each nearby venue
        venues_list.append([
            (name, lat, lng, v['venue']['name'], v['venue']['location']['lat'],
             v['venue']['location']['lng'],v['venue']['categories'][0]['name']) for v in results])
    nearby_venues = pd.DataFrame([item for venue_list in venues_list for item in venue_list])
    nearby_venues.columns = ['Neighborhood', 'Neighborhood Latitude', 'Neighborhood Longitude',
        'Venue', 'Venue Latitude', 'Venue Longitude', 'Venue Category']

    return (nearby_venues)

#function to check if the neighborhoods are listed in the neighborhood data
def venues_check(loc_neighborhoods,loc_venues):
    loc_neighborhoods = loc_neighborhoods.loc[loc_neighborhoods["neighborhood"].isin(loc_venues["Neighborhood"]
    return loc_neighborhoods

#function to apply one hot encoding to the dataset
def onehot(loc_venues):
    # one hot encoding
    loc_onehot = pd.get_dummies(loc_venues[['Venue Category']], prefix="", prefix_sep="")
    # add neighborhood column back to dataframe
    loc_onehot['Neighborhood'] = loc_venues['Neighborhood']
    loc_grouped = loc_onehot.groupby('Neighborhood').mean().reset_index()
    return loc_grouped

#function to return most common venues
def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[1:]
    row_categories_sorted = row_categories.sort_values(ascending=False)
    return row_categories_sorted.index.values[0:num_top_venues]

#function to append the data into the dataframe
def most_common(loc_grouped):
    num_top_venues = 10
    indicators = ['st', 'nd', 'rd']
    # create columns according to number of top venues
    columns = ['Neighborhood']
    for ind in np.arange(num_top_venues):
        try:
            columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
        except:
            columns.append('{}th Most Common Venue'.format(ind+1))
    # create a new dataframe
    neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
    neighborhoods_venues_sorted['Neighborhood'] = loc_grouped['Neighborhood']
    for ind in np.arange(loc_grouped.shape[0]):
        neighborhoods_venues_sorted.iloc[ind, 1:] = return_most_common_venues(loc_grouped.iloc[ind, :], num_top
    return neighborhoods_venues_sorted


#function to apply kmeans clustering on the dataset
def model(loc_grouped,neighborhoods_venues_sorted,loc_neighborhoods):

    kclusters = 4
    loc_grouped_clustering = loc_grouped.drop('Neighborhood', 1)
    # run k-means clustering
    kmeans = KMeans(n_clusters=kclusters, random_state=0).fit(loc_grouped_clustering)
    #add clustering labels
    neighborhoods_venues_sorted.insert(0, 'Cluster Labels', kmeans.labels_)
    loc_merged = loc_neighborhoods
    # merge loc_grouped with loc_data to add latitude/longitude for each neighborhood
    loc_merged = loc_merged.join(neighborhoods_venues_sorted.set_index('Neighborhood'), on='neighborhood')
    return loc_merged

#function to call all previous functions together
def main_function(location):
    latlang = get_location(location)
    loc_neighborhoods = import_dataset(location)
    loc_neigh = getNeighborhoodLatLong(Neighborhoods = loc_neighborhoods["neighborhood"],loc=location)
    loc_neighborhoods = add_coordinates(loc_neigh,loc_neighborhoods)
    loc_venues = api_call(names=loc_neighborhoods['neighborhood'],latitudes=loc_neighborhoods['Latitude'],
                                longitudes=loc_neighborhoods['Longitude'])
    loc_neighborhoods =venues_check(loc_neighborhoods,loc_venues)
    loc_grouped = onehot(loc_venues)
    neighborhoods_venues_sorted = most_common(loc_grouped)
    loc_merged =model(loc_grouped,neighborhoods_venues_sorted,loc_neighborhoods)
    return loc_merged

#function to return the most common venue of particular cluster
def most_venues(df):
    df_tmp =pd.DataFrame(df.groupby("neighborhood")["1st Most Common Venue"].apply(lambda x: ','.join(x.unique
    df_tmp.rename(columns={'neighborhood':'neighborhood' ,'top_venues':'1st Most Common Venue'})
    venues = df_tmp["1st Most Common Venue"].values
    counter ={}
    if len(venues)==0:
        return None
    else:
        for ven in venues:
            if ven in counter:
                counter[ven]=counter[ven] +1
            else:
                counter[ven]=0
        max_val = list(counter.values())
        max_ke = list(counter.keys())
        return max_ke[max_val.index(max(max_val))]
```

In [ ]:

In [ ]:

Helps GUI in fetching data from API

In [1]:

