SOLENT UNIVERSITY, SOUTHAMPTON FACULTY OF BUSINESS, LAW, AND DIGITAL TECHNOLOGY.

M.Sc Applied Artificial Intelligence and Data Science

Academic Year 2021-2022

Ayorinde Janet Olaide

Classification of lithology facies using machine learning model

Supervisor: Dr Muntasir Al-Asfoor

September 2022

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

Acknowledgement

First, I would like to thank God for the gift of life, and for the wisdom, knowledge and understanding that was used to carryout this study. My special thanks also go to my supervisor, Dr Muntasir Al-Asfoor, for his time and invaluable contribution to the success of this dissertation. I would also like to appreciate other lecturers that took their time to teach and support me, Dr. Olufemi Isiaq, Dr. Shakeel Ahmad, Dr. Prins Butt and Dr. Drishty Sobnath. Special appreciation also goes to my husband Egbebi Adebiyi, my daughter, mother and sisters for their support and unwavering encouragement.

Abstract

Understanding the geology of a reservoir requires accurate identification of its lithological facies. Lithological facies classification is the process of determining rock lithology by analyzing indirect measurements such as well logs. Well logging is a standard tool for determining reservoir parameters. The interpretation of well logging data, on the other hand, may take some time but machine learning (ML) can automate the process in less time.

The objective of the study is to investigate the use of machine learning and well logs in classifying rock facies, and to improve the accuracy of different machine learning model compared to previous published research on the same dataset. A total of 118 wells, with 12 known lithology from a field in North Sea were available for analysis and classification, the dataset is divided into three for training, testing and validation purpose.

Nine machine learning model which include Random Forest (RF), Decision tree, Support vector machine (SVM), Gradient boosting (GB), Categorical boosting (CatBoost), Light gradient boosting (LGBM), Extreme gradient boosting (XGB), K-Nearest Neighbour (KNN), and logistic regression was designed and trained with a sample of the dataset. The random forest model outperforms the other methods in classifying the lithofacies in the dataset with 91% accuracy. For each model, I also tested various scenarios including feature engineering, machine learning imputation, data augmentation and outlier removal to improve the performance of the models.

Random Forest, CatBoost, XGB and Gradient boosting performed best without any form of feature engineering, outlier filter and ML imputation. Logistic regression, decision tree and SVM performed best when the dataset has ML imputation (scenario 2), KNN and LGBM performed best with feature engineering. The top 3 model were then used to train the entire train set and evaluated with the open and hidden dataset. The RF base model performed best with an accuracy of 78% and 80%, on the open and hidden data respectively, the accuracy on the open data increased by 1% when the data is trained with hyperparameters and augmented data. The accuracy of XGB model increased by 4% and 3% when the data was trained with additional feature.

According to the findings that I obtained, machine learning algorithms have a significant application in the automatic classification of lithology facies with high accuracy and efficiency. It has the potential to significantly improve the rock physical property estimation process while simultaneously reducing the amount of manual labour required.

Contents

List of Tables

List of figures

Acronyms

ML Machine learning

RF Random Forest

SVM Support vector machine

GB Gradient boosting

CatBoost Categorical boosting

LGBM Light gradient boosting model

XGB Extreme gradient boosting

KNN K-Nearest Neighbour

MLA Machine learning algorithms

STOOIP stock tank oil originally in place

CHAPTER ONE

INTRODUCTION

1.1 General Statement

Machine learning (ML) algorithms are designed to locate and correctly predict patterns in multivariate data as quickly as possible. The application of machine and deep learning techniques to investigate volumes of diverse multivariate geospatial data holds enormous promise for industry and research in the geosciences. The oil and gas business are critical to increasing energy demands to increase profit. To overcome the problems connected with diverse exploration and production operations, newer unconventional wells are drilled for the extraction of hydrocarbons, which necessitates technological innovations. The oil and gas business demands increasingly creative technologies to remain competitive in the globalized energy market. These technologies must enable continuous, cost-effective, high-quality, and long-term production.

The fields of big data analytics, machine learning, and deep learning have become particularly attractive study subjects for a wide range of applications as a direct result of the development of methods involving high levels of computational complexity. (Wang and Alexander 2015; Anifowose *et al*. 2014). These methods are capable of processing large volumes of data, extracting meaningful information from raw data, and identifying hidden patterns in the data. These advanced techniques can quickly filter out noise, reduce dimensionality, model nonlinear relationships, and are occasionally useful in dealing with reservoir uncertainties (Bhattacharya *et al*. 2016; Wang and Alexander 2015).

As a result, ML algorithms have been developed to handle complicated categorization and estimation challenges (Chaki *et al*. 2015; Avseth and Mukerji 2002). Machine learning has various advantages, including being cost-effective, allowing for speedy mitigation of real-world problems, real-time deployment, allowing for real-world automation, and being found to be more resilient and reliable. ML models, on the other hand, have significant challenges with data

reliance, data availability, immature infrastructure, and multidisciplinary knowledge (Bhattacharya *et al*. 2016; Wang and Alexander 2015)

Machine learning models can help solve real challenges in the petroleum industry. As a result, hybrid computational models, such as ensemble models, are becoming increasingly popular. These techniques are critical, especially when high classification or estimate accuracy is desired, because they can improve a ML model's performance by improving its modelling strategy (Anifowose *et al*. 2014).

The classification of facies is a crucial stage in reservoir modelling. During oil and gas exploration and production, a precise understanding of facies is crucial for enhancing the characterization of reservoir features. It aids in the better knowledge of the geology, petrophysical, and reservoir properties by geologists and engineers.

The oil and gas industry has paid close attention to the rapid development of data-driven modelling methods in recent years (Sebtosheikh and Salehi 2015 and Xie *et al*. 2018). Research is being conducted on several methodologies, and these approaches are being implemented in a variety of contexts, such as well log interpretation, seismic signal analysis, and seismic interpretation (Zu *et al*. 2018; Qu *et al*. 2019; Chen 2020; Lopez *et al*. 2020).

Geoscience data are usually characterized by a limited number, distribution of direct observations, irreducible noise, high intraclass variability and interclass similarity. As a result, the machine learning method or algorithms chosen, as well as the specifics of how they are applied, must be appropriate for the context of geoscience data. Therefore, this study will primarily focus on the use of feature engineering and ensemble models for lithofacies identification using well logs.

1.2. Problem statement

The identification and mapping of porous and permeable sandstone reservoirs storing commercial volumes of hydrocarbons is critical to the success of any conventional hydrocarbon exploration programme (Primmer *et al.*, 1997). Traditionally, geologists manually review well-logs to recognise various lithofacies layers based on their experience. However, when complex well-logs are interpreted manually, there is always a significant risk of human error. Well log data usually

overlap, and they are not linearly connected which causes manual interpretation of lithofacies to fail. It is also time consuming and expensive to manually interpret well logs.

Utilizing machine learning algorithms to systematically speed up the process and accurately predict the rock lithofacies from well log is one of the potential solutions to this problem. To achieve the desired level of accuracy in the outputs, the data-driven algorithms require a significant amount of data, which must be utilised during the training process in a balanced manner.

Building a reservoir model to produce hydrocarbons requires an understanding of the petrophysical characteristics of rocks and their spatial distribution in relation to lithofacies (Bai *et al*. 2012). The more well logs that need to be interpreted, the more challenging the task becomes. Therefore, it is necessary to automate the reservoir characterization process. As a result, geologists may create more accurate quantitative evaluation models of various rock qualities, which can enhance oil production. Machine learning methods, such as ensembles, may be able to address issues in the oil and gas sector. In this thesis, the ensemble approach and feature engineering are specifically examined to address problems with lithofacies identification.

1.3. Research question

1. What makes some supervised machine learning algorithms so good at predicting lithofacies from geophysical well log data?
2. Which machine learning algorithm will best classify lithology in this data?
3. Can data augmentation improve the performance of the classification model?
4. Can creating new features from the existing ones improves the classification accuracy?

1.4. Objectives

This study attempts to apply machine learning methods for lithofacies classification and aims to achieve the following objectives:

1. To classify lithofacies using ensemble classifiers.
2. To enhance the prediction accuracy of lithofacies classification using ML techniques as compared to earlier published research on the same dataset.

CHAPTER TWO

THEORETICAL BACKGROUND AND LITERATURE REVIEW

2.1. Machine Learning

Machine learning uses the ability of computers to learn, execute calculations, store data, and receive instruction swiftly to process and train data for classification or prediction purpose (Sivia 1996; Burl *et al*. 1998). Reinforcement learning is one of the four primary paradigms that comprise machine learning, along with unsupervised learning, semi-supervised learning, and supervised learning.

There is a wide variety of machine learning algorithms, and each one has its own specific set of advantages and disadvantages for the purpose of resolving geoscience issues. Machine learning algorithms (MLA) uses an automatically adaptive approach to recognise patterns in data and then apply the learned relationships to other data sets with patterns that are similar. MLAs have the capability to generate predictions for classification and regression problems inductively, which is particularly helpful when the process being investigated is represented by high-dimensional multivariate input data (Witten and Frank 2005; Kotsiantis 2007; Kanevski *et al*. 2009).

Common examples of supervised machine learning algorithms include Decision Trees, Random Forests (RF), Support Vector Machines (SVM), Linear and Logistic Regression, Naive Bayes, Linear Discriminant Analysis, k-Nearest Neighbor (KNN) algorithms, and Neural Networks. Other examples include Linear Regression, Naive Bayes, and Linear Discriminant Analysis. Each of these independent machine learning models can be placed into one of three broad categories: linear models, nonlinear models, or Ensemble models.

Unsupervised learning techniques, which are sometimes referred to as exploratory data analysis techniques in statistics, are responsible for automatically determining the organisation of a dataset. Because of this, relying on a priori categorization or other limits and assumptions that have been pre-imposed is reduced. Unsupervised methods are thought to be more robust than supervised methods because they estimate the manifold that best represents a dataset only based on the input data.

Likewise, unsupervised methods are resistant to extreme observations or outliers in the dataset. Cluster analysis techniques are a group of techniques that contain most unsupervised machine learning techniques. These techniques allow clusters to be inferred using Euclidian, probabilistic, or similarity distance measurements. Hidden Markov Models, Self-Organizing Maps (SOM), Gaussian Mixture Models, Hierarchical Clustering, and k-Means Clustering are some of the most common types of clustering methods.

2.2. Well and well logs

Subsurface formations and wireline logs (also called well logs) are used as part of the research process. Therefore, knowing the structure of formations in a well is essential. It is equally important to understand the concept of a well as it is to understand what a formation is and how it is distinct from groups. The physical and mineralogical characteristics of rocks and their interactions with those around them are used to classify them into lithostratigraphic units, such as formations and groups.

A lithostratigraphic unit can be composed of sedimentary, igneous, or metamorphic-equivalent rocks. These are the three primary types of rock that occur naturally. The idea that more recent layers are constructed on top of older ones is the most important concept of stratigraphy(Geological Survey of Norway, 2015). The classification of lithostratigraphic units can be done using formations and groups. A series of beds that is distinct from other beds both above and below it and is thick enough to be shown on geological maps is referred to as a formation. While a formation is the most fundamental formal unit of lithostratigraphic classification, a group can be defined as the succession of two or more connected formations that share significant and diagnostic lithologic features (Salvador and Murphy 1998).

The most accurate approach for learning the structure of the lithostratigraphic units in a well would be obtained through the extraction of core samples taken from the reservoir rock. However, since this process requires a lot of labour and is expensive, it is not commonly used in the oil industry (Dubois *et al*. 2007). Considering the abundance of additional important methods, core samples alone are not sufficient to obtain sufficient information about a well to adequately interpret the properties

of the reservoir. Wireline or downhole logging is the method that has gained the most popularity in the modern oil and gas industry.

Wireline logging is a method that plots precise information about geological formations against each depth point of the well (Leyland 2017).This information is obtained by sending down a cable into the borehole that is equipped with several different measuring instruments. After reaching each predetermined depth in the well, the various measurement devices collect data on a variety of signals, including resistivity, density, and porosity, amongst others. This can be done either during the drilling operations or after they have been completed (Vakarelov 2016).

2.2.1. Gamma ray log

The gamma ray log is a tool that is used to determine the total amount of natural gamma radiation that is emitted by a formation. The isotope series composed of uranium, radium, thorium, and potassium-40, are the primary contributors to this gamma radiation. It is common practise to denote the gamma ray log with the symbol GR. We can detect and quantify the presence of radioactive elements thanks to the ability of the isotopes that are produced when these chemical elements interact to build structures at the nuclear energy level. These isotopes do this by emitting gamma rays (Søland and Thue 2019). After being released by an isotope in the formation, gamma rays gradually lose energy because of collisions with other atoms in the rock over a period (compton scattering). The process of Compton scattering continues right up until the gamma ray's energy becomes so low that the formation can completely absorb it. The amount of Compton scattering will be directly proportional to the density of the formation (Glover 2014). The gamma ray is an instrument that can be utilised to assist in the interpretation of lithology. Even though the gamma ray log on its own is unable to determine distinct lithology formations, when combined with other logs, it can provide useful information lithology (Glover 2014).

2.2.2. Resistivity Log

Electrical resistivity is a measure of how much a material can impede the flow of an electrical current. A material's resistivity or specific resistance is its resistance over a unit cube (typically 1 m3) at a given temperature. The resistivity of underground formations can be measured with electrical logs (or "resistivity logs"). Resistivity readings can be taken at three depths: shallow, medium, and deep. It is impossible to overstate the importance of electrical logs to a Petro-physicist's toolkit. This is because they can be used to calculate the stock tank oil originally in place (STOOIP) (Asquith *et al*. 2004).

A rock is made up of millions of grains, the density of which is determined by the porosity of the given rock. The space between the grains of a rock with high porosity is larger, making the rock more permeable. Electrical currents are typically not carried through rock matrix, but rather through salty formation waters. Dry rock is an excellent electrical insulator, meaning that it does not transmit electrical currents. Interstitial water in the pore space or absorbed in the solid matrix of rocks (e.g., clay minerals) will be less resistive or more conductive.

Resistivity log can also be used for qualitative purposes, such as lithology indications, facies and electro-facies analysis, correlation, overpressure determination, shale porosity determination, indications of compaction, and source rock examination.

2.2.3. SP log

In general, the spontaneous potential instrument separates porous, permeable sandstones from surrounding shales by measuring natural electrical potentials that arise in boreholes. The "natural battery" results from the interaction of two solutions with varying ion concentrations when drilling mud with a salinity that differs from formation fluids is utilised. Ions spread from a concentrated solution (usually formation water) to a diluted one. The SP instrument measures the natural potential in millivolts, which is produced by the ion flow, which is electrical current (Schlumberger *et al*. 1934)

The spontaneous potential log (SP), also known as self-potential, measures the self-potential differential between the borehole and the surface in the absence of any

artificially applied current. SP can be measured in two ways: potential gradient and potential amplitude. The potential gradient technique uses a pair of electrodes spaced at a constant distance (typically between 5 and 10 m) apart and a division by that distance to determine the gradient (Singha *et al*. 2021).

There are primarily four applications for the SP log:

• The identification of beds that are permeable.

• Estimating Rw.

• A measure of the amount of shale present in a formation.

• Correlation.

2.2.4. Sonic Log

The sonic or acoustic log is used to measure the amount of time it takes for an elastic wave to move through the formation. This can be used to calculate the velocity of elastic waves as they move through the formation. The bore hole-centred sonic log tool which comprises of an acoustic transmitter and two receivers are separated from one another and from the transmitter. This is done to ensure that the sonic pulse is radiated in a symmetrical manner and that measurements are taken concurrently on all sides of the hole. In addition, this helps ensure that the hole is not distorted. The time it takes for an elastic wave, or a "sound" pulse, to travel from a transmitter to a receiver on the device is measured. The transmitted pulse is both intense and fleeting in duration. This is attenuated and dispersed (the wave energy is spread out over time and space) as it travels through the rock in its various forms (loss of energy through absorption of energy by the formations). Since different rock types have varying degrees of transparency, the log can be used to determine the porosity of a hole that is filled with liquid (Rider 1991).

The main uses of sonic log are:

• Recording "seismic" velocity and travel time in a borehole.

• Synthetic seismograms can be made possible with the help of the "seismic" data provided.

• Estimation of porosity (together with the FDC and CNL tools).

• Lithostratigraphic correlation.

• Lithology identification.

• Source rock detection

• Fracture detection.

• Detection of over-pressures and compaction in rocks.

2.2.5. Density Log.

The purpose of the density log is to determine the overall density of the rock. It establishes the relationship between density and porosity in the rock. Gas-containing deposits and evaporites are also detected with density log. It is primarily employed in determining the total porosity of the formation. Furthermore, it helps in the recognition of evaporites and the detection of gas-bearing formations (Rider 2011).

Like other active open hole tools, a density log operates by projecting gamma rays into a formation and measuring its scattering at Cesium-137, which produces gamma rays with a high energy, is the source used by the sondes. The scattering back gamma rays that detectors pick up depend on the formation's electron density.

The radioactive source is installed on the well bore wall in a protected sidewall skid and emits medium gamma rays into the formation. Gamma ray waves can be thought of as energetic particles. The gamma ray loses some of its energy to the electron during the creation process as these energy particles (photons) collide with them. This is referred to as Compton scattering. The greater the number of electrons in the formation, the more energy is wasted due to collisions. The energy loss is proportional to porosity if the matrix density is known. Lower bulk density is implied by lower electron density. Density log is a very good method for lithological identification when used with neutron log. To calculate porosity, the formation density log is typically used. Other important uses of density log are finding gas deposits and locating mineral composition (particularly evaporites). Combining formation density log data with neutron log data is one of the most reliable methods for identifying lithologies in a borehole.

2.2.6. Neutron Porosity Log

The neutron logging equipment sends neutrons into the formation, where they lose energy and produce high-energy gamma rays. Because of the interaction with hydrogen atoms, the dispersed neutrons lose energy. The number of hydrogen atoms in a formation has the greatest impact on the neutron log. Its primary application is to determine the porosity of a formation. As a result, the energy of absorbing neutrons decreases as the number of hydrogen atoms increases. It indicates that in a porous rock, the count on the receiving end is low, and vice versa.

The neutron porosity log and the density log are frequently combined on suitable scales. The combination of the two logs is one of the most accurate indicators of the lithology of the subsurface that is currently known. This is because both the neutron log and the density log evaluate the porosity of a formation, and discrepancies between the two logs can be helpful in determining the nature of certain formations (Rider 1991).

2.3. Literature Review

In reservoir description and characterization, lithofacies classification is a critical and necessary task. In the exploration and development of hydrocarbon, reliable lithofacies recognition is helpful in increasing the precision and reducing the uncertainty of reservoir estimation (Xiong *et al*. 2010; Liu *et al*. 2017; Zhang *et al*. 2018). Facies is a sedimentary unit that can be identified from its surroundings by its petrophysical features. Facies that can be distinguished based on its mineralogy and grain size is called lithology facies or lithofacies. Using well logs to determine lithofacies or rock types is a crucial task for the field development plan. With the help of core measurements and seismic data, geologists integrate several types of well logs and analyze their physical qualities to derive petrophysical features of the subsurface (Hong *et al*. 2020). Outcrops, core data, and petrography are common methods for recognizing and identifying lithology, however outcrops may not appropriately represent what is in the subsurface, and core data are expensive to acquire.

Earliest work of lithofacies classification focused on using well log data together with multivariate statistical approaches. In the early 1980s and 1990s, lithofacies classification employed the technique of grouping and clustering features. Delfiner *et al*. 1987 and Busch *et al*. 1987 used a discriminant function analysis to identify lithofacies. Gill *et al*. (1993) used the correlation of zones between well and multivariate clustering to classify lithofacies.

Other ML algorithms like Support Vector Machine (SVM), Naïve Bayes (NB), Artificial Neural Network (ANN), fuzzy logic, Principal Component Analysis (PCA) became popular for solving geoscience problem in 1990. For example, Li and Anderson-Sprecher, 2006 used Naïve bayes. Zhang *et al*. (1999) and Dubois *et al*. (2007) used Artificial Neural Network (ANN), while Al-Anazi and Gates (2010), Sebtosheikh *et al*. (2015) and Hall (2016) used Support Vector Machine (SVM) in their research for lithofacies classification.

Qi and Carr (2006) and Wang and Carr (2012) both used ANN for the classification of lithofacies based on well logs, and they discovered that ANN produces a significant accuracy. According to the work of Graves (2012), neural network models struggle with sequential data because there is no record of previous entries in its internal structure, therefore most research that uses neural network combines it with a post processing step that considers the likelihood of succession such as probabilistic statistical analysis. Deep neural networks (DNNs) are one of the most popular new approaches to automatic lithofacies classification. One of the advantages of DNN is the ability to get high level features from input data because of the large number of layers (Santos *et al*. 2021).

Convolutional Neural Network (CNN) is a type of DNN that consists of multiple convolutional filter layers. Silva *et al*. 2015 and Lindberg *et al*. 2015 both identified lithofacies using CNNs. Tschannen *et al*. (2017) used an inception convolution network to predict lithofacies, their prediction was satisfactory in the first order but failed at replicating the high accuracy of a geologist interpretation. In a gas reservoir at Ordos basin, Lin *et al*. (2020) used a Long Short Term Memory (LSTM) network using Adam optimizer to analyze data from well logs. According to their findings, the LSTM network can correctly categorize rock lithofacies in thin interbedding layers and carbonate reservoirs that have thin interbedding lithofacies. Recently,

Xie *et al*. (2018) evaluated five common ML algorithms and discovered that selecting parameters and reservoir type have a significant effect on the accuracy of the ML method used.

Hall (2016) used support vector machine to solve the problem of geophysical lithofacies classification in the Panoma field, and he achieved an accuracy of 43%. Using same panoma field data, the Society of Exploration Geophysicists (SEG) held a machine learning competition in 2016 to forecast lithofacies (Hall 2016). Participants employed a variety of models, such as boosted tree models, neural networks, k-nearest neighbours (k-NN) and support vector machines (Hall and Hall 2017). Boosted tree method was the most accurate model, with an accuracy of 0.64 (Hall and Hall 2017). Imamverdiyev and Sukhostat, 2019 developed a 1D-CNN using three optimizers, and they compared the model to recurrent neural network (RNN), LSTM, SVM and k-NN algorithm. They showed that 1D-CNN (Adagrad) is better compared to other method with an accuracy of 76.87%.

Based on the well logs, Li and Zhang (2016) investigated the use of data-driven models to make predictions about the presence of sand, shale, and a mixture of the two. Several distinct data analytics algorithms, including logistic regression, gaussian discriminant analysis, random forest, and support vector machine, were put through their paces to identify the model that provided the most accurate predictions. Liu *et al*. (2020) improved the accuracy of lithology identification by developing a Multikernal Relevance Vector Machine using a set of inverted elastic attributes. There method preserves the benefits of traditional Support Vector Machine algorithms while optimising processes with Bayesian analyses. When compared to traditional methods, their results show advantages such as better generalisation and accuracy in identifying rock facies.

Using stratigraphic interpretation and well logs, Kim *et al*. (2018) developed random forest models to assist in the classification of seismic facies. The significance of each input feature in seismic facies classification is also determined by their model. This not only helps in the selection of important features, but it also reduces the amount of computational power that is required for the subsequent establishment of more complex models.

To assist in the differentiation of lithology types such as sand, silt, and clay, Lopez *et al*. (2020) developed several data-driven models including least-squares polynomial approximation, random forest, and support vector machine, amongst others. Their research demonstrates that data-driven models can differentiate between different types of lithologies by analysing the values of electrical resistivity and seismic wave velocity. The findings also indicate that the random forest model's forecasted lithology has a higher statistical correlation with the lithology that was found in their research.

Geologic domain knowledge was employed by Bestagini *et al*. (2017) to create a set of augmented characteristics for categorising well log data that were used by the top teams in the competition. They estimated gradients of each of the well log values and included non-linear characteristics. This demonstrates how the characteristics of the rock change as depth increases, which may imply the environment of deposition. With the help of this feature, the machine learning system can perceive the surroundings of a sample, providing context that it can utilise to distinguish between different rock kinds.

Chen and Zeng (2018) showed that the performance of the classifier may be enhanced by utilising petrophysical features computed from the base well logs. They adapted Archie's equation to include the log ratio of two measures, resistivity, and neutron porosity, which increased the precision of their predictions.

Another competition based on lithofacies classification was organized by FORCE with the objective of correctly predicting lithology patterns using well logs. The top 3 team achieved an accuracy of 80.1% and 79.9% using extreme gradient boost, and 79.9% using Random Forest respectively. Each of these models appears to excel in some areas, thus combining them to create an ensemble model with a high predictive ability seems like the logical next step for this study.

Table 2.1: Summary of related works on the use of machine and deep learning on lithology classification

S/N	Authors	Summary of research
1	Delfiner *et al*. (1987).	Used Bayesian rule to assign depth level to a lithofacies using a database.
2	Busch *et al*. (1987).	They compared different model by employing the statistical method of discriminant analysis and chose model the best model that predict lithology with 75 % accuracy.
3	Qi and Carr (2006).	They classified lithofacies with ANN, using well logs and they reported absolute accuracies range from 70.37 to 90.82 percent.
4	Wang and Carr (2012).	Their study discovered that the neural network performs better for shale lithofacies prediction than the discriminant analysis.
5	Gill *et al*. (1993).	They successfully Used hierarchical clustering algorithm for zoning and recognizing log facies, and this means that log facies might be helpful for interwell correlations and locating reversal faults in boreholes.
6	Silva *et al*. (2015).	This study focused on using a back-propagation neural network algorithm for petrographic classification using well logs, and their result shows the effectiveness of ANN with an accuracy of 85.62 percent.
7	Lindberg *et al*. (2015).	The authors used convolutional hidden Markov model for facies classification, and they reported that the combination of neutron porosity log, resistivity and gamma ray log as a subset yielded the best prediction.

8	Tschannen *et al*. (2017).	They trained an inception network, to extract reservoir stratigraphy from well data. The model was suitable in the first instance, but it was unable to precisely repeat the geologists' work since it lacked higher resolution core sample data and additional well readings.
9	Lin *et al*. (2020).	On a tight gas sandstone reservoir, the LSTM-based Adam optimizer can more accurately identify rock facies than the Sgdm and Rmsprop optimizers can, provided that the sample size and number of hidden layer neurons are appropriately set.
10	Hall (2016).	With the use of a support vector machine, the author was able to solve the geophysical lithofacies classification issue in the Panoma field with an accuracy of 43%.
11	Hall and Hall (2017).	The Society of Exploration Geophysicists (SEG) held a machine learning competition in 2016 to forecast lithofacies. It was reported that beyond using the most suitable model, the use of domain knowledge to create new features improved the classification model's accuracy.
12	Imamverdiyev and Sukhostat (2019).	The authors used different algorithm to classify lithofacies and their result shows that 1D-CNN model is more accurate compared to SVM, k-NN, RNN, and LSTM.
13	Li and Anderson-Sprecher (2006).	Core and well log data was used to compare the classification of naïve bayes to a linear discriminant analysis, and their result indicates that both methods performed satisfactorily.
14	Zhang *et al*. (1999).	For post-stack seismic lithology prediction, they combined DNN and CNNs with continuous wavelet transforms (CWTs). In terms of prediction

		accuracy, CWT-CNN models outperform DNN, CNN, and CWT-DNN models.
15	Dubois *et al*. (2007).	The authors compared four models for facies classification and discovered that to match the performance of neural networks, the non-parametric approaches such as k-nearest neighbour and fuzzy logic would need to be significantly improved.
16	Al-Anazi and Gates (2010).	The lithology and permeability predictions made by SVM are contrasted with those made by conventional regression and back-propagation neural networks. According to statistical error analysis, the SVM method outperforms neural network methods in terms of identification of the lithology and permeability estimates.
17	Sebtosheikh *et al*. (2015).	This study used SVM in predicting lithology, and their results indicate that SVM is a practical method for lithology prediction, and that the radial basis function kernel, which produces the lowest misclassification rate error, is more accurate than other kernel functions.
18	**Merembayev et al. (2021)**	The authors employed wavelet transformation in machine learning to classify lithologies of well logs from Norway and Kazakhstan, and they reported an accuracy of 94.8% and 98% respectively.
19	Masapanta (2021)	This study investigated the use of different machine learning model such as tree based gradient boosting and neural network to classify lithologies in a Norway field, and he achieved an accuracy of 82.5% on the test data.

20	Force (2020)	A machine learning competition for facies classification was held in 2020 using the Norway data provided by FORCE, and the top 3 model achieved an accuracy of 80%, 78% and 79& on the blind dataset.

CHAPTER THREE

METHODOLOGY

3.1. Dataset

This research will make use of the datasets that were collected during the "Machine-Learning Lithology Prediction Contest" that took place in 2020 and was organised by FORCE, a cooperating forum that is run by Norwegian oil and gas companies and authorities. The data can be downloaded from https://drive.google.com/drive/folders/0B7brcfeGK8CRUhfRW9rSG91bW8?resourcekey=0-NsLk7JL-IDDxUKPVp0dZrw. The dataset is made up of 118 wells drilled off the coast of Norway, with locations spanning the Viking Graben to the south and north. Geophysical well logs are single-point observations of rock physical characteristics recorded in a well at a certain depth. Three distinct data subsets, each having a different function, are also included in the data. There are 98, 10, and 10 wells in the training, open test, and hidden test subsets, respectively.

Well logs, interpreted lithofacies, and lithostratigraphy are all included in the dataset. The well logs contain the well's name, the measured depth, the wireline measurement locations, as well as the well logs CALI, RDEP, RHOB, DHRO, SGR, GR, RMED, RMIC, NPHI, PEF, RSHA, DTC, SP, BS, ROP, DTS, DCAL, and MUDWEIGHT. An explanation of the abbreviations is shown in the table 3.1.

Table 2.1: Description of the dataset features.

S/N	FEATURES	DESCRIPTION
1	FORCE_2020_LITHOFACIES_CONFIDENCE	Qualitative measurement of interpretation confidence
2	GR	Gamma Ray Log
3	FORCE_2020_LITHOFACIES_LITHOLOGY	Interpreted lithofacies
4	RSHA	Shallow Reading Restitivity measurement

5	RMED	Medium Deep Reading Restitivity measurement
6	RDEP	Deep Reading Restitivity measurement
7	RMIC	Micro Resisitivity measurement
8	SP	Self-Potential Log
9	RXO	Flushed Zone Resistivity measurement
10	DTS	Shear wave sonic log (us/ft)
11	ROPA	Average Rate of Penetration
12	DTC	Compressional waves sonic log (us(ft))
13	NPHI	Neutron Porosity log
14	PEF	Photoelectric Absorption Factor log
15	RHOB	Bulk Density Log
16	DTC	Compressional waves sonic log (us(ft))
17	DRHO	Density Correction log
18	SGR	Spectra Gamma Ray log
19	BS	Borehole size
20	DCAL	Differential Caliper log
21	MUDWEIGHT	Weight of Drilling Mud
22	ROP	Rate of Penetration
23	CALI	Caliper log
24	DEPTH_MD	Measured Depth

25	X_loc	X location of sample
26	Y_loc	Y location of sample
27	X_loc	Z(TVDSS) location of sample

3.2. Exploratory Data Analysis (EDA)

Exploratory data analysis is a process that involves investigating data to discover relevant information and preexisting patterns. Understanding the nature of the data and developing early potential strategies or methodologies for addressing the challenges associated with lithofacies classification is the primary purpose of EDA in this study. EDA makes use of data visualisation, which includes plotting the well log data, creating cross plots and correlation plots to better understand the dataset, and many other similar activities.

3.3. Data Preparation

Data preparation and cleaning is an essential part of machine learning, and it's also one of the first steps. The Pandas library comes with a variety of utilities that can be used for data manipulation, cleaning, and visualisation (Matplotlib). When cleaning data, certain steps must be taken, such as removing outliers, dropping variables that are not required for the training of the model, dropping null or missing values, and replacing values that are missing. The purpose of outlier detection and removal is to eliminate anomalies that resulted from measurement errors. When attempting to predict rock lithofacies using well log data, it is assumed that each type of rock lithofacies will produce its own unique set of log readings. All the well logs need to be consistent and balanced throughout the lithological formation that is being analysed to make an accurate prediction of the lithofacies.

3.4. Data Processing

Initial pre-processing of the original raw logs is required to remove any depth-related anomalies, any inaccuracy from borehole logging, and the presence of hydrocarbons or other fluids. Feature scaling is used to normalise data variables or features such that they all fall into the same range, such as -1 to 1 (Youn and Jeong 2009). In most cases, the range of raw data values fluctuates greatly, which means that without normalisation, objective functions in machine learning algorithms may not perform well.

3.5. Feature engineering

This process will involve the use of domain knowledge to choose the most suitable features for prediction, transforming or creating new features from raw data with the aim of increasing the accuracy of the prediction.

3.6. Data Augmentation

This is a technique that can be used to artificially expand the size of a training set by creating modified data from the existing one. This is accomplished using a technique known as Data Transformation. If you want to avoid overfitting your model, or if the initial dataset is too small to train on, or even if you just want to get better performance out of your model, it is a good method to use.

3.7. Classification Algorithms

The data contains numerous discrete labels (or integer values) therefore, this might be termed a multilabel classification problem. Several machine learning methods developed during the previous decade might be utilised to tackle this challenge. Several algorithms were tested during the workflow's development. Random forest, decision tree classifier, k-nearest neighbour (KNN), SVM, and linear models were among the techniques investigated.

3.7.1. Decision tree

A non-parametric supervised learning method applied to regression and classification problems. It is structured in a hierarchical manner, and its components include a root node, branches, internal nodes, and leaf nodes.

3.7.2. Random Forest

A random forest (RF) approach is used to classify an instance label based on the label (i.e., facies label) predicted most frequently by an ensemble of decision trees (Breiman 2001; James *et al*. 2013). The term "random forest" (RF) refers to a method of machine learning that, as its name suggests, includes elements of randomness within its fundamental operations. Each tree is generated and fitted using a bagging technique that is based on the random selection (with replacement) of samples from the training set. This improves the stability and accuracy of the machine learning classification (Breiman 2001; James *et al*. 2013). The random forest model helps reduce bias in the model and improves the performance of the model by taking into consideration predictions made by many independently created decision trees that were generated at random and trained using groups of data points that were chosen at random.

3.7.3. K–Nearest Neighbors (KNN)

KNN is an easy supervised learning method. During training, a KNN model stores the coordinates and values of every data point it encounters. Using the input data, the model determines the k-nearest points to make its predictions. The majority class of the nearest points is the prediction in a classification problem, while the average of the nearest points is the prediction in a regression problem. While this method appears simple, it has proven to be quite effective when dealing with large amounts of data. Changing a model's hyperparameter can result in a noticeable shift in performance (Mohamed *et al*. 2019).

3.7.4. Support Vector Machine (SVM)

Support Vector Machines (SVMs) are one of the most widely used machine learning methods. SVMs find the line that maximises the separation between the points of each class in a 2-dimensional, 2-class classification problem. The margin is the

distance between the line and the nearest point classified one way or the other. Many lines can be used to separate the points, but the goal is to find the line with the greatest margin. Support vectors are the points closest to the separating line (Muhammad *et al*. 2019). A suitable solution to the categorization problem in practise may include the use of multiple planes. We can see that a few data points are isolated in one region because a collection of planes divides the space of all points into parts (Figure 3.1). This is yet another method for identifying outliers.

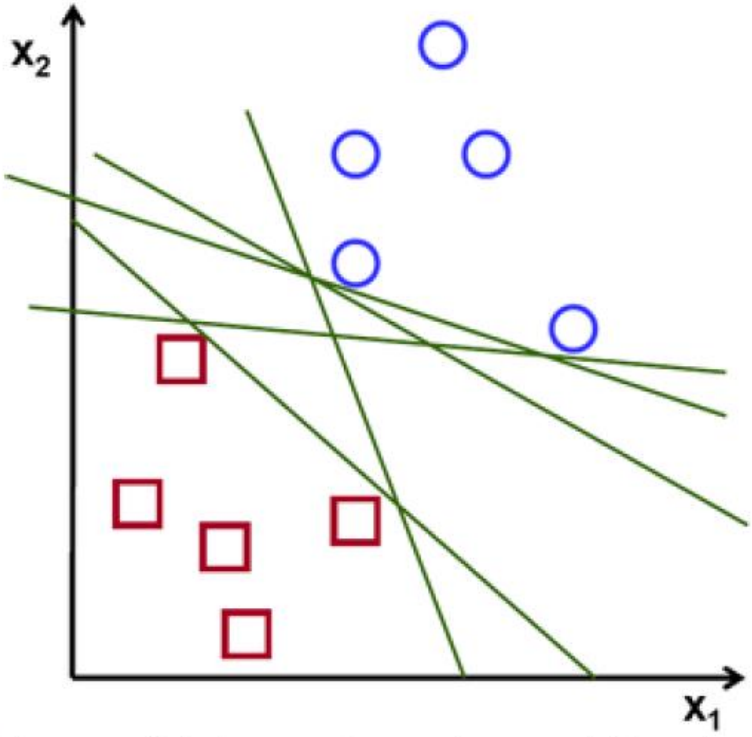


Figure 3.1. Illustration of the various hyperplane that could be selected to divide two groups of data points into distinct categories (Gandhi, 2018).

