University of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study program / Specialization: Petroleum Geosciences Engineering	Spring semester, 2020 Open
Writer: Khoirrashif Pratikna	…………………………………… (Writer's signature)
Supervisor: Arild Buland	
Thesis title: A study of Machine Learning application on net-to-gross prediction using seismic horizons. Case Study: The Brent Group - Statfjord Field	
Credits (ECTS): 30	
Keywords: Machine Learning Statfjord Field Brent Group Net-to-Gross Seismic Horizons	Pages : 50 + enclosures : 17 Stavanger, 15 July 2020

A study of Machine Learning application on net-to-gross prediction using seismic horizons. Case Study: The Brent Group - Statfjord Field

By

Khoirrashif Pratikna

MSc Thesis

Presented to the Faculty of Science and Technology

University of Stavanger

2020

Acknowledgements

First and foremost, I would like to thank my parents and my sisters to whom I dedicated my master thesis to. Thank you for sending me continuous support, constant prayers, and encouragement through various stages in my journey to finish the master degree.

My utmost gratitude to my supervisor Arild Buland for all the advices, excellent guidance, and for the opportunity to work on a such interesting and challenging topic. I also would like to express my gratitude to Evan Thomas Delaney for his continuous assistance and valuable discussion.

Special thanks to Equinor ASA for kindly providing the dataset and hardware for this thesis.

Last but not least, I would like to thank my Indonesian friends in Stavanger for all the cordial support and laughter.

Abstract

A study of Machine Learning application on net-to-gross prediction using seismic horizons. Case Study: The Brent Group - Statfjord Field

Khoirrashif Pratikna, The University of Stavanger

Supervisor: Arild Buland

As petroleum geosciences enter the era of big data, this field of study encompass difficult optimization and decision-making in real-world problems. The increasing number, difficulty, and scale of such specific problems has become too complicated for geoscientists to rely on a single discipline for effective solution. Machine Learning (ML) provides extensive capability to be the solution in this area.

This thesis, as a part of ongoing research, focuses on the application of various ML algorithms in predicting the net to gross value of the Brent Group reservoir zone in the Statfjord Field. For this purpose, several objectives were defined. As the first fundamental step, features were generated directly from the TWT and amplitudes of the respective seismic horizons. Secondly, predictive models were built from both training and testing phase using the features. The final task was estimating and mapping the value of net to gross property of the pre-defined reservoir zone. Furthermore, classification task and sand thickness prediction were also included as additional comparisons to the main task.

The results indicate outstanding performance demonstrated by Decision Tree and Random Forest algorithms despite the limitation on the dataset. Insufficient amount of data as well as data cleaning problems have been the main constraints in this study. This unarguably led to high variance in the data which yielded less accurate and less reliable prediction models.

The ML clearly have potential to accomplish the defined task better once the obstacles are handled properly in the future studies. Some improvements such as better data cleaning process, more involvement of well logs data, AVO inversion analysis, and utilization of more advance algorithms are strongly suggested in order to boost the models' performance.

Contents

List of tables

List of figures

1 Introduction

Petroleum Geosciences, as well as many other fields, is a domain which encompasses difficult optimization and decision-making in real-world problems. As a result, integration, big-data handling, uncertainty, and risk management are considered as fundamental issues in petroleum geosciences. The increasing number, difficulty, and scale of such specific problems has become too complicated for geoscientists to rely on a single discipline for effective solution. Consequently, establishing new concepts intended to decent integration of disciplines (e.g., petroleum engineering, geology, and geophysics), fusing data, reducing risk, and handling uncertainty have become top priority tasks in this field of study (Cranganu et al., 2015).

As petroleum geosciences enters the era of big data, machine learning (ML) provides extensive capability to be the solution in this area. Machine learning approaches are a set of algorithms which is possible to convert data to actionable intelligence. These techniques belong to a class of methods in which the solutions are principally derived from data instead of physics-based models (Nwachukwu, 2018).

Recent works show how ML approaches have been employed as problem-solving tool in oil and gas industry. Among various ML algorithms, artificial neural network (ANN) and support vector machine (SVM) are the most preferred when dealing with geoscience problems (Lary et al., 2016). An example of implementation of ANN was to evaluate bottom hole pressure (BHP) in multi-phase annular flow while under balanced drilling (UBD) operations (Ashena et al., 2010). ANN was also used in drilling hydraulics simulations to predict hydraulic pressure losses (Fruhwirth et al., 2006), as well as in drilling optimization in terms of investigating the effects of vibration parameters on rate of penetration (ROP) (Elahifar et al., 2012), and also for permeability prediction (Naeeni et al., 2010).

The SVM technique has also been considered as effective and accurate method with powerful prediction capability. This is confirmed by its potential to successfully estimate the lithofacies, and petrophysical properties such as porosity and permeability (Al-Anazi and Gates, 2010a,b,c,d)

1.1 Aim of the Study

As a part of ongoing research, the aim of the study is to obtain a better understanding of how machine learning perform prediction on net-to-gross value of the oil-bearing reservoirs in the Brent Group based on the structure and amplitude derived from seismic horizons.