```
#importing required libraries
import requests
import time
import pandas as pd
```

In [2]:

```
#importing dataset
df_post = pd.read_excel(r'area_codes.xlsx')
```

In [3]:

```
#function to return bedroom prices as per the location passed through API
def get_bed_prices(location):

    postcodes = df_post.postcode[df_post.AreaName==location].unique()
    records =[]
    for bed,postcode in zip(range(2,6),postcodes):
        response = requests.get('https://api.propertydata.co.uk/prices?key=L1SGXGR1LQ&postcode={}&bedrooms={}'.
        data = response.json()
        if data['status']=='error':
            continue
        elif data['status']== 'success':
            for row in data['data']['raw_data']:
                records.append(row)
        time.sleep(5)
    df = pd.DataFrame.from_dict(records, orient='columns')
    return df
```

In [4]:

```
#function to return crime data as per the location passed through API
def get_crime_data(location):
    postcodes = df_post.postcode[df_post.AreaName==location].unique()
    records =[]
    for postcode in postcodes:
        response = requests.get('https://api.propertydata.co.uk/crime?key=L1SGXGR1LQ&postcode={}'.format(postc
        data = response.json()
        if data['status']=='error':
            continue
        elif data['status']== 'success':
            records.append((postcode,data['crimes_last_12m'],data['population'],data['crime_rating'],data['cri
        time.sleep(5)
    df = pd.DataFrame.from_dict(records, orient='columns')
    return df
```

In [5]:

```
#function created to speed up the GUI after calling the bedroom api for all cities
def get_bed_prices1(location):
    df = pd.read_csv(r'Bedroom_Data/{}_Bed.csv'.format(location))
    return df
```

In [6]:

```
#function created to speed up the GUI after calling the crime data api for all cities
def get_crime_data1(location):
    df = pd.read_csv('Crime_Data\{}_crime.csv'.format(location))
    return df
```

Graphical User Interface

In [1]:

```python
#importing libraries
from tkinter import *
from tkinter.ttk import *
import pandas as pd
import tkintermapview
from geopy.geocoders import Nominatim
import geocoder
import requests
import numpy as np
import random
import matplotlib.pyplot as plt
import joblib
import seaborn as sns
from sklearn.cluster import KMeans
import folium # plotting library
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import warnings
import time
warnings.filterwarnings('ignore')
%run assist.ipynb
%run api_calls.ipynb
from keras.models import load_model
from sklearn.preprocessing import MinMaxScaler
```

In [2]:

```python
#importing various datasets
df = pd.read_excel(r'top25.xlsx')
df_post = pd.read_excel(r'area_codes.xlsx')
df_model = pd.read_csv(r'indices_dataset/Indices-2022-05_ONS.csv')
df_land = pd.read_excel(r'indices_dataset/UK-HPI-full-file-2022-05_land_registry_1.xlsx')
df_reg = pd.read_excel(r'indices_dataset/pp-2022-modified.xlsx')
df_price = pd.read_excel(r'price_post_coordinates.xlsx')
df_dummy = pd.read_csv(r'dummy_csv.csv')
df_dummy.drop(['Unnamed: 0','Price'],axis=1,inplace=True)
df_city = pd.read_csv(r'city.csv')
```

In [6]:

```python
### creating window for gui
window = Tk()
window.title("House Price Prediction System")
width = window.winfo_screenwidth()
height = window.winfo_screenheight()

# setting tkinter window size
window.geometry("%dx%d" % (width, height))
heading = Label(text="House Price Prediction System", font=("Arial Black", 22))
heading.pack()

#creating different tabs in the window
tab_parent = Notebook(window)
tab1 = Frame(tab_parent)
tab2 = Frame(tab_parent)
tab3 = Frame(tab_parent)
tab_parent.add(tab1, text='Your Interest City Search')
tab_parent.add(tab2, text="Know the Price")
tab_parent.add(tab3, text="Research")
#creating a list of countries name
countries = list(df.Country.unique())

#function to pick county name as per country selected
def pick_county(e):
    if cm1.get()=="England":
        data = list(df.county[df.Country=='England'].values)
        cm2.config(value=data)
    elif cm1.get()=="Wales":
        data = list(df.county[df.Country=='Wales'].values)
        cm2.config(value=data)
    elif cm1.get()=="Scotland":
        data = list(df.county[df.Country=='Scotland'].values)
        cm2.config(value=data)
    elif cm1.get()=="Northern Ireland":
        data = list(df.county[df.Country=='Northern Ireland'].values)
        cm2.config(value=data)

#function to pick city name as per county name selected
def pick_city(e):
    data = list(df.City[df.county == cm2.get()].values)
    cm3.config(value=data)