3.7.5. Categorical boosting (CatBoost).

The recently developed method of machine learning known as Categorical Boosting, also known by its abbreviation CatBoost, is built from the terms Category and Boosting. It handles categorical characteristics or predictors without the need for separate encoding of categorical data, which is commonly needed by other machine learning approaches as part of the pre-processing stage. This is where the term "Cat" comes from, as it refers to the fact that it handles categorical characteristics or

predictors. The term "Boost" is a reference to its functionality, which is based on the gradient boosting technique that was covered in the section 3.7.8 below (Ghori *et al*. 2019)

3.7.6. Light Gradient Boosting

The Light Gradient Boosting model (LGBM) is an extremely effective gradient boosting algorithm, which excludes a significant portion of the data instances that have minor gradients when calculating the amount of information gained. This algorithm is like the conventional Gradient Boosting Decision Trees (GBDT) machine-learning models with almost the same efficiency as, but it is significantly quicker throughout the training phase (Ke *et al*. 2017).

3.7.7. Extreme Gradient Boosting

As an ensemble machine learning technique, gradient boosting can be applied to problems of predictive modelling in both classification and regression. The ensembles are constructed with the help of decision tree models. Individual trees are then added to the ensemble and adjusted to compensate for the errors in prediction made by earlier models. This is an illustration of a machine learning model that uses a boosting ensemble. When compared to other Gradient Boosting implementations, extreme gradient boost stands out because it uses a more regularised model formalisation to control over-fitting, leading to improved performance and a reduction in overfitting. according to (Chen and Guestrin 2016).

3.7.8. Gradient Boosting (GB).

For both regression and classification tasks, the machine learning technique known as "gradient boosting" can generate an ensemble of low-quality prediction models. This method builds a model incrementally and makes it more generic by allowing optimization of any loss function that can be differentiated. To put it simply, gradient boosting is an iterative process that combines multiple weak learners into a single robust one. Each additional poor learner prompts a re-fit of the model to generate a more precise prediction of the response variable. As a group, the new weak learners correlate most strongly with the negative gradient of the loss

function. Gradient boosting's goal is to strengthen prediction by combining several weak prediction models.

3.7.9. Logistic regression.

Despite its name, logistic regression is a classification model rather than a regression model. For binary and linear classification problems, logistic regression is a simpler and more efficient method. It is a classification model that is simple to implement and delivers excellent results with linearly separable classes (Subasi 2021). It is a widely used classification method in industry and it uses the same theory as linear regression; however, it is a probabilistic approach used for solving binary or multiclass issues by employing a logistic function (Masapanta 2021).

3.8. Model Evaluation

It is essential to make use of assessment metrics to determine whether a model was successful after it was developed. In a problem involving pattern recognition or classification, each individual item in the overall population has both a real label and a predicted label associated with it. Since both the actual label and the predicted label have the potential to be either positive or negative, there are a total of four possible outcomes that can take place because of the actual label and the predicted label. To begin, what is known as a true positive prediction is when both the expected labels and the actual labels have a positive value (TP). Second, a true negative prediction is what we call the situation when both the anticipated labels and the actual labels are negative (TN).

Third, a false positive prediction (FP) takes place when the actual label is negative, but the anticipated label is positive. The fourth type of prediction is known as a false negative (FP), which takes place when the expected label is negative, but the actual label is positive (Zhou 2020). True positive and true negative predictions are valid because the anticipated values correspond to the actual values (Figure 3.2). On the other hand, predictions that are false positive or false negative could be false predictions because the values that were anticipated for them do not match the values that occur.

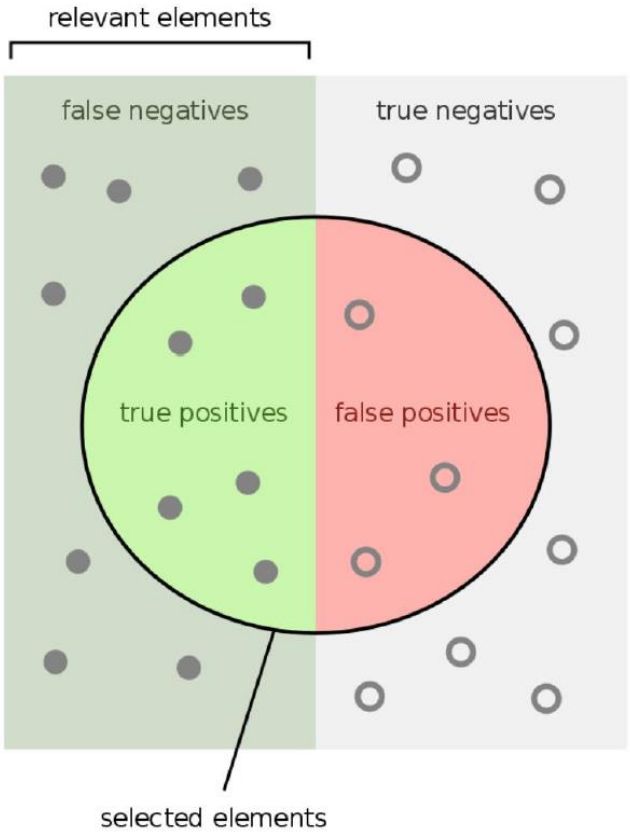


Figure 3. 2. Illustration of predicted values and actual values (Chris, 2019).

The percentage of correct predictions made across all the samples determines a measurement known as accuracy. Figure 3.3 provides a visual representation of some common metrics, including precision, recall, and specificity. The term "precision," which is also known as "positive predictive value," refers to the percentage of accurate positive predictions made across all the selected elements (positive predictions). The proportion of accurate predictions across all relevant factors is referred to as recall, which is also known as sensitivity or the true positive rate. The genuine negative rate, which is also referred to as specificity, indicates the percentage of non-relevant factors that result in accurate negative predictions.

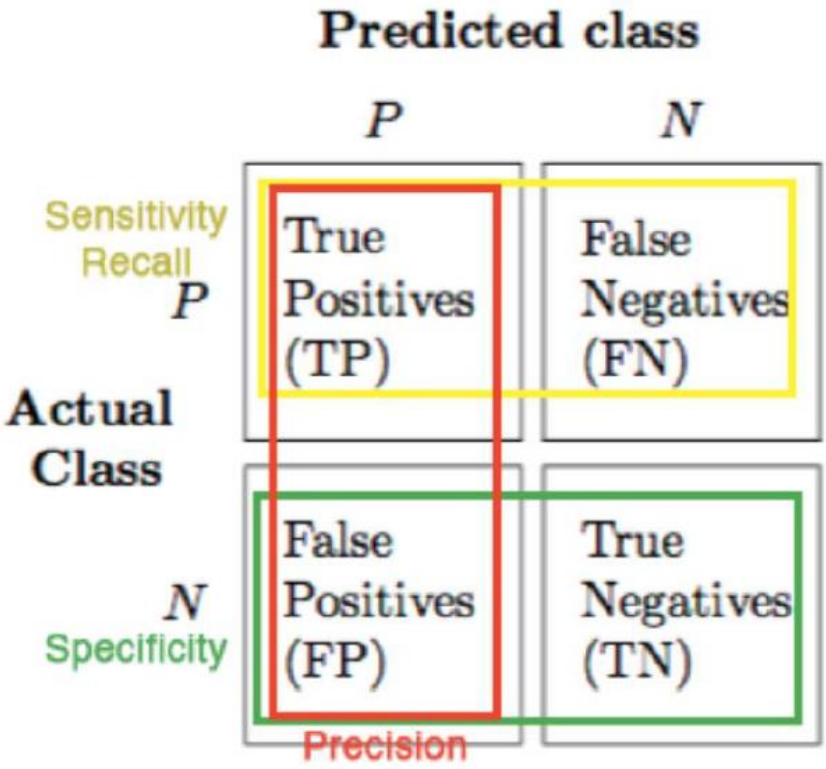


Figure 3. 3. Illustration of recall, precision, and specificity (Tran, 2016).

In addition, the F1 score is a measurement that takes into consideration both the accuracy and the recall (Zhou 2020). It is the harmonic average of the accuracy and recall scores. The equations for calculating the parameters are shown down below.

Accuracy= $(TP+TN)/(TP+TN+FP+FN)$.....3.1

Precision= $TP/(TP+FP)$.....3.2

Recall= $TP/(TP+FN)$.....3.3

F1=2 $(Precision - Recall)/(Precision+Recall)$.....3.4

CHAPTER FOUR

RESULTS

4.1. Exploratory Data Analysis

4.1.1. Descriptive statistics

Exploration of the data revealed that it contains several glaring inaccuracies; for instance, the maximum value of GR is 1076, which is significantly higher than what is typical for any lithology. For sandstone, shale, and carbonate, the value of gamma ray can range anywhere from 0 to 150, except for organic rich shales and lithology that contains radioactive minerals like K-feldspar, mica, and zircon. This information comes from geology. The descriptive statistics presented in table 4.1, as well as the histogram and boxplot presented in figure 4.1, make it abundantly clear that the GR data are skewed, which indicates that they include outliers.

According to Table 4.1, the median value of RMED was 1.44, while the mean was 4.98 due to the influence of extreme values. The maximum value of RMED was 1988, and the lowest value was -0.00842. Boxplots and the histogram can both be used to identify the data set's extreme values, also known as outliers.

Table 3.1: Descriptive statistics of the raw data.

	DEPTH_MD	RSHA	RMED	RDEP	RHOB	GR	ROP	DTS	DRHO
count	1170511	630650	1131518	1159496	1009242	1170511	535071	174613	987857
mean	2184.087	10.69466	4.986978	10.69103	2.284987	70.9137	137.368	204.655	0.012196
std	997.1821	100.6426	54.67269	113.948	0.253284	34.23149	1539.384	71.06846	7.477798
min	136.086	0.0001	-0.00842	0.031701	0.720971	0.109284	-0.11798	69.16318	-7429.34
0.25	1418.597	0.85412	0.914086	0.91024	2.092203	47.62722	5.628	155.9367	-0.00925
0.5	2076.605	1.39902	1.443584	1.439	2.321228	68.36763	17.8	188.2007	0.001752
0.75	2864.393	3.099348	2.68093	2.55722	2.48858	89.03551	34.81279	224.6451	0.021702
max	5436.632	2193.905	1988.616	1999.887	3.45782	1076.964	47015.13	676.5781	2.836938

The outliers in Gamma ray were removed and replaced with a median value, which makes the Gamma ray log to become a normal distribution as shown in figure 4.2

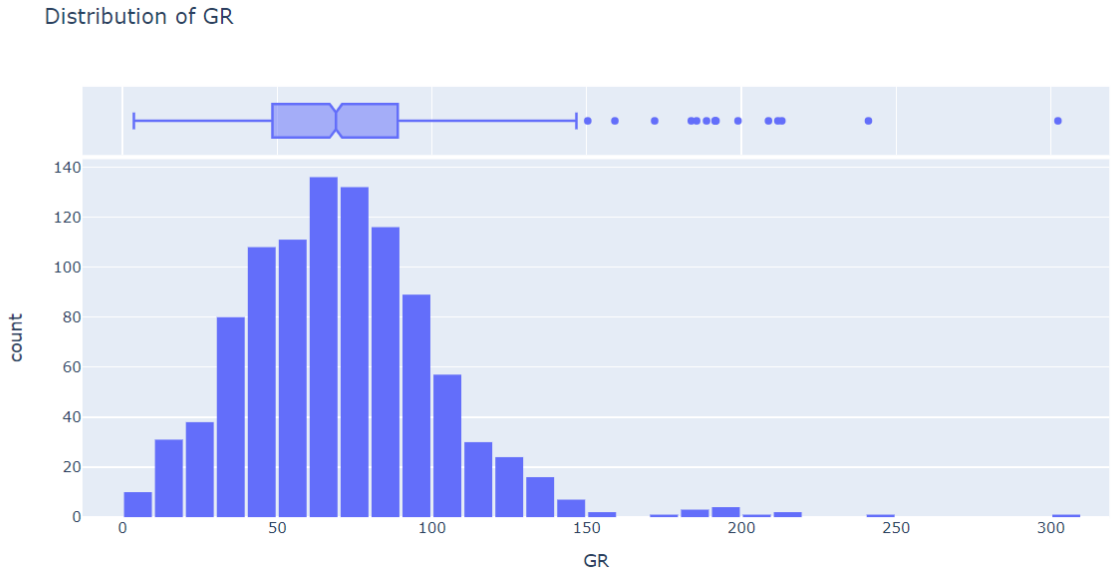


Figure 4.1 Histogram distribution and boxplot of Gamma ray log before outlier removal

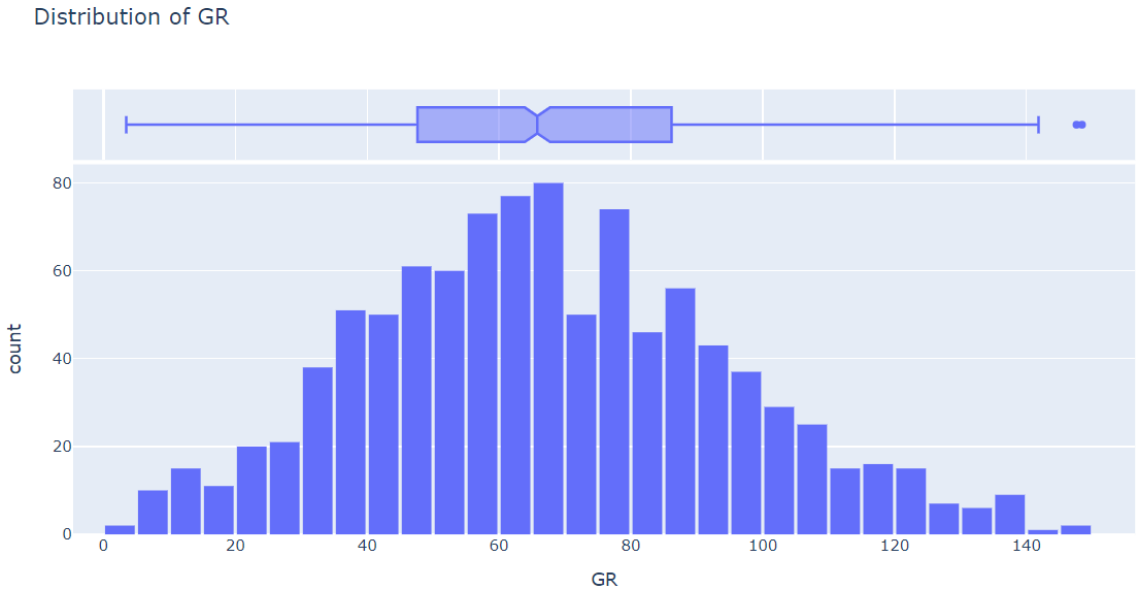


Figure 4.2 Histogram distribution and boxplot of Gamma ray log after outlier removal

4.1.2 Pearson correlation

It is imperative to eliminate the features with high correlation to minimise or reduce the amount of overfitting that occurs during the training stage. This assists in comprehending the internal structure of the data, which in turn helps reduce the dimensionality. Calculating the degree of correlation that exists between each

variable in the dataset was accomplished with the Pearson Correlations, and the results are depicted in figure 4.3.

Pearson correlation has a value between +1 and -1, where +1 implies positive correlation and -1 denotes negative correlation (Table 4.2). As a result, a stronger relationship between variables is indicated when the absolute value of the correlation coefficient gets closer to one.

Table 4.2 Interpretation of correlation coefficient values

Correlation coefficient value	Interpretation
±1	Perfect positive/negative relationship
±0.8	Strong positive/negative relationship
±0.6	Moderate strong positive/negative relationship
0	No relationship

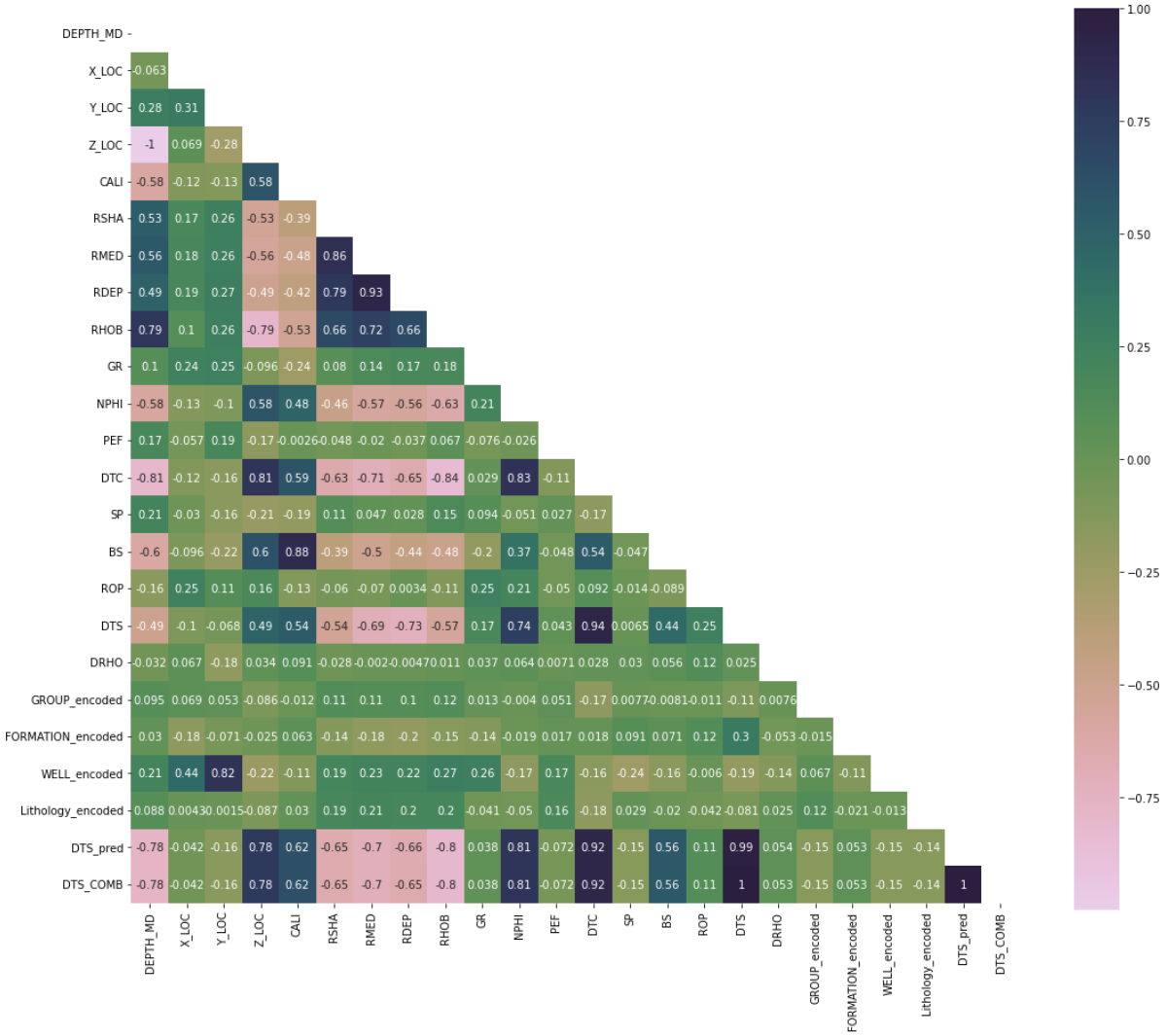


Figure 4.3 Pearson correlation of the dataset feature

There is a strong Pearson correlation of 0.93 between RDEP and RMED log, which is expected because both are resistivity measurement. Likewise, there is a correlation of 0.94 between DTC and DTS log, which is also expected as both are sonic log measurement. Strong correlations between variables may lead to performance issues for some machine learning models due to the increase in dimensionality that results from keeping correlated variables. This is because keeping correlated variables results in an increase in the number of dimensions. Additionally, they might make the model less scalable and lengthen the amount of time it takes to run.

It was also observed that there is a strong correlation of 0.83 between NPHI and DTC logs. This was to be expected because it is known that compressional waves are dependent on the amount of solid minerals present in the rock. This means that the

less mineral there is in a rock, the higher the porosity, and consequently, the higher the compressional sonic log.

On the other hand, there is a negative Pearson correlation of 0.84 between RHOB and DTC, which is also to be expected if we consider the rock compaction and fluid saturation. This is because the higher the compaction, the higher the bulk density and the compressional wave velocity, and the lower the compressional slowness.

4.1.3. Density-Neutron cross plot.

For more complex lithology formations, a combination of density and neutron logs can be an invaluable source of porosity information. More accurate estimates of porosity can be obtained using the combination than using either tool alone, due to the ability to make determine lithology and fluid content. Pure lithologies like sandstone, limestone, or dolomite that is filled with oil or water, can be identified with the help of the density-neutron cross plot. The cross plot showed that most of the lithology are overlapping; only the basement, anhydrite, and halite stood out due to their low porosity (Figure 4.4).

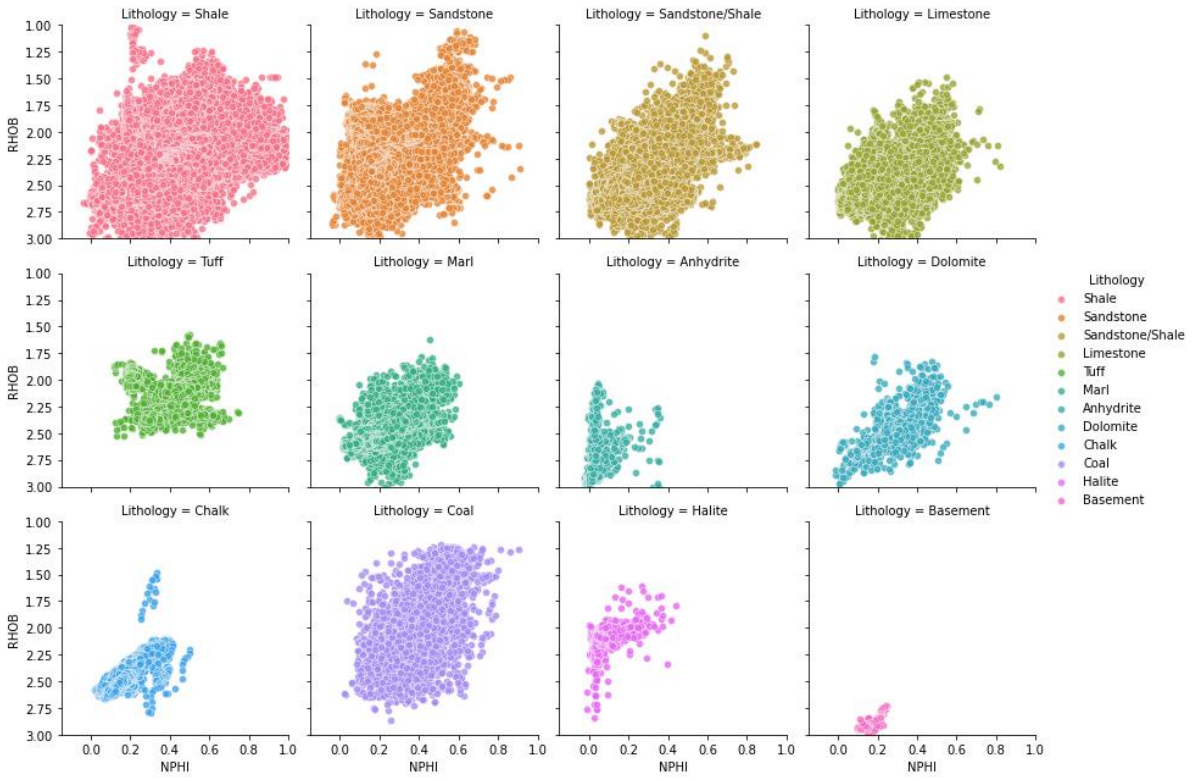


Figure 4.4. Density-neutron cross plot coloured by lithology

The cross plot shown below reveals that the Rotliegendes Group is primarily composed of sandstone, whereas the Nordaland Group is a mixture of shale and sandstone/shale lithologies (Figure 4.5).

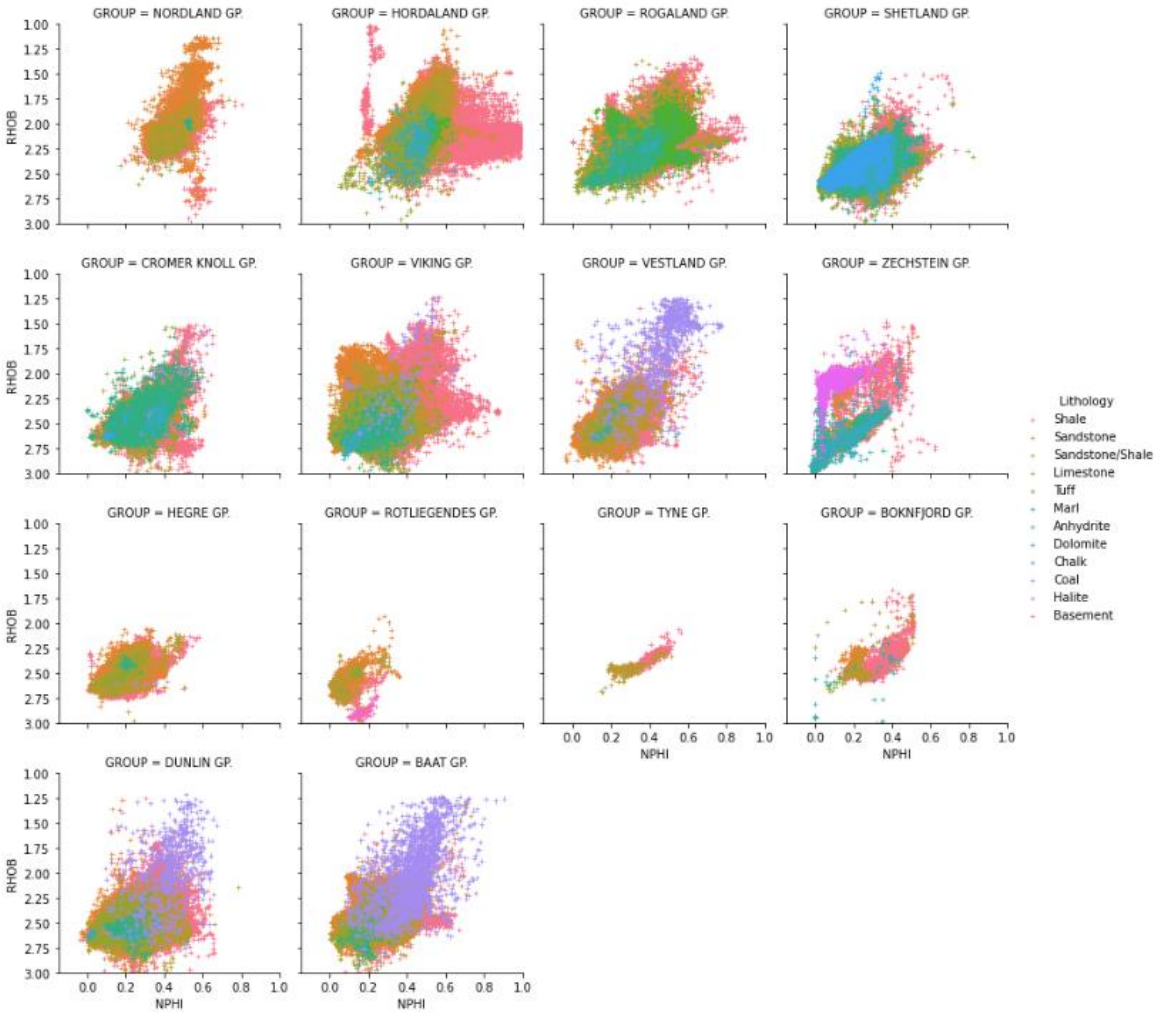


Figure 4.5: Density-neutron cross plot per group coloured by lithology

4.1.4. Exploring the features

There is a total of 12 lithofacies represented in the data, with shale, sandy shale, and sandstone accounting for more than 70% of the total data. The percentage of each lithology's appearance in the training, open test, and hidden test subsets is detailed in Table 4.3 below.

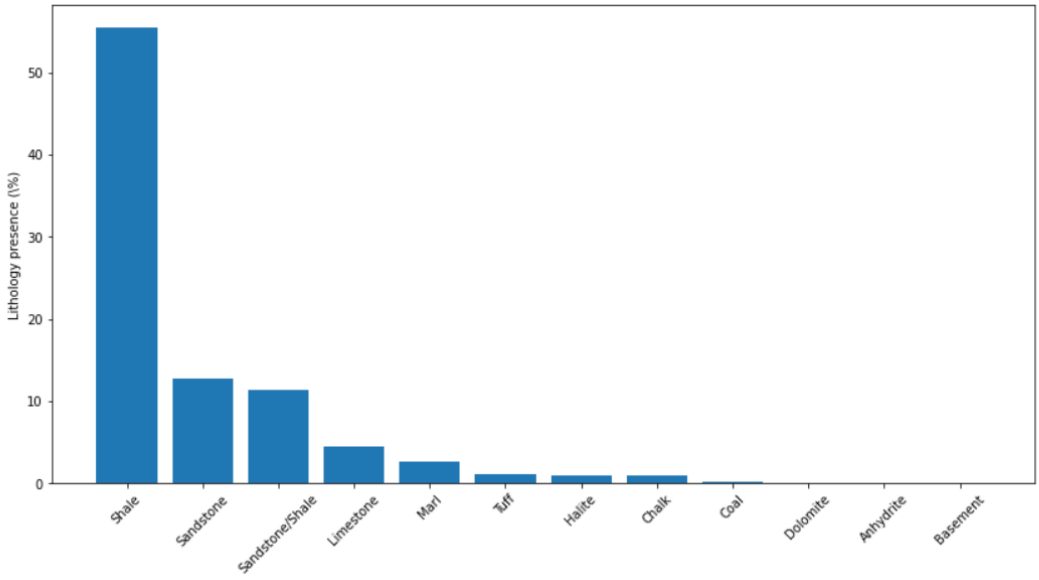


Figure 4.6: Bar plot showing the lithology presence in the dataset

Figure 4.6 also provides a more visual representation of the lithology distributions across the various data subsets. The different lithologies may be extremely important to categorise, but there is a significant class imbalance between them. Furthermore, it is critical to remember that many lithology types could be categorised as mineral mixtures. Similar readings for petrophysical properties, as depicted in figure 4.4, are expected to exist between lithology classes that are thought to be related.

Table 4.3 Lithofacies presence percentages summary.

Lithology	Code	Training	Open	Hidden
Sandstone	0	14.4	17.6	11.5
Shaly	1	12.9	12.8	10
Shale	2	61.6	61.4	58.7
Marl	3	2.8	2.4	3.6
Dolomite	4	0.1	0.3	0.2
Limestone	5	4.8	3.5	3.8
Chalk	6	0.9	0.5	2.4
Halite	7	0.7	-	5.3
Anhydrite	8	0.1	0.1	0.5
Tuff	9	1.3	0.9	0.8

Coal	10	0.3	0.5	0.2
Basement	11	0.01	-	

As can be seen in Figure 4.7, the geology of the North Sea is predominately characterised by shale, sand, and carbonates, the majority of which were deposited during the Jurassic, Cretaceous, and Cenozoic time periods respectively. It is not a surprise that shale and sandy shale lithology predominate in the North Sea given the history of its geological development, which was first marked by a significant marine incursion that stretched the entire length of the North Sea during the transition from the Triassic to the Jurassic periods of time. Huge deltaic systems consisting of sand, shale, and coal began to emerge in the late Jurassic period in the northern North Sea and on the Horda Platform after the maritime incursion had come to an end. The most significant Jurassic rifting episode took place in the North Sea region during the Late Jurassic and continued into the Early Cretaceous. This event occurred at the end of the Jurassic period. Throughout the course of this tectonic episode, significant block faulting caused uplift and tilting, which led to the formation of significant local topography, complete with erosion and sediment supply. Both the most important source rock and the Draupne Formation, which is an important seal for hydrocarbon traps in the North Sea region, were produced by the accumulation of extensive sequences of shale in anoxic basins. The Draupne Formation serves as a crucial seal for hydrocarbon traps in the North Sea region. (NPD, 2015).

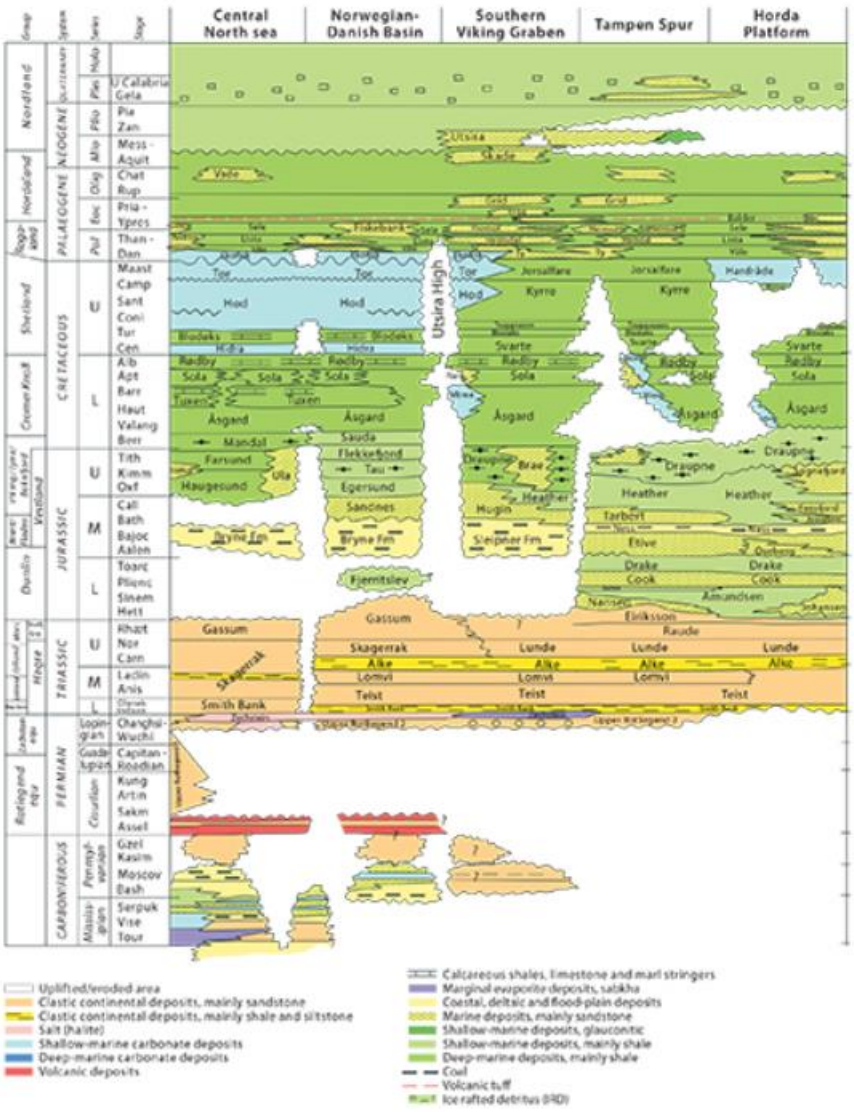


Figure 4.7: Geology and stratigraphic section of the North Sea (NPD 2015).

The datasets include a wide variety of features that can be used as input data. These data elements include twenty types of well log readings in addition to six additional metadata characteristics that includes well names, interpretation confidence, location, and lithostratigraphic data. However, like many situations that occur in the real world, the dataset has missing values for some of its features. These missing values could be the result of a wide variety of factors, including economic concerns, borehole problems, logging instrument failure, telemetry issues, or simply being excluded on purpose. The SGR, DCAL, ROPA, RMIC, MUDWEIGHT, and DTS logs

account for most of the missing data in the training set, which are present in only 13, 22, 25, 27, 28, and 32 wells, respectively as shown in figure 4.8.