1.2 Objectives

The Objectives of the study are defined as follows:

- Generating features from the respective horizons (TWT and amplitudes)
- Building training and testing predictive models by applying a number of machine learning algorithms (e.g. SVM, Decision Tree, Random Forest, and etc.)
- Estimation of net to gross property of the defined reservoir zone for the whole study area
- Determining which features have the most significant impact as well as the best machine learning algorithm for this study case

2 The Statfjord Field

Statfjord field is a producing oil field situated on the southwestern part of the Tampen Spur within the East Shetland Basin which is located in the 33/9 and 33/12 Norwegian sector (Fig. 2.1). The field was discovered in 1974 and started producing in 1979. The Statfjord field is considered as the largest oil field in the Northern North field due to its hydrocarbon content area which extends for 24 km by 4 km (Roberts et al., 1987).

The current owners of Statfjord field are Equinor Energy AS (44.37%, operator), Var Energi AS (21.37%), Spirit Energy Norway AS (19.77%), Spirit Energy Resources Limited (14.53%) (Norwegian Petroleum Directorate (NPD), 2020).

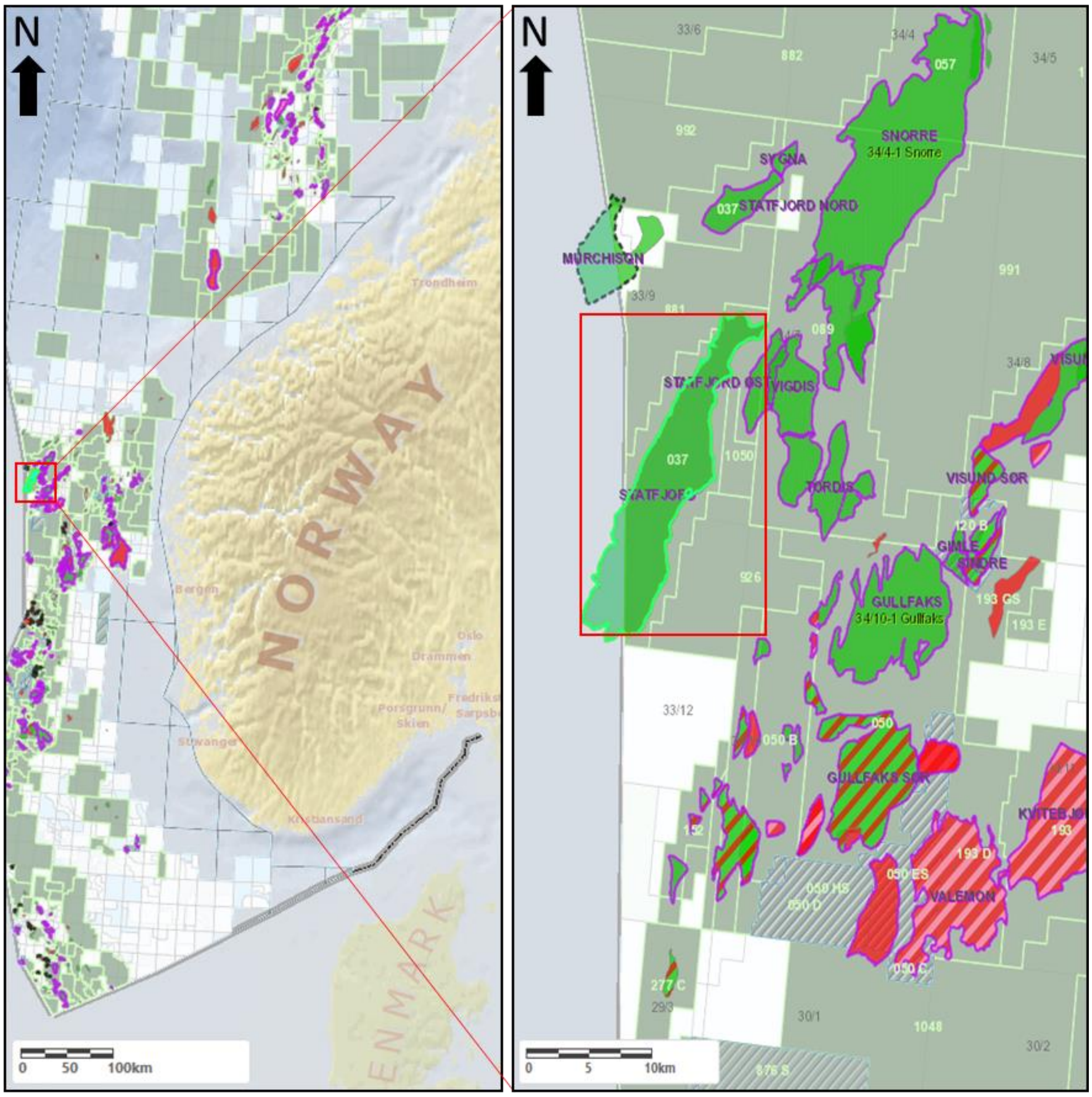


Figure 2.1: Location of the Statfjord Field (modified from www.npd.no).

2.1 Structural settings

The Statfjord field (Kirk, 1980) is located on the west border of the North Sea rift system in one part of a platform inside the East Shetland Basin (Gabrielsen, 1986; Gabrielsen et al., 1990) and approximately 220 kilometres northwest of Bergen (Fig. 2.2b). The East Shetland Basin is surrounded by several parts of the North Sea rift system such as The East Shetland Platform (south and west), the More Basin (north) and Tampen Spur (northeast) and the North Viking Graben (east) (Fig. 2.2a). The Statfjord is gently sloped toward the northwest (