#getting the co-ordinates of the location
def location(location):
    geolocator = Nominatim(user_agent="uk")
    address = str(location +", United Kingdom")
    location = geocoder.arcgis(address)
    latlang = location.latlng
    return latlang

#class to plot the graphs using clustering
class interests:
    def graph(self):
        self.loc = cm3.get()
        self.df_final = main_function(self.loc)
        latlang =location(self.loc)
        lbl55= Label(tab1, text="Most Common Venues", font=("Helvetica", 12,))
        lbl55.place(x=50, y=100)
        map_widget =tkintermapview.TkinterMapView(tab1,width=500,height=400,corner_radius=0)
        map_widget.set_position(latlang[0],latlang[1])
        map_widget.set_zoom(11)
        map_widget.place(x=50,y=120)
        for lat,long,name,cluster in zip(self.df_final['Latitude'], self.df_final['Longitude'], self.df_final['
            if cluster == 0 :
                map_widget.set_marker(lat,long, text=name,marker_color_circle='red',marker_color_outside='red')
            elif cluster == 1 :
                map_widget.set_marker(lat,long, text=name,marker_color_circle='blue',marker_color_outside='blue
            elif cluster == 2 :
                map_widget.set_marker(lat,long, text=name,marker_color_circle='green',marker_color_outside='gre
            elif cluster == 3 :
                map_widget.set_marker(lat,long, text=name, marker_color_circle='purple',marker_color_outside='p
        lbl8 = Label(tab1, text="Most Common Venues :",font=('bold'))
        lbl8.place(x=600, y=150)
        lbl9 = Label(tab1, text="Red :  {}".format(most_venues(self.df_final[self.df_final['Cluster Labels']==(
        lbl9.place(x=600, y=200)
        lbl10 = Label(tab1, text="Blue :  {} ".format(most_venues(self.df_final[self.df_final['Cluster Labels']
        lbl10.place(x=600, y=230)
        lbl11 = Label(tab1, text="Green :  {} ".format(most_venues(self.df_final[self.df_final['Cluster Labels'
        lbl11.place(x=600, y=260)
        lbl56 = Label(tab1, text="Purple :  {} ".format(most_venues(self.df_final[self.df_final['Cluster Labels
        lbl56.place(x=600, y=290)

        lbl57= Label(tab1, text="Average House Prices", font=("Helvetica", 12))
        lbl57.place(x=800, y=100)
        map_widget1 =tkintermapview.TkinterMapView(tab1,width=500,height=400,corner_radius=0)
        map_widget1.set_position(latlang[0],latlang[1])
        map_widget1.set_zoom(11)
        map_widget1.place(x=800,y=120)
        temp_loc = df_price[df_price.city1==self.loc]
        max_val = temp_loc.Avg_asking_price.max()
        min_val =temp_loc.Avg_asking_price.min()
        labels=['low','medium','high']
        bins = np.linspace(min_val,max_val,4)
        temp_loc['bins'] = pd.cut(temp_loc['Avg_asking_price'], bins=bins, labels=labels, include_lowest=True)
        for lat,long,post,bins in zip(temp_loc['Latitude'], temp_loc['Longitude'],temp_loc['Postcode'], temp_lo
            if bins == "low" :
                map_widget1.set_marker(lat,long, text=post,marker_color_circle='pink',marker_color_outside='pin
            elif bins == "medium" :
                map_widget1.set_marker(lat,long, text=post,marker_color_circle='orange',marker_color_outside='o
            elif bins == "high" :
                map_widget1.set_marker(lat,long, text=post,marker_color_circle='cyan',marker_color_outside='cya
        lbl58 = Label(tab1, text="House Prices:", font='bold')
        lbl58.place(x=600, y=350)
        temp_low = temp_loc[temp_loc.bins=='low']
        min_low = temp_low.Avg_asking_price.min()
        max_low = temp_low.Avg_asking_price.max()
        lbl59 = Label(tab1, text="Pink :£{}-£{}".format(min_low,max_low),foreground='pink')
        lbl59.place(x=600, y=370)
        temp_med = temp_loc[temp_loc.bins=='medium']
        min_med = temp_med.Avg_asking_price.min()
        max_med = temp_med.Avg_asking_price.max()
        lbl60 = Label(tab1, text="Orange :£{}-£{}".format(min_med,max_med),foreground='orange')
        lbl60.place(x=600, y=400)
        temp_high = temp_loc[temp_loc.bins=='low']
        min_high = temp_high.Avg_asking_price.min()
        max_high = temp_high.Avg_asking_price.max()
        lbl61 = Label(tab1, text="Cyan :£{}-£{}".format(min_high,max_high),foreground='cyan')
        lbl61.place(x=600, y=430)