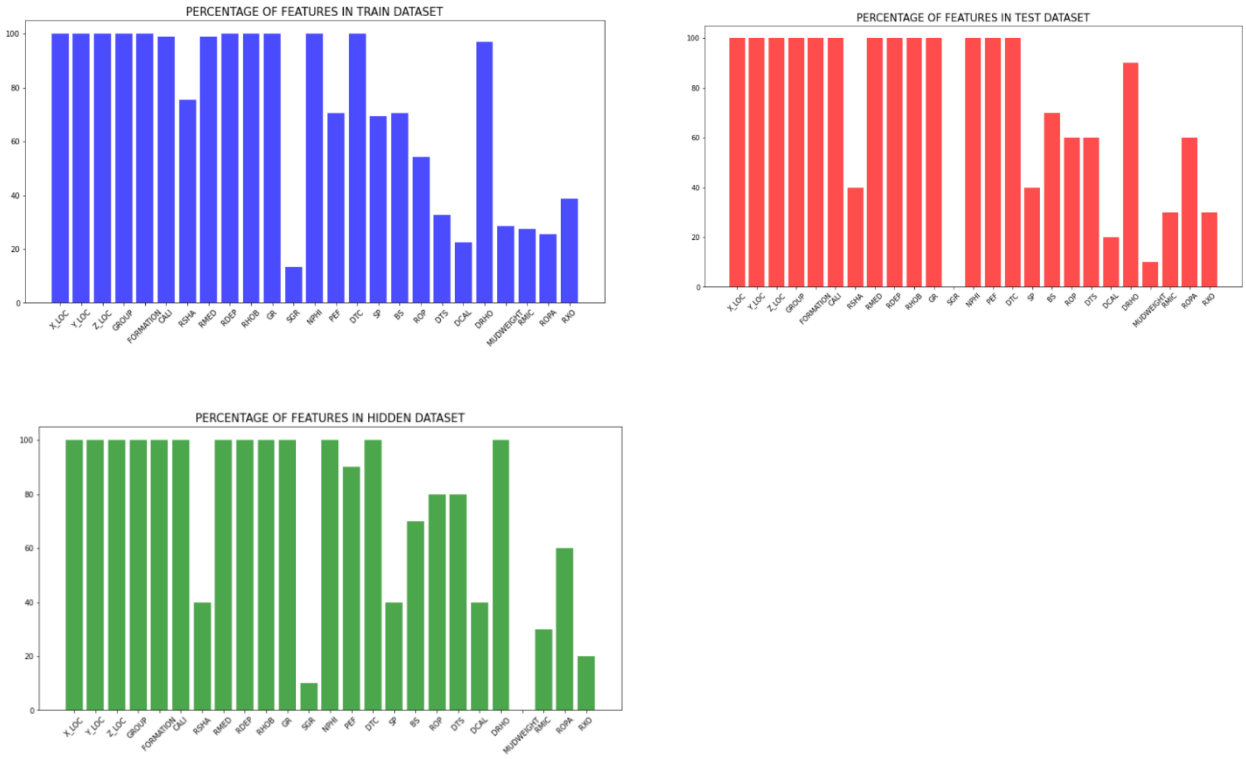


Figure 4.8: Percentage distribution of features in train, test and hidden dataset.

In addition, the overlapping of the mixed-based lithofacies, such as sandstones and shaly-sandstones, could suggest that certain measurements in the dataset corresponding to these classes were misread or mislabelled. Alternatively, it could be an inherent characteristic of the formations due to the presence of radioactive minerals such as k-feldspar, zircon, or mica. In addition, this could be because the boundaries between facies are continuous rather than discrete, resulting in the feature space containing measurements from various lithology classes being superimposed.

4.2. Data preparation

The first and most critical stage in developing a predictive model is data preparation (Akinnikawe 2018). This entails data cleaning, outlier removal, and data quality control.

4.2.1. Data cleaning and outlier removal

Data cleaning is a phase in data processing that involve removing 'NAN' data values (Fayyad 1996). "NAN" stands for "Not a number," and it is a particular case in both floating-point representations of real numbers and floating-point operations. Even while certain machine learning algorithms can deal with "NAN" data, data cleaning might potentially mitigate the negative impact of polluted "NAN" data.

From observations GR, WELL, and FORCE_2020_LITHOFACIES_LITHOLOGY which is the target variable has no missing value while, ROPA, RMIC, DTS, DCAL, SGR, RXO AND MUDWEIGHT have more than 70% of missing values (Figure 4.9).

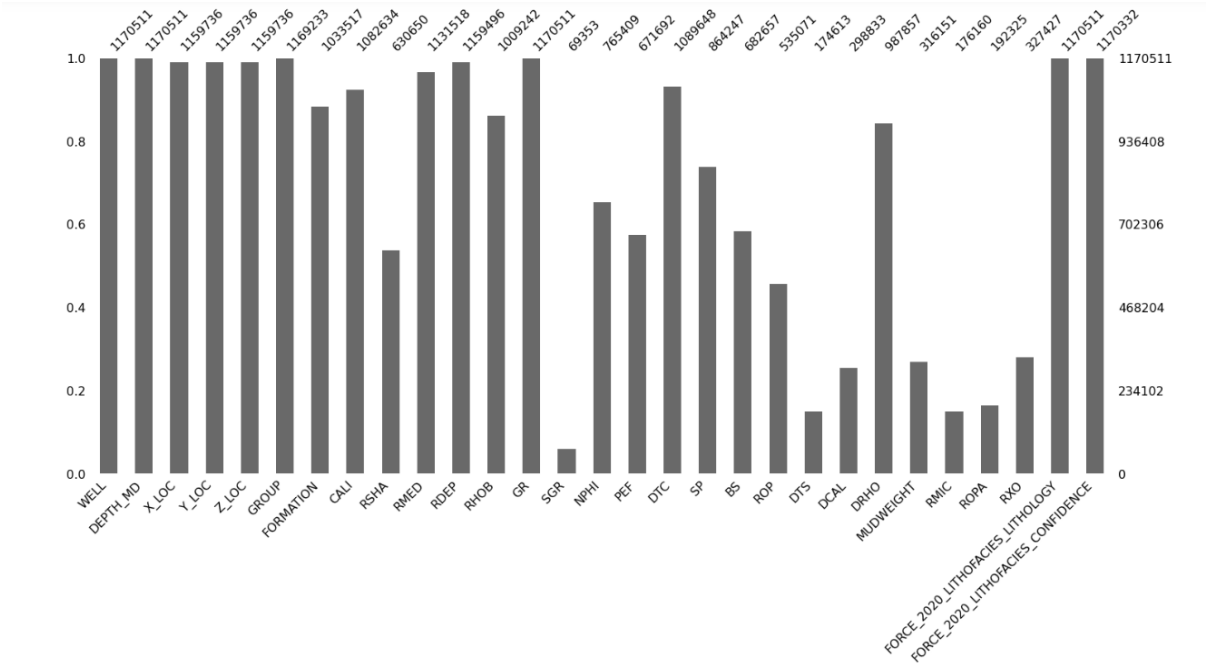


Figure 4.9: Bar plot showing the percentage of missing values in each feature of the dataset

In respect to well, there are 4 wells with more than 60% of missing data (31/6-5 ,33/9-1,35/11-12,31/5-4 S). Well 31/5-4 S has 81% of missing data, and 16 other wells have more than 50% missing values. The average missing values per well in the

dataset is 41%. All these missing values can either be replaced or dropped before modelling, but for this study they were replaced.

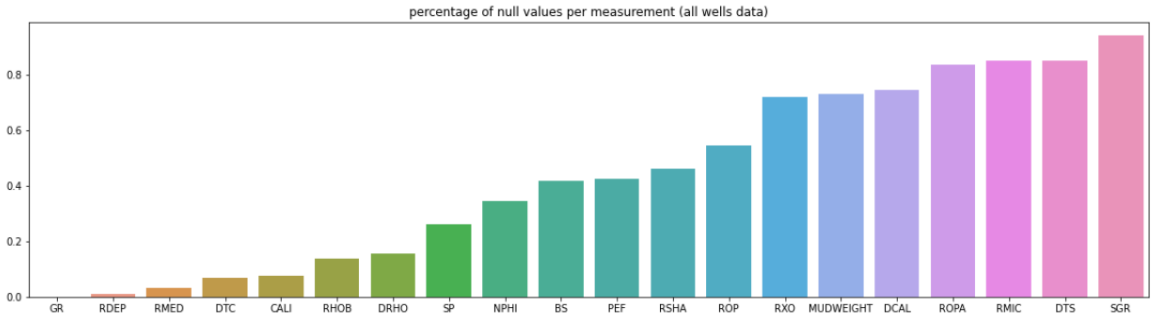


Figure 4.10: Bar plot showing the percentage of null values per measurement.

Outliers are unusual points in a dataset. They are points that do not fit into the dataset's normal or anticipated statistical distribution and can arise for a variety of causes, including sensor and measurement mistakes, inadequate data sampling, and unexpected occurrences.

Outliers may be identified in a dataset using a variety of strategies, some of which utilise visual techniques such as scatterplots (e.g. cross plots) and boxplots, while others rely on univariate statistical methods (e.g. Z-score) or even unsupervised machine learning algorithms. For this study, four methods such as isolation forest, standard deviation outlier, local outlier factor and one class support vector machines was tested on the train dataset. Comparing the four method of outlier removal with the data before outlier was removed using box plot revealed that standard deviation filter is the better method out of the four (Figure 4.11). The local outlier factor removed 30% of the dataset which is the highest percentage among the four methods used (Table 4.4).

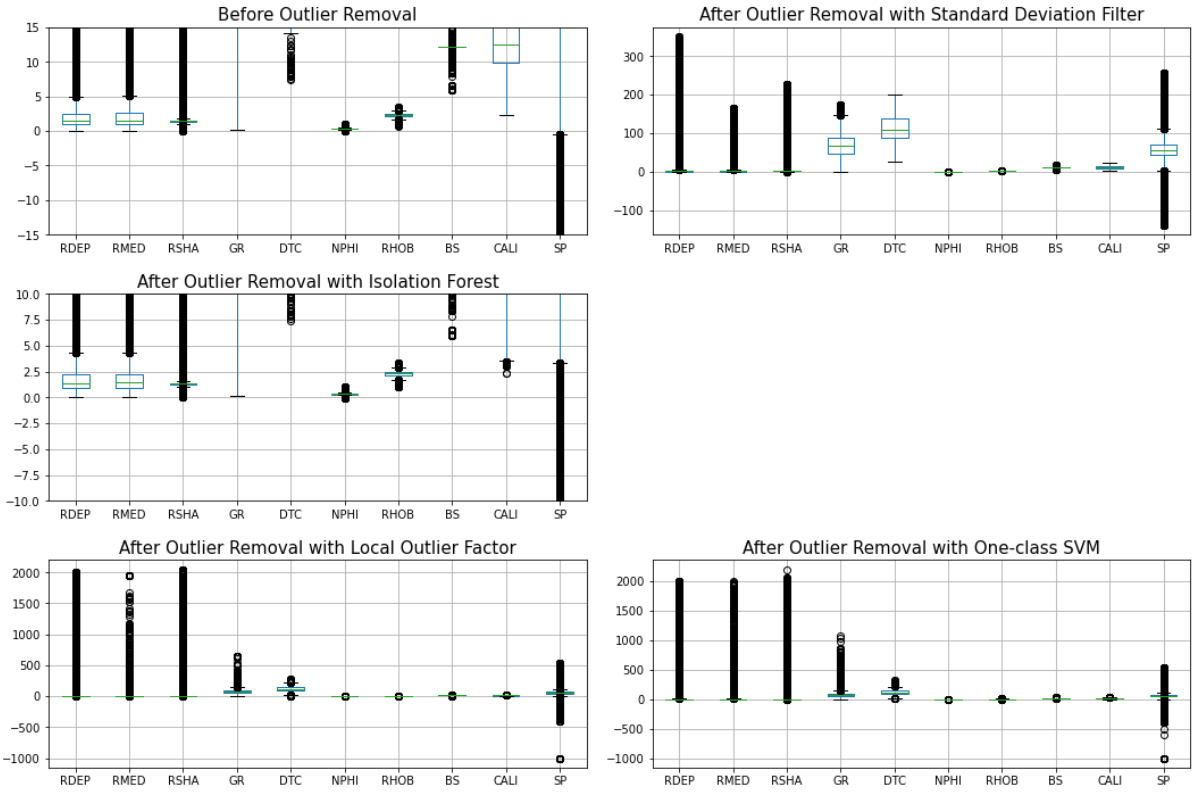


Figure 4.11: Box plot comparing the distribution of some logs in the data before outlier removal and 4 other outlier method.

Table 4.4: Table showing the percentage of points removed for different outlier methods

Outlier methods	Points after outliers removal	Number of points removed	Percentage of point removed
Local outlier Factor	819358	351153	30
Isolation Forest	1053460	117051	10
Standard Deviation	1046472	124039	11
One-class SVM	1053725	116786	10

4.2.2. Well log quality check

The quality of well logs can also be affected by bad hole, which can be caused by borehole enlargement, shale swelling, compacted rocks and variations in stresses acting on those rocks such as the mud weight. Measurements of the borehole's diameter and profile can be obtained from the Caliper Log, and the bit size curve can be used to determine the diameter of the bit used to bore the well (Figure 4.12).

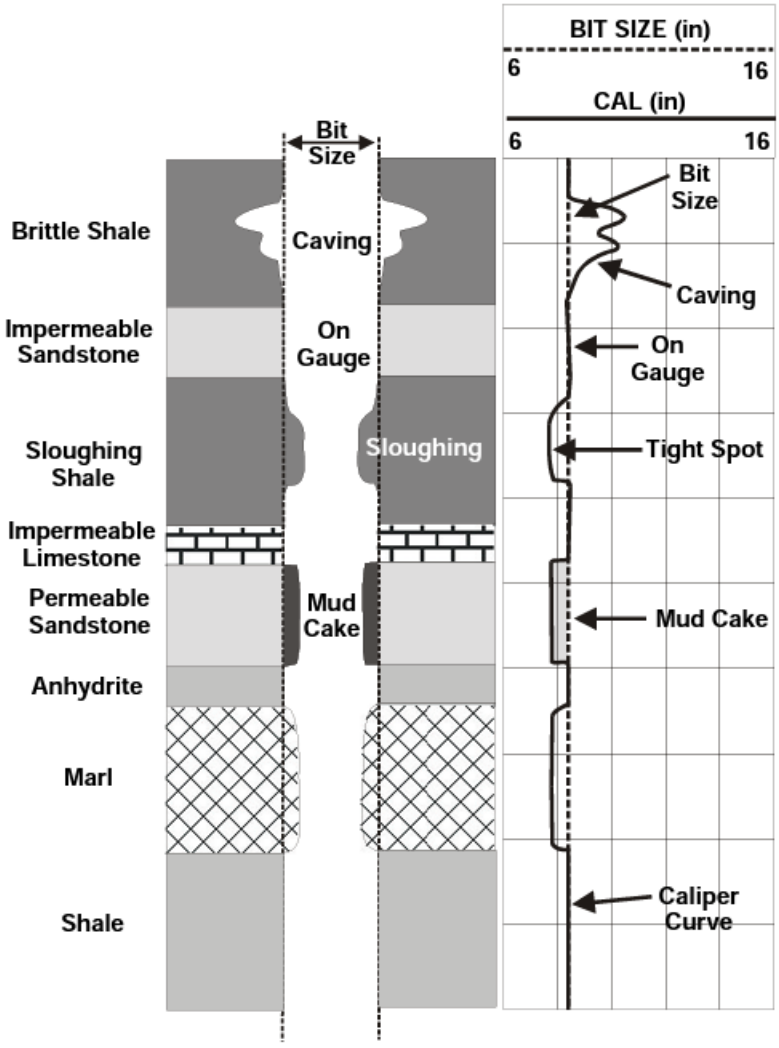


Figure 4.12: Typical caliper responses to various lithologies (Glover 2014)

To identify the wells affected by bad hole, a log was created by subtracting the caliper log from the bit size log. Negative values indicate that the borehole has decreased (e.g., shale swelling), whereas positive numbers indicate that the borehole has collapsed. From figure 4.13 below well 15/9–15,16/14-1 and 16/1–2,

contain bad hole data which might be caused by shale sloughing and enlarged borehole (cave in or collapse).

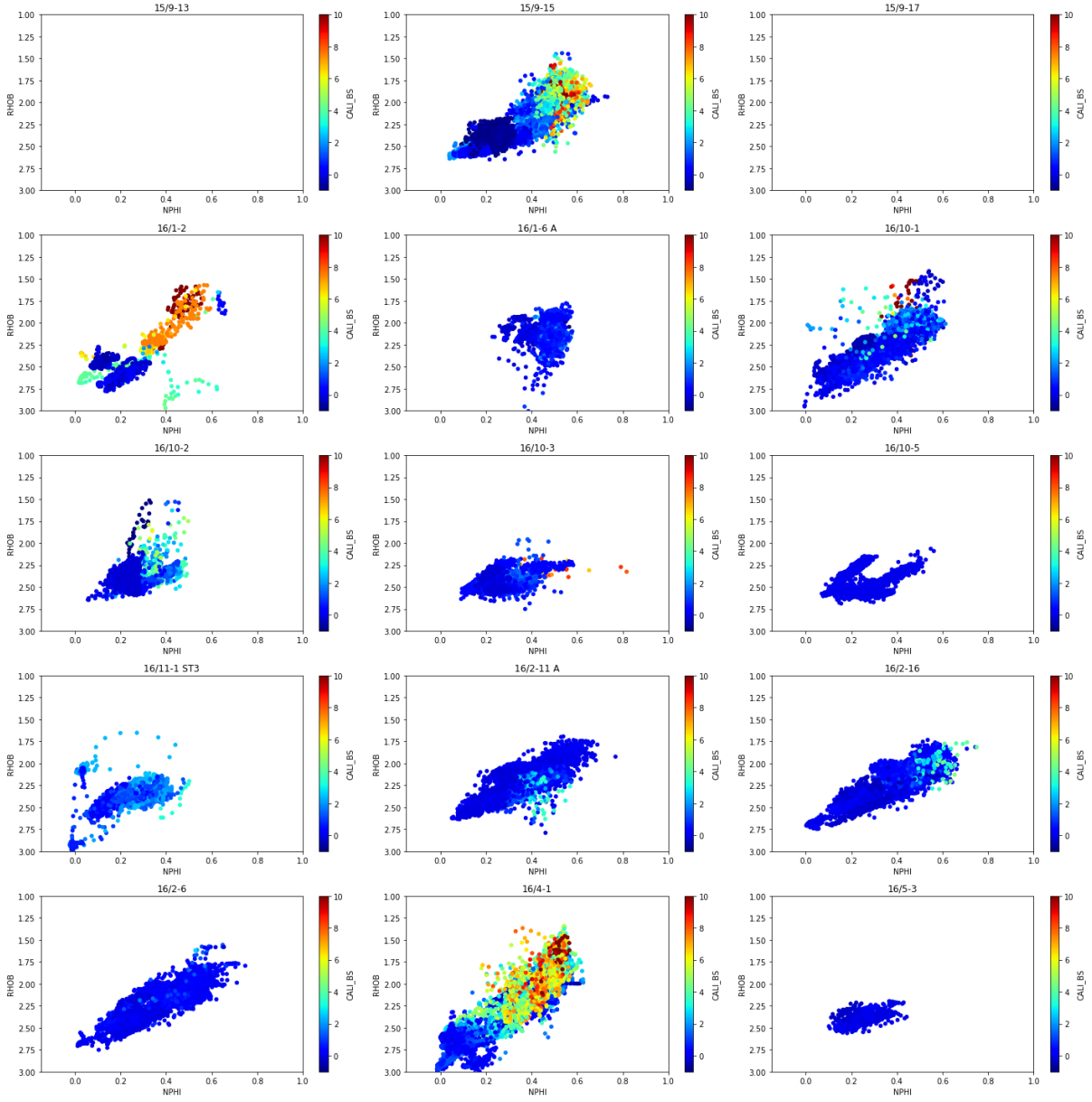


Figure 4.13: Cross plot of neutron density log showing bad hole data

4.2.3. Data pre-processing

With exception of few, most machine learning algorithms don't perform well when numerical values have different scales which is what is evident in our dataset. Since most of the well log reading are in different scales and several magnitudes in terms of range, it is important we bring all variable to same scale and range. This is done to improve the performance of the machine learning algorithms. To achieve this, I tested four different normalisation methods: MinMax scaler, standardisation,

robustscaler, and normalisation. Based on the boxplot in figure 4.14 the MinMax scaler method was chosen as the best normalisation method for the data.

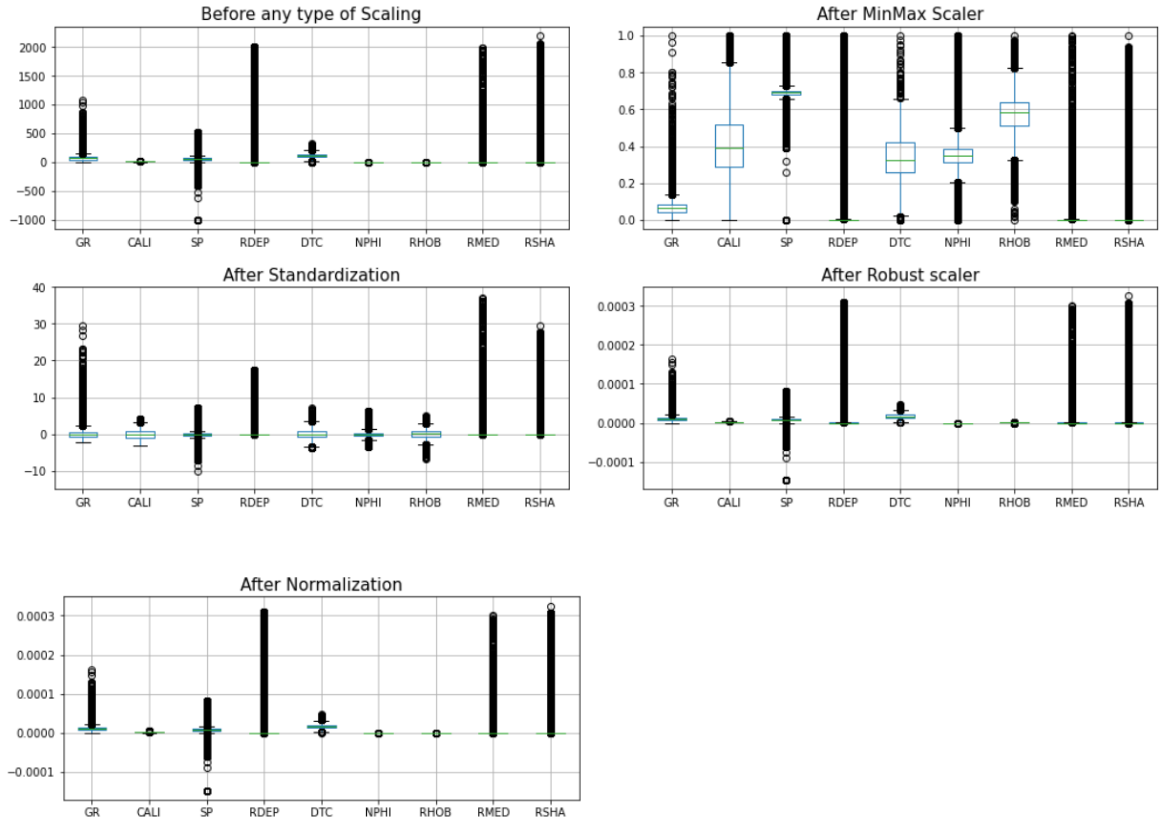


Figure 4.14: Wireline logs prior to scaling and after application of a min-max scaler, standard-scaler, and normalizer.

On the other hand, the categorical features such as well, group, formation, and lithology (target label) were label encoded by using a cat encoding. The missing values were replaced by the median values for all the well logs in the dataset. Another method of handling missing values is to predict the well log and impute the prediction to replace the null values. From feature selection and knowledge of geology, GR, NPHI, RHOB, DTS, and DTC logs are some of the most important features for identifying lithofacies. DTS, NPHI, RHOB and DTC have 85%,35%,14%, and 7% of missing values respectively. The extreme gradient boosting XGB regressor was used to train and predict the four logs and this was used to replace the null values in the logs.

4.3. Feature Engineering

Regarding feature engineering, we employed several techniques to enhance the predictive ability of the models.

4.3.1. Feature selection

When it comes to machine learning, one of the most important goals of the feature selection process is to choose the input features for the machine learning model in a way that considers the relationship between the features and the model output. Feature selection can improve model performance with a lower error rate, while also improving model generalisation and avoiding the problem of overfitting. To select the best characteristics, the K-best, Logistic regression, and variance methods were utilised. The RDEP, GR, NPHI, DTC, and RSHA features were chosen as the top five by each of the research approaches. For K-best and logistic regression, the least important five features are DRHO, ROP, SP, PEF, and RMED. For the variance method, the least important five features are X LOC, ROP, SP, PEF, and RMED (Figure 4.15 and 4.16).

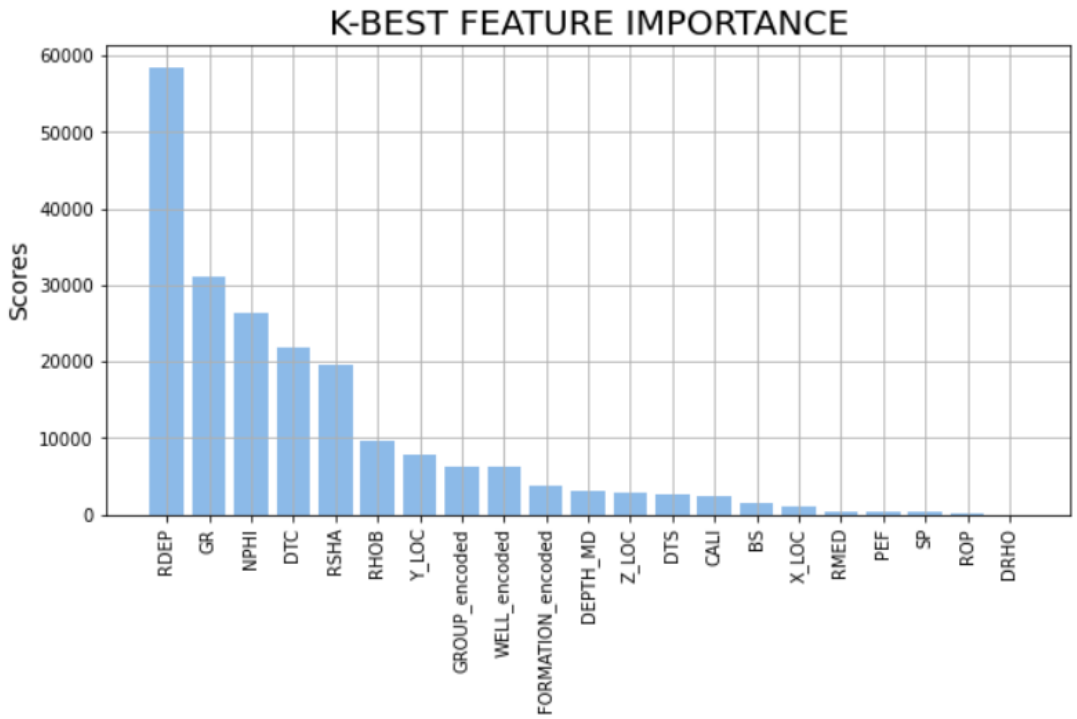


Figure 4.15: Bar chart of the distribution of K best feature importance

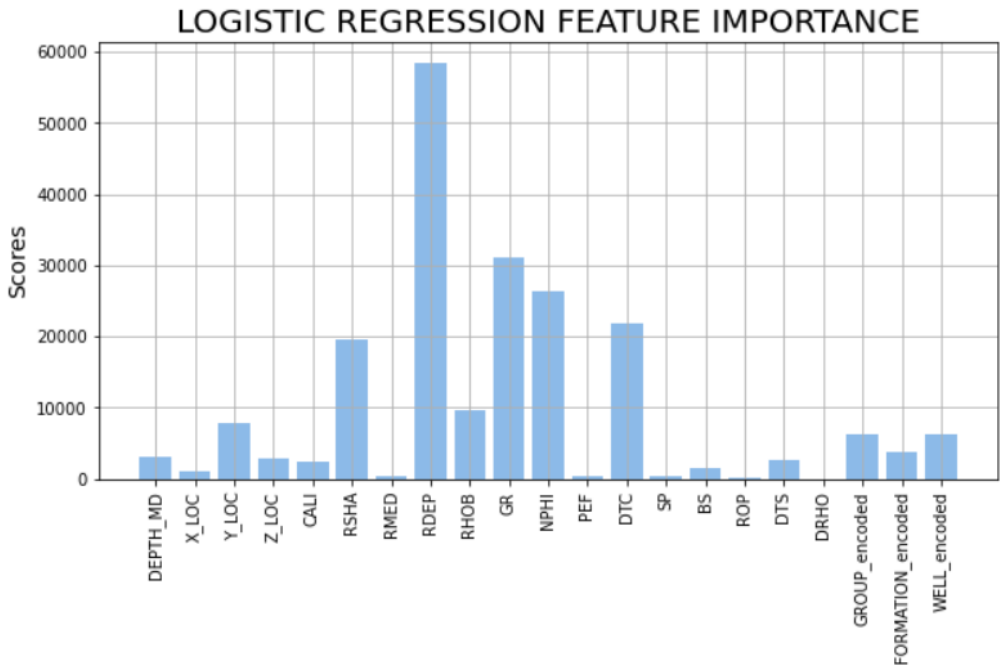


Figure 4.16: Bar chart of the distribution of K best feature importance

4.3.2. Feature extraction

In addition to the 29 features that were provided, four of which were imputed and improved with the help of machine learning, seven additional features were designed and included in the original datasets so that they could be used during the training, validation, and prediction stages.

1. Shale volume calculation: the shale volume was estimated from GR log using the equation below.

   $VSH\_GR=GR-GR\_ma/GR\_SH-GR\_ma$ 4.1

2. Porosity: is a measurement that indicates how much fluid a rock can hold. When determining a rock's porosity, the pore volume of the rock is divided by the rock's bulk volume (Tiab and Donaldson 1996). The term "total porosity" refers to the proportion of a rock's total pore space to the total volume of the rock itself. Neutron, density, and sonic logs are used in the calculation to arrive at the total porosity. The effective porosity of a material is calculated by taking the total porosity and subtracting the proportion of the pore space that is taken up by shale or clay (Crain 2019). The porosity can be determined by using the equation that is provided below.

$\emptyset$total=$\rho ma+\rho B/\rho ma+\rho f$ 4.2

$\emptyset effective=\emptyset t\times(1-vsh)$ 4.3

3. Bulk modulus(K)
4. Shear modulus(G)
5. P-wave impedance (PI)
6. S-wave impedance (SI)

4.4. Baseline Model.

There were nine baseline ML models that were designed for this study to evaluate which model best classifies the lithology. A cross-validation methodology was applied to ten stratified K-Folds taken from the training set, each of which contained a sample of 100,000 observations. This method takes the train dataset and divides it into ten subsets, then iterates using nine folds for training and one fold for testing. It also ensures that each data subset has the same lithology as the original training set, which helps to generalise the performance and prevents bias toward the lithology that is most common. Given that each model was trained and validated using only the training data without any attempt to regularise their learning process, these results may be susceptible to overfitting. This is because each model was trained and validated using only the training data.

Four different scenarios based on data pre-processing were also tested on each model to determine which type of data pre-processing improves the model performance.

The first scenario is the dataset imputed with median values, in this case the Random Forest classifier performed best with 91.4% accuracy, and support vector classifier is the least with 47.6% accuracy.

The second scenario is the dataset imputed with predicted well logs for DTC, DTS, RHOB and NPHI while other null values were imputed with the median value, in this case the Random Forest classifier performed best with 91.3% accuracy, and support vector classifier is the least with 48.8% accuracy. There is no improvement with random forest, but there is a slight improvement with the support vector and Logistic regression.

The third scenario is the dataset with additional extracted feature such as PHIT, PHIE, SI, PI, and so on, while other null values were imputed with the median value, in this case the Random Forest classifier performed best with 91.2% accuracy, and support vector classifier is the least with 47.6% accuracy. There is a slight improvement of 2% and 17% with KNN model when comparing scenario 1 and 3 and scenario 1 and 2 respectively. On the other hand, there is little or no improvement for other models.

The fourth scenario is the dataset with outliers removed by SVM method, while other null values were imputed with the median value, in this case the Random Forest classifier performed best with 91.2% accuracy, and support vector classifier is the least with 47.4% accuracy. Overall, removing the outliers did not improve the performance of any of the model, instead the performance reduced (Table 4.5).

The fifth scenario is the dataset with data augmentation, comparing this to the base model, only the logistic model accuracy increased by 0.5%, the model accuracy however decreased in decision tree, RF, XGB, CatBoost, and KNN.

Table 4.5: Comparing different scenario and model for 100,000 sample of the data.

Model	Scenario 1: Base Model with median imputation	Scenario 2: Model with predicted missing log	Scenario 3: Model with new features	Scenario 4: Model with SVM filter	Scenario 5: Model with data augmentation
Logistic Regression	0.478	0.505	0.491	0.475	0.483
Decision Trees Classifier	0.864	0.865	0.864	0.862	0.851
Random Forest Classifier	0.914	0.913	0.912	0.912	0.911
XGB Classifier	0.897	0.896	0.896	0.894	0.894

CatBoost	0.891	0.891	0.89	0.888	0.884
Gradient Boosting Classifier	0.836	0.834	0.834	0.831	0.836
LGBM	0.785	0.784	0.787	0.771	0.786
K-Nearest Neighbor Classifier	0.88	0.748	0.896	nan	0.867
Support Vector Machines	0.476	0.488	0.476	0.474	0.476

Some of this model will be further examined, modified, and tested on the open and hidden dataset, hyperparameter tuning will also be performed to select the best parameter for each model.

4.5. Random Forest

4.5.1 Random Forest base model

A base model was designed and trained to serve as a basis of comparison with subsequent model, to determine if creation of new features, hyperparameter tuning, data augmentation will improve the performance of the model.

With 100,000 sample of the dataset, an accuracy of 91% was achieved by the Random Forest base model after it was trained and validated on 10 different stratified k-folds as discussed earlier in section 4.4. To investigate further, RF base model was trained on the entire train dataset, and this achieved an accuracy of 100%, 78% and 80% on the train set, open test set and hidden set respectively (Table 4.6).

Table 4.6: Reports for the training, open, and hidden datasets generated by the Random Forest Classification algorithm.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	1	1	1	0.8	0.83	0.81	0.77	0.77	0.77
Sandstone/Shale (1)	1	1	1	0.83	0.93	0.87	0.85	0.96	0.9
Shale (2)	1	1	1	0.5	0.27	0.35	0.55	0.25	0.34
Marl (3)	1	1	1	0.43	0.55	0.48	0.59	0.63	0.61
Dolomite (4)	1	1	1	0.71	0.02	0.03	0.62	0.47	0.53
Limestone (5)	1	1	1	0	0	0	0	0	0
Chalk (6)	1	1	1	0.46	0.08	0.13	0.4	0.23	0.29
Halite (7)	1	1	1	0	0	0	0.82	0.6	0.69
Anhydrite (8)	1	1	1	_	_	_	0.99	0.99	0.99
Tuff (9)	1	1	1	0.75	0.51	0.6	0.88	0.62	0.73
Coal (10)	1	1	1	_	_	_	_	_	_
Basement (11)	1	1	1	0.75	0.49	0.6	0.7	0.46	0.55
Penalty matrix score	-2.99			-0.583			-0.553		
accuracy	1			0.78			0.8		
RMSE	0.0018			1.311			1.323		
Weighted avg	1	1	1	0.75	0.78	0.75	0.77	0.8	0.78

4.5.2. Random forest with hyperparameter

A hyper-parameter optimization process was carried out based on a grid parameter search technique. The parameter values that were used can be found in table 4.7 below. The hyper-parameter grid search was carried out on a 50,000 sample of the dataset to save time, while cross validating the training with 10 stratified folds to prevent overfitting the training data. The optimal hyper-parameters are provided below, and best score of 90% was achieved during the tuning.

Table 4.7: Hyperparameter values for Random forest.

Hyper parameter	Values	Best value
n_estimators	100, 150, 200	150
max_depth	10,15,20	20
criterion	gini, entropy	entropy
max_features	auto, sqrt	auto

min_samples_leaf	1,2,4	1

After training a new model with the best hyper-parameters, the final model achieved accuracies of 98%, 79%, and 80%, respectively, on the train, open, and hidden set. Table 4.8 provides a visual representation of the detailed classification reports that are broken down by each class of lithology.

Table 4.8: Reports for the training, open, and hidden datasets generated by the Random Forest Classification algorithm with hyperparameter.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	0.99	0.97	0.98	0.81	0.82	0.82	0.78	0.78	0.78
Sandstone/Shale (1)	0.98	1	0.99	0.83	0.95	0.88	0.85	0.95	0.9
Shale (2)	0.97	0.94	0.96	0.61	0.29	0.4	0.59	0.26	0.36
Marl (3)	1	0.91	0.95	0.41	0.57	0.47	0.59	0.61	0.6
Dolomite (4)	1	1	1	0	0	0	0.62	0.47	0.53
Limestone (5)	1	0.66	0.8	0	0	0	0	0	0
Chalk (6)	0.99	0.98	0.99	0.62	0.07	0.13	0.34	0.27	0.3
Halite (7)	1	1	1	0	0	0	0.73	0.62	0.67
Anhydrite (8)	1	1	1	_	_	_	0.99	0.99	0.99
Tuff (9)	1	0.98	0.99	0.77	0.52	0.62	0.88	0.65	0.75
Coal (10)	1	1	1	_	_	_	_	_	_
Basement (11)	0.99	0.99	0.99	0.72	0.48	0.58	0.66	0.55	0.6
Penalty matrix score	-0.0501			-0.551			-0.559		
accuracy	0.98			0.79			0.8		
RMSE	0.293			1.304			1.361		
Weighted avg	0.98	0.98	0.98	0.77	0.79	0.77	0.78	0.8	0.78

4.5.3. Random forest with data augmentation

Furthermore, a new model was trained with augmented data. This model achieved an accuracy of 100% on the training set, 78% on the open set, and 80% on the hidden set. Table 4.9 provides a visual representation of the detailed classification reports by lithology class. There is no significant improvement with the performance of model compared to the model without data augmentation.