    #function to work when know more is clicked
    def know_more(self):
        new_win = Toplevel(tab1)
        new_win.geometry("%dx%d" % (width, height))
        lbl4 = Label(new_win, text="Most Common Venues in {} :".format(self.loc), font=("Helvetica", 15))
        lbl4.place(x=110, y=20)
        table = Treeview(new_win)
        table['columns'] = ('Neighborhood', 'Latitude','Longitude','1st Most Common Venue',
                            '2nd Most Common Venue','3rd Most Common Venue','4th Most Common Venue','5th Most C
        table.column("#0", width=0, stretch=NO)
        table.column("Neighborhood", anchor=CENTER, width=100)
        table.column("Latitude", anchor=CENTER, width=150)
        table.column("Longitude", anchor=CENTER, width=150)
        table.column("1st Most Common Venue", anchor=CENTER, width=150)
        table.column("2nd Most Common Venue", anchor=CENTER, width=150)
        table.column("3rd Most Common Venue", anchor=CENTER, width=150)
        table.column("4th Most Common Venue", anchor=CENTER, width=150)
        table.column("5th Most Common Venue", anchor=CENTER, width=150)
        table.heading("#0", text="", anchor=CENTER)
        table.heading("Neighborhood", text="Neighborhood", anchor=CENTER)
        table.heading("Latitude", text="Latitude", anchor=CENTER)
        table.heading("Longitude", text="Longitude", anchor=CENTER)
        table.heading("1st Most Common Venue", text="1st Most Common Venue", anchor=CENTER)
        table.heading("2nd Most Common Venue", text="2nd Most Common Venue", anchor=CENTER)
        table.heading("3rd Most Common Venue", text="3rd Most Common Venue", anchor=CENTER)
        table.heading("4th Most Common Venue", text="4th Most Common Venue", anchor=CENTER)
        table.heading("5th Most Common Venue", text="5th Most Common Venue", anchor=CENTER)
        output_data = []
        for i in range(len(self.df_final)):
            output_data.append((self.df_final.iloc[i,0], self.df_final.iloc[i,1],self.df_final.iloc[i,2],self.d
                               self.df_final.iloc[i,5],self.df_final.iloc[i,6],self.df_final.iloc[i,7],self.df_
        for out in output_data:
            table.insert(parent='', index='end', text='', values=out)
        table.place(x=120,y=50)
        lbl5 = Label(new_win, text="House Prices per bedroom count in {} :".format(self.loc), font=("Helvetica"
        lbl5.place(x=110, y=290)
        df_beds = get_bed_prices1(self.loc)
        if len(df_beds)==0:
            lbl7 = Label(new_win, text="Insufficient data for  {} Present !!! Sorry!!! :".format(self.loc), for
            lbl7.place(x=120, y=350)
        else:
            figure1 = plt.Figure(figsize=(8, 4))
            ax1 = figure1.add_subplot(111)
            plot1 = FigureCanvasTkAgg(figure1, new_win)
            dict_data = {'2 Bedrooms':df_beds.price[df_beds.bedrooms==2],'3:Bedrooms':df_beds.price[df_beds.bed
                         '4 Bedrooms':df_beds.price[df_beds.bedrooms==4],'5 Bedrooms':df_beds.price[df_beds.bed
            ax1.boxplot(dict_data.values())
            ax1.set_xticklabels(dict_data.keys())
            plot1.get_tk_widget().place(x=120, y=320)
            temp2 =df_beds[df_beds.bedrooms==2]
            temp3 =df_beds[df_beds.bedrooms==3]
            temp4 =df_beds[df_beds.bedrooms==4]
            temp5 =df_beds[df_beds.bedrooms==5]
            lbl13 = Label(new_win, text="Average Price of 2 Bedrooms:  £{:.2f}".format(temp2.price.mean()),
                                                                                                  font=("Hel
            lbl13.place(x=800, y=350)
            lbl13 = Label(new_win, text="Average Price of 3 Bedrooms:  £{:.2f}".format(temp3.price.mean()),
                                                                                                  font=("Hel
            lbl13.place(x=800, y=375)
            lbl13 = Label(new_win, text="Average Price of 4 Bedrooms:  £{:.2f}".format(temp4.price.mean()),
                                                                                                  font=("Hel
            lbl13.place(x=800, y=400)
            lbl13 = Label(new_win, text="Average Price of 5 Bedrooms:  £{:.2f}".format(temp5.price.mean())
                                                                                                  , font=("H
            lbl13.place(x=800, y=425)
            lbl6 = Label(new_win, text="Crimes in {} :".format(self.loc), font=("Helvetica", 15))
            lbl6.place(x=120, y=620)
        df_crime = get_crime_data1(self.loc)
        if len(df_crime) == 0:
            lbl17 = Label(new_win, text="Crimes in 12 months: No Sufficient Data Present! Sorry!!!", font=("Hel
            lbl17.place(x=350, y=625)
            lbl19 = Label(new_win, text="Crime Rating: No Sufficient Data Present! Sorry!!!", font=("Helvetica"
            lbl19.place(x=700, y=625)

        else:
            mean_crimes = df_crime['1'].mean()
            lbl17 = Label(new_win, text="Crimes in 12 months: {:.2f}".format(mean_crimes), font=("Helvetica", 1
            lbl17.place(x=350, y=625)
            crime_rating = df_crime['3'].mode()[0]
            lbl19 = Label(new_win, text="Crime Rating: {}".format(crime_rating), font=("Helvetica", 10))
            lbl19.place(x=700, y=625)
        b5 = Button(new_win, text="EXIT", command =new_win.destroy)
        b5.place(x=1200,y=550)

# function to pick county name as per country selected
def pcounty1(e):
    if cm7.get()=='England':
        data = list(df_reg.County[df_reg.Country=='England'].unique())
        data.sort()
        cm8.config(value=data)

    elif cm7.get()=="Wales":
        data = list(df_reg.County[df_reg.Country=='Wales'].unique())
        data.sort()
        cm8.config(value=data)

#function to pick district name as per county selected
def pdistrict1(e):
    data = list(df_reg.District[df_reg.County == cm8.get()].unique())
    data.sort()
    cm9.config(value=data)

#function to pickcity name as per district selected
def pcity1(e):
    data =list(df_reg.Town_City[df_reg.District == cm9.get()].unique())
    data.sort()
    cm10.config(value=data)

#function to pick areacode as per city selected
def pareacode1(e):
    data =list(df_reg.areacode[df_reg.Town_City == cm10.get()].unique())
    data.sort()
    cm11.config(value=data)


#function to calculate the property prices using XGBoost machine learning model
def calc_property_price():
    rank = df_city.Rank[df_city.city==cm8.get()]
    c_rank =rank.values[0]
    header = df_dummy.columns
    df_prediction = pd.DataFrame(columns=header)
    df_prediction = df_prediction.append(pd.Series(0, index=df_prediction.columns), ignore_index=True)
    areacode = cm11.get()
    if (cm12.get() =="Detached"):
        property_type='D'
    elif (cm12.get()=="Semi-Detached"):
        property_type='S'
    elif cm12.get() == "Terraced":
        property_type ='T'
    elif cm12.get() == "Flats/Maisonettes":
        property_type ='F'
    elif cm12.get() == "Other":
        property_type ='O'
    if cm13.get()=="Newly Built":
        property_age =1
    elif cm13.get()=="Established":
        property_age =0
    if cm14.get()=="Freehold":
        property_tenure=0
    elif cm14.get()=="Leasehold":
        property_tenure =1
    if cm15.get()=="single residential property sold":
        property_ppd =0
    elif cm15.get()=="buy-to-lets/others":
        property_ppd =1
    property_epc = cm16.get()
    property_year = cm17.get()
    fl_area = total_floor_area_e.get()
    energy_tar = cm18.get()
    df_prediction['Old/New']=property_age
    df_prediction['Property_Type_{}'.format(property_type)]=1
    df_prediction['Duration']=property_tenure
    df_prediction['PPD\xa0Category Type']=property_ppd
    df_prediction['construction_age_band_ {}'.format(property_year)]=1
    df_prediction['current_energy_rating_{}'.format(property_epc)]=1
    df_prediction['total_floor_area']=fl_area
    df_prediction['energy_tariff_{}'.format(energy_tar)]=1
    df_prediction['areacode_{}'.format(areacode)]=1
    df_prediction['county_rank']=c_rank
    preds = model_xgboost.predict(df_prediction)
    lbl62 = Label(tab2, text="Estimate Property Price :£{:.2f}".format(preds[0]),font=("Arial Black", 12))
    lbl62.place(x=750, y=250)