Table 4.9. Reports for the training, open, and hidden datasets from the Random Forest Classification algorithm, completed with data augmentation.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	1	1	1	0.79	0.82	0.81	0.77	0.78	0.78
Sandstone/Shale (1)	1	1	1	0.83	0.93	0.88	0.85	0.97	0.91
Shale (2)	1	1	1	0.54	0.31	0.39	0.57	0.26	0.36
Marl (3)	1	1	1	0.43	0.56	0.49	0.61	0.62	0.62
Dolomite (4)	1	1	1	0.75	0.1	0.18	0.62	0.47	0.54
Limestone (5)	1	1	1	0	0	0	0	0	0
Chalk (6)	1	1	1	0.33	0.05	0.09	0.44	0.26	0.33
Halite (7)	1	1	1	0	0	0	0.76	0.64	0.7
Anhydrite (8)	1	1	1	_	_	_	0.99	0.99	0.99
Tuff (9)	1	1	1	0.79	0.52	0.62	0.88	0.63	0.73
Coal (10)	1	1	1	_	_	_	_	_	_
Basement (11)	1	1	1	0.72	0.49	0.59	0.7	0.57	0.63
Penalty matrix score	-1.708			-0.578			-0.535		
accuracy	1			0.78			0.8		
RMSE	0.009			1.323			1.294		
Weighted avg	1	1	1	0.76	0.78	0.76	0.77	0.8	0.78

4.5.4. Random forest with data augmentation and hyperparameters

Furthermore, a new model was trained with augmented data and hyperparameters. This model achieved an accuracy of 99% on the training set, 79% on the open set, and 80% on the hidden set. Table 4.10 provides a visual representation of the detailed classification reports by lithology class. There is no significant improvement with the performance of model compared to the model without data augmentation.

Table 4.10. Reports for the training, open, and hidden datasets from the Random Forest Classification algorithm, completed with hyperparameter and data augmentation.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	1	0.98	0.99	0.78	0.82	0.8	0.77	0.78	0.77
Sandstone/Shale (1)	0.98	1	0.99	0.82	0.94	0.88	0.85	0.96	0.9
Shale (2)	0.98	0.95	0.97	0.6	0.28	0.38	0.58	0.27	0.37
Marl (3)	1	0.93	0.96	0.47	0.57	0.51	0.62	0.61	0.62
Dolomite (4)	1	1	1	0	0	0	0.62	0.47	0.53
Limestone (5)	1	0.75	0.86	0	0	0	0	0	0
Chalk (6)	1	0.99	0.99	0.42	0.05	0.09	0.34	0.27	0.3
Halite (7)	1	1	1	0	0	0	0.75	0.62	0.68
Anhydrite (8)	1	1	1	_	_	_	0.99	0.99	0.99
Tuff (9)	1	0.99	1	0.83	0.52	0.64	0.88	0.66	0.75
Coal (10)	1	1	1	_	_	_	_	_	_
Basement (11)	1	0.99	0.99	0.72	0.47	0.57	0.69	0.57	0.62
Penalty matrix score	-0.038			-0.565			-0.557		
accuracy	0.99			0.79			0.8		
RMSE	0.243			1.322			1.341		
Weighted avg	0.99	0.99	0.98	0.76	0.79	0.76	0.78	0.8	0.78

4.5.5. Random forest with feature engineering

To further investigate if the model performance can be boosted, a new model was trained with hyper-parameters and 7 additional created features. This model achieved an accuracy of 98% on the training set, 77% on the open test, and 80% on the hidden test. Table 4.11 provides a visual representation of the detailed classification reports by lithology class. There is no significant improvement with the overall performance of the model compared to the model without feature engineering.

Table 4.11: Reports for the training, open, and hidden datasets from the Random Forest Classification algorithm, completed with hyperparameter and feature engineering.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	1	1	1	0	0	0	0.74	0.82	0.77
Sandstone/Shale (1)	1	1	1	_	_	_	_	_	_
Shale (2)	1	1	1	0	0	0	0.64	0.47	0.54
Marl (3)	1	0.98	0.99	0.84	0.54	0.66	0.87	0.66	0.75
Dolomite (4)	1	0.75	0.86	0	0	0	0	0	0
Limestone (5)	1	1	1	0	0	0	0.99	0.99	0.99
Chalk (6)	1	0.92	0.96	0.52	0.48	0.5	0.6	0.64	0.62
Halite (7)	0.99	0.98	0.99	0.39	0.03	0.06	0.34	0.26	0.29
Anhydrite (8)	0.99	0.98	0.99	0.89	0.7	0.79	0.76	0.78	0.77
Tuff (9)	0.98	0.95	0.96	0.64	0.16	0.26	0.57	0.28	0.38
Coal (10)	0.98	1	0.99	0.77	0.99	0.86	0.86	0.95	0.9
Basement (11)	0.99	0.99	0.99	0.75	0.34	0.47	0.65	0.59	0.62
Penalty matrix score	-0.056			-0.747			-0.681		
accuracy	0.98			0.77			0.8		
RMSE	0.316			1.07			1.075		
Weighted avg	0.98	0.98	0.98	0.75	0.77	0.73	0.78	0.8	0.79

4.6. Extreme Gradient boosting (XGB) model

4.6.1. Extreme Gradient boosting (XGB) base model

With 100,000 sample of the dataset, an accuracy of 89.7% was achieved by the XGB base model after it was trained and validated on 10 different stratified k-folds as discussed earlier in section 4.4.

The XGB base model on the entire train set was able to achieve an accuracy of 82% on the train set, 76% in the open set and 78% in the hidden set, respectively (Table 4.12.). However, because XGB has performed exceptionally well in several machine learning competitions for classification and regression tasks, I believed that careful selection of appropriate hyper-parameters might be able to improve its overall performance.

Table 4.12: XGB base model Classification reports for the training, open, and hidden datasets.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	0.78	0.72	0.74	0.74	0.78	0.76	0.75	0.56	0.64
Sandstone/Shale (1)	0.83	0.97	0.89	0.81	0.93	0.87	0.83	0.97	0.9
Shale (2)	0.76	0.35	0.48	0.42	0.24	0.31	0.49	0.27	0.34
Marl (3)	0.84	0.58	0.69	0.42	0.22	0.29	0.52	0.51	0.51
Dolomite (4)	0.9	0.9	0.9	0	0	0	0.56	0.58	0.57
Limestone (5)	0.79	0.19	0.3	0	0	0	0.29	0.49	0.37
Chalk (6)	0.79	0.55	0.65	0.38	0.07	0.12	0.48	0.25	0.33
Halite (7)	0.93	0.89	0.91	0.96	0.61	0.75	0.94	0.35	0.51
Anhydrite (8)	0.99	1	0.99	_	_	_	0.99	0.96	0.97
Tuff (9)	0.87	0.41	0.56	0.83	0.43	0.57	0.85	0.48	0.61
Coal (10)	1	0.83	0.91	_	_	_	_	_	_
Basement (11)	0.75	0.78	0.76	0.7	0.72	0.71	0.6	0.51	0.55
Penalty matrix score	-0.4742			-0.636			-0.614		
accuracy	0.82			0.76			0.78		
RMSE	1.196			1.339			1.388		
Weighted avg	0.81	0.82	0.8	0.72	0.76	0.73	0.75	0.78	0.75

4.6.2. Extreme Gradient boosting (XGB) with hyperparameters

Because the hyperparameter tuning for XGB model was expensive and time consuming, I was not able to perform a Gridsearch to select the best parameters. However, I used a set of hyperparameters from a previous work (Masapanta 2021) as shown in table 4.13.

Table 4.13: Selected hyperparameters for XGB model.

Hyper parameter	Selected value
n_estimators	1000
max_depth	4
booster	gbtree
objective	multi:softprob
learning_rate	0.075
random_state	42
subsample	1

colsample_bytree	1
verbose	2020
reg_lambda	1500

After training a new model with the selected hyper-parameters above, the final model achieved accuracies of 86%, 80%, and 81%, respectively, on the train set, open set, and hidden set. Table 4.14 provides a visual representation of the detailed classification reports that are broken down by each class of lithology. Compared to the base model, there is 4% and 3% increase in the open and hidden set accuracy when hyperparameters was used.

Table 4.14. Reports for the training, open, and hidden datasets from the XGB Classification algorithm completed with hyperparameter.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	0.83	0.8	0.82	0.84	0.81	0.82	0.74	0.78	0.76
Sandstone/Shale (1)	0.88	0.97	0.92	0.83	0.96	0.89	0.87	0.94	0.9
Shale (2)	0.77	0.57	0.65	0.6	0.29	0.39	0.55	0.36	0.43
Marl (3)	0.84	0.65	0.73	0.47	0.52	0.5	0.71	0.67	0.69
Dolomite (4)	0.89	0.9	0.9	0	0	0	0.8	0.79	0.8
Limestone (5)	0.63	0.12	0.2	0	0	0	0	0	0
Chalk (6)	0.8	0.66	0.73	0.64	0.15	0.25	0.29	0.24	0.26
Halite (7)	0.91	0.88	0.9	0.97	0.29	0.44	0.82	0.5	0.62
Anhydrite (8)	0.99	0.99	0.99	_	_	_	0.99	0.99	0.99
Tuff (9)	0.82	0.53	0.65	0.79	0.58	0.67	0.83	0.67	0.74
Coal (10)	0.98	0.44	0.6	_	_	_	_	_	_
Basement (11)	0.8	0.85	0.83	0.72	0.65	0.68	0.62	0.76	0.68
Penalty matrix score	-0.369			-0.534			-0.54		
accuracy	0.86			0.8			0.81		
RMSE	1.069			1.249			1.374		
Weighted avg	0.86	0.86	0.86	0.77	0.8	0.77	0.79	0.81	0.8

4.6.3. Extreme Gradient boosting (XGB) with data augmentation

To boost the classification accuracy, a new model was trained with data augmentation. This model achieved an accuracy of 93% on the training set, 76% on the open set, and 79% on the hidden set. Table 4.15 provides a visual representation of the detailed classification reports by lithology class. Compared to the base model,

there is a 11% accuracy increase in the train set, while in the open and hidden set there is no significant improvement with the performance of model.

Table 4.15: Reports for the training, open, and hidden datasets from the XGB Classification algorithm completed with data augmentation.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	0.9	0.9	0.9	0.8	0.8	0.8	0.71	0.76	0.73
Sandstone/Shale (1)	0.94	0.98	0.96	0.84	0.88	0.86	0.89	0.91	0.9
Shale (2)	0.86	0.76	0.81	0.42	0.35	0.38	0.54	0.46	0.5
Marl (3)	0.9	0.79	0.85	0.45	0.6	0.51	0.61	0.62	0.62
Dolomite (4)	0.99	0.99	0.99	0	0	0	0.63	0.48	0.54
Limestone (5)	0.94	0.78	0.85	0	0	0	0.4	0.15	0.21
Chalk (6)	0.91	0.88	0.89	0.12	0.07	0.09	0.28	0.28	0.28
Halite (7)	1	1	1	1	0.22	0.36	0.77	0.74	0.75
Anhydrite (8)	1	1	1	_	_	_	0.98	1	0.99
Tuff (9)	0.96	0.95	0.96	0.76	0.6	0.67	0.7	0.63	0.66
Coal (10)	1	1	1	_	_	_	_	_	_
Basement (11)	0.95	0.98	0.96	0.76	0.61	0.68	0.61	0.54	0.57
Penalty matrix score	-0.199			-0.648			-0.587		
accuracy	0.93			0.76			0.79		
RMSE	0.614			1.383			1.431		
Weighted avg	0.92	0.93	0.92	0.74	0.76	0.75	0.79	0.79	0.79

4.6.4. Extreme Gradient boosting (XGB) with data augmentation and hyperparameters

To boost classification accuracy, a new model was trained with selected hyper-parameters and data augmentation. This model achieved an accuracy of 89% on the training set, 79% on the open set, and 80% on the hidden set. Table 4.16 provides a visual representation of the detailed classification reports by lithology class. Comparing this model to the model with data augmentation and no hyperparameter, it can be observed that there is no improvement in the accuracy of the train set, while there is a 3% and 1% increase in accuracy in the open and hidden set respectively.

Table 4.16: Reports for the training, open, and hidden datasets from the XGB Classification algorithm completed with augmentation and hyperparameter.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	0.87	0.85	0.86	0.85	0.8	0.82	0.74	0.83	0.78
Sandstone/Shale (1)	0.92	0.97	0.94	0.83	0.94	0.88	0.88	0.93	0.9
Shale (2)	0.81	0.67	0.74	0.54	0.35	0.42	0.54	0.33	0.41
Marl (3)	0.85	0.72	0.78	0.48	0.6	0.53	0.64	0.64	0.64
Dolomite (4)	0.94	0.94	0.94	0.96	0.07	0.14	0.68	0.5	0.58
Limestone (5)	0.72	0.29	0.41	0	0	0	0.05	0.02	0.03
Chalk (6)	0.85	0.78	0.82	0.66	0.14	0.23	0.29	0.29	0.29
Halite (7)	0.95	0.94	0.94	1	0.16	0.28	0.81	0.66	0.73
Anhydrite (8)	1	1	1	_	_	_	0.99	0.99	0.99
Tuff (9)	0.84	0.75	0.79	0.74	0.67	0.7	0.75	0.73	0.74
Coal (10)	1	0.87	0.93	_	_	_	_	_	_
Basement (11)	0.88	0.93	0.9	0.78	0.65	0.71	0.62	0.65	0.63
Penalty matrix score	-0.282			-0.543			-0.565		
accuracy	0.89			0.79			0.8		
RMSE	0.859			1.225			1.407		
Weighted avg	0.89	0.89	0.89	0.78	0.79	0.78	0.79	0.8	0.79

4.6.5. Extreme Gradient boosting (XGB) with feature engineering

To further investigate the possibility of improving the model performance, a new model was trained with the selected hyper-parameters and 7 additional created features. This model achieved an accuracy of 88% on the training set, 80% on the open set, and 80% on the hidden set. Table 4.17 provides a visual representation of the detailed classification reports by lithology class. There is no significant improvement with the performance of model compared to the model without feature engineering.

Table 4.17: Reports for the training, open, and hidden datasets from the XGB Classification algorithm completed with hyperparameter and feature engineering.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	0.92	0.91	0.92	1	0.07	0.13	0.78	0.59	0.67
Sandstone/Shale (1)	0.99	0.8	0.88	_	_	_	_	_	_
Shale (2)	0.92	0.92	0.92	1	0	0	0.74	0.49	0.59
Marl (3)	0.81	0.65	0.72	0.61	0.68	0.64	0.82	0.68	0.74
Dolomite (4)	0.63	0.15	0.24	0	0	0	0.07	0.01	0.02
Limestone (5)	0.99	0.99	0.99	_	_	_	0.99	0.99	0.99
Chalk (6)	0.85	0.67	0.75	0.5	0.52	0.51	0.62	0.65	0.63
Halite (7)	0.82	0.72	0.77	0.66	0.15	0.25	0.3	0.27	0.29
Anhydrite (8)	0.85	0.83	0.84	0.83	0.81	0.82	0.73	0.81	0.77
Tuff (9)	0.79	0.62	0.69	0.58	0.33	0.42	0.53	0.31	0.39
Coal (10)	0.9	0.97	0.93	0.83	0.95	0.88	0.88	0.93	0.9
Basement (11)	0.84	0.89	0.86	0.74	0.63	0.68	0.61	0.66	0.63
Penalty matrix score	-0.414			-0.671			-0.694		
accuracy	0.88			0.8			0.8		
RMSE	0.757			1.041			1.079		
Weighted avg	0.87	0.88	0.87	0.78	0.8	0.77	0.78	0.8	0.79

4.7. KNN model

4.7.1. KNN base model

The non-parametric K-nearest neighbour base model of 100,000 samples produced an accuracy of 88% through cross validation on 10-stratified k-folds. However, despite the promising result demonstrated in the 100,000 sample base model, when KNN base model was trained on the entire train dataset, it achieved 96%, 57% and 55% accuracy on train set, open set and hidden set respectively (Table 4.18). To investigate the possibility of improving the model performance, a hyper-parameter optimization was also carried out.

Table 4.18: KNN base model Classification reports for the training, open, and hidden datasets.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	0.95	0.95	0.95	0.53	0.47	0.5	0.48	0.52	0.5
Sandstone/Shale (1)	0.97	0.98	0.98	0.78	0.71	0.75	0.77	0.63	0.69
Shale (2)	0.92	0.91	0.92	0.2	0.21	0.21	0.17	0.27	0.21
Marl (3)	0.95	0.86	0.9	0.13	0.4	0.2	0.37	0.56	0.45
Dolomite (4)	0.97	0.97	0.97	0	0	0	0.45	0.52	0.48
Limestone (5)	0.88	0.63	0.73	0	0	0	0	0	0
Chalk (6)	0.95	0.93	0.94	0.24	0.12	0.16	0.36	0.26	0.3
Halite (7)	0.96	0.93	0.95	0	0	0	0.71	0.58	0.64
Anhydrite (8)	0.99	1	0.99	_	_	_	0.99	0.65	0.78
Tuff (9)	0.92	0.8	0.85	0.55	0.38	0.45	0.77	0.17	0.28
Coal (10)	0.98	1	0.99	_	_	_	_	_	_
Basement (11)	0.95	0.97	0.96	0.13	0.25	0.17	0.03	0.19	0.05
Penalty matrix score	-0.108			-1.137			-1.223		
accuracy	0.96			0.57			0.55		
RMSE	0.56			2.005			2.806		
Weighted avg	0.96	0.96	0.96	0.61	0.57	0.59	0.63	0.55	0.58

4.7.2. KNN with hyperparameters

A hyper-parameter optimization process was carried out based on a grid parameter search technique. The parameter values that were used can be found in table 4.19 below. The hyper-parameter grid search was carried out on a 50,000 sample of the dataset while cross validating the training with 10 stratified folds to prevent overfitting the training data. The optimal hyper-parameters are provided below, and the process achieved a score of 88%.

Table 4.19: Hyperparameter values for KNN model

Hyper parameter	Values	Best value
n_estimators	1,10,1	1
leaf_size	20,40,1	20
weights	uniform, distance	uniform
metric	minkowski, chebyshev	minkowski

After training a new model with the most appropriate hyper-parameters, the final model achieved accuracies of 100%, 61%, and 56%, respectively, on the training, open test, and hidden test. Table 4.20 provides a visual representation of the detailed classification reports that are broken down by each class of lithology. Comparing this model to the base model in section 4.7.1,there is 4%,4% and 1% in the train, open and hidden dataset respectively.

Table 4.20: KNN Classification reports for the training, open, and hidden datasets with hyperparameters.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	1	1	1	0.57	0.59	0.58	0.53	0.54	0.53
Sandstone/Shale (1)	1	1	1	0.81	0.73	0.77	0.77	0.6	0.68
Shale (2)	1	1	1	0.23	0.25	0.24	0.21	0.37	0.27
Marl (3)	1	1	1	0.17	0.46	0.25	0.35	0.57	0.43
Dolomite (4)	1	1	1	0	0	0	0.42	0.46	0.44
Limestone (5)	1	1	1	0.02	0.01	0.01	0	0	0
Chalk (6)	1	1	1	0.29	0.15	0.2	0.38	0.29	0.33
Halite (7)	1	1	1	0.96	0.54	0.69	0.55	0.62	0.59
Anhydrite (8)	1	1	1	0	0	0	0.98	0.7	0.82
Tuff (9)	1	1	1	0.58	0.4	0.47	0.57	0.16	0.25
Coal (10)	1	1	1	_	_	_	_	_	_
Basement (11)	1	1	1	0.2	0.14	0.16	0.1	0.47	0.17
Penalty matrix score	0			-1.036			-1.186		
accuracy	1			0.61			0.56		
RMSE	0			1.722			2.474		
Weighted avg	1	1	1	0.64	0.61	0.62	0.64	0.56	0.59

4.7.3. KNN with augmented data and hyperparameters.

To boost classification accuracy, a new model was trained with selected hyper-parameters and augmented data. This model achieved an accuracy of 95% on the training set, 57% on the open set, and 59% on the hidden set. Table 4.21 provides a visual representation of the detailed classification reports by lithology class. Comparing this model to the model with hyperparameter in section 4.7.2., it can be observed that there is a 5% and 4% decrease in accuracy of the train and open set respectively, while there is a 3% increase in the accuracy of the hidden set.

Table 4.21: KNN Classification reports for the training, open, and hidden datasets on augmented data with hyperparameters.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	0.93	0.93	0.93	0.54	0.46	0.5	0.46	0.54	0.5
Sandstone/Shale (1)	0.96	0.98	0.97	0.75	0.72	0.74	0.76	0.69	0.72
Shale (2)	0.9	0.86	0.88	0.2	0.22	0.21	0.21	0.28	0.24
Marl (3)	0.94	0.83	0.88	0.15	0.35	0.21	0.53	0.53	0.53
Dolomite (4)	0.97	0.96	0.96	0	0	0	0.45	0.52	0.48
Limestone (5)	0.85	0.55	0.67	0.01	0.01	0.01	0	0	0
Chalk (6)	0.94	0.91	0.92	0.27	0.12	0.17	0.33	0.26	0.29
Halite (7)	0.95	0.9	0.93	0	0	0	0.78	0.17	0.28
Anhydrite (8)	0.99	1	0.99	_	_	_	0.98	0.61	0.75
Tuff (9)	0.93	0.78	0.85	0.58	0.32	0.41	0.49	0.08	0.13
Coal (10)	0.99	0.99	0.99	_	_	_	_	_	_
Basement (11)	0.92	0.95	0.94	0.16	0.31	0.21	0.03	0.2	0.06
Penalty matrix score	-0.142			-1.104			-1.139		
accuracy	0.95			0.57			0.59		
RMSE	0.648			1.947			2.824		
Weighted avg	0.95	0.95	0.95	0.6	0.57	0.58	0.64	0.59	0.61

4.7.4. KNN model with feature engineering

To further investigate the possibility of improving the model performance, a new model was trained with hyper-parameters and 7 additional created features. This model achieved an accuracy of 100% on the train set, 60% on the open set, and 55% on the hidden set. Table 4.22 provides a visual representation of the detailed classification reports by lithology class. Comparing this to the base model, there is an increase of 4% and 3% on the train and open set respectively, while there is no improvement in the hidden set.

Table 4.22 KNN Classification reports for the training, open, and hidden datasets with hyperparameters and feature engineering.

	Training set			Open set			Hidden set		
Lithology	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Sandstone (0)	1	1	1	0.92	0.66	0.77	0.3	0.65	0.41
Sandstone/Shale (1)	1	1	1	_	_	_	_	_	_
Shale (2)	1	1	1	0	0	0	0.39	0.41	0.4
Marl (3)	1	1	1	0.6	0.42	0.49	0.67	0.35	0.46
Dolomite (4)	1	1	1	0.01	0	0.01	0	0	0
Limestone (5)	1	1	1	_	_	_	0.97	0.3	0.46
Chalk (6)	1	1	1	0.19	0.45	0.27	0.32	0.55	0.41
Halite (7)	1	1	1	0.27	0.14	0.19	0.41	0.31	0.35
Anhydrite (8)	1	1	1	0.55	0.56	0.56	0.59	0.49	0.54
Tuff (9)	1	1	1	0.22	0.23	0.22	0.26	0.43	0.32
Coal (10)	1	1	1	0.78	0.74	0.76	0.74	0.62	0.68
Basement (11)	1	1	1	0.25	0.15	0.19	0.1	0.48	0.16
Penalty matrix score		0		-1.35			-1.56		
accuracy		1		0.6			0.55		
RMSE		0		1.388			1.965		
Weighted avg	1	1	1	0.62	0.6	0.61	0.63	0.55	0.57

CHAPTER FIVE

DISCUSSION

In this study, I investigated how accurately machine learning model like RF, SVM, Logistic regression, decision tree, KNN, XGB, Light GBM, GB, and CatBoost can be able to classify lithology label in well logs. To achieve this, the models were trained with or without hyperparameters, with or without feature engineering, and with or without augmented data to investigate which model performed best. For the 100,000-sample data, Random forest performed best, which is consistent with Merembayev et al. 2021. Random Forest, CatBoost, XGB and KNN performed best without any form of feature engineering, outlier filter and ML imputation, Logistic regression, decision tree and support vector machine performed best when the dataset has ML imputation (scenario 2), KNN and LGBM performed best with feature engineering. None of the model performed better when augmented data was used

For deeper study, 3 top models (RF, XGB, KNN) in the first investigation were used to train the entire train dataset and evaluated on the open and hidden set. A comparison of the model performance for different scenario is shown in table 6.1.

Table 6.1. Comparison of the train, open and hidden accuracy of Random forest, Extreme gradient boosting, and K nearest neighbour.

	Random Forest			Extreme Gradient Boosting			K-Nearest Neighbour		
	Train set	Open set	Hidden set	Train set	Open set	Hidden set	Train set	Open set	Hidden set
Base model	100	78	80	82	76	78	96	57	55
Model with hyperparameters	98	79	80	86	80	81	100	61	56
Model with data augmentation	100	78	80	93	76	79	95	57	59
Model with data augmentation and hyperparameters	99	79	80	89	79	80	_	_	_
Model with feature engineering	98	77	80	88	80	80	100	60	55

For random forest trained on the entire train dataset, and evaluated on the open dataset, it was observed that on the open set RF performed best at classifying sandstone/shale lithology, and it was able to differentiate between sandstone/shale and sandstone but misclassified the shale lithology. With feature engineering, the model was not able to classify sandstone, shale/sandstone, and shale but coal was correctly classified with feature engineering. There was improvement with the classification of chalk from 7% to 48%, coal from 0% to 99% and anhydrite from 0% to 70%. On the hidden dataset, RF performed best at classifying anhydrite, sandstone/shale, and halite. With feature engineering, sandstone accurate

classification improved from 77% to 82%, while sandstone/shale could not be classified correctly. The classification of coal also increased from 0% to 95%, and limestone from 0% to 99%. There is little or no improvement with the classification of all the lithology when Augmented data was used with or without hyperparameters.

Overall, RF model was unable to generalize its performance on unseen objects, and it performed quite well with classifying lithology facies that are not similar to each other.

For XGB trained on the entire train dataset and evaluated on the open dataset, it was observed that XGB performed well at classifying sandstone/shale lithology, and it was able to differentiate between sandstone/shale and sandstone but misclassified the shale lithology more. In the open set, XGB model was not able to make any accurate classification for dolomite, limestone, anhydrite, and coal, whether with hyperparameters, feature engineering, data augmentation or not. While on the hidden set, anhydrite was accurately classified (96%), as well as sandstone/shale (97%). When trained with hyperparameters, the overall accuracy of open and hidden set increased by 4% and 3% respectively. With data augmentation, marl lithology accuracy increased from 22% to 60%, and with feature engineering it increased to 68% (figure 6.1). XGB also classifies basement better than RF.

Feature engineering was able to improve the classification of the lithology with fewer observation like marl, basement, anhydrite, and coal compared to lithologies like shale, sandstone/shale and sandstone that contributes to more than 70% of the dataset. XGB model trained with hyperparameter performed better for most of the lithology like sandstone, shale, sandstone/shale, marl, tuff, chalk except basement and halite. The XGB model could not classify dolomite and limestone in all the different scenario, dolomite was misclassified as marl while limestone was misclassified as sandstone/shale (figure 6.2). In the hidden set, the base model classified sandstone/shale (97%), and anhydrite (99%) correctly, while other lithology was averagely classified.

Overall, the model performance was better in hidden set compared to the open set and this may be because the distribution of the features is more balanced in the hidden set.

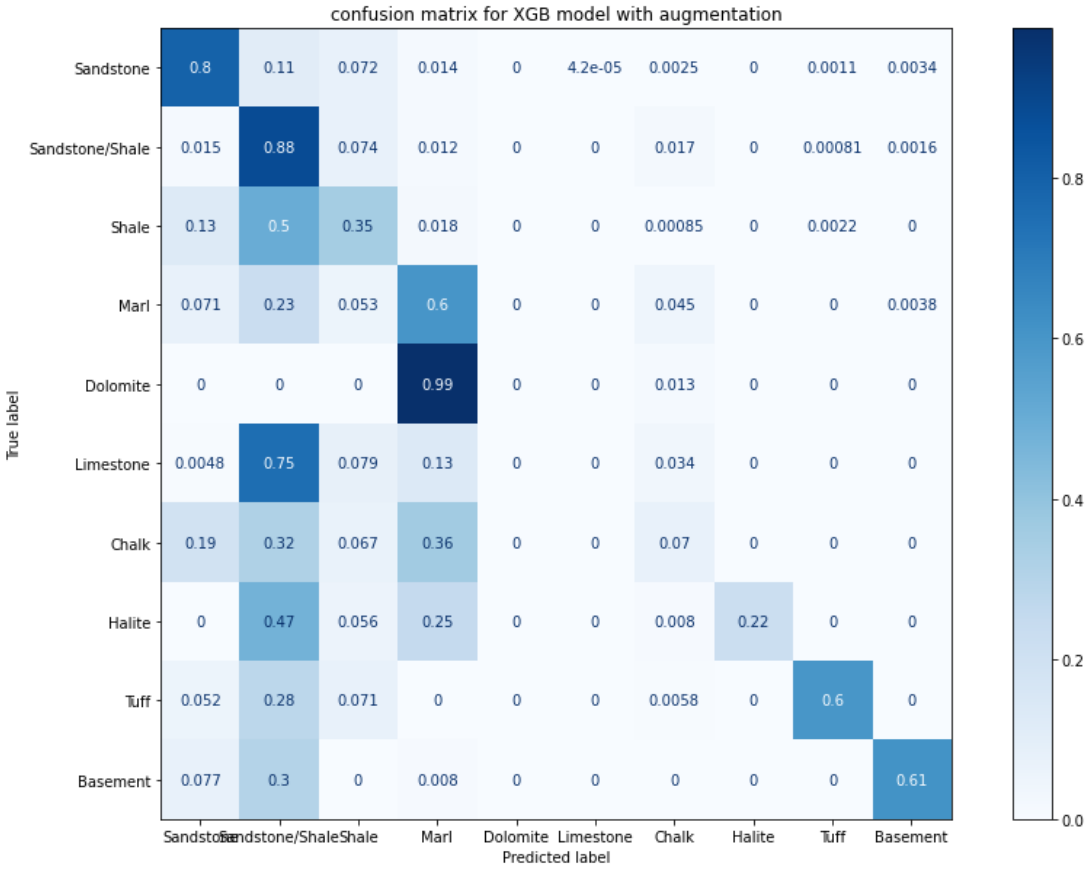


Figure 6.1. Normalised confusion matrix of XGB model with augmented data.

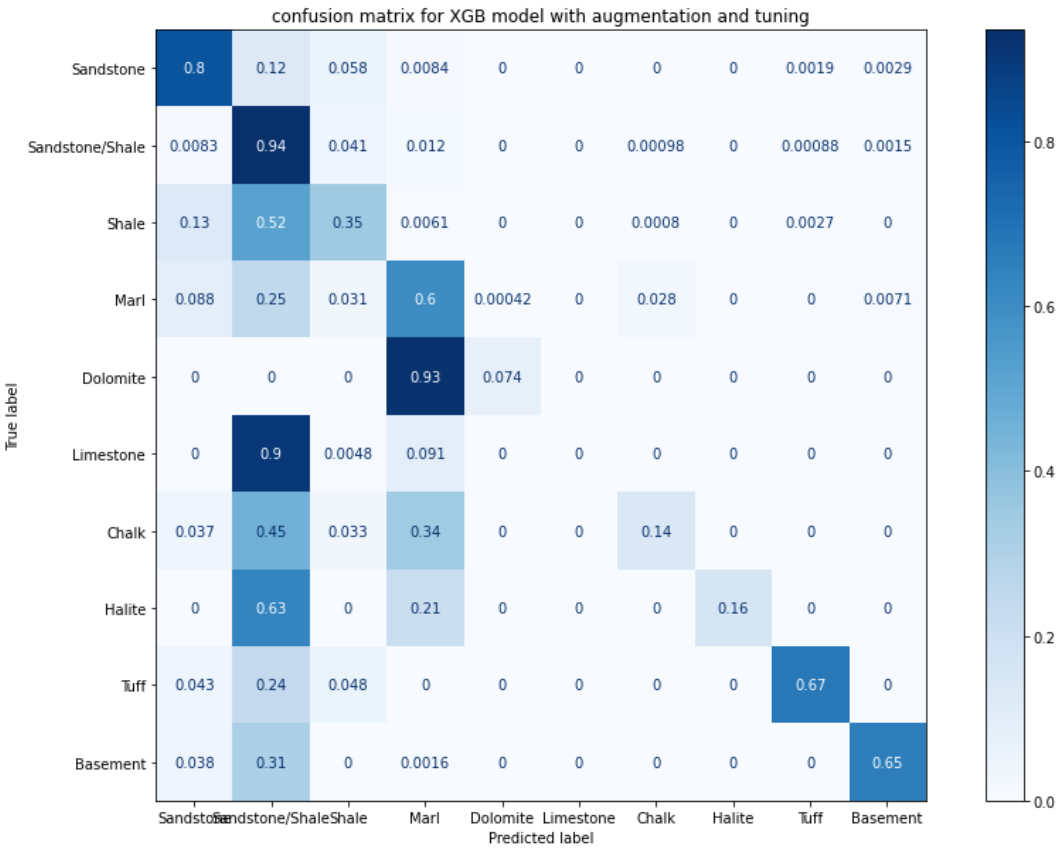


Figure 6.2. Normalised confusion matrix of XGB model with augmented data and hyperparameter.

Though, KNN model achieved an accuracy of 88% in the 100,000 sample base model, its overall performance in the open and hidden set was low. In the open set, KNN base model could not classify dolomite, limestone, halite, anhydrite and coal, with hyperparameters halite and limestone were moderately and poorly classified respectively while dolomite, anhydrite and coal were misclassified. In the hidden set, dolomite, halite, and anhydrite were moderately classified

Base model and model with augmented data with or without hyperparameter misclassified coal, but with feature engineering coal was classified with an accuracy of 74% and 62% in open and hidden set respectively.

Shale lithology is generally misclassified in all the model, but feature engineering improves the performance of the classification from 27% to 41% in the hidden data set. There was little improvement in the overall classification of most lithology with

the model trained with augmented data with or without tuning, and dolomite and halite were misclassified (Figure 6.3).

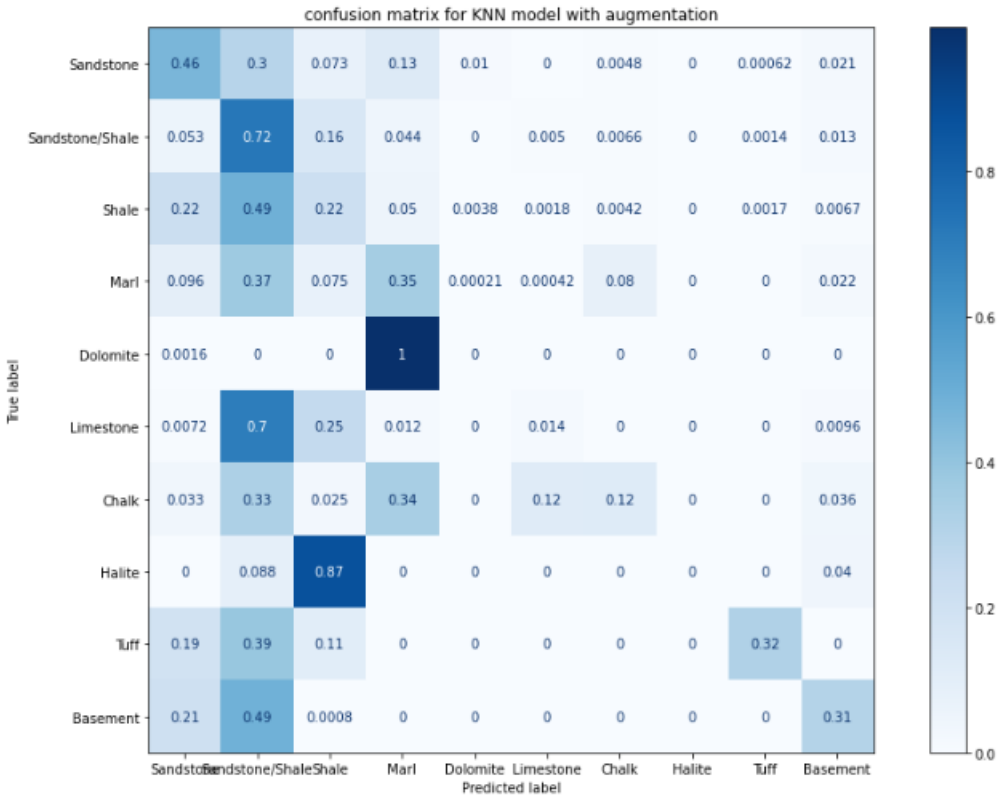


Figure 6.3. Normalised confusion matrix of KNN model with augmented data.

Comparing the actual label and predicted label for the three model as shown in figure 6.4 below, none of the model can accurately classify the thin beds of shales and chalk. KNN performed worst as it misclassifies most of the thin layers of lithology.