#function to calculate the prices of development and display profit percentage
def calc_profit():
    url ='https://api.propertydata.co.uk/development-calculator?key=L1SGXGR1LQ&postcode={}&purchase_price={}&sq
    response = requests.get(url)
    data = response.json()
    lbl29=Label(tab2, text="Stamp Duty: £{}".format(data['results']['stampDuty']['value']), font=("Arial Black"
    lbl29.place(x=850,y=300)
    lbl30=Label(tab2, text="Legal fees: £{}".format(data['results']['legals']['value']), font=("Arial Black", 1
    lbl30.place(x=850,y=320)
    lbl31=Label(tab2, text="Architects fees: £{}".format(data['results']['architects']['value']), font=("Arial
    lbl31.place(x=850,y=340)
    lbl32=Label(tab2, text="Construction Cost: £{}".format(data['results']['construction']['value']), font=("Ar
    lbl32.place(x=850,y=360)
    lbl33=Label(tab2, text="subTotal Costs: £{}".format(data['results']['subTotalCosts']['value']), font=("Aria
    lbl33.place(x=850,y=380)
    lbl34=Label(tab2, text="Contingency Amount: £{}".format(data['results']['contingency']['value']), font=("Ar
    lbl34.place(x=850,y=400)
    lbl35=Label(tab2, text="finance costs: £{}".format(data['results']['finance']['value']), font=("Arial Black
    lbl35.place(x=850,y=420)
    lbl36=Label(tab2, text="total Costs: £{}".format(data['results']['totalCosts']['value']), font=("Arial Blac
    lbl36.place(x=850,y=440)
    lbl37=Label(tab2, text="Average asking price:(persqf) £{}".format(data['results']['valuePsf']['value']), fo
    lbl37.place(x=850,y=460)
    lbl38=Label(tab2, text="Total Value: £{}".format(data['results']['totalValue']['value']), font=("Arial Blac
    lbl38.place(x=850,y=480)
    lbl39=Label(tab2, text="Profit: £{}".format(data['results']['profit']['value']), font=("Arial Black", 11))
    lbl39.place(x=850,y=500)
    lbl40=Label(tab2, text="Profit Percentage: {}".format(data['results']['profitPc']['value']), font=("Arial B
    lbl40.place(x=850,y=540)

#function to forecast HPI and display on frame 3
def HPI_pred(temp_input,x_input):
    lst_output=[]
    n_steps=30
    i=0
    while(i<30):

        if(len(temp_input)>30):
            x_input=np.array(temp_input[1:])
            x_input=x_input.reshape(1,-1)
            x_input = x_input.reshape((1, n_steps, 1))
            yhat = model_hpi.predict(x_input, verbose=0)
            temp_input.extend(yhat[0].tolist())
            temp_input=temp_input[1:]
            lst_output.extend(yhat.tolist())
            i=i+1
        else:
            x_input = x_input.reshape((1, n_steps,1))
            yhat = model_hpi.predict(x_input, verbose=0)
            temp_input.extend(yhat[0].tolist())
            lst_output.extend(yhat.tolist())
            i=i+1
    return lst_output


#loading models
model_hpi = load_model('HPI_model.H5')
model_xgboost = joblib.load('xgboost_model1.sav')


#calling class modules
inter = interests()

#widgets for tab1
lbl1 = Label(tab1, text="Please choose the country :", font=("Helvetica", 15))
lbl1.place(x=20, y=50)
cm1 = Combobox(tab1,values=countries)
cm1.place(x=270, y=53)
cm1.bind("<<ComboboxSelected>>",pick_county)
lbl2 = Label(tab1, text="County :", font=("Helvetlstica", 15))
lbl2.place(x=450, y=50)
cm2 = Combobox(tab1,values=[" "])
cm2.place(x=540,y=53)
cm2.bind("<<ComboboxSelected>>",pick_city)
lbl3 = Label(tab1, text="City :", font=("Helvetica", 15))
lbl3.place(x=750, y=50)
cm3 = Combobox(tab1,values=[" "])
cm3.place(x=810,y=53)
b1 = Button(tab1, text="Go!",command =inter.graph)
b1.place(x=1000,y=53)
lb =Label(tab1,text=cm3.get())
lb.place(x=1200,y=500)
b2 = Button(tab1, text="Know More!",command=inter.know_more)
b2.place(x=600,y=500)
b3 = Button(tab1, text="EXIT", command =window.destroy)
b3.place(x=1200,y=550)

#tab2 widgets
lbl20 =Label(tab2, text="Please enter the following details to Know the House Price :", font=("Arial Black", 12
lbl20.place(x=50, y=10)
c_name =['England','Wales']
lbl45 = Label(tab2, text ="Select the Country :")
lbl45.place(x=60,y=40)
cm7 = Combobox(tab2,values=c_name)
cm7.place(x=290,y=40)
cm7.bind("<<ComboboxSelected>>",pcounty1)
lbl46 =Label(tab2, text ="Select the County :")
lbl46.place(x=60,y=70)
cm8 = Combobox(tab2,values=[" "])
cm8.place(x=290,y=70)
cm8.bind("<<ComboboxSelected>>",pdistrict1)
lbl47 = Label(tab2,text="Select the District :")
lbl47.place(x=60,y=100)
cm9 =Combobox(tab2,values= [" "])
cm9.place(x=290,y=100)
cm9.bind("<<ComboboxSelected>>",pcity1)
lbl48 = Label(tab2,text="Select the Town/City :")
lbl48.place(x=60,y=130)
cm10 =Combobox(tab2,values= [" "])
cm10.place(x=290,y=130)
cm10.bind("<<ComboboxSelected>>",pareacode1)
lbl49 = Label(tab2,text="Select the Post code :")
lbl49.place(x=60,y=160)
cm11 =Combobox(tab2,values= [" "])
cm11.place(x=290,y=160)
lbl50 = Label(tab2,text="Select the Property Type:")
lbl50.place(x=60,y=190)
prop_type = ['Detached', 'Semi-Detached', 'Terraced','Flats/Maisonettes','Other']
cm12 = Combobox(tab2,values=prop_type)
cm12.place(x=290,y=190)
lbl51 = Label(tab2,text="Select the Age of Property:")
lbl51.place(x=60,y=220)
prop_age = ['Newly Built','Established']
cm13 = Combobox(tab2,values=prop_age)
cm13.place(x=290,y=220)
lbl52 = Label(tab2,text="Select the Tenure of Property:")
lbl52.place(x=60,y=250)
prop_tenure = ['Freehold','Leasehold']
cm14 = Combobox(tab2,values=prop_tenure)
cm14.place(x=290,y=250)
lbl52 = Label(tab2,text="Select the type of Price Paid transaction:")
lbl52.place(x=60,y=280)
prop_ppd = ['single residential property sold','buy-to-lets/others']
cm15 = Combobox(tab2,values=prop_ppd)
cm15.place(x=290,y=280)
lbl53 = Label(tab2,text="Select Energy Ratings:")
lbl53.place(x=600,y=40)
prop_epc = ['A','B','C','D','E','F','G','O']
cm16 = Combobox(tab2,values=prop_epc)
cm16.place(x=790,y=40)
lbl53 = Label(tab2,text="Select Year Built:")
lbl53.place(x=600,y=70)
prop_year = ['1950-1966','1900-1929','1930-1949','1967-1975','before 1900','1983-1990','1976-1982','1996-2002',
 '2003-2006','2007-2011','2007 onwards','2012 onwards']
prop_year.sort()
cm17 = Combobox(tab2,values=prop_year)
cm17.place(x=790,y=70)
lbl54 = Label(tab2,text="Enter Total Floor Area:")
lbl54.place(x=600,y=100)
total_floor_area_e = Entry(tab2,width=23)
total_floor_area_e.place(x = 790, y = 100)
lbl55 = Label(tab2,text="Select Energy Tariff:")
lbl55.place(x=600,y=130)
prop_tariff =['Single','Unknown','dual','off-peak 7 hour','standard tariff',
              'off-peak 10 hour','dual (24 hour)','off-peak 18 hour','24 hour']
prop_tariff.sort()
cm18 = Combobox(tab2,values=prop_tariff)
cm18.place(x=790,y=130)
b8 = Button(tab2, text="Calculate Price!", command=calc_property_price)
b8.place(x=700,y=200)
lbl22 =Label(tab2, text="Please enter the following details to Know the Profit on Property :", font=("Arial Bla
lbl22.place(x=50, y=320)
lbl23 =Label(tab2, text="Full UK PostCode :", font=("Helvetica", 10))
lbl23.place(x=50, y=370)
postcode_e1 = Entry(tab2,width=23)
postcode_e1.place(x = 500, y = 370)
lbl24 =Label(tab2, text="Initial purchase price(in pounds) :", font=("Helvetica", 10))
lbl24.place(x=50, y=400)
ini_e2 = Entry(tab2,width=23)
ini_e2.place(x = 500, y = 400)
lbl25 =Label(tab2, text="The size of the property before development(in square feet) :", font=("Helvetica", 10)
lbl25.place(x=50, y=430)
sqp_e3 = Entry(tab2,width=23)
sqp_e3.place(x = 500, y = 430)
lbl26 =Label(tab2, text="The size of the property after development(in square feet) :", font=("Helvetica", 10))
lbl26.place(x=50, y=460)
sqa_e4 = Entry(tab2,width=23)
sqa_e4.place(x = 500, y = 460)
lbl27 =Label(tab2, text="Type of development project (refurbish / demolition) :", font=("Helvetica", 10))
lbl27.place(x=50, y=490)
data_dev =['refurbish','demolition']
dev_cm5=Combobox(tab2,values=data_dev)
dev_cm5.place(x = 500, y = 490)
lbl28 =Label(tab2, text="Interior finish quality (premium / medium / basic) :", font=("Helvetica", 10))
lbl28.place(x=50, y=520)
data_fin =['premium','medium','basic']
fin_cm6=Combobox(tab2,values=data_fin)
fin_cm6.place(x = 500, y = 520)
b4 = Button(tab2, text="Check!", command =calc_profit)
b4.place(x=550,y=550)
b6 = Button(tab2, text="EXIT", command =window.destroy)
b6.place(x=1200,y=550)