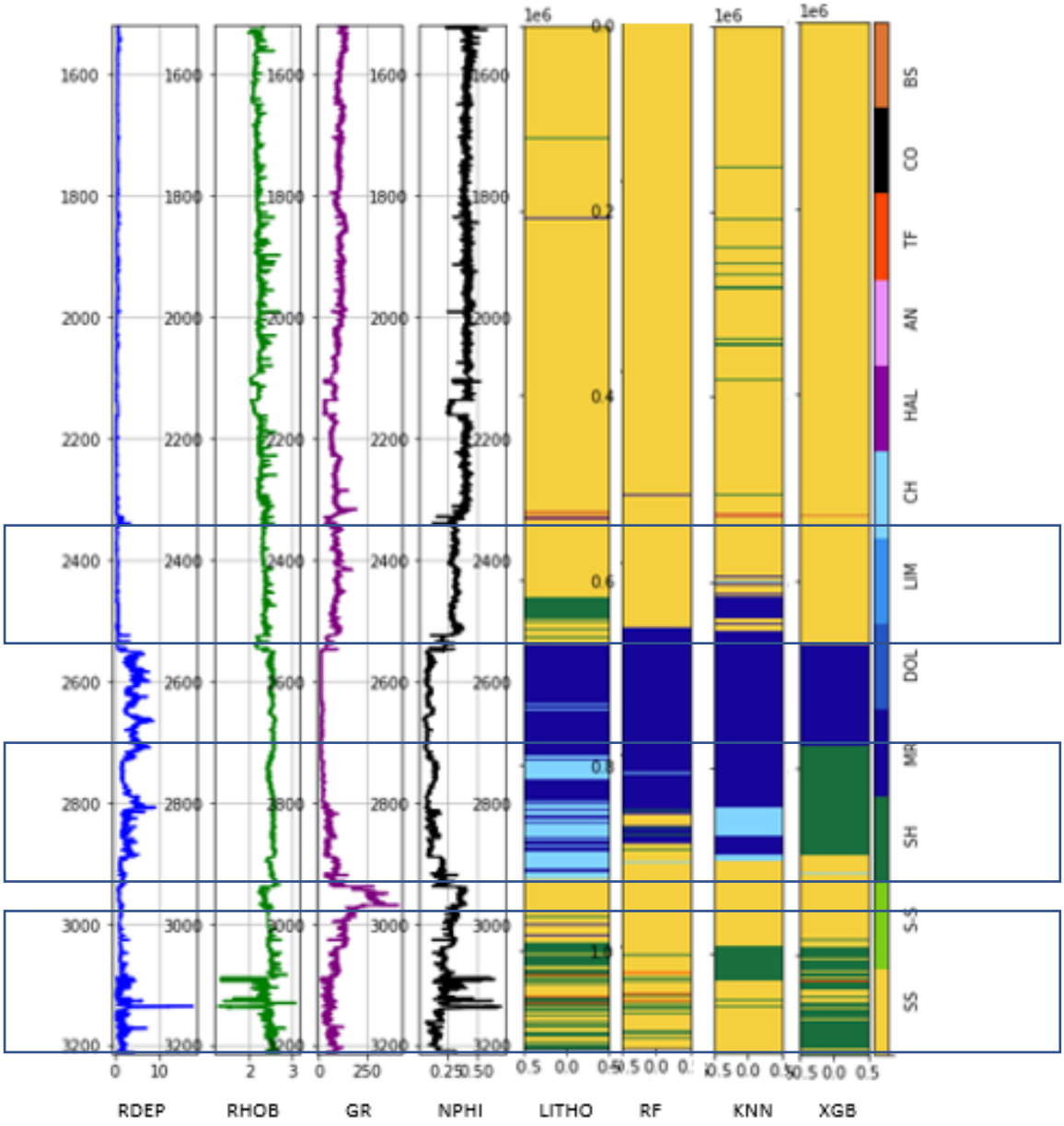


Figure 6.4. Prediction analysis of well 15/9-23 comparing the actual label to the predicted label in RF, XGB and KNN.

CHAPTER SIX

CONCLUSION

Within the scope of this study, I investigated the capability of nine different machine learning algorithms to classify rock facies. The various machine learning algorithms that I have tried includes Random Forest (RF), Decision tree, Support Vector Machine (SVM), Gradient boosting, Categorical boosting (CatBoost), Light gradient boosting (LGBM), Extreme gradient boosting (XGB), K-Nearest Neighbour (KNN), and logistic regression, both with and without feature engineering, outlier removal, and ML imputation. According to the findings, the Random Forest model produces the best results out of all the algorithms that were taken into consideration, with an accuracy of 91%. The XGB model and the CatBoost model came in second and third, with an accuracy of 89.6% and 89.1% respectively. When the entire model was trained and tested on the open and hidden set, XGB model trained with hyperparameters performed best with accuracy of 86%,80% and 81% on the train, open and hidden dataset respectively. With RF model trained with hyperparameters, the accuracy is of 98%,79% and 80% on the train, open and hidden dataset respectively. Therefore XGB is a better model for this data because produce good accuracy on the both training and test data, while in RF the model could not generalise on the test data because of overfitting.

The thesis's main contribution can be broken down into three sections. The first major takeaway from the research is that machine learning can accurately predict rock facies in the FORCE data set, with an average accuracy of 81%. Second, Models trained with hyperparameters performed better, as well as model trained with augmented data. To conclude, feature engineering on the test data was most effective with KNN and LGBM, feature engineering also improved the classification of some lithology, especially lithologies like coal, limestone and marl with few observations point in the data. It was also observed that the removal of outliers did not improve the model’s performance.

Considering these encouraging outcomes, future work will focus on analysing deep learning approaches to feature learning and classification (e.g., convolutional neural networks).

Recommendation

First and foremost, there is need to address the problem of unbalanced dataset. This can be accomplished by weighting samples in accordance with the frequency of the classes. If we give fewer common classes a higher weight, the model may be able to classify them more accurately. In addition, we could try the upsampling or downsampling method to train our model on a dataset that is more evenly distributed.

Then the tuning of the hyperparameters should be improved. Either by performing additional iterations of our existing grid search or by utilising bayesian search, which typically yields superior results.

Alternate types of models are another option that can enhance the quality of our findings. For instance, LSTM architecture can be used to simultaneously classify all the samples that come from the same well. In this way, our model will classify a sample based on information obtained from its surroundings. Additionally, this kind of model might be able to capture the structure of the lithofacies that are found underground.

References

Akinnikawe, O., S. Lyne, and J. Roberts, 2018. Synthetic well log generation using machine learning techniques. *Unconventional Resources Technology Conference*. doi: 10.15530/URTEC-2018-2877021.

Al-Anazi, A., and I. D. Gates, 2010. A support vector machine algorithm to classify lithofacies and model permeability in heterogeneous reservoirs. *Engineering Geology*, v. 114, no. 3, p. 267– 277, doi:10.1016/j.enggeo.2010.05.005.

Anifowose, F., J. Labadin, and A. Abdulraheem, 2014. Improving the prediction of petroleum reservoir characterization with a stacked generalization ensemble model of support vector machines. *Applied Soft Computing*, v. 26, 483-496.

Asquith, G., D. Krygowski, S. Henderson, and N. Hurley, 2004. Basic well log analysis. *American Association of Petroleum Geologists*.

Avseth, P. and T. Mukerji, 2002. Seismic lithofacies classification from well logs using Statistical rock physics. *Petrophysics,* v. 43, pp. 70-81.

Bai, R., Z. Jin-gong, and L. Wei, "Reservoir geological model of lithological reservoir of Chang 6 3 in Heshui area, Ordos Basin," *Journal of Jilin University (Earth Science Edition)*, vol. 42, no. 6, pp. 1601–1609, 2012.

Bestagini, P., V. Lipari, and S. Tubaro, 2017. A machine learning approach to facies classification using well logs. *Seg Technical Program Expanded Abstracts*.

Bhattacharya, B., R.C. Timothy, and M. Pal. 2016. Comparison of Supervised and unsupervised approaches for mudstone lithofacies classification: case studies from the Bakken and Mahatango-Marcellus Shale, USA. *Natural Gas Science and Engineering*, v. 33, pp. 1119-1133.

Breiman L., 2001. Random forests. *Machine Learning* 45(1):5–32

Burl, M.C., L. Asker, P. Smyth, U. Fayyad, P. Perona, L. Crumpler, and J. Aubele, 1998.'Learning to recognize volcanoes on Venus', *Machine Learning*, vol. 30, no. 2, pp.165-194.

Busch, J., W. Fortney, and L.J.S. Berry, 1987. Determination of lithology from well logs by statistical analysis. *Society of Petroleum Engineer Formation Evaluation*, 2(04), 412–418.

Chaki, S., A. Routray, W.K. Mohanty, and M. Jenamani, 2015. A novel multiclass SVM based framework to classify lithology from well logs: a real-world application, In *IEEE INDICON*.2015.

Chen, J. and Y. Zeng, 2018. Application of machine learning in rock facies classification with physics-motivated feature augmentation. *arXiv preprint arXiv:1808.09856*.

Chen, T., and C. Guestrin, 2016. XGBoost: A Scalable Tree Boosting System, *in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*: MIT Press, p. 785–794, doi:10.1145/2939672.2939785.

Chen, Y., 2020. Automatic micro seismic event picking via unsupervised machine learning. *Geophysical Journal International*, Volume 212, Issue 1, January 2018, Pages 88–102. https://doi.org/10.1093/gji/ggx420.

Chris, L., 2019. Evaluating ML Models: Precision, Recall, F1 and Accuracy, [viewed 20 August,2022]. Available from: https://medium.com/analytics-vidhya/evaluating-ml-models-precision-recall-f1-and-accuracy-f734e9fcc0d3.

Delfiner, P., O. Peyret, and O.J.S. Serra, 1987. Automatic determination of lithology from well logs. *Society of Petroleum Engineer Formation Evaluation*, 2(03), 303–310.

Dubois, M. K., G. C. Bohling, and S. Chakrabarti, 2007. Comparison of four approaches to a rock facies classification problem. *Computers and Geosciences*, v. 33, no. 5, p. 599–617, doi:10.1016/j.cageo.2006.08.011.

Force,2020. Blog post litho facies competition geological summary. October 2020 [viewed 01 September 2020]. Available from:https://docs.google.com/document/d/13XAftsBVHIm01ZN0lP56Q4hZ9hgdYR1G_6KeV2DdzOA/edit

Gandhi, R., 2018. Support Vector Machine - Introduction to Machine Learning Algorithms, SVM model from scratch, [viewed 01 september, 2022]. Available from: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

Geological Survey of Norway, 2015. Stratigraphy. [viewed 01 september, 2022]. Available from: https://www.ngu.no/en/topic/stratigraphy

Ghori, K. M., R. A. Abbasi, M. Awais, M. Imran, A. Ullah, and L. Szathmary, 2019. Performance Analysis of Different Types of Machine Learning Classifiers for Non-Technical Loss Detection: *IEEE Access*, v. 8, p. 16033–16048, doi:10.1109/ACCESS.2019.2962510.

Gill, D., A. Shomrony, and H.J.A. Fligelman, 1993. Numerical zonation of log suites and logfacies recognition by multivariate clustering. *American Association of Petroleum Geoscience Bulletin*, 77(10), 1781–1791.

Glover, P. (2014). Petrophysics MSc course notes [viewed 15 August, 2022]. Available from:Chromeextension://efaidnbmnnnibpcajpcglclefindmkaj/https://homepagessee.leeds.ac.uk/~earpwjg/PG_EN/CD%20Contents/GGL66565%20Petrophysics%20English/Chapter%201.PDF

Graves, A. 2012. “Supervised Sequence Labelling with Recurrent Neural Networks”. *Studies in Computational Intelligence,Berlin, Germany:* Springer, 2012.

Guarido, M., 2019. Machine learning strategies to perform facies classification. Presented at *CSEG GeoConvention*.

Hall, B., 2016. Facies classification using machine learning. *The Leading Edge*, 35(10), 906-909. doi:10.1190/tle35100906.1.

Hall, M., and B. Hall, 2017. Distributed Collaborative Prediction: Results of the Machine Learning Contest. *The Leading Edge* 36 (3), 267–269. doi:10.1190/tle36030267.1.

Hong, Y., S. Wang, J. Bae, J. Yoo and S. Yoon, 2020. Automated facies identification using unsupervised clustering. *In Offshore Technology Conference*, May 2020. OnePetro.

Imamverdiyev Y. and L. Sukhostat, 2019. Lithological facies classification using deep convolutional neural network. *Journal of Petroleum Science and Engineering* 174, pp. 216 – 228.

James G, D.Witten, T. Hastie, and R. Tibshirani, 2013. An introduction to statistical learning, vol 112. Springer, New York.

Kanevski, M., A. Pozdnoukhov, V. and Timonin. 2009. Machine Learning model for Spatial na.

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, 2017, LightGBM: A Highly Efficient Gradient Boosting Decision Tree: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, v. 30, p. 3146–3154.

Kim, Y., R. Hardisty, E. Torres, and K. Marfurt, 2018. Seismic-facies Classification Using Random Forest Algorithm, *SEG Technical Program Expanded Abstracts*: 2161-2165

Kotsiantis, S.B., 2007. Supervised machine learning: a review of classification techniques. *Informatica*, vol. 31, pp. 249-268.

Leyland, L., 2017. The importance and applications of well logging [viewed 27 August, 2022]. Available from: https://medium.com/@welllogging/the-importance-and-applications-of-well-logging-3fe97fd1fa0e

Li, P., and Y. Zhang, 2016. Facies Characterization of a Reservoir in the North Sea Using Machine Learning Techniques, *Class Project Final Report for CS229, Stanford University*.

Li, Yumei and Anderson-Sprecher, Richard, 2006. Facies identification from well logs: A comparison of discriminant analysis and Naïve Bayes classifier". *Journal of Petroleum Science and Engineering*. 53. 149-157. 10.1016/j.petrol.2006.06.001.

Lin, J., H. Li, N. Liu, J. Gao, and Z. Li, 2020. Automatic lithology identification by applying LSTM to logging data: A case study inX tight rock reservoirs," *IEEE Geoscience Remote Sensing Letter*, early access, Jun. 19, 2020, doi: 10.1109/LGRS.2020.3001282

Lindberg, D. V., E. Rimstad, and H. More, 2015. Inversion of well logs into facies accounting for spatial dependencies and convolution effects. *Journal of Petroleum Science Engineering*. vol. 134, pp. 237–246, Oct. 2015.

Liu, X., J. Li, X. Chen, L. Zhou, and K. Guo, 2017. Bayesian discriminant analysis of lithofacies integrate the fisher transformation and the kernel function estimation: Interpretation, 5, no. 2. SE1-SE10.

Liu, X., X. Chen, J. Li, X. Zhou, and Y. Chen, 2020. Facies identification based on multikernel relevance vector machine. *IEEE Transactions on Geoscience and Remote Sensing*. https://doi.org/10.1109/TGRS.2020.2981687.

Lopez, D., J. Lorenzo, X. Zhou, 2020. Soil type data analytics prediction using electrical resistivity and S-wave velocities for shallow (<20 m) unconsolidated sediments." *Paper presented at the SEG International Exposition and Annual Meeting, Virtual, October 2020*. doi: https://doi.org/10.1190/segam2020-3428165.1

Masapanta, J., 2021. Machine and Deep Learning for Lithofacies Classification from Well Logs in the North Sea (Master's thesis, uis).

Merembayev, T., D. Kurmangaliyev, B. Bekbauov, and Y. Amanbek, 2021. A Comparison of machine learning algorithms in predicting lithofacies: Case studies from Norway and Kazakhstan. *Energies*, 14(7), p.1896.

Mohamed, I.M., S. Mohamed, I. Mazher, and P. Chester, 2019. September. Formation lithology classification: Insights into machine learning methods. In *SPE Annual Technical Conference and Exhibition*. OnePetro.

NPD, 2015. CO2 atlas for the Norwegian Continental Shelf: [viewed 10 july 2022]. Available from: https://www.npd.no/en/facts/publications/co2-atlases/co2-atlas-for-the-norwegian-continental-shelf/

Primmer, T.J., C.A Cade, I. Evans, J. Gluyas, M.S Hopkins, N.H. Oxtoby, P. Smalley, E.A. Warren, R. Worden, 1997. Global patterns in sandstone diagenesis: Their application to reservoir quality prediction for petroleum exploration. In: Reservoir quality prediction in sandstones and carbonates (eds. Kupecz, J.A., Gluyas, J. and Bloch, S.) *AAPG Memoir*. 69. 61-78.

Qi, L. and T. R. Carr. 2006. "Neural network prediction of carbonate lithofacies from well logs, Big Bow and Sand Arroyo Creek fields, Southwest Kansas". *Computational Geoscience*, vol. 32, no. 7, pp. 947–964, Aug. 2006.

Qu, S., Z. Guan, E. Verschur, and Y. Chen, 2019. Automatic high-resolution micro-seismic event detection via supervised machine learning: *Geophysical Journal International*, 218, no. 3, 2106-2121.

Rider, M. 1991. The geological interpretation of well logs.

Salvador, A. and M. A Murphy, 1998. International stratigraphic guide — an abridged version. *Episodes.*, 22 (4), 255.

Santos, D. T. d., M. Roisenberg and M. d. S. Nascimento, 2021. Deep Recurrent Neural Networks Approach to Sedimentary Facies Classification Using Well Logs. in *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1-5, 2022, Art no. 3001405, doi: 10.1109/LGRS.2021.3053383.

Schlumberger, C., M. Schlumberger, and E. Leonardon, 1934. A New Contribution to Subsurface Studies by Means of Electrical Measurements in Drill Holes. 103: 73-288. http://dx.doi.org/doi:10.2118/934273-G.

Sebtosheikh, M. A., R. Motafakkerfard, M. A. Riahi, S. Moradi, and N. Sabety. 2015. "Support vector machine method, a new technique for lithology prediction in an Iranian heterogeneous carbonate reservoir using petrophysical well logs".

Carbonates and Evaporites, v. 30, no. 1, p. 59–68, doi:10.1007/s13146-014-0199-0.

Sebtosheikh, M.A., A. Salehi, 2015. Lithology prediction by support vector classifiers using inverted seismic attributes data and petrophysical logs as a new approach and investigation of training data set size effect on its performance in a heterogeneous carbonate reservoir. *Journal of Petroleum Science and Engineering*. 134, 143–149.

Silva, A. A., I. A. Lima Neto, R. M. Misságia, M. A. Ceia, A. G. Carrasquilla, and N. L. Archilha, 2015. Artificial neural networks to support petrographic classification of carbonate-siliciclastic rocks using well logs and textural information. *Journal Applied Geophysics*, vol. 117, pp. 118–125, Jun. 2015.

Singha, D.K., N. Rai, Madhvi, M. Maurya, U. Shankar, R. Chatterjee. 2021. Interpretation of Self-Potential (SP) Log and Depositional Environment in the Upper Assam Basin, India. In: Biswas, A. (eds) Self-Potential Method: Theoretical Modeling and Applications in Geosciences. *Springer Geophysics*. Springer, Cham. https://doi.org/10.1007/978-3-030-79333-3_10

Sivia, D.S., 1996. *Data Analysis: A Bayesian Tutorial*, Oxford University Press, New York, p. 189.

Søland, P. and M.V. Thue, 2019. Machine learning for automated stratigraphy classification: an empirical study to label subsurface formations in the Johan Sverdrup field (Master's thesis).

Subasi, A., 2020. Practical Machine Learning for Data Analysis Using Python. Academic Press.

Tiab, D. and E.C,Donaldson, 1996. Theory and practice of measuring reservoir rock and fluid transport properties. Student Edition.

Tran, 2016. Precision, Recall, Sensitivity and Specificity, [viewed 4 September 2022]. Available from: https://newbiettn.github.io/2016/08/30/precision-recall-sensitivity-specificity/.

Tschannen, V., M. Delescluse, M. Rodriguez, and J. Keuper. 2017. “Facies classification from well logs using an inception convolutional network,” 2017, arXiv:1706.00613. [Online]. Available: http://arxiv.org/abs/1706.00613.

Vakarelov, B. (2016). The "art" of well log correlation: practical tips and other musings. [viewed 27 August, 2022]. Available from: https://www.linkedin.com/pulse/art-well-log-correlation-practical-tips-other-musings-boyan-vakarelov/

Wang, G. and T. Carr, 2012. Marcellus shale lithofacies prediction by multiclass neural network classification in the Appalachian Basin, Mathematics. *Geoscience*, vol. 44, pp. 975–1004, Nov. 2012.

Wang, L. and C. A. Alexander, 2015. Big data in the design and manufacturing engineering” American Journal of Economics and Business Administration, v. 7(2), pp. 60-67.

Witten, I., and E. Frank, 2005. Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn, *Morgan Kaufmann Series in Data Management Systems,Elsevier/Morgan Kaufman*, San Francisco, California, p. 525.

Xie, Y., C. Zhu, W. Zhou, Z. Li, X. Liu, M. Tu, 2018. Evaluation of machine learning methods for formation lithology identification: A comparison of tuning processes and model performances, *Journal of Petroleum Science and Engineering*, Volume 160, Pages 182- 193, https://doi.org/10.1016/j.petrol.2017.10.028.

Xiong, W., Z. H. Wan, M. S. Chen, and H. Y. Zhang, 2010. Semi-automatic determination of the number of seismic facies in waveform classification. *in Procedure of 72nd European Association of Geoscientists & Engineers Conference Exhibition Incorporating SPE EUROPEC*, pp. 3829–3833.

Xiong, W., Z. Wan, M. Chen, and H. Zhang, 2010. Semi-automatic determination of the number of seismic facies in waveform classification: *72nd Annual International Meeting, EAGE, Expanded Abstracts*, 3829.

Yiying, W. and Z. Yeze, 2019. Cryptocurrency Price Analysis with Artificial Intelligence, *2019 5th International Conference on Information Management* (ICIM), pp. 97-101, doi: 10.1109/INFOMAN.2019.8714700.

Youn, E. and M. K. Jeong, 2009. Class dependent feature scaling method using naive Bayes classifier for text datamining. *Pattern Recognition Letters.* **30***: 477–485.* doi:10.1016/j.patrec.2008.11.013.

Zhang, G., Z. Wang, and Y. Chen, 2018. Deep learning for seismic lithology prediction. *Geophysical Journal International*, pp. 1368–1387, Aug. 2018.

Zhang, Y., H. A. Salisch, and J. G. McPherson, 1999. Application of neural networks to identify lithofacies from well logs. *Exploration Geophysics*, v. 30, no. 2, p. 45–49, doi:10.1071/eg999045.

Zhou, X., 2020. Data-Driven Modeling and Prediction for Reservoir Characterization and Simulation Using Seismic and Petrophysical Data Analyses. *Louisiana State University and Agricultural & Mechanical College*.

Zu, S., H. Zhou, R. Wu, W. Mao, and Y. Chen, 2018. Hybrid-sparsity constrained dictionary learning for iterative deblending of extremely noisy simultaneous-source data: *IEEE Transaction on Geosciences and Remote Sensing*, 57, no. 4, 2249.

APPENDIXES

Appendix A: Data loading and exploration

In [1]:

```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
from tqdm import tqdm
import time
```

In [2]:

```
train = pd.read_csv("CSV_train.csv",low_memory=False,delimiter=';')
test=pd.read_csv("CSV_test.csv",low_memory=False,delimiter=',')
hidden=pd.read_csv("CSV_hidden_test.csv",low_memory=False,delimiter=',')
```

In [3]:

```
hidden
```

Out[3]:

	WELL	DEPTH_MD	X_LOC	Y_LOC	Z_LOC	GROUP	FORMATION
0	15/9-23	1518.2800	433906.7500	6460000.5	-1493.241821	HORDALAND GP.	Skade Fm.
1	15/9-23	1518.4320	433906.7500	6460000.5	-1493.393799	HORDALAND GP.	Skade Fm.
2	15/9-23	1518.5840	433906.7500	6460000.5	-1493.545776	HORDALAND GP.	Skade Fm.
3	15/9-23	1518.7360	433906.7500	6460000.5	-1493.697754	HORDALAND GP.	Skade Fm.
4	15/9-23	1518.8880	433906.7500	6460000.5	-1493.849609	HORDALAND GP.	Skade Fm.
...	...	...	...	...	...	...	...
122392	35/9-7	2973.2988	536096.0625	6793022.0	-2943.444580	BAAT GP.	Etive Fm.
122393	35/9-7	2973.4508	536096.0625	6793022.0	-2943.595947	BAAT GP.	Etive Fm.
122394	35/9-7	2973.6028	536096.0625	6793022.0	-2943.747559	BAAT GP.	Etive Fm.
122395	35/9-7	2973.7548	536096.0625	6793022.0	-2943.899170	BAAT GP.	Etive Fm.
122396	35/9-7	2973.9068	536096.0625	6793022.0	-2944.050537	BAAT GP.	Etive Fm.

122397 rows × 29 columns

In [4]:

```
Alldata=pd.concat([train, test], axis=0).reset_index()
Alldata.loc[train.index, 'Dataset'] = 'Train'
Alldata.loc[train.index.max()+1:, 'Dataset'] = 'Test'
```

In [5]:

```
Alldata
```

Out[5]:

	index	WELL	DEPTH_MD	X_LOC	Y_LOC	Z_LOC	GROUP	FORI
0	0	15/9-13	494.5280	437641.96875	6470972.5	-469.501831	NORDLAND GP.	
1	1	15/9-13	494.6800	437641.96875	6470972.5	-469.653809	NORDLAND GP.	
2	2	15/9-13	494.8320	437641.96875	6470972.5	-469.805786	NORDLAND GP.	
3	3	15/9-13	494.9840	437641.96875	6470972.5	-469.957794	NORDLAND GP.	
4	4	15/9-13	495.1360	437641.96875	6470972.5	-470.109772	NORDLAND GP.	
...	...	...	...	...	...	...	...	
1307292	136781	35/9-8	3224.3896	536225.93750	6794880.5	-3199.876465	BAAT GP.	F
1307293	136782	35/9-8	3224.5416	536225.93750	6794880.5	-3200.028320	BAAT GP.	F
1307294	136783	35/9-8	3224.6936	536225.93750	6794880.5	-3200.180176	BAAT GP.	F
1307295	136784	35/9-8	3224.8456	536225.93750	6794880.5	-3200.332031	BAAT GP.	F
1307296	136785	35/9-8	3224.9976	536225.93750	6794880.5	-3200.483887	BAAT GP.	F

1307297 rows × 31 columns

In [6]:

```
train_wells = Alldata['WELL'][Alldata.Dataset=='Train'].unique()
print('No of train wells: %s' % len(train_wells))
```

```
No of train wells: 98
```

In [7]:

```
test_wells = Alldata['WELL'][Alldata.Dataset=='Test'].unique()
print('No of test wells: %s' % len(test_wells))
```

```
No of test wells: 10
```

In [8]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1170511 entries, 0 to 1170510
Data columns (total 29 columns):
 #   Column                              Non-Null Count    Dtype
---  ------                              --------------    -----
 0   WELL                                1170511 non-null  object
 1   DEPTH_MD                            1170511 non-null  float64
 2   X_LOC                               1159736 non-null  float64
 3   Y_LOC                               1159736 non-null  float64
 4   Z_LOC                               1159736 non-null  float64
 5   GROUP                               1169233 non-null  object
 6   FORMATION                           1033517 non-null  object
 7   CALI                                1082634 non-null  float64
 8   RSHA                                630650 non-null   float64
 9   RMED                                1131518 non-null  float64
 10  RDEP                                1159496 non-null  float64
 11  RHOB                                1009242 non-null  float64
 12  GR                                  1170511 non-null  float64
 13  SGR                                 69353 non-null    float64
 14  NPHI                                765409 non-null   float64
 15  PEF                                 671692 non-null   float64
 16  DTC                                 1089648 non-null  float64
 17  SP                                  864247 non-null   float64
 18  BS                                  682657 non-null   float64
 19  ROP                                 535071 non-null   float64
 20  DTS                                 174613 non-null   float64
 21  DCAL                                298833 non-null   float64
 22  DRHO                                987857 non-null   float64
 23  MUDWEIGHT                           316151 non-null   float64
 24  RMIC                                176160 non-null   float64
 25  ROPA                                192325 non-null   float64
 26  RXO                                 327427 non-null   float64
 27  FORCE_2020_LITHOFACIES_LITHOLOGY    1170511 non-null  int64
 28  FORCE_2020_LITHOFACIES_CONFIDENCE   1170332 non-null  float64
dtypes: float64(25), int64(1), object(3)
memory usage: 259.0+ MB
```

In [10]:

```
#percentage missing values
empty_data = train[train.isna().any(axis=1)]
per_missing = train.isnull().sum()/len(train)*100
print(per_missing)
```

```
WELL                                  0.000000
DEPTH_MD                              0.000000
X_LOC                                 0.920538
Y_LOC                                 0.920538
Z_LOC                                 0.920538
GROUP                                 0.109183
FORMATION                            11.703777
CALI                                  7.507576
RSHA                                 46.121822
RMED                                  3.331280
RDEP                                  0.941042
RHOB                                 13.777658
GR                                    0.000000
SGR                                  94.074981
NPHI                                 34.608987
PEF                                  42.615490
DTC                                   6.908350
SP                                   26.164983
BS                                   41.678720
ROP                                  54.287401
DTS                                  85.082327
DCAL                                 74.469868
DRHO                                 15.604638
MUDWEIGHT                            72.990344
RMIC                                 84.950163
ROPA                                 83.569142
RXO                                  72.027004
FORCE_2020_LITHOFACIES_LITHOLOGY      0.000000
FORCE_2020_LITHOFACIES_CONFIDENCE     0.015292
dtype: float64
```

In [11]:

```
# statistics of the data
train.describe()
```

Out[11]:

	DEPTH_MD	X_LOC	Y_LOC	Z_LOC	CALI	RSH
count	1.170511e+06	1.159736e+06	1.159736e+06	1.159736e+06	1.082634e+06	630650.0000
mean	2.184087e+03	4.856310e+05	6.681276e+06	-2.138527e+03	1.318568e+01	10.6946
std	9.971821e+02	3.455641e+04	1.281524e+05	9.709426e+02	3.798907e+00	100.6425
min	1.360860e+02	4.268988e+05	6.406641e+06	-5.395563e+03	2.344000e+00	0.0001
25%	1.418597e+03	4.547996e+05	6.591327e+06	-2.811502e+03	9.429712e+00	0.8541
50%	2.076605e+03	4.769203e+05	6.737311e+06	-2.042785e+03	1.255575e+01	1.3990
75%	2.864393e+03	5.201532e+05	6.784886e+06	-1.391866e+03	1.671075e+01	3.0993
max	5.436632e+03	5.726328e+05	6.856661e+06	-1.110860e+02	2.827900e+01	2193.9045

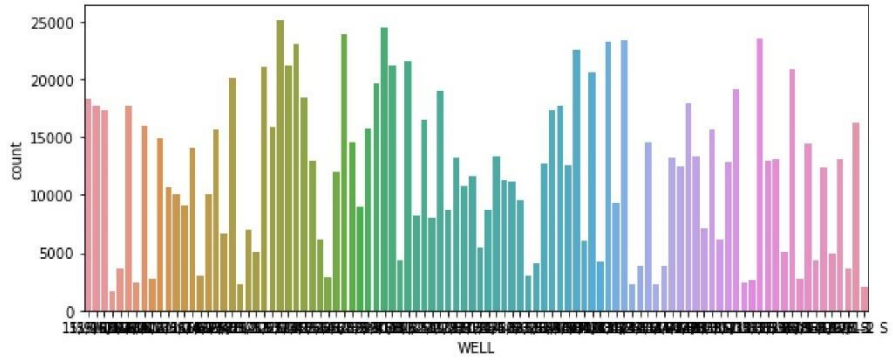
8 rows × 26 columns

In [12]:

```
def understand_dist(dataset,feature_type):

    if feature_type == "Categorical":

        categorical_features=[feature for feature in dataset.columns if dataset[feature].dt
        dataframes=[]
        for feature in categorical_features:
            dataframe=dataset[feature].value_counts().rename_axis(feature).reset_index(name
            dataframes.append(dataframe)

            plt.figure(figsize=(10,4))
            sns.countplot(x=feature,data = dataset)
            plt.show()
            print(dataframe,'\n')




    elif feature_type == "Numeric":

        numerical_features=[feature for feature in dataset.columns if dataset[feature].dtyp

        for feature in numerical_features:
            plt.figure(figsize=(10,4))
            sns.distplot(dataset[feature])
            plt.show()
            print("Description :\n\n"+str(dataset[feature].describe())+"\n\n")
```

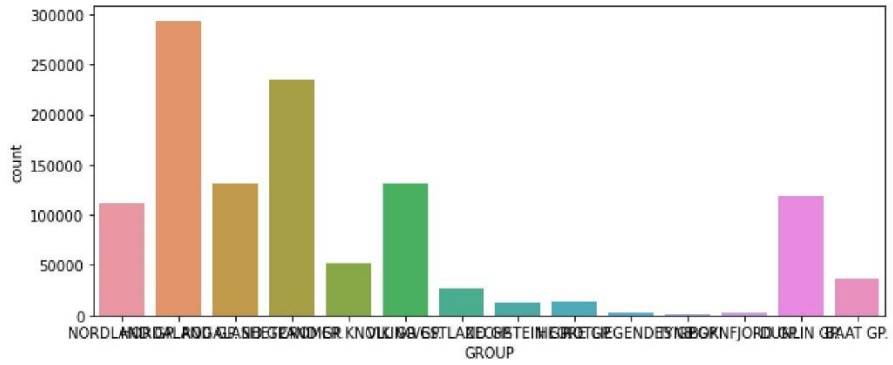
In [13]:

```
import seaborn as sns
understand_dist(train, "Categorical")
```



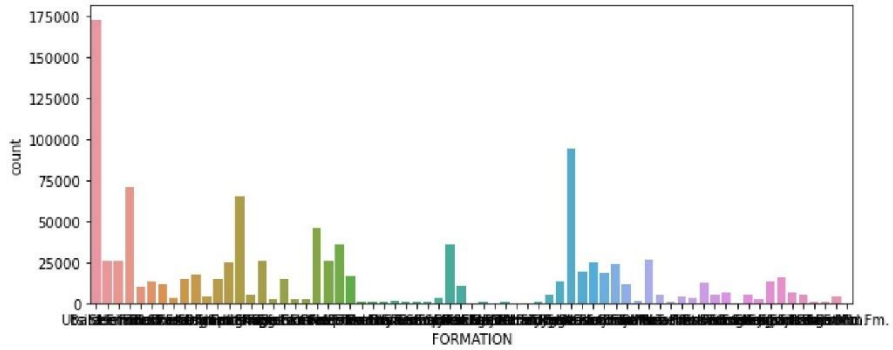
	WELL	counts
0	25/2-7	25131
1	29/6-1	24464
2	25/7-2	23879
3	35/11-6	23544
4	34/2-4	23403
..	...	...
93	25/11-15	2284
94	34/5-1 S	2273
95	34/3-1 A	2251
96	7/1-2 S	2040
97	16/1-2	1734

[98 rows x 2 columns]



	GROUP	counts

0	HORDALAND GP.	293155
1	SHETLAND GP.	234028
2	VIKING GP.	131999
3	ROGALAND GP.	131944
4	DUNLIN GP.	119085
5	NORDLAND GP.	111490
6	CROMER KNOLL GP.	52320
7	BAAT GP.	35823
8	VESTLAND GP.	26116
9	HEGRE GP.	13913
10	ZECHSTEIN GP.	12238
11	BOKNFJORD GP.	3125
12	ROTLIEGENDES GP.	2792
13	TYNE GP.	1205



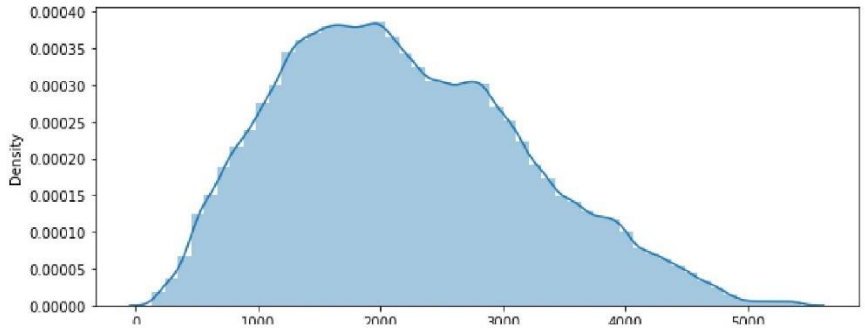
	FORMATION	counts
0	Utsira Fm.	172636
1	Kyrre Fm.	94328
2	Lista Fm.	71080
3	Heather Fm.	65041
4	Skade Fm.	45983
..	...	...
64	Broom Fm.	235
65	Intra Balder Fm. Sst.	177
66	Farsund Fm.	171
67	Flekkefjord Fm.	118
68	Egersund Fm.	105

[69 rows x 2 columns]

In [14]:

```
understand_dist(train, "Numeric")
```

```
C:\Users\Ayori\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```



In [15]:

```
wells = train['WELL'].unique()
len(wells)
```

Out[15]:

```
98
```

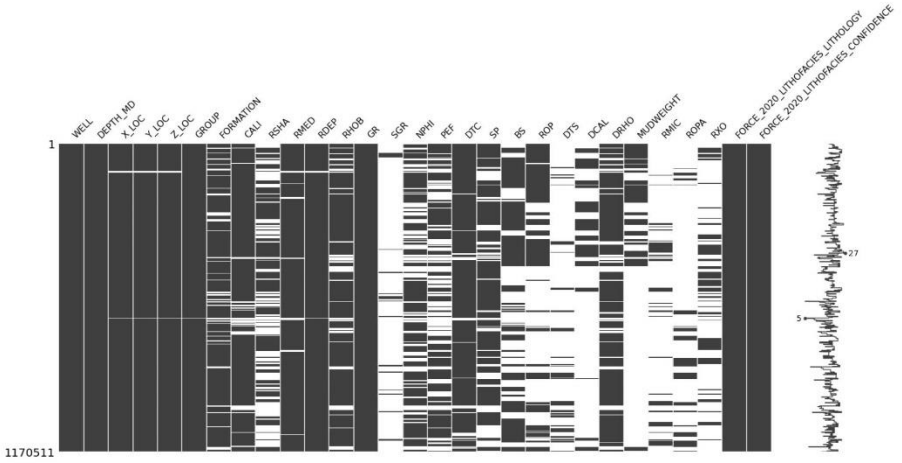
missing values

The data is presented as is and includes a large amount of missing values. Missing data within well logging can arise for a number of reasons including:

Tool failures & problems Missing by choice (i.e. tools not run due to budgetary constraints) Human error Vintage datasets Issues arising from the borehole environment

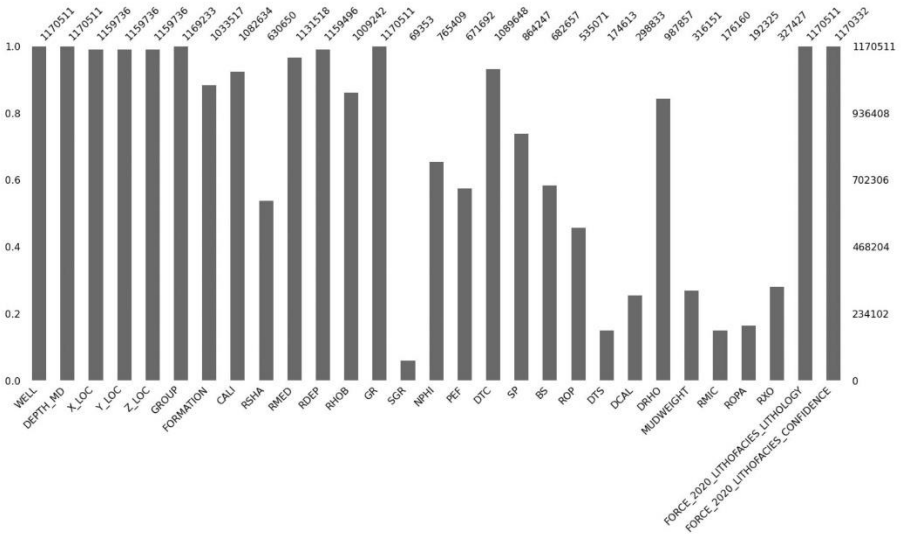
In [16]:

```
import missingno as msno
msno.matrix(train);
```



In [17]:

```
msno.bar(train);
```



In [19]:

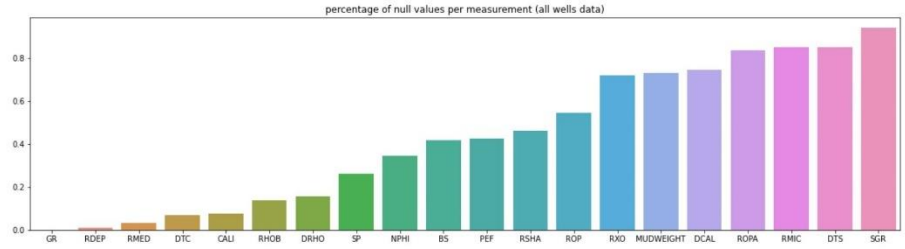
```
measures_list = train.columns.values[7:-2]
nan = train[measures_list].isna().mean().sort_values()
nan
```

Out[19]:

```
GR           0.000000
RDEP         0.009410
RMED         0.033313
DTC          0.069084
CALI         0.075076
RHOB         0.137777
DRHO         0.156046
SP           0.261650
NPHI         0.346090
BS           0.416787
PEF          0.426155
RSHA         0.461218
ROP          0.542874
RXO          0.720270
MUDWEIGHT    0.729903
DCAL         0.744699
ROPA         0.835691
RMIC         0.849502
DTS          0.850823
SGR          0.940750
dtype: float64
```

In [20]:

```
plt.figure(figsize=(20,5))
splot = sns.barplot(x=nan.index,y=nan)
splot.set_title("percentage of null values per measurement (all wells data)")
plt.show()
```



In [21]:

```
nan_values_per_well = train.isna()[measures_list]
nan_values_per_well["WELL"] = train.WELL
nan_values_per_well= nan_values_per_well.groupby("WELL").mean()
##order wells per missing values count overall
missing_data_threshold = 0.6
wells_missing_data = nan_values_per_well.mean(axis=1).sort_values()
wells_missing_data.plot(kind="bar",figsize=(30,5))
plt.axhline(y=missing_data_threshold, color='green', linestyle='--',label="missing percenta
plt.axhline(y=wells_missing_data.mean(), color='blue', linestyle='-',label="mean of missing
plt.title("percentage of missing data per well")
plt.legend()
plt.show()
```

In [22]:

```
wells_missing_data[wells_missing_data>0.5]
```

Out[22]:

```
WELL
34/7-20        0.517994
31/3-3         0.518303
34/7-21        0.519305
31/4-5         0.521012
31/2-7         0.525094
25/9-1         0.530642
34/10-35       0.535666
31/6-8         0.543978
31/3-2         0.547474
34/10-33       0.554904
36/7-3         0.559496
30/3-5 S       0.560942
31/3-1         0.571309
34/2-4         0.573657
16/11-1 ST3    0.594725
29/6-1         0.599107
31/6-5         0.600725
33/9-1         0.622905
35/11-12       0.668109
31/5-4 S       0.817138
dtype: float64
```

In [23]:

```
feature_missing_data_threshold = 0.70

wells_missing_data2 = (nan_values_per_well>feature_missing_data_threshold).sum(axis=1).sort
wells_missing_data2.plot(kind="bar",figsize=(30,5))
plt.axhline(y=nan_values_per_well.shape[1]//2, color='green', linestyle='--',label="half nu
plt.axhline(y=wells_missing_data2.mean(), color='blue', linestyle='-',label="average value
plt.title("number of measurements with more than {}% of missing data per well".format(featu

plt.legend()
plt.show()
```

In [24]:

```
wells_missing_data2[wells_missing_data2>=10]
```

Out[24]:

```
WELL
33/9-1         10
31/6-5         10
31/6-8         10
34/7-20        10
31/4-10        10
31/3-2         10
34/2-4         10
31/2-9         10
34/10-33       10
25/4-5         10
31/3-1         10
30/3-5 S       11
16/11-1 ST3    11
34/10-35       11
36/7-3         11
31/2-7         11
29/6-1         12
35/11-12       14
31/5-4 S       18
dtype: int64
```

Observations GR has no missing value ROPA,RMIC,DTS,SGR have more than 80% of missing values (remove them from data?keep them?) According to the boxplots, There are some features we could put aside because they have a very high level of null values percentage for almost all wells (more than 75% of wells): SGR, DTS, DCAL,RMIC, RXO. MUDWEIGHT also have more than 40% empty values for 75% of wells. outliers (wells)

there are 4 wells with more than 60% of missing data (31/6-5 ,33/9-1,35/11-12,31/5-4 S).

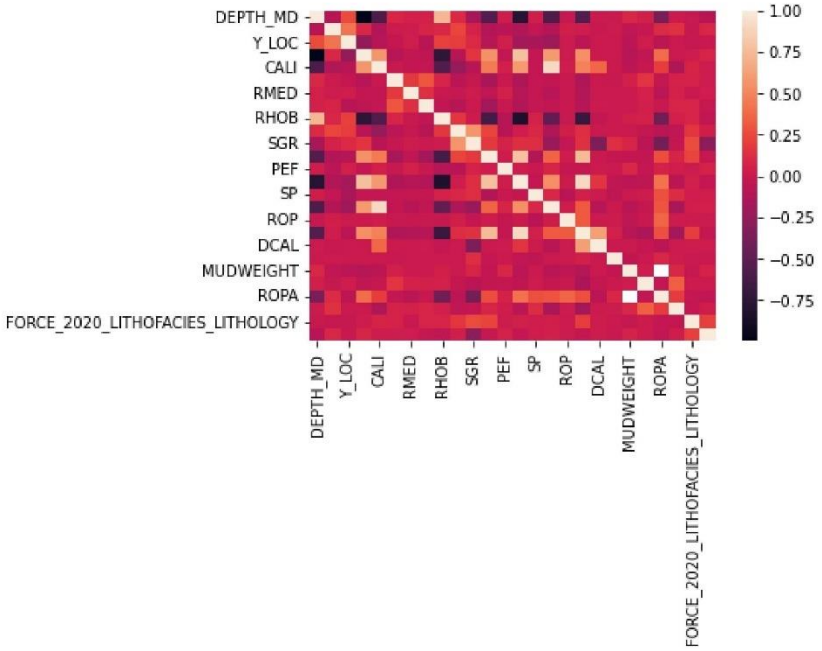
Well 31/5-4 S has 81% of missing data. We can put them aside during modelling.

Depending on the results we can also try to remove other wells with more than 50% of missing values (average missing values percentage is 41%).

There are 4 wells with 10+/20 features which have more than 90% of missing values : (well,number) 34/10-33 10, 34/7-20 10, 30/3-5 S 11, 36/7-3 11 There are 16 wells with 10+/20 features which have more than 75% of missing values : (well,number) 31/6-8 10 31/2-9 10 34/2-4 10 31/3-2 10 31/6-5 10 34/10-33 10 33/9-1 10 34/7-20 10 36/7-3 11 16/11-1 ST3 11 30/3-5 S 11 34/10-35 11 31/2-7 11 29/6-1 12 35/11-12 14 31/5-4 S 18 We can remove those wells or impute missing values during modelisation

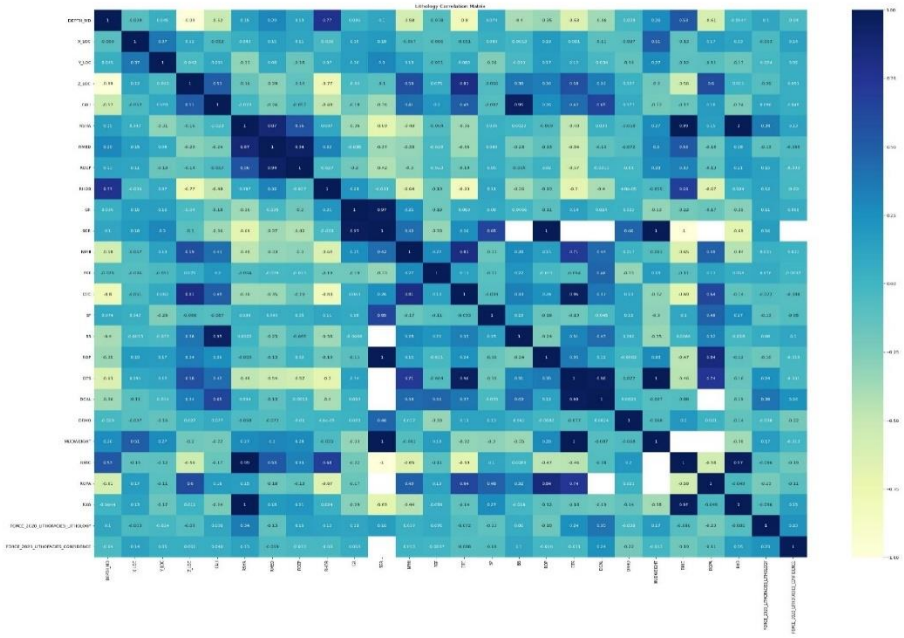
In [25]:

```
corr = train.corr()
round(corr,2)
sns.heatmap(corr);
```



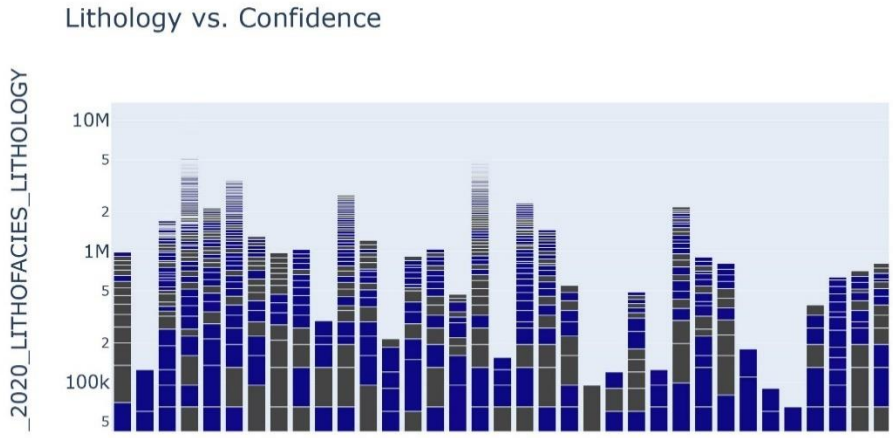
In [26]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(38, 28))
sns.heatmap(train.sample(200).corr(), cmap='YlGnBu', annot=True)
plt.title('Lithology Correlation Matrix');
```



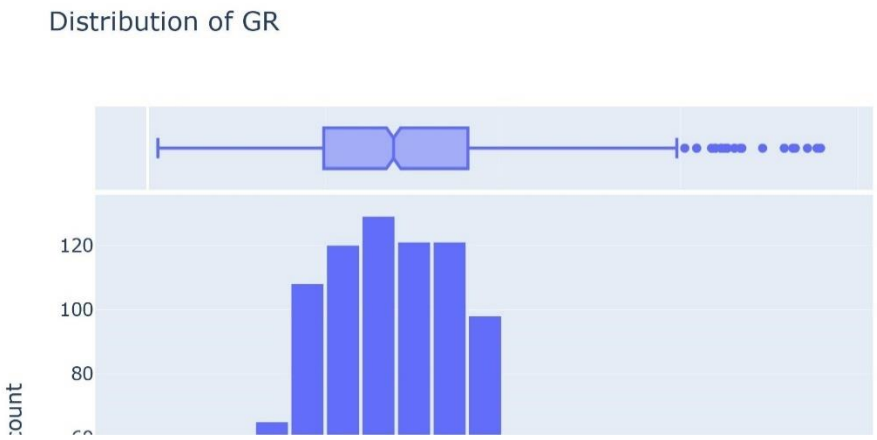
In [27]:

```
import plotly.express as px
px.bar(train.sample(1000),
           title='Lithology vs. Confidence',
           x='FORMATION',
           y='FORCE_2020_LITHOFACIES_LITHOLOGY',
           log_y=True,
#            log_x=True,
           color='RSHA')
```



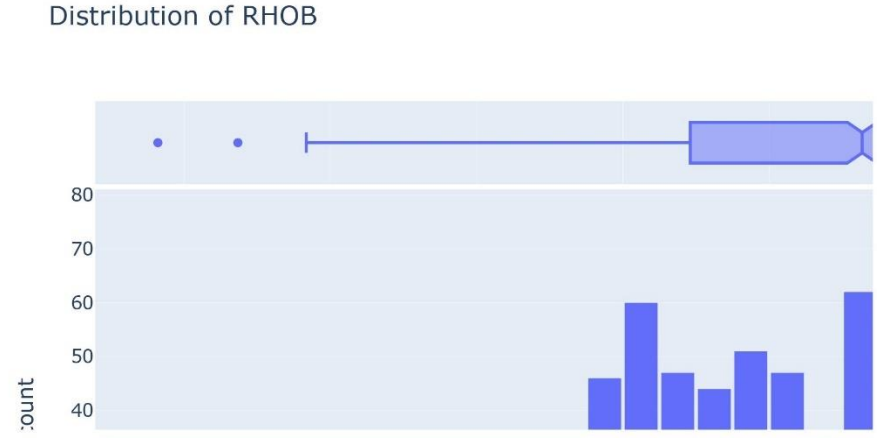
In [29]:

```
fig = px.histogram(train.sample(1000),
                   x='GR',
                   marginal='box',
                   nbins=47,
                   title='Distribution of GR')
fig.update_layout(bargap=0.1)
fig.show()
```



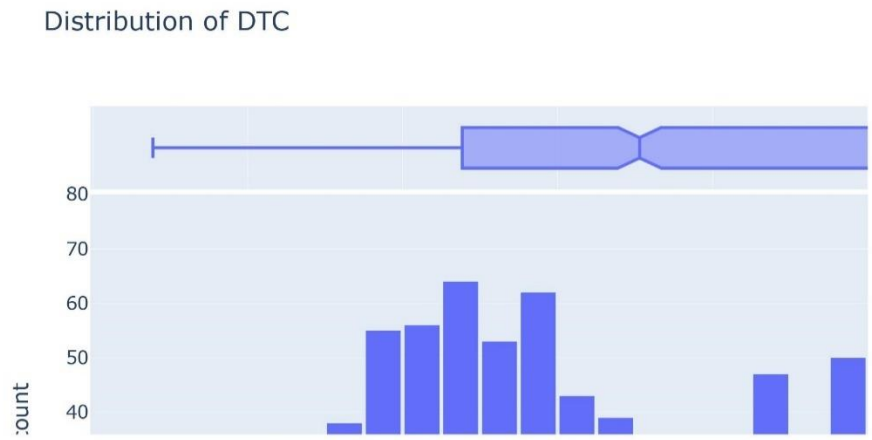
In [30]:

```
fig = px.histogram(train.sample(1000),
                   x='RHOB',
                   marginal='box',
                   nbins=47,
                   title='Distribution of RHOB')
fig.update_layout(bargap=0.1)
fig.show()
```



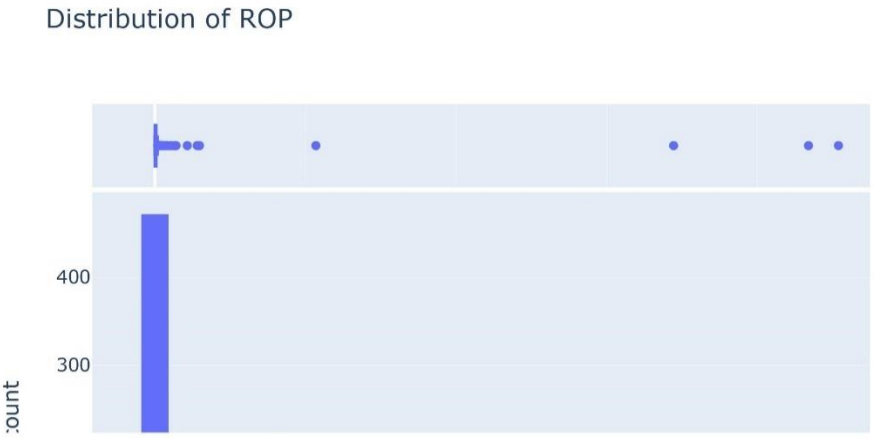
In [31]:

```
fig = px.histogram(train.sample(1000),
                   x='DTC',
                   marginal='box',
                   nbins=47,
                   title='Distribution of DTC')
fig.update_layout(bargap=0.1)
fig.show()
```



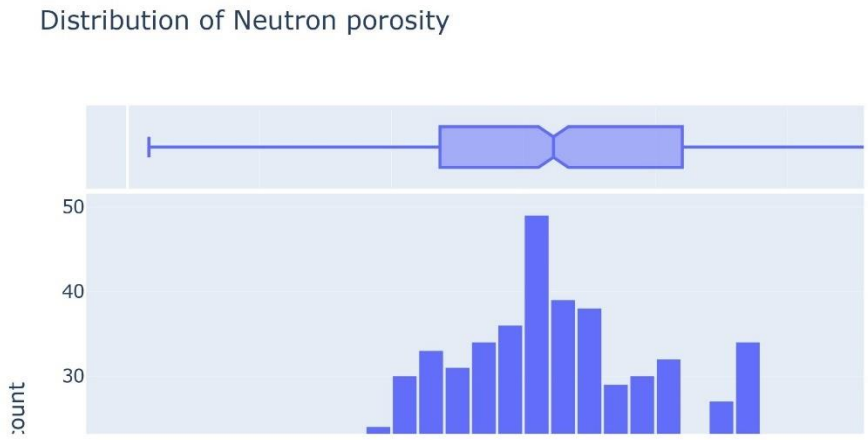
In [32]:

```
fig = px.histogram(train.sample(1000),
                   x='ROP',
                   marginal='box',
                   nbins=47,
                   title='Distribution of ROP')
fig.update_layout(bargap=0.1)
fig.show()
```



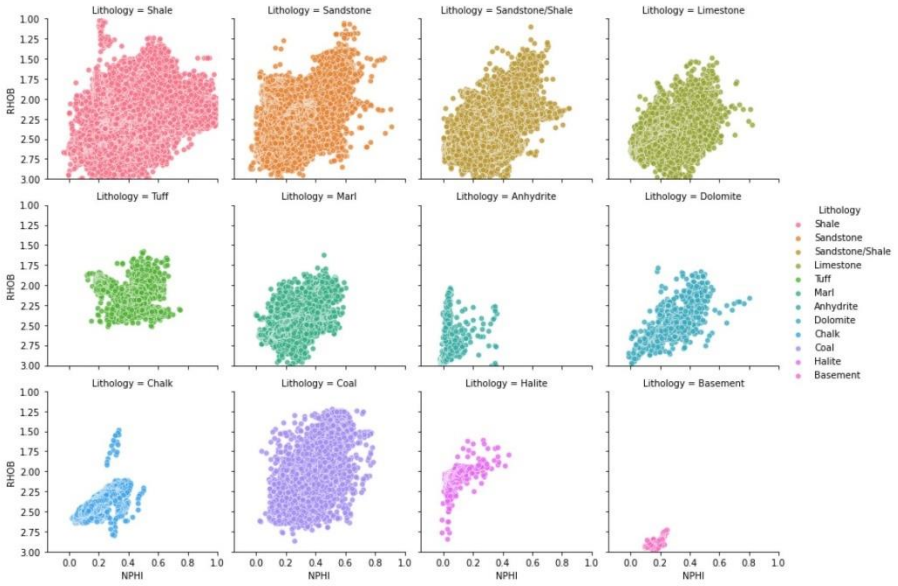
In [33]:

```
fig = px.histogram(train.sample(1000),
                   x='NPHI',
                   marginal='box',
                   nbins=47,
                   title='Distribution of Neutron porosity',
                   )
fig.update_layout(bargap=0.1)
fig.show()
```



In [37]:

```
plot_dn = sns.FacetGrid(train, col='Lithology', col_wrap=4, hue='Lithology')
plot_dn.map(sns.scatterplot, 'NPHI', 'RHOB', alpha=0.8)
plot_dn.set(xlim=(-0.15, 1))
plot_dn.set(ylim=(3, 1))
plot_dn.add_legend();
```

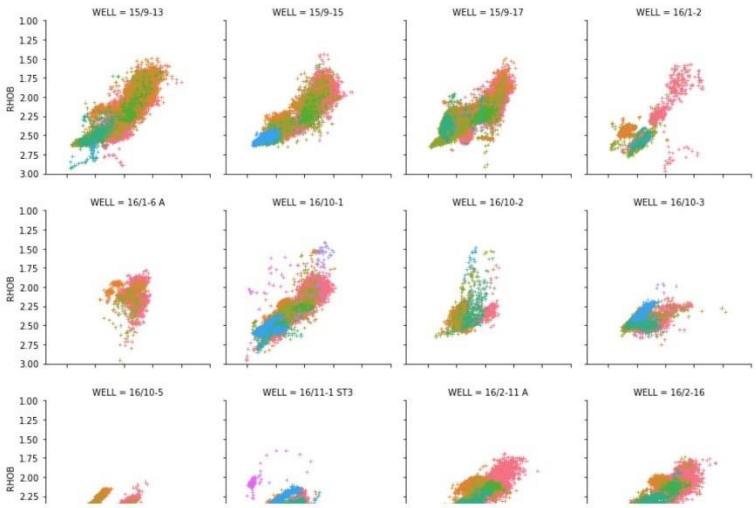


Density-Neutron Distribution by Lithology and Well

We can further enhance the density neutron data by looking at the lithology distribution across multiple wells. Using the converted LITH data column we can create shading for different lithology types by supplying LITH to the hue parameter. We can then supply the WELL column from the dataframe into the col parameter.

In [38]:

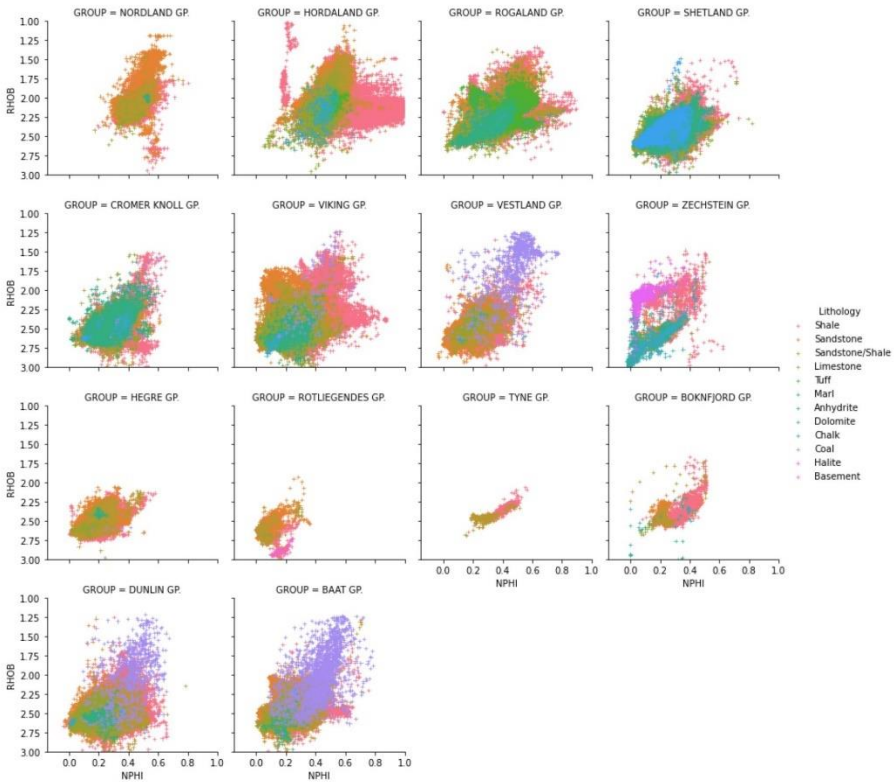
```
plot_dn = sns.FacetGrid(train, col='WELL', col_wrap=4, hue='Lithology')
plot_dn.map(sns.scatterplot, 'NPHI', 'RHOB', alpha=0.8,linewidth=1, size=0.1, marker='+')
plot_dn.set(xlim=(-0.15, 1))
plot_dn.set(ylim=(3, 1))
plot_dn.add_legend();
```



Density-Neutron Distribution by Lithology and Geological Group

In [39]:

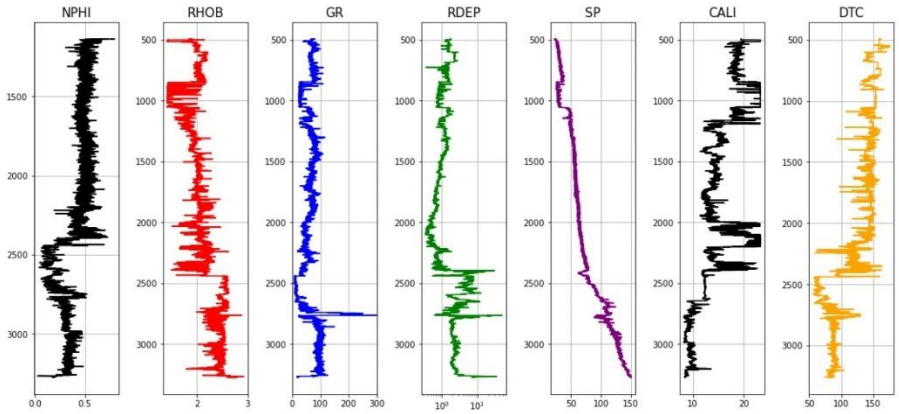
```
plot_dn = sns.FacetGrid(train, col='GROUP', col_wrap=4, hue='Lithology')
plot_dn.map(sns.scatterplot, 'NPHI', 'RHOB', alpha=0.8,linewidth=1, size=0.1, marker='+')
plot_dn.set(xlim=(-0.15, 1))
plot_dn.set(ylim=(3, 1))
plot_dn.add_legend();
```



In [40]:

```
# Display train well data
well = train[train.WELL=='15/9-13']

# define what logs are we going to us
logs = ['NPHI', 'RHOB', 'GR', 'RDEP', 'SP', 'CALI', 'DTC']

# create the subplots; ncols equals the number of logs
fig, ax = plt.subplots(nrows=1, ncols=len(logs), figsize=(15,7))

# Color list
colors = ['black', 'red', 'blue', 'green', 'purple', 'black', 'orange']

for i in range(len(logs)):
  if i == 3:
    # for resistivity, semilog plot
    ax[i].semilogx(well[logs[i]], well['DEPTH_MD'], color=colors[i])
  else:
    # for non-resistivity, normal plot
    ax[i].plot(well[logs[i]], well['DEPTH_MD'], color=colors[i])

  ax[i].set_title(logs[i], size=15)
  ax[i].invert_yaxis()
  ax[i].grid(True)

ax[2].set_xlim(0, 300)
plt.tight_layout(1.1)
plt.show()
```

```
C:\Users\Ayori\AppData\Local\Temp/ipykernel_16448/2563653784.py:26: Matplotl
ibDeprecationWarning:

Passing the pad parameter of tight_layout() positionally is deprecated since
Matplotlib 3.3; the parameter will become keyword-only two minor releases la
ter.
```



Identifying Bad Hole Data

To visualise where we have badhole data caused by borehole enlargement. Deterioration of the borehole wall can happen for a number of different reasons including undercompacted rocks and variations in stresses acting on those rocks such as the mud weight.

In [41]:

```
train_BS = train.dropna(subset=['BS', 'CALI']).copy()
train_BS.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 657745 entries, 18350 to 1170510
Data columns (total 30 columns):
 #   Column                              Non-Null Count   Dtype  
---  ------                              --------------   -----  
 0   WELL                                657745 non-null  object 
 1   DEPTH_MD                            657745 non-null  float64
 2   X_LOC                               649822 non-null  float64
 3   Y_LOC                               649822 non-null  float64
 4   Z_LOC                               649822 non-null  float64
 5   GROUP                               657745 non-null  object 
 6   FORMATION                           600992 non-null  object 
 7   CALI                                657745 non-null  float64
 8   RSHA                                375101 non-null  float64
 9   RMED                                641156 non-null  float64
 10  RDEP                                649659 non-null  float64
 11  RHOB                                583803 non-null  float64
 12  GR                                  657745 non-null  float64
 13  SGR                                 19899 non-null   float64
 14  NPHI                                475420 non-null  float64
 15  PEF                                 412870 non-null  float64
 16  DTC                                 614148 non-null  float64
 17  SP                                  429251 non-null  float64
 18  BS                                  657745 non-null  float64
 19  ROP                                 404358 non-null  float64
 20  DTS                                 158498 non-null  float64
 21  DCAL                                298696 non-null  float64
 22  DRHO                                571527 non-null  float64
 23  MUDWEIGHT                           214471 non-null  float64
 24  RMIC                                153746 non-null  float64
 25  ROPA                                140222 non-null  float64
 26  RXO                                 161936 non-null  float64
 27  FORCE_2020_LITHOFACIES_LITHOLOGY    657745 non-null  int64  
 28  FORCE_2020_LITHOFACIES_CONFIDENCE   657629 non-null  float64
 29  Lithology                           657745 non-null  object 
dtypes: float64(25), int64(1), object(4)
memory usage: 155.6+ MB
```

In [42]:

```
train_BS['BS'].isna().sum()
```

Out[42]:

```
0
```

In [43]:

```
train_BS['CALI'].isna().sum()
```

Out[43]:

```
0
```

In [44]:

```
numer_of_BS_wells = train_BS['WELL'].nunique()
print(f"We have {numer_of_BS_wells} wells that have bit size log")
```

```
We have 68 wells that have bit size log
```

In [45]:

```
wells_BS = train_BS['WELL'].unique()
wells_BS
```

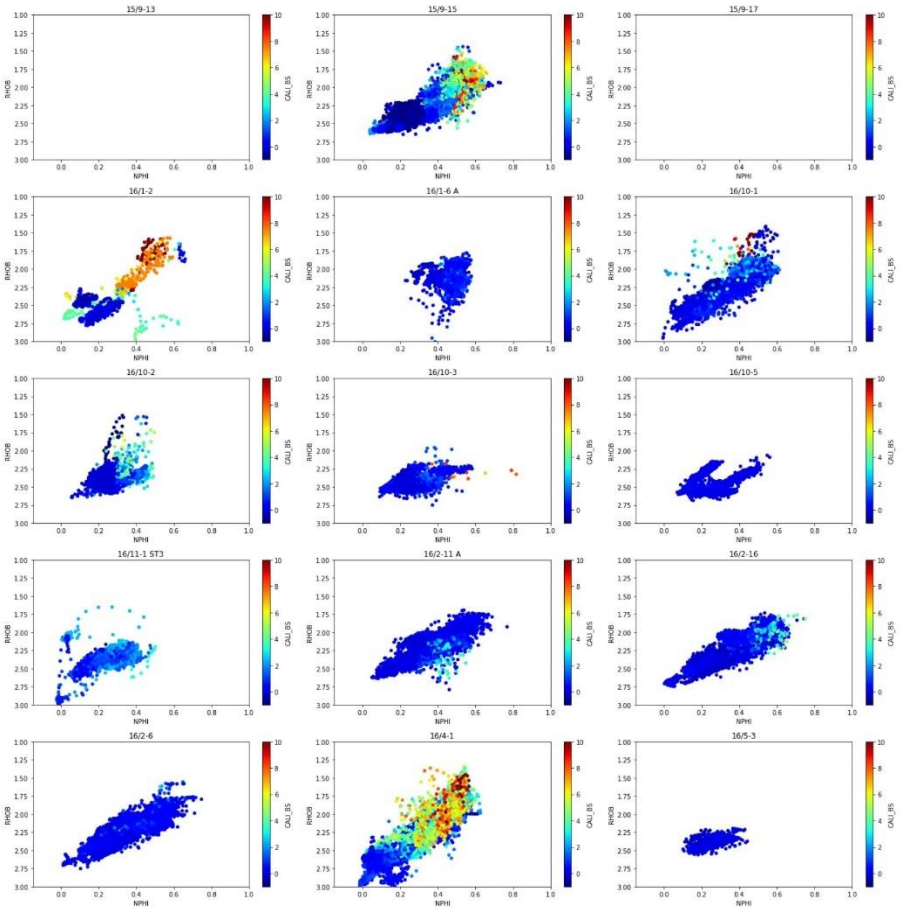
Out[45]:

```
array(['15/9-15', '16/1-2', '16/1-6 A', '16/10-1', '16/10-2', '16/10-3',
       '16/10-5', '16/11-1 ST3', '16/2-11 A', '16/2-16', '16/2-6',
       '16/4-1', '16/5-3', '16/7-4', '25/11-15', '25/11-5', '25/2-13 T4',
       '25/2-14', '25/2-7', '25/3-1', '25/4-5', '25/5-4', '25/6-1',
       '25/6-2', '25/6-3', '25/7-2', '25/8-5 S', '25/8-7', '25/9-1',
       '26/4-1', '31/2-1', '31/2-19 S', '31/3-1', '31/3-2', '31/3-3',
       '31/3-4', '31/6-5', '31/6-8', '32/2-1', '33/5-2', '33/6-3 S',
       '33/9-1', '33/9-17', '34/10-33', '34/11-1', '34/12-1', '34/3-1 A',
       '34/4-10 R', '34/5-1 A', '34/5-1 S', '34/7-13', '34/8-1',
       '34/8-7 R', '35/11-10', '35/11-11', '35/11-12', '35/11-13',
       '35/11-15 S', '35/11-7', '35/12-1', '35/3-7 S', '35/4-1',
       '35/8-6 S', '35/9-10 S', '35/9-2', '35/9-5', '7/1-1', '7/1-2 S'],
      dtype=object)
```

We need to calculate the difference between the bit size and the calliper logs to analyse the well conditions. If the result is a negative value, this indicates that the borehole has shrunken in size resulting from shale swelling. On the other hand, a positive value indicates that the borehole has increased in size due to caving resulting from a fragile formation.

In [46]:

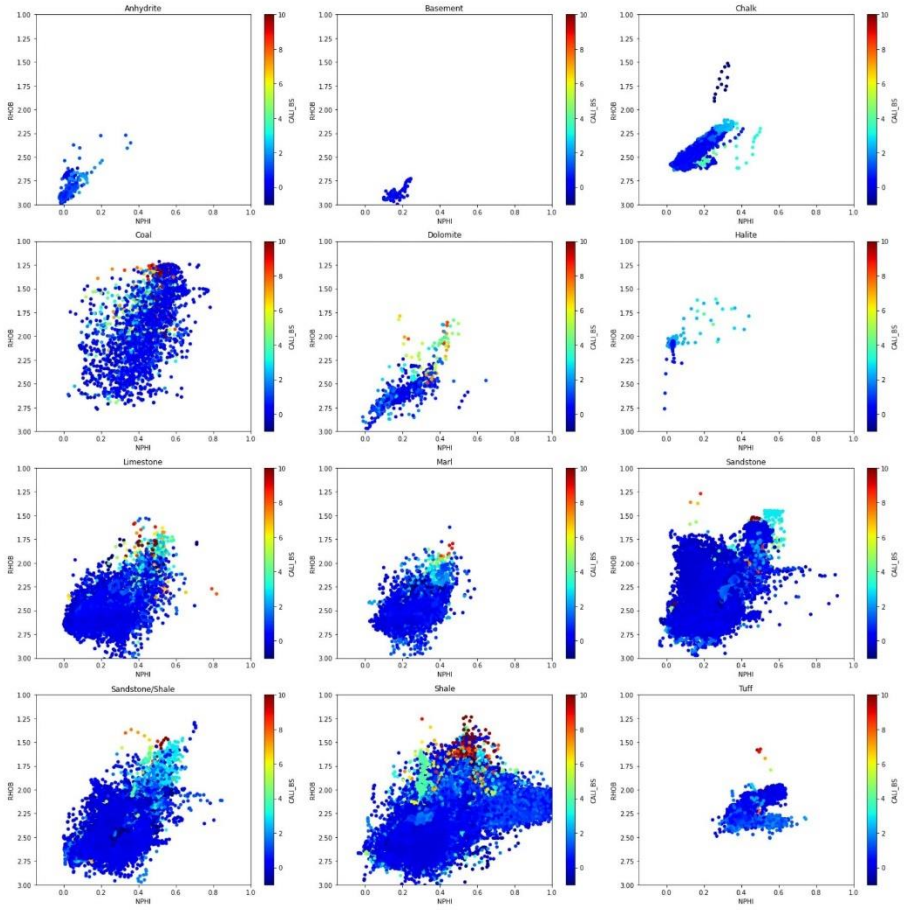
```
train['CALI_BS'] = train['CALI'] - train['BS']
well_group = train.groupby('WELL')
fig, axs = plt.subplots(5, 3, figsize=(20,20))
for (name, df), ax in zip(well_group, axs.flat):
    df.plot(kind='scatter', x='NPHI', y='RHOB', ax=ax, c='CALI_BS', cmap='jet', vmin=-1, vm
    ax.set_xlim(-0.15,1)
    ax.set_ylim(3,1)
    ax.set_title(name)
plt.tight_layout()
```



well 15/9–15 and 16/1–2 and 16/4-1 have badhole data

In [47]:

```
well_group_lith = train.groupby('Lithology')
fig, axs = plt.subplots(4, 3, figsize=(20,20))
for (name, df), ax in zip(well_group_lith, axs.flat):
    df.plot(kind='scatter', x='NPHI', y='RHOB', ax=ax, c='CALI_BS', cmap='jet', vmin=-1, vm
    ax.set_xlim(-0.15,1)
    ax.set_ylim(3,1)
    ax.set_title(name)
plt.tight_layout()
```



Appendix B: Data Pre-processing

In [59]:

```
# storing length of datasets
train_len = train.shape[0]
test_len = test.shape[0]
All_data = pd.concat((train,test,hidden)).reset_index(drop=True)
#dropping columns with high missing values
drop_cols = ['SGR', 'ROPA', 'RXO', 'MUDWEIGHT','DCAL','RMIC','FORCE_2020_LITHOFACIES_CONFID
All_data_drop = All_data.drop(drop_cols, axis=1)
# encoding categorical variables
All_data_drop['GROUP_encoded'] = All_data_drop['GROUP'].astype('category')
All_data_drop['GROUP_encoded'] = All_data_drop['GROUP_encoded'].cat.codes

All_data_drop['FORMATION_encoded'] = All_data_drop['FORMATION'].astype('category')
All_data_drop['FORMATION_encoded'] = All_data_drop['FORMATION_encoded'].cat.codes

All_data_drop['WELL_encoded'] = All_data_drop['WELL'].astype('category')
All_data_drop['WELL_encoded'] = All_data_drop['WELL_encoded'].cat.codes

All_data_drop['Lithology_encoded'] = All_data_drop['FORCE_2020_LITHOFACIES_LITHOLOGY'].asty
All_data_drop['Lithology_encoded'] = All_data_drop['Lithology_encoded'].cat.codes
```

In [60]:

```
#dropping categorial features replaces beforehan by encoded features
drop2 = All_data_drop.drop(['GROUP', 'FORMATION','WELL','FORCE_2020_LITHOFACIES_LITHOLOGY',

# splitting dataset into training, test, and hidden sets
train_prep = drop2[:train_len].copy()
test_prep = drop2[train_len:(train_len+test_len)].copy()
hidden_prep = drop2[(train_len+test_len):].copy()
```

In [61]:

```
train_prep1= train_prep.copy()
test_prep1= test_prep.copy()
hidden_prep1= hidden_prep.copy()
```

In [63]:

```
#Inputing missing values by introducing median
from sklearn.impute import SimpleImputer
miss = SimpleImputer(missing_values=np.nan, strategy='median')
miss.fit(train_prep)
train_imp = miss.fit_transform(train_prep)
train_imp=pd.DataFrame(train_imp, columns=['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC', 'CALI', '
       'RHOB', 'GR', 'NPHI', 'PEF', 'DTC', 'SP', 'BS', 'ROP', 'DTS', 'DRHO',
         'GROUP_encoded',
       'FORMATION_encoded', 'WELL_encoded','Lithology_encoded'])
train_imp
```

Out[63]:

	DEPTH_MD	X_LOC	Y_LOC	Z_LOC	CALI	RSHA	RMED	RDEP	RHC
0	494.5280	437641.96875	6470972.5	-469.501831	19.480835	1.39902	1.611410	1.798681	1.8841
1	494.6800	437641.96875	6470972.5	-469.653809	19.468800	1.39902	1.618070	1.795641	1.8897
2	494.8320	437641.96875	6470972.5	-469.805786	19.468800	1.39902	1.626459	1.800733	1.8965
3	494.9840	437641.96875	6470972.5	-469.957794	19.459282	1.39902	1.621594	1.801517	1.8919
4	495.1360	437641.96875	6470972.5	-470.109772	19.453100	1.39902	1.602679	1.795299	1.8800
...	...	...	...	...	...	...	...	...	
1170506	3169.3124	476920.31250	6737311.0	-2042.784973	8.423170	1.39902	1.443584	1.439000	2.5279
1170507	3169.4644	476920.31250	6737311.0	-2042.784973	8.379244	1.39902	1.443584	1.439000	2.5376
1170508	3169.6164	476920.31250	6737311.0	-2042.784973	8.350248	1.39902	1.443584	1.439000	2.4918

In [66]:

```
#Predicting DTS
from sklearn.metrics import max_error
from xgboost import XGBRegressor

model1000 = XGBRegressor()
model1000.fit(X_dts_train, Y_dts_train.values.ravel(), early_stopping_rounds=100, eval_set=

train_pred = model1000.predict(X_dts_train)
val_pred = model1000.predict(X_dts_val)

print('Train error:', max_error(Y_dts_train, train_pred))
print('Validation error:', max_error(Y_dts_val, val_pred))
```

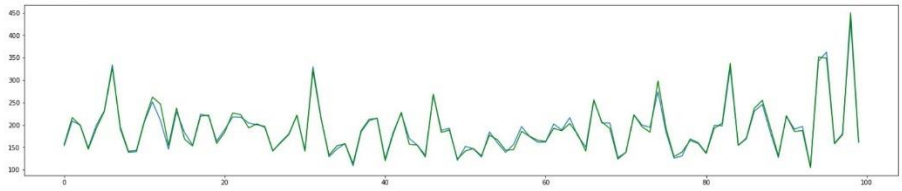
```
C:\Users\Ayori\anaconda3\lib\site-packages\xgboost\compat.py:36: FutureWarni
ng:

pandas.Int64Index is deprecated and will be removed from pandas in a future
version. Use pandas.Index with the appropriate dtype instead.

C:\Users\Ayori\anaconda3\lib\site-packages\xgboost\data.py:250: FutureWarnin
g:

pandas.Int64Index is deprecated and will be removed from pandas in a future
version. Use pandas.Index with the appropriate dtype instead.


[0]     validation_0-rmse:151.68997
[99]    validation_0-rmse:10.83745
Train error: 166.67475890855468
Validation error: 181.6628417940625
```

In [67]:

```
#Visual check on the predictions (Validation Set)
plt.figure(figsize=(25, 5))
plt.plot(list(range(100)), Y_dts_val[25000:25100])
plt.plot(list(range(100)), val_pred[25000:25100], color='g')
```

Out[67]:

```
[<matplotlib.lines.Line2D at 0x1f63ee0d340>]
```



In [68]:

```
# Filling nan values before predicting DTS
X_train_DTS = train_prep.drop(['DTS'], axis=1)
X_train_DTS2 = X_train_DTS.apply(lambda x: x.fillna(x.median()), axis=0)

X_test_DTS = test_prep.drop(['DTS'], axis=1)
X_test_DTS2 = X_test_DTS.apply(lambda x: x.fillna(x.median()), axis=0)

X_hidden_DTS = hidden_prep.drop(['DTS'], axis=1)
X_hidden_DTS2 = X_hidden_DTS.apply(lambda x: x.fillna(x.median()), axis=0)

#Predicting DTS (COMPLETE DATASETS)
train_prep['DTS_pred'] = model1000.predict(X_train_DTS2)
test_prep['DTS_pred'] = model1000.predict(X_test_DTS2)
hidden_prep['DTS_pred'] = model1000.predict(X_hidden_DTS2)

#Inputing nan values in DTS with DTS_PREDICTED
train_prep['DTS_COMB'] = train_prep['DTS']
train_prep['DTS_COMB'].fillna(train_prep['DTS_pred'], inplace=True)

test_prep['DTS_COMB'] = test_prep['DTS']
test_prep['DTS_COMB'].fillna(test_prep['DTS_pred'], inplace=True)

hidden_prep['DTS_COMB'] = hidden_prep['DTS']
hidden_prep['DTS_COMB'].fillna(hidden_prep['DTS_pred'], inplace=True)
```

In [69]:

```
X_dts.columns
```

Out[69]:

```
Index(['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC', 'CALI', 'RSHA', 'RMED', 'RDE
P',
       'RHOB', 'GR', 'NPHI', 'PEF', 'DTC', 'SP', 'BS', 'ROP', 'DRHO',
       'GROUP_encoded', 'FORMATION_encoded', 'WELL_encoded',
       'Lithology_encoded'],
      dtype='object')
```

In [70]:

```
['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC', 'CALI', 'RSHA', 'RMED', 'RDEP',
       'RHOB', 'GR', 'NPHI', 'PEF', 'DTC', 'SP', 'BS', 'ROP', 'DTS', 'DRHO',
       'GROUP_encoded', 'FORMATION_encoded', 'WELL_encoded',
       'Lithology_encoded']
```

Out[70]:

```
['DEPTH_MD',
 'X_LOC',
 'Y_LOC',
 'Z_LOC',
 'CALI',
 'RSHA',
 'RMED',
 'RDEP',
 'RHOB',
 'GR',
 'NPHI',
 'PEF',
 'DTC',
 'SP',
 'BS',
 'ROP',
 'DTS',
 'DRHO',
 'GROUP_encoded',
 'FORMATION_encoded',
 'WELL_encoded',
 'Lithology_encoded']
```

Feature selection

In [71]:

```
#univariate feature selection based on variance prior normalization
train_imp.var().sort_values(ascending=False)
```

Out[71]:

```
Y_LOC                1.630050e+10
X_LOC                1.183845e+09
ROP                  1.086799e+06
DEPTH_MD             9.943722e+05
Z_LOC                9.341349e+05
RDEP                 1.286276e+04
RSHA                 5.478753e+03
SP                   4.333577e+03
RMED                 2.889932e+03
GR                   1.171795e+03
WELL_encoded         1.029100e+03
DTC                  8.382095e+02
DTS                  7.878111e+02
FORMATION_encoded    5.732392e+02
PEF                  6.985704e+01
DRHO                 4.719175e+01
CALI                 1.337578e+01
GROUP_encoded        9.834499e+00
BS                   6.492793e+00
Lithology_encoded    2.958274e+00
RHOB                 5.546984e-02
NPHI                 1.113932e-02
dtype: float64
```

In [72]:

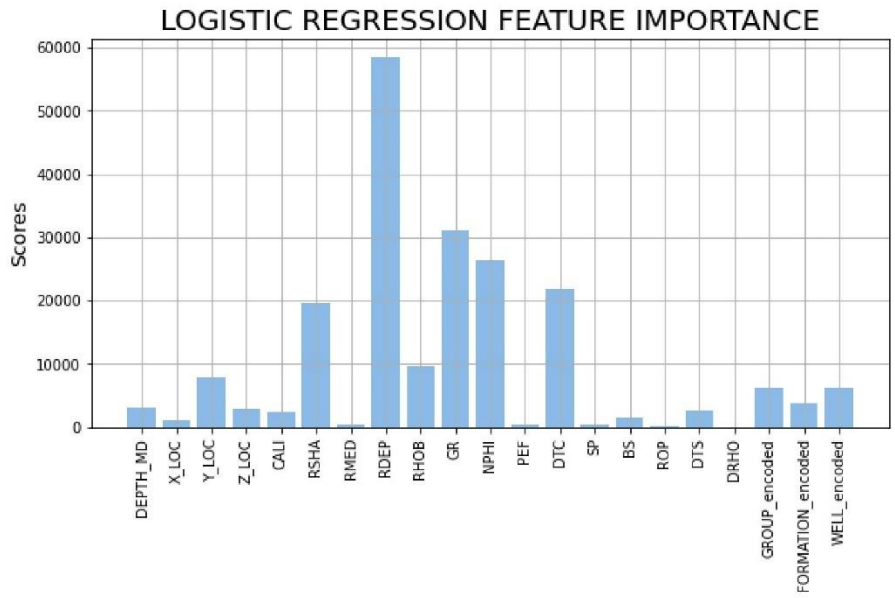
```
import sklearn.feature_selection
x_header=['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC', 'CALI', 'RSHA', 'RMED', 'RDEP',
       'RHOB', 'GR', 'NPHI', 'PEF', 'DTC', 'SP', 'BS', 'ROP', 'DTS', 'DRHO',
       'GROUP_encoded', 'FORMATION_encoded', 'WELL_encoded']
y_header=['Lithology_encoded']
x_train = train_imp[x_header]
y_train = train_imp[y_header]

feature = sklearn.feature_selection.SelectKBest(k=15)
selected_features = feature.fit(x_train, y_train)

data = pd.DataFrame({'Feature':list(x_train.columns), 'Scores':selected_features.scores_})
data = data.sort_values(by='Scores', ascending=False)
display(data.head(20))
```

```
C:\Users\Ayori\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: D
ataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
```

	Feature	Scores
7	RDEP	58466.905990
9	GR	30989.161981
10	NPHI	26283.435828
12	DTC	21921.793977
5	RSHA	19570.572074
8	RHOB	9618.469484
2	Y_LOC	7723.230331
18	GROUP_encoded	6271.927504
20	WELL_encoded	6172.054183
19	FORMATION_encoded	3690.058986
0	DEPTH_MD	3029.220186
3	Z_LOC	2970.368774
16	DTS	2614.047492
4	CALI	2318.560951
14	BS	1463.302955
1	X_LOC	1072.678467
6	RMED	399.446953
11	PEF	342.075575
13	SP	334.400425
15	ROP	44.671834

In [73]:

```
#Visualizing Accuracies
f, ax = plt.subplots(figsize=(10,5))
x_pos = np.arange(len(x_train.columns))

#Create Bars
ax.bar(x_pos, np.array(selected_features.scores_), color=(0.5, 0.7, 0.9, 0.9))
ax.set_ylabel('Scores', size=14)
ax.set_title('LOGISTIC REGRESSION FEATURE IMPORTANCE', size=20)

plt.xticks(x_pos, x_train.columns, rotation=90)
plt.grid(True)
plt.show()
```



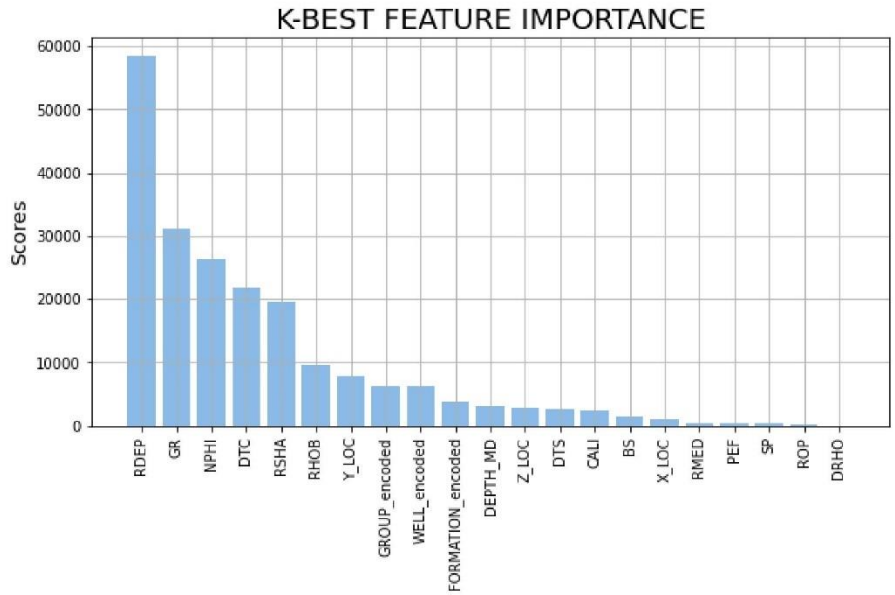
In [74]:

```
#Ranking Features based on the K-Best Method
from sklearn.feature_selection import SelectKBest, chi2, f_classif
selected_features = SelectKBest(f_classif, k=5 ).fit(x_train, y_train)

selected_features_df = pd.DataFrame({'feature': list(x_train.columns),
                                     'scores': selected_features.scores_})
df_new = selected_features_df.sort_values(by='scores', ascending=False)
```

```
C:\Users\Ayori\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: D
ataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
```

In [75]:

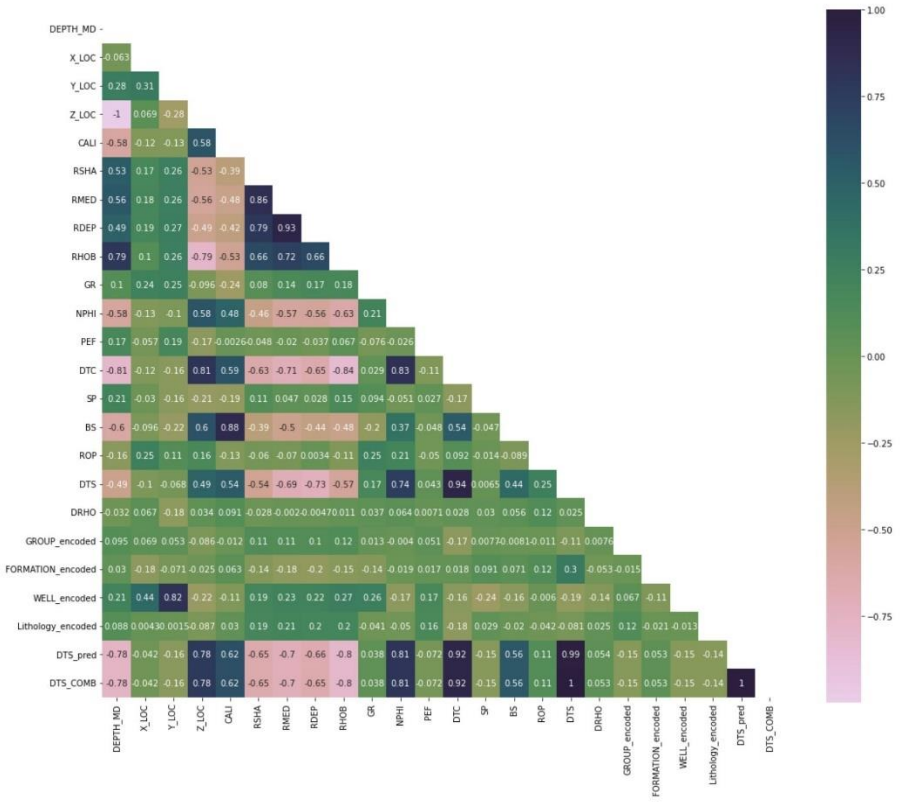
```
#Ploting K-Best Feature Importances
#Visualizing Accuracies
f, ax = plt.subplots(figsize=(10,5))
x_pos = np.arange(len(x_train.columns))

#Create Bars
ax.bar(x_pos, df_new.scores, color=(0.5, 0.7, 0.9, 0.9))
ax.set_ylabel('Scores', size=14)
ax.set_title('K-BEST FEATURE IMPORTANCE', size=20)

plt.xticks(x_pos, df_new.feature, rotation=90)
plt.grid(True)
plt.show()
```



In [76]:

```
#Spearman's Correlation

# Generate a mask for the upper triangle
mask = np.zeros_like(train_prep.corr(method = 'spearman') , dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Generate a custom diverging colormap
cmap = sns.cubehelix_palette(n_colors=12, start=-2.25, rot=-1.3, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
plt.figure(figsize=(18,15))
sns.heatmap(train_prep.corr(method = 'spearman') ,annot=True,  mask=mask, cmap=cmap, vmax=1

plt.show()
```

```
C:\Users\Ayori\AppData\Local\Temp/ipykernel_16448/3325240931.py:4: Deprecati
onWarning:

`np.bool` is a deprecated alias for the builtin `bool`. To silence this warn
ing, use `bool` by itself. Doing this will not modify any behavior and is sa
fe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/d
evdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/rel
ease/1.20.0-notes.html#deprecations)
```



In [77]:

```
#using a cut off to select features - Spearman Correlation
def spearmancorr(data, threshold):
    column_corr = set()  #Set all the names of the correlates columns
    corr_matrix = data.corr(method='spearman')
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold:
                colname = corr_matrix.columns[i]
                column_corr.add(colname)
    return column_corr
```

In [78]:

```
corr_features = spearmancorr(train_prep, 0.8)
corr_features
```

Out[78]:

```
{'BS',
 'DTC',
 'DTS',
 'DTS_COMB',
 'DTS_pred',
 'RDEP',
 'RMED',
 'WELL_encoded',
 'Z_LOC'}
```


In [79]:

```
corr_features = spearmancorr(train_prep, 0.65)
corr_features
```

Out[79]:

```
{'BS',
 'DTC',
 'DTS',
 'DTS_COMB',
 'DTS_pred',
 'RDEP',
 'RHOB',
 'RMED',
 'WELL_encoded',
 'Z_LOC'}
```

In [2]:

```
#feature importance using random forest
from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=200)
rf_model.fit(x_train,y_train.values.ravel())
train_feature = x_train.columns
importances = rf_model.feature_importances_
indices = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(indices)),importances[indices], color='r', align='center')
plt.yticks(range(len(indices)),[train_feature[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

In [ ]:

```
from sklearn.ensemble import IsolationForest
from sklearn.covariance import EllipticEnvelope
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
train_imp1= train_imp.copy()

#Method 1: Local Oultlier Factor
lof = LocalOutlierFactor(contamination=0.3)
yhat = lof.fit_predict(train_imp1)
mask = yhat != -1
train_out = train_imp1[mask]
#display(train_data_lof)
```

In [ ]:

```
train_out
```

In [ ]:

```
#Method 2: Isolation Forest
isolation = IsolationForest(contamination=0.1)
yhat1 = isolation.fit_predict(train_imp1)
mask1 = yhat1 != -1
train_iso = train_imp1[mask1]
```

In [ ]:

```
train_iso
```

In [ ]:

```
#Method 3: Standard Deviation Method (traditional)
train_std = train_imp1[np.abs(train_imp1-train_imp1.mean()) <= (3*train_imp1.std())]
train_std = train_std.dropna()
```

In [ ]:

```
#Method 4: One-class SVM
svm = OneClassSVM(nu=0.1)
that = svm.fit_predict(train_imp1)
mask = that != -1
train_svm = train_imp1[mask]
```

In [ ]:

```
#Checking outlier revoved data lenght
print('Number of points before outliers removed                    :', len(train_imp1))
print('Number of points after outliers removed with Outlier Factor    :', len(train_out))
print('Number of points after outliers removed with Isolation Forest  :', len(train_iso))
print('Number of points after outliers removed with Standard Deviation:', len(train_std))
print('Number of points after outliers removed with One-class SVM     :', len(train_svm))
```

In [ ]:

```
#Box plotting data after outlier removal
box_features = ['RDEP','RMED', 'RSHA','GR', 'DTC', 'NPHI', 'RHOB', 'BS', 'CALI','SP']

plt.figure(figsize=(15,10))

#Before outliers removal
plt.subplot(3,2,1)
train_imp1[box_features].boxplot()
plt.ylim([-15, 15])
plt.title('Before Outlier Removal', size=15)

#Methods of outlier removal
plt.subplot(3,2,2)
train_std[box_features].boxplot()
plt.title('After Outlier Removal with Standard Deviation Filter', size=15)

plt.subplot(3,2,3)
train_iso[box_features].boxplot()
plt.ylim([-10, 10])
plt.title('After Outlier Removal with Isolation Forest', size=15)

plt.subplot(3,2,5)
train_out[box_features].boxplot()
plt.title('After Outlier Removal with Local Outlier Factor', size=15)

plt.subplot(3,2,6)
train_svm[box_features].boxplot()
plt.title('After Outlier Removal with One-class SVM', size=15)

plt.tight_layout(1.7)
plt.show()
```

In [ ]:

```
#Pair plot after outlier removal
pairplot_features = ['RDEP','RMED', 'RSHA','GR', 'DTC', 'NPHI', 'RHOB', 'BS', 'CALI','SP']
sns.pairplot(train_iso, diag_kind='kde', vars = pairplot_features, plot_kws = {'alpha': 0.6
```

Normalisation

In [ ]:

```
#Inputing missing values by introducing median
from sklearn.impute import SimpleImputer
miss = SimpleImputer(missing_values=np.nan, strategy='median')
miss.fit(test_prep1)
test_imp = miss.fit_transform(test_prep1)
test_imp=pd.DataFrame(test_imp, columns=['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC', 'CALI', 'RS
       'RHOB', 'GR', 'NPHI', 'PEF', 'DTC', 'SP', 'BS', 'ROP', 'DTS', 'DRHO',
         'GROUP_encoded',
       'FORMATION_encoded', 'WELL_encoded','Lithology_encoded'])
test_imp
```

In [ ]:

```
#Inputing missing values by introducing median
from sklearn.impute import SimpleImputer
miss = SimpleImputer(missing_values=np.nan, strategy='median')
miss.fit(hidden_prep1)
hidden_imp = miss.fit_transform(hidden_prep1)
hidden_imp=pd.DataFrame(hidden_imp, columns=['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC', 'CALI',
       'RHOB', 'GR', 'NPHI', 'PEF', 'DTC', 'SP', 'BS', 'ROP', 'DTS', 'DRHO',
         'GROUP_encoded',
       'FORMATION_encoded', 'WELL_encoded','Lithology_encoded'])
hidden_imp
```

In [ ]:

```
x_train.iloc[:,:18]
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
from sklearn.preprocessing import StandardScaler, Normalizer, MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import PowerTransformer
x_test = test_imp[x_header]
y_test = test_imp[y_header]
x_hidden = hidden_imp[x_header]
y_hidden = hidden_imp[y_header]
##Min-Max scaler
scaler = MinMaxScaler()
x_train_scaled = x_train.copy()
x_test_scaled = x_test.copy()
x_hidden_scaled = x_hidden.copy()

x_train_scaled.iloc[:,:18] = scaler.fit_transform(x_train_scaled.iloc[:,:18])
x_test_scaled.iloc[:,:18] = scaler.transform(x_test_scaled.iloc[:,:18])
x_hidden_scaled.iloc[:,:18] = scaler.transform(x_hidden_scaled.iloc[:,:18])
```

In [ ]:

```
##standard scaler
sc = StandardScaler()
x_train_sc = x_train.copy()
x_test_sc = x_test.copy()
x_hidden_sc = x_hidden.copy()

x_train_sc.iloc[:,:18] = sc.fit_transform(x_train_sc.iloc[:,:18])
x_test_sc.iloc[:,:18] = sc.transform(x_test_sc.iloc[:,:18])
x_hidden_sc.iloc[:,:18] = sc.transform(x_hidden_sc.iloc[:,:18])
```

In [ ]:

```
#Robustscaler
from sklearn.preprocessing import RobustScaler
rs= RobustScaler()
x_train_rs = x_train.copy()
x_test_rs = x_test.copy()
x_hidden_rs = x_hidden.copy()

x_train_rs.iloc[:,:18] = rs.fit_transform(x_train_rs.iloc[:,:18])
x_test_rs.iloc[:,:18] = rs.transform(x_test_rs.iloc[:,:18])
x_hidden_rs.iloc[:,:18] = rs.transform(x_hidden_rs.iloc[:,:18])
```

In [ ]:

```
##Normalization
norm = Normalizer()
x_train_norm = x_train.copy()
x_test_norm = x_test.copy()
x_hidden_norm = x_hidden.copy()

x_train_norm.iloc[:,:18] = norm.fit_transform(x_train_norm.iloc[:,:18])
x_test_norm.iloc[:,:18] = norm.transform(x_test_norm.iloc[:,:18])
x_hidden_norm.iloc[:,:18] = norm.transform(x_hidden_norm.iloc[:,:18])
```

In [ ]:

```
#Box plotting data after Scaling ---- NO LOG

#box_features = ['RDEP','RMED', 'RSHA','GR', 'DTC', 'NPHI', 'RHOB', 'BS', 'CALI']
box_features = ['GR', 'CALI', 'SP', 'RDEP', 'DTC', 'NPHI', 'RHOB','RMED','RSHA']
plt.figure(figsize=(15,10))

# 0. Before Normalization
plt.subplot(3,2,1)
x_train[box_features].boxplot()
plt.title('Before any type of Scaling', size=15)

# 1. After MinMax Scaler
plt.subplot(3,2,2)
x_train_scaled[box_features].boxplot()
plt.title('After MinMax Scaler', size=15)

# 2. After Standardization
plt.subplot(3,2,3)
x_train_sc[box_features].boxplot()
plt.ylim([-15, 40])
plt.title('After Standardization', size=15)

# 3. After robust scaler
plt.subplot(3,2,4)
x_train_norm[box_features].boxplot()
plt.title('After Robust scaler', size=15)

# 4. After Normalization
plt.subplot(3,2,5)
x_train_norm[box_features].boxplot()
plt.title('After Normalization', size=15)

plt.tight_layout(1.7)
plt.show()
```

Comparing the performance of scaling

In [ ]:

```
#Fitting a logistic regression model  #### KEEP THIS ONE

lr = LogisticRegression(C=1e-3, solver="saga", max_iter=10000, verbose=1)
#lr = LogisticRegression()

#Score with no Scale/Normalization
lr.fit(x_train, y_train.values.ravel())
print('The acuracy score withouth Normalization/scaling', lr.score(x_test, y_test))
```

In [ ]:

```
# 1. Score after Scaling
lr.fit(x_train_scaled, y_train.values.ravel())
print('1. The accuracy score after Scaling', lr.score(x_test_scaled, y_test))
```

In [ ]:

```
# 2. Score after Standardization
lr.fit(x_train_sc, y_train.values.ravel())
print('2. The acuracy score after Standardization', lr.score(x_test_sc, y_test))
```

In [ ]:

```
# 3. score after robustscaler

lr.fit(x_train_rs, y_train.values.ravel())
print('3. The acuracy score after Standardization', lr.score(x_test_rs, y_test))
```

In [ ]:

```
# 4. Score after Normalization
lr.fit(x_train_norm, y_train.values.ravel())
print('4. The accuracy score after Normalization', lr.score(x_test_norm, y_test))
```

In [ ]:

```
y_test
```

In [ ]:

```
#can we make new features by combining previous ones?
import featuretools as ft
from woodwork.logical_types import Categorical
df_features = train1
es = ft.EntitySet(id = 'observations')
es = es.add_dataframe(dataframe_name='observations',dataframe = df_features.reset_index(),
                              index = 'index',
                              logical_types={"FORMATION": Categorical,
                                             "GROUP": Categorical})
features, ft_names = ft.dfs(entityset = es, target_dataframe_name = 'observations',
                            trans_primitives = ['multiply_numeric'],
                            #trans_primitives = ['add_numeric', 'multiply_numeric',
                            #                    'subtract_numeric', 'divide_numeric', 'dif
                            max_depth=2)
features.columns
```

Volume of shale (vsh): gamma ray (gr) , spontaneous potential (sp) Porosity (phi): neutron porosity (nphi), density (den), sonic (dt) Water saturation (sw): deep resistivity (deep_res)

In [ ]:

```
# Volume of shale by gamma ray

gr_cl = 17    # gr clean or minimum
gr_sh = 157    # gr of shales or maximum
GR= train_imp1['GR']
vsh = (GR - gr_cl)/(gr_sh - gr_cl)    # Similar to the unity-base normalization

vsh = np.maximum(np.minimum(vsh, 1), 0.0001)
```

In [ ]:

```
vsh
```

In [ ]:

```
from sklearn.model_selection import train_test_split

# NPHI training Data
train_nphi = train_prep[train_prep.NPHI.notna()]

#Labels and faatures
X_nphi = train_nphi.drop(['NPHI'], axis=1)
Y_nphi = train_nphi['NPHI']

#Imputation
X_nphi_inp = X_nphi.apply(lambda x: x.fillna(x.median()), axis=0)

#Spliting into nphi_train and nphi_validation sets
X_nphi_train, X_nphi_val, Y_nphi_train, Y_nphi_val = train_test_split(X_nphi_inp, Y_nphi, t

print('Splitted training data shape is {} and validation data shape is {}'.format(X_nphi_tr
#Predicting NPHI
from sklearn.metrics import max_error
from xgboost import XGBRegressor

model1000 = XGBRegressor()
model1000.fit(X_nphi_train, Y_nphi_train.values.ravel(), early_stopping_rounds=100, eval_se

train_pred1 = model1000.predict(X_nphi_train)
val_pred1 = model1000.predict(X_nphi_val)

print('Train error:', max_error(Y_nphi_train, train_pred1))
print('Validation error:', max_error(Y_nphi_val, val_pred1))
#Visual check on the predictions (Validation Set)
plt.figure(figsize=(25, 5))
plt.plot(list(range(100)), Y_nphi_val[25000:25100])
plt.plot(list(range(100)), val_pred1[25000:25100], color='g')

# Filling nan values before predicting nphi
X_train_nphi = train_prep.drop(['NPHI'], axis=1)
X_train_nphi2 = X_train_nphi.apply(lambda x: x.fillna(x.median()), axis=0)

X_test_nphi = test_prep.drop(['NPHI'], axis=1)
X_test_nphi2 = X_test_nphi.apply(lambda x: x.fillna(x.median()), axis=0)

X_hidden_nphi = hidden_prep.drop(['NPHI'], axis=1)
X_hidden_nphi2 = X_hidden_nphi.apply(lambda x: x.fillna(x.median()), axis=0)

#Predicting nphi (COMPLETE DATASETS)
train_prep['NPHI_pred'] = model1000.predict(X_train_nphi2)
test_prep['NPHI_pred'] = model1000.predict(X_test_nphi2)
hidden_prep['NPHI_pred'] = model1000.predict(X_hidden_nphi2)

#Inputing nan values in nphi with nphi_PREDICTED
train_prep['NPHI_COMB'] = train_prep['NPHI']
train_prep['NPHI_COMB'].fillna(train_prep['NPHI_pred'], inplace=True)

test_prep['NPHI_COMB'] = test_prep['NPHI']
test_prep['NPHI_COMB'].fillna(test_prep['NPHI_pred'], inplace=True)

hidden_prep['NPHI_COMB'] = hidden_prep['NPHI']
hidden_prep['NPHI_COMB'].fillna(hidden_prep['NPHI_pred'], inplace=True)
```