#tab3 widgets
#plotting prediction
df_mod = df_model[df_model.Region_Name=='United Kingdom']
lbl41 =Label(tab3, text="House Price Index Prediction :", font=("Helvetica", 10))
lbl41.place(x=50, y=60)
figure2 = plt.Figure(figsize=(7,3))
ax2 = figure2.add_subplot(111)
plot2 = FigureCanvasTkAgg(figure2, tab3)
data1=df_mod[['Index']].values
scaler = MinMaxScaler(feature_range=(0,1))
data1 = scaler.fit_transform(np.array(data1).reshape(-1,1))
x_input =data1[-30:]
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
preds = HPI_pred(temp_input,x_input)
predicted_value =scaler.inverse_transform(preds).tolist()
actual_value =(df_mod[['Index']].values).tolist()[-300:]
actual_value.extend(predicted_value)
day_new=np.arange(1,331)
ax2.plot(day_new,actual_value)
plot2.get_tk_widget().place(x=50, y=100)

#plotting prices as per land registry
df_uk = df_land[df_land.RegionName=='United Kingdom']
df_scot = df_land[df_land.RegionName=='Scotland']
df_eng = df_land[df_land.RegionName=='England']
df_wales= df_land[df_land.RegionName=='Wales']
df_NI = df_land[df_land.RegionName=='Northern Ireland']
lbl42 =Label(tab3, text="Average House Prices Country Wise :", font=("Helvetica", 10))
lbl42.place(x=650, y=60)
figure3 = plt.Figure(figsize=(7,3))
ax3 = figure3.add_subplot(111)
plot3 = FigureCanvasTkAgg(figure3, tab3)
sns.lineplot(data = df_scot, x='Date',y='AveragePrice', label='Scotland',ax=ax3)
sns.lineplot(data = df_eng, x='Date',y='AveragePrice', label='England',ax=ax3)
sns.lineplot(data = df_wales, x='Date',y='AveragePrice', label='Wales',ax=ax3)
sns.lineplot(data = df_NI, x='Date',y='AveragePrice', label='Northern Ireland',ax=ax3)
plot3.get_tk_widget().place(x=650, y=100)

#plot for average house price per region
property_type =['Detached', 'Semi-Detached', 'Terraced','Flat']
uk_property = [df_uk['DetachedPrice'].mean(),df_uk['SemiDetachedPrice'].mean(),
               df_uk['TerracedPrice'].mean(),df_uk['FlatPrice'].mean()]
lbl43 =Label(tab3, text="Average House Prices as per Property type in United Kingdom :", font=("Helvetica", 10)
lbl43.place(x=50, y=350)
figure4 = plt.Figure(figsize=(7,3))
ax4 = figure4.add_subplot(111)
plot4 = FigureCanvasTkAgg(figure4, tab3)
ax4.bar(property_type,uk_property)
plot4.get_tk_widget().place(x=50, y=380)

#plotting average sales volume
country_name = ['ENGLAND', 'SCOTLAND','WALES','NORTHERN IRELAND']
sales_lst = [df_eng['SalesVolume'].mean(),df_scot['SalesVolume'].mean(),df_wales['SalesVolume'].mean(),
             df_NI['SalesVolume'].mean()]
lbl44 =Label(tab3, text="Average Sales Volume of Properties as per Country:", font=("Helvetica", 10))
lbl44.place(x=650, y=350)
figure5 = plt.Figure(figsize=(7,3))
ax5 = figure5.add_subplot(111)
plot5 = FigureCanvasTkAgg(figure5, tab3)
ax5.bar(country_name,sales_lst)
plot5.get_tk_widget().place(x=650, y=380)
b7 = Button(tab3, text="EXIT", command =window.destroy)
b7.place(x=1200,y=550)

tab_parent.pack(expand=1, fill="both")
window.mainloop()
```