In [ ]:

```
from sklearn.model_selection import train_test_split

# RHOB training Data
train_rhob = train_prep[train_prep.RHOB.notna()]

#Labels and faatures
X_rhob = train_rhob.drop(['RHOB'], axis=1)
Y_rhob = train_rhob['RHOB']

#Imputation
X_rhob_inp = X_rhob.apply(lambda x: x.fillna(x.median()), axis=0)

#Spliting into rhob_train and rhob_validation sets
X_rhob_train, X_rhob_val, Y_rhob_train, Y_rhob_val = train_test_split(X_rhob_inp, Y_rhob, t

print('Splitted training data shape is {} and validation data shape is {}'.format(X_rhob_tr
#Predicting rhob
from sklearn.metrics import max_error
from xgboost import XGBRegressor

model1000 = XGBRegressor()
model1000.fit(X_rhob_train, Y_rhob_train.values.ravel(), early_stopping_rounds=100, eval_se

train_pred2 = model1000.predict(X_rhob_train)
val_pred2 = model1000.predict(X_rhob_val)

print('Train error:', max_error(Y_rhob_train, train_pred2))
print('Validation error:', max_error(Y_rhob_val, val_pred2))
#Visual check on the predictions (Validation Set)
plt.figure(figsize=(25, 5))
plt.plot(list(range(100)), Y_rhob_val[25000:25100])
plt.plot(list(range(100)), val_pred2[25000:25100], color='g')

# Filling nan values before predicting rhob
X_train_rhob = train_prep.drop(['RHOB'], axis=1)
X_train_rhob2 = X_train_rhob.apply(lambda x: x.fillna(x.median()), axis=0)

X_test_rhob = test_prep.drop(['RHOB'], axis=1)
X_test_rhob2 = X_test_rhob.apply(lambda x: x.fillna(x.median()), axis=0)

X_hidden_rhob = hidden_prep.drop(['RHOB'], axis=1)
X_hidden_rhob2 = X_hidden_rhob.apply(lambda x: x.fillna(x.median()), axis=0)

#Predicting rhob (COMPLETE DATASETS)
train_prep['RHOB_pred'] = model1000.predict(X_train_rhob2)
test_prep['RHOB_pred'] = model1000.predict(X_test_rhob2)
hidden_prep['RHOB_pred'] = model1000.predict(X_hidden_rhob2)

#Inputing nan values in rhob with rhob_PREDICTED
train_prep['RHOB_COMB'] = train_prep['RHOB']
train_prep['RHOB_COMB'].fillna(train_prep['RHOB_pred'], inplace=True)

test_prep['RHOB_COMB'] = test_prep['RHOB']
test_prep['RHOB_COMB'].fillna(test_prep['RHOB_pred'], inplace=True)

hidden_prep['RHOB_COMB'] = hidden_prep['RHOB']
hidden_prep['RHOB_COMB'].fillna(hidden_prep['RHOB_pred'], inplace=True)
```

Appendix C: Base model for 100,000 sample of data

In [1]:

```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
from tqdm import tqdm
import time

train = pd.read_csv("CSV_train.csv",low_memory=False,delimiter=';')
test=pd.read_csv("CSV_test.csv",low_memory=False,delimiter=',')
hidden=pd.read_csv("CSV_hidden_test.csv",low_memory=False,delimiter=',')
```


In [2]:

```
# storing length of datasets
train_len = train.shape[0]
test_len = test.shape[0]
All_data = pd.concat((train,test,hidden)).reset_index(drop=True)

lithology_keys = {30000: 'Sandstone',
                  65030: 'Sandstone/Shale',
                  65000: 'Shale',
                  80000: 'Marl',
                  74000: 'Dolomite',
                  70000: 'Limestone',
                  70032: 'Chalk',
                  88000: 'Halite',
                  86000: 'Anhydrite',
                  99000: 'Tuff',
                  90000: 'Coal',
                  93000: 'Basement'}
All_data['Lithology'] = All_data['FORCE_2020_LITHOFACIES_LITHOLOGY'].map(lithology_keys)
All_data
```

In [3]:

```
#dropping columns with high missing values
drop_cols = ['SGR', 'ROPA', 'RXO', 'MUDWEIGHT','DCAL','RMIC','FORCE_2020_LITHOFACIES_CONFID
All_data_drop = All_data.drop(drop_cols, axis=1)
# encoding categorical variables
All_data_drop['GROUP_encoded'] = All_data_drop['GROUP'].astype('category')
All_data_drop['GROUP_encoded'] = All_data_drop['GROUP_encoded'].cat.codes

All_data_drop['FORMATION_encoded'] = All_data_drop['FORMATION'].astype('category')
All_data_drop['FORMATION_encoded'] = All_data_drop['FORMATION_encoded'].cat.codes

All_data_drop['WELL_encoded'] = All_data_drop['WELL'].astype('category')
All_data_drop['WELL_encoded'] = All_data_drop['WELL_encoded'].cat.codes

All_data_drop['Lithology_encoded'] = All_data_drop['FORCE_2020_LITHOFACIES_LITHOLOGY'].asty
All_data_drop['Lithology_encoded'] = All_data_drop['Lithology_encoded'].cat.codes
```


In [5]:

```
#dropping categorial features replaces beforehan by encoded features
drop2 = All_data_drop.drop(['GROUP', 'FORMATION','WELL','FORCE_2020_LITHOFACIES_LITHOLOGY',
```


In [8]:

```
#Inputing missing values by introducing median
from sklearn.impute import SimpleImputer
miss = SimpleImputer(missing_values=np.nan, strategy='median')
miss.fit(drop2)
All_imp = miss.fit_transform(drop2)
All_imp=pd.DataFrame(All_imp, columns=['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC', 'CALI', 'RSHA
       'RHOB', 'GR', 'NPHI', 'PEF', 'DTC', 'SP', 'BS', 'ROP', 'DTS', 'DRHO',
         'GROUP_encoded',
       'FORMATION_encoded', 'WELL_encoded','Lithology_encoded'])
All_imp
```

In [11]:

```
from sklearn.preprocessing import StandardScaler, Normalizer, MinMaxScaler
x_header=['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC', 'CALI', 'RSHA', 'RMED', 'RDEP',
       'RHOB', 'GR', 'NPHI', 'PEF', 'DTC', 'SP', 'BS', 'ROP', 'DTS', 'DRHO',
       'GROUP_encoded', 'FORMATION_encoded', 'WELL_encoded']
y_header=['Lithology_encoded']
x_train = train_imp[x_header]
y_train = train_imp[y_header]
x_test = test_imp[x_header]
y_test = test_imp[y_header]
x_hidden = hidden_imp[x_header]
y_hidden = hidden_imp[y_header]

##Min-Max scaler
scaler = MinMaxScaler()
x_train_scaled = x_train.copy()
x_test_scaled = x_test.copy()
x_hidden_scaled = x_hidden.copy()

x_train_scaled.iloc[:,:18] = scaler.fit_transform(x_train_scaled.iloc[:,:18])
x_test_scaled.iloc[:,:18] = scaler.transform(x_test_scaled.iloc[:,:18])
x_hidden_scaled.iloc[:,:18] = scaler.transform(x_hidden_scaled.iloc[:,:18])
```

In [12]:

```
#Supervised Algorithms
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error, accuracy_score, recall_score, precision_sco
from sklearn.neighbors import KNeighborsRegressor
from pprint import pprint
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import GaussianNB
import xgboost
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
#Comparing base models accuracies by using k-fold cross validation - 10 folds

from sklearn.model_selection import cross_val_score

new_train = pd.concat((x_train_scaled, pd.DataFrame(y_train, columns=["Lithology_encoded"])

#Randomly sampling data
sampled_train = new_train.sample(n=100000, random_state=0)


#Spliting training data
x_train_sam = sampled_train.drop(["Lithology_encoded"], axis=1)
y_train_sam = sampled_train["Lithology_encoded"]
```

```
C:\Users\Ayori\anaconda3\lib\site-packages\xgboost\compat.py:36: FutureWarni
ng: pandas.Int64Index is deprecated and will be removed from pandas in a fut
ure version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

In [13]:

```
estimator = LogisticRegression(C=1e-3, solver='saga', max_iter=4000)
score = cross_val_score(estimator, x_train_sam, y_train_sam.values.ravel(), cv=10, scoring=
score
```

Out[13]:

```
0.4781277833239425
```

In [14]:

```
estimator1 = DecisionTreeClassifier()
score = cross_val_score(estimator1, x_train_sam, y_train_sam.values.ravel(), cv=10, scoring
score
```

Out[14]:

```
0.8637868813236393
```

In [15]:

```
estimator2 = RandomForestClassifier()
score = cross_val_score(estimator2, x_train_sam, y_train_sam.values.ravel(), cv=10, scoring
score
```

Out[15]:

```
0.9143748285935347
```

In [17]:

```
estimator3 = XGBClassifier()
score3 = cross_val_score(estimator3, x_train_sam, y_train_sam.values.ravel(), cv=10, scorin
score3
```

```
C:\Users\Ayori\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWa
rning: The use of label encoder in XGBClassifier is deprecated and will be
removed in a future release. To remove this warning, do the following: 1)
Pass option use_label_encoder=False when constructing XGBClassifier objec
t; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1,
2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\Ayori\anaconda3\lib\site-packages\xgboost\data.py:250: FutureWarn
ing: pandas.Int64Index is deprecated and will be removed from pandas in a
future version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):

[15:39:53] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'multi:softprob' was cha
nged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like
to restore the old behavior.

C:\Users\Ayori\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWa
rning: The use of label encoder in XGBClassifier is deprecated and will be
removed in a future release. To remove this warning, do the following: 1)
```

In [18]:

```
estimator4 = GradientBoostingClassifier()
score4 = cross_val_score(estimator4, x_train_sam, y_train_sam.values.ravel(), cv=10, scorin
score4
```

Out[18]:

```
0.8359888930126212
```

In [19]:

```
from sklearn.neighbors import KNeighborsClassifier
estimator5 =KNeighborsClassifier()
score5 = cross_val_score(estimator5, x_train_sam, y_train_sam.values.ravel(), cv=10, scorin
score5
```

Out[19]:

```
0.879973736174572
```

In [20]:

```
from sklearn.neighbors import KNeighborsClassifier
estimator6 =SVC()
score6 = cross_val_score(estimator6, x_train_sam, y_train_sam.values.ravel(), cv=10, scorin
score6
```

Out[20]:

```
0.4762872789526972
```

In [21]:

```
from catboost import CatBoostClassifier
estimator7 =CatBoostClassifier()
score7 = cross_val_score(estimator7, x_train_sam, y_train_sam.values.ravel(), cv=10, scorin
score7
```

```
Learning rate set to 0.099538
0:      learn: 1.9572400        total: 443ms    remaining: 7m 22s
1:      learn: 1.7220980        total: 753ms    remaining: 6m 15s
2:      learn: 1.5628252        total: 1.06s    remaining: 5m 53s
3:      learn: 1.4377097        total: 1.37s    remaining: 5m 42s
4:      learn: 1.3368788        total: 1.79s    remaining: 5m 55s
5:      learn: 1.2600264        total: 2.09s    remaining: 5m 46s
6:      learn: 1.1952599        total: 2.38s    remaining: 5m 38s
7:      learn: 1.1424774        total: 2.69s    remaining: 5m 33s
8:      learn: 1.0902235        total: 2.99s    remaining: 5m 29s
9:      learn: 1.0447627        total: 3.29s    remaining: 5m 25s
10:     learn: 1.0057126        total: 3.57s    remaining: 5m 20s
11:     learn: 0.9718828        total: 3.84s    remaining: 5m 16s
12:     learn: 0.9433615        total: 4.13s    remaining: 5m 13s
13:     learn: 0.9171024        total: 4.42s    remaining: 5m 11s
14:     learn: 0.8936737        total: 4.72s    remaining: 5m 10s
15:     learn: 0.8685208        total: 5.01s    remaining: 5m 8s
16:     learn: 0.8485486        total: 5.32s    remaining: 5m 7s
17:     learn: 0.8294735        total: 5.61s    remaining: 5m 6s
```

In [23]:

```
from lightgbm import LGBMClassifier
estimator8 =LGBMClassifier()
score8 = cross_val_score(estimator8, x_train_sam, y_train_sam.values.ravel(), cv=10, scorin
score8
```

Out[23]:

```
0.7853414136709407
```

Appendix D: Data augmentation

In [13]:
```
def augment_features_window(X, N_neig):

    # Parameters
    N_row = X.shape[0]
    N_feat = X.shape[1]

    # Zero padding
    X = np.vstack((np.zeros((N_neig, N_feat)), X, (np.zeros((N_neig, N_feat)))))

    # Loop over windows
    X_aug = np.zeros((N_row, N_feat*(2*N_neig+1)))
    for r in np.arange(N_row)+N_neig:
        this_row = []
        for c in np.arange(-N_neig,N_neig+1):
            this_row = np.hstack((this_row, X[r+c]))
        X_aug[r-N_neig] = this_row

    return X_aug

# Feature gradient computation function
def augment_features_gradient(X, depth):

    # Compute features gradient
    d_diff = np.diff(depth).reshape((-1, 1))
    d_diff[d_diff==0] = 0.001
    X_diff = np.diff(X, axis=0)
    X_grad = X_diff / d_diff

    # Compensate for last missing value
    X_grad = np.concatenate((X_grad, np.zeros((1, X_grad.shape[1]))))

    return X_grad

#Feature augmentation function
def augment_features(X, well, depth, N_neig=1):

    # Augment features
    X_aug = np.zeros((X.shape[0], X.shape[1]*(N_neig*2+2)))
    for w in np.unique(well):
        w_idx = np.where(well == w)[0]
        X_aug_win = augment_features_window(X[w_idx, :], N_neig)
        X_aug_grad = augment_features_gradient(X[w_idx, :], depth[w_idx])
        X_aug[w_idx, :] = np.concatenate((X_aug_win, X_aug_grad), axis=1)

    # Find padded rows
    padded_rows = np.unique(np.where(X_aug[:, 0:7] == np.zeros((1, 7)))[0])

    return X_aug, padded_rows
```

In [14]:
```
x_train_arg, padded_rows = augment_features(x_train_scaled.values, train_well, train_depth)
x_test_arg, padded_rows = augment_features(x_test_scaled.values, test_well, test_depth)
x_hidden_arg, padded_rows = augment_features(x_hidden_scaled.values, hidden_well, hidden_depth)
```

In [15]:
```
RAINING DATA before argumentation is {} and its shape after argumentation is {}'.format(x_train_scaled.shape, x_train_arg.shape))
EST DATA before argumentation is {} and its shape after argumentation is {}'.format(x_test_scaled.shape, x_test_arg.shape))
IDDEN DATA before argumentation is {} and its shape after argumentation is {}'.format(x_hidden_scaled.shape, x_hidden_arg.shape))
```

```
The shape of the TRAINING DATA before argumentation is (1170511, 21) and its shape after argumentation is (1170511, 84)
The shape of the TEST DATA before argumentation is (136786, 21) and its shape after argumentation is (136786, 84)
The shape of the HIDDEN DATA before argumentation is (122397, 21) and its shape after argumentation is (122397, 84)
```

Appendix F: Feature engineering

In [16]:

```
# calculating p-impedance
train_imp['PI'] = train_imp.RHOB * (1e6/train_imp.DTC)
test_imp['PI'] = test_imp.RHOB * (1e6/test_imp.DTC)
hidden_imp['PI'] = hidden_imp.RHOB * (1e6/hidden_imp.DTC)

# calculating s-impedance
train_imp['SI'] = train_imp.RHOB * (1e6/train_imp.DTS)
test_imp['SI'] = test_imp.RHOB * (1e6/test_imp.DTS)
hidden_imp['SI'] = hidden_imp.RHOB * (1e6/hidden_imp.DTS)

#calculating Shear modulus (G)
train_imp['G'] = ((1e6/train_imp.DTS)**2) * train_imp.RHOB
test_imp['G'] = ((1e6/test_imp.DTS)**2) * test_imp.RHOB
hidden_imp['G'] = ((1e6/hidden_imp.DTS)**2) * hidden_imp.RHOB

#calculating Bulk modulus (K)
train_imp['K'] = (((1e6/train_imp.DTC)**2) * train_imp.RHOB) - (4 * train_imp.G/3)
test_imp['K'] = (((1e6/test_imp.DTC)**2) * test_imp.RHOB) - (4 * test_imp.G/3)
hidden_imp['K'] = (((1e6/hidden_imp.DTC)**2) * hidden_imp.RHOB) - (4 * hidden_imp.G/3)

# calculate the shale volume
train_imp["VSHALE"] = (train_imp.GR - np.min(train_imp.GR)) / (np.max(train_imp.GR) - np.min(train_imp.GR))
test_imp["VSHALE"] = (test_imp.GR - np.min(test_imp.GR)) / (np.max(test_imp.GR) - np.min(test_imp.GR))
hidden_imp["VSHALE"] = (hidden_imp.GR - np.min(hidden_imp.GR)) / (np.max(hidden_imp.GR) - np.min(hidden_imp.GR))
#train_imp1.head()

# calculate the total porosity
train_imp['PHIT'] = np.sqrt(((((train_imp.NPHI)*(train_imp.NPHI)+(train_imp.RHOB)*(train_imp.RHOB))))/2)
test_imp['PHIT'] = np.sqrt(((((test_imp.NPHI)*(test_imp.NPHI)+(test_imp.RHOB)*(test_imp.RHOB))))/2)
hidden_imp['PHIT'] = np.sqrt(((((hidden_imp.NPHI)*(hidden_imp.NPHI)+(hidden_imp.RHOB)*(hidden_imp.RHOB))))/2)
#train_imp1.tail()

# calculate effective porosity
train_imp['PHIE'] = train_imp.PHIT*(1-train_imp.VSHALE)
train_imp = train_imp[train_imp['PHIE'] !=0]
train_imp['PHIE'] = train_imp['PHIE'].abs()

test_imp['PHIE'] = test_imp.PHIT*(1-test_imp.VSHALE)
test_imp = test_imp[test_imp['PHIE'] !=0]
test_imp['PHIE'] = test_imp['PHIE'].abs()

hidden_imp['PHIE'] = hidden_imp.PHIT*(1-hidden_imp.VSHALE)
hidden_imp = hidden_imp[hidden_imp['PHIE'] !=0]
hidden_imp['PHIE'] = hidden_imp['PHIE'].abs()
#train_imp1.tail()
```

Appendix G: Base model for entire dataset

This is similar for other models, only that different scenarios were added.

In [1]:

```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
from tqdm import tqdm
import time

train = pd.read_csv("CSV_train.csv",low_memory=False,delimiter=';')
test=pd.read_csv("CSV_test.csv",low_memory=False,delimiter=',')
hidden=pd.read_csv("CSV_hidden_test.csv",low_memory=False,delimiter=',')
```


In [3]:

```
# storing length of datasets
train_len = train.shape[0]
test_len = test.shape[0]
All_data = pd.concat((train,test,hidden)).reset_index(drop=True)

lithology_keys = {30000: 'Sandstone',
                  65030: 'Sandstone/Shale',
                  65000: 'Shale',
                  80000: 'Marl',
                  74000: 'Dolomite',
                  70000: 'Limestone',
                  70032: 'Chalk',
                  88000: 'Halite',
                  86000: 'Anhydrite',
                  99000: 'Tuff',
                  90000: 'Coal',
                  93000: 'Basement'}
All_data['Lithology'] = All_data['FORCE_2020_LITHOFACIES_LITHOLOGY'].map(lithology_keys)
All_data
```

Out[3]:

In [4]:

```
#dropping columns with high missing values
drop_cols = ['SGR', 'ROPA', 'RXO', 'MUDWEIGHT','DCAL','RMIC','FORCE_2020_LITHOFACIES_CONFID
All_data_drop = All_data.drop(drop_cols, axis=1)
```


In [8]:

```
#Inputing missing values by introducing median
from sklearn.impute import SimpleImputer

numeric_header=['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC',
       'CALI', 'RSHA', 'RMED', 'RDEP', 'RHOB', 'GR', 'NPHI', 'PEF', 'DTC',
       'SP', 'BS', 'ROP', 'DTS', 'DRHO', 'FORCE_2020_LITHOFACIES_LITHOLOGY'
       ]
categorical_header=['WELL','GROUP', 'FORMATION','Lithology']
numeric=All_data_drop.select_dtypes(include=[np.number])
categorical= All_data_drop.select_dtypes(exclude=[np.number])
miss = SimpleImputer(missing_values=np.nan, strategy='median')
miss.fit(numeric)
numeric_imp = miss.fit_transform(numeric)
numeric_imp=pd.DataFrame(numeric_imp, columns=numeric_header)
miss2 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
miss2.fit(categorical)
categorical_imp = miss2.fit_transform(categorical)
categorical_imp=pd.DataFrame(categorical_imp, columns=categorical_header)
frames = [numeric_imp,categorical_imp]

result = pd.concat(frames,axis=1, join='inner')
result
```

In [9]:

```
# encoding categorical variables
result['GROUP_encoded'] = result['GROUP'].astype('category')
result['GROUP_encoded'] = result['GROUP_encoded'].cat.codes

result['FORMATION_encoded'] = result['FORMATION'].astype('category')
result['FORMATION_encoded'] = result['FORMATION_encoded'].cat.codes

result['WELL_encoded'] = result['WELL'].astype('category')
result['WELL_encoded'] = result['WELL_encoded'].cat.codes

result['Lithology_encoded'] = result['FORCE_2020_LITHOFACIES_LITHOLOGY'].astype('category')
result['Lithology_encoded'] = result['Lithology_encoded'].cat.codes
```

In [10]:

```
#dropping categorial features replaces beforehan by encoded features
# drop2 = All_data_drop.drop(['GROUP', 'FORMATION','WELL','FORCE_2020_LITHOFACIES_LITHOLOGY

# # splitting dataset into training, test, and hidden sets
# train_prep = drop2[:train_len].copy()
# test_prep = drop2[train_len:(train_len+test_len)].copy()
# hidden_prep = drop2[(train_len+test_len):].copy()
```

In [11]:

```
# train_prep1= train_prep.copy()
# test_prep1= test_prep.copy()
# hidden_prep1= hidden_prep.copy()
```

In [12]:

```
train_imp = result[:train_len].copy()
test_imp = result[train_len:(train_len+test_len)].copy()
hidden_imp = result[(train_len+test_len):].copy()
```

In [13]:

```
print(train_imp.shape)
print(test_imp.shape)
print(hidden_imp.shape)
```

```
(1170511, 27)
(136786, 27)
(122397, 27)
```

In [14]:

```
from sklearn.preprocessing import StandardScaler, Normalizer, MinMaxScaler
x_header=['DEPTH_MD', 'X_LOC', 'Y_LOC', 'Z_LOC', 'CALI', 'RSHA', 'RMED', 'RDEP',
       'RHOB', 'GR', 'NPHI', 'PEF', 'DTC', 'SP', 'BS', 'ROP', 'DTS', 'DRHO',
       'GROUP_encoded', 'FORMATION_encoded', 'WELL_encoded']
y_header=['Lithology_encoded']
x_train = train_imp[x_header]
y_train = train_imp[y_header]
x_test = test_imp[x_header]
y_test = test_imp[y_header]
x_hidden = hidden_imp[x_header]
y_hidden = hidden_imp[y_header]

##Min-Max scaler
scaler = MinMaxScaler()
x_train_scaled = x_train.copy()
x_test_scaled = x_test.copy()
x_hidden_scaled = x_hidden.copy()

x_train_scaled.iloc[:,:18] = scaler.fit_transform(x_train_scaled.iloc[:,:18])
x_test_scaled.iloc[:,:18] = scaler.transform(x_test_scaled.iloc[:,:18])
x_hidden_scaled.iloc[:,:18] = scaler.transform(x_hidden_scaled.iloc[:,:18])
```

07/09/2022, 02:49 New XGB model - Jupyter Notebook

In [16]:

```
A = np.load('penalty_matrix.npy')
def score(y_true, y_pred):
    S = 0.0
    y_true = y_true.astype(int)
    y_pred = y_pred.astype(int)
    for i in range(0, y_true.shape[0]):
        S -= A[y_true[i], y_pred[i]]
    return S/y_true.shape[0]
```

In [17]:

```
#Supervised Algorithms
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error, accuracy_score, recall_score, precision_sco
from sklearn.metrics import classification_report, accuracy_score
from sklearn.neighbors import KNeighborsRegressor
from pprint import pprint
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import GaussianNB
import xgboost
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
#Comparing base models accuracies by using k-fold cross validation - 10 folds

from sklearn.model_selection import cross_val_score

model_xgb = XGBClassifier()

model_xgb.fit(x_train_scaled, y_train.values.ravel(), early_stopping_rounds=100, eval_set=[

train_pred_xgb = model_xgb.predict(x_train_scaled)
open_pred_xgb = model_xgb.predict(x_test_scaled)
hidden_pred_xgb = model_xgb.predict(x_hidden_scaled)
#Printing Reports
```

```
C:\Users\Ayori\anaconda3\lib\site-packages\xgboost\compat.py:36: FutureWarni
ng: pandas.Int64Index is deprecated and will be removed from pandas in a fut
ure version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
C:\Users\Ayori\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarn
ing: The use of label encoder in XGBClassifier is deprecated and will be rem
oved in a future release. To remove this warning, do the following: 1) Pass
option use_label_encoder=False when constructing XGBClassifier object; and
2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [n
um_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\Ayori\anaconda3\lib\site-packages\xgboost\data.py:250: FutureWarnin
g: pandas.Int64Index is deprecated and will be removed from pandas in a futu
re version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
C:\Users\Ayori\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: D
ataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  return f(*args, **kwargs)

[02:03:54] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the d
efault evaluation metric used with the objective 'multi:softprob' was change
d from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to r
estore the old behavior.
[0]     validation_0-mlogloss:1.69426
[99]    validation_0-mlogloss:4.48642
```

In [18]:

```
print('----------------------TRAIN SET REPORT--------------------')
print("Open set RMSE:", np.sqrt(mean_squared_error(y_train, train_pred_xgb)))
print('Open set penalty matrix score:', score(y_train.values, train_pred_xgb))
print('Open set report:', classification_report(y_train, train_pred_xgb))
print('----------------------OPEN SET REPORT--------------------')
print("Open set RMSE:", np.sqrt(mean_squared_error(y_test, open_pred_xgb)))
print('Open set penalty matrix score:', score(y_test.values, open_pred_xgb))
print('Open set report:', classification_report(y_test, open_pred_xgb))
print('----------------------HIDDEN SET REPORT--------------------')
print("Hidden set RMSE:", np.sqrt(mean_squared_error(y_hidden, hidden_pred_xgb)))
print('Hidden set penalty matrix score:', score(y_hidden.values, hidden_pred_xgb))
print('Hidden set report:', classification_report(y_hidden, hidden_pred_xgb))
```

```
----------------------TRAIN SET REPORT--------------------
Open set RMSE: 1.196822021310681
Open set penalty matrix score: [-0.47421158]
Open set report:               precision    recall  f1-score   support

           0       0.78      0.72      0.74    168937
           1       0.83      0.97      0.89    720803
           2       0.76      0.35      0.48    150455
           3       0.84      0.58      0.69     56320
           4       0.90      0.90      0.90     10513
           5       0.79      0.19      0.30      1688
           6       0.79      0.55      0.65     33329
           7       0.93      0.89      0.91      1085
           8       0.99      1.00      0.99      8213
           9       0.87      0.41      0.56      3820
          10       1.00      0.83      0.91       103
          11       0.75      0.78      0.76     15245

    accuracy                           0.82   1170511
```

In [20]:

```
from mpl_toolkits.axes_grid1 import make_axes_locatable
import matplotlib.colors as colors
facies_colors = ['#F4D03F','#7ccc19','#196F3D','#160599','#2756c4','#3891f0','#80d4ff','#87
facies_labels = ['SS', 'S-S', 'SH', 'MR', 'DOL','LIM', 'CH','HAL', 'AN', 'TF', 'CO', 'BS']


#Facies_color_map
facies_color_map = {}
for ind, label in enumerate(facies_labels):
    facies_color_map[label] = facies_colors[ind]

def pred_log(logs, well_num, facies_colors, n_pred):
    wells = logs['WELL'].unique()
    logs = logs[logs['WELL'] == wells[well_num]]
    logs = logs.sort_values(by='DEPTH_MD')        #Sorting log by depth
    cmap_facies = colors.ListedColormap(facies_colors[0:len(facies_colors)], 'indexed')

    top = logs.DEPTH_MD.min()
    bot = logs.DEPTH_MD.max()

    f, ax = plt.subplots(nrows=1, ncols=(12+n_pred), figsize=(15, 12))
    log_colors = ['black', 'red', 'blue', 'green', 'purple','black', 'red', 'blue', 'green'

    for i in range(7,18):
      ax[i-7].plot(logs.iloc[:,i], logs.DEPTH_MD, color=log_colors[i])
      ax[i-7].set_ylim(top, bot)
      #ax[i-7].set_xlim(logs.iloc[:,i].min(), logs.iloc[:,i].max())

      ax[i-7].set_xlabel(str(logs.columns[i]))
      ax[i-7].invert_yaxis()
      ax[i-7].grid()

    for j in range((-1-n_pred), 0):
      label = np.repeat(np.expand_dims(logs.iloc[:,j].values, 1), 100, 0)
      im = ax[j].imshow(label, interpolation='none', aspect='auto', cmap=cmap_facies, vmin=
      ax[j].set_xlabel(str(logs.columns[j]))

    divider = make_axes_locatable(ax[-1])
    cax = divider.append_axes("right", size="20%", pad=0.05)
    cbar=plt.colorbar(im, cax=cax)
    cbar.set_label((12*' ').join(['SS', 'S-S', 'SH', 'MR', 'DOL','LIM', 'CH','HAL', 'AN', '
    cbar.set_ticks(range(0,1)); cbar.set_ticklabels('')

    f.suptitle('WELL LOGS '+str(wells[well_num]), fontsize=14,y=0.94)
```

In [26]:

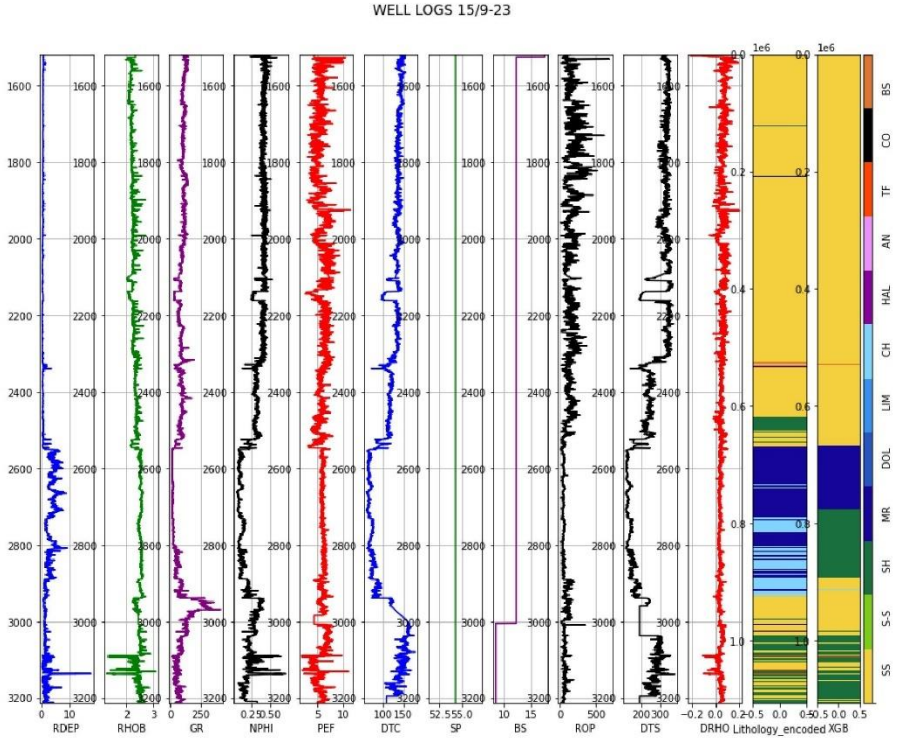
```
#Storing results
test_xgb = test_imp.copy()
hidden_xgb = hidden_imp.copy()
#Saving Results
test_xgb['XGB'] = open_pred_xgb
hidden_xgb['XGB'] = hidden_pred_xgb

test_xgb.to_csv('test_xgb.csv', index=False)
hidden_xgb.to_csv('hidden_xgb.csv', index=False)
```

In [27]:

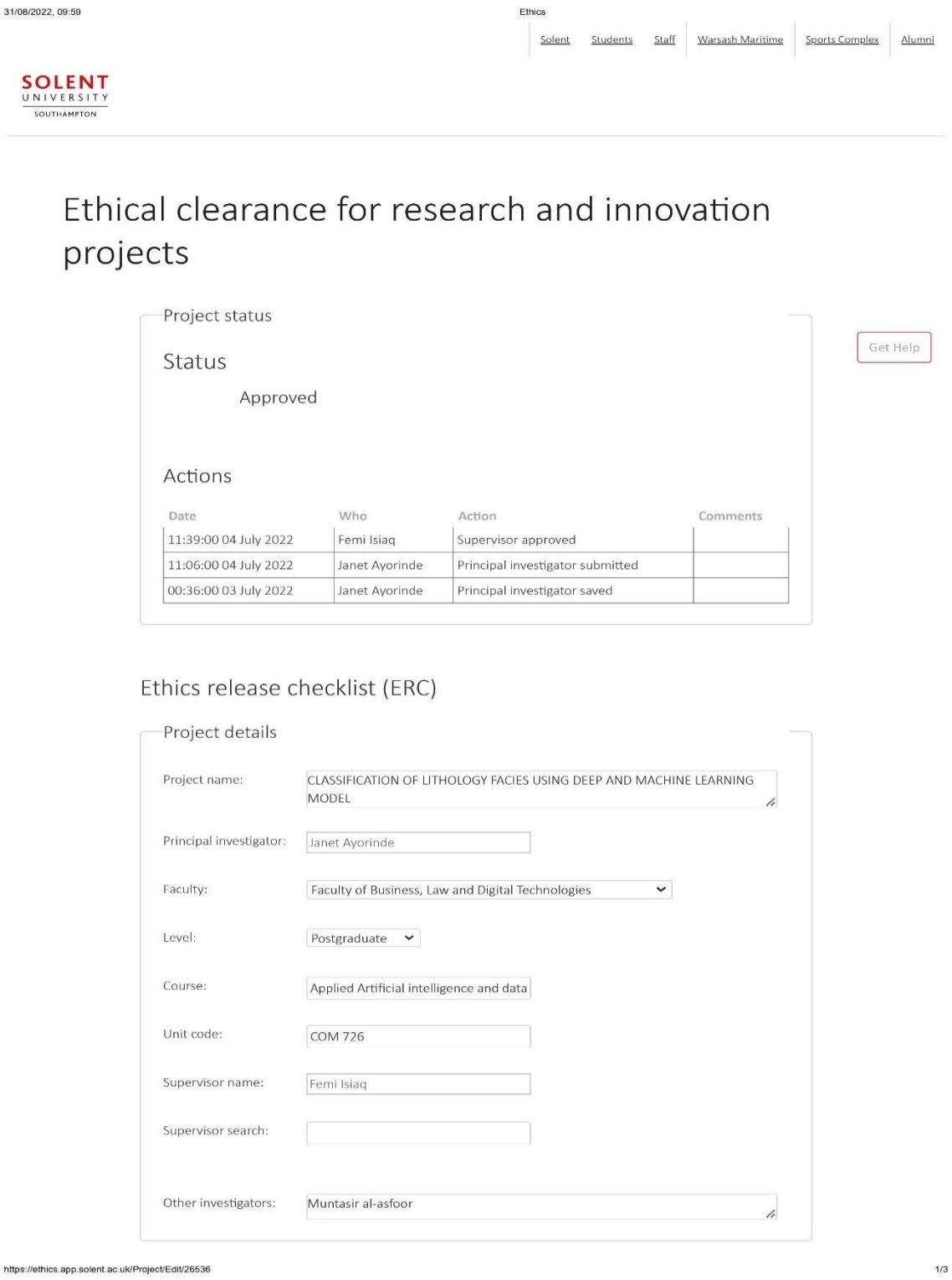
```
test_xgb = pd.read_csv('test_xgb.csv')
hidden_xgb = pd.read_csv('hidden_xgb.csv')
```

In [28]:

```
#Plotting predictions - HIDDEN DATASET
for i in range(1):
  pred_log(hidden_xgb, i, facies_colors, 1)
```



Appendix H: Ethics application



31/08/2022, 09:59 Ethics

Solent Students Staff Warsash Maritime Sports Complex Alumni

SOLENT UNIVERSITY SOUTHAMPTON

Ethical clearance for research and innovation projects

Project status

Status

Approved

Get Help

Actions

Date	Who	Action	Comments
11:39:00 04 July 2022	Femi Isiaq	Supervisor approved	
11:06:00 04 July 2022	Janet Ayorinde	Principal investigator submitted	
00:36:00 03 July 2022	Janet Ayorinde	Principal investigator saved	

Ethics release checklist (ERC)

Project details

Project name:	CLASSIFICATION OF LITHOLOGY FACIES USING DEEP AND MACHINE LEARNING MODEL
Principal investigator:	Janet Ayorinde
Faculty:	Faculty of Business, Law and Digital Technologies
Level:	Postgraduate
Course:	Applied Artificial intelligence and data
Unit code:	COM 726
Supervisor name:	Femi Isiaq
Supervisor search:	
Other investigators:	Muntasir al-asfoor

https://ethics.app.solent.ac.uk/Project/Edit/26536 1/3

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 [https://www.gov.uk/government/organisations/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 [https://ico.org.uk/for-organisations/guide-to-data-protection/]	○	◉

Get Help

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 endeavour 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 [https://ico.org.uk/for-organisations/guide-to-data-protection-404/] 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 [https://www.gov.uk/government/publications/prevent-duty-guidance/prevent-duty-guidance-for-higher-education-institutions-in-england-and-wales].

Get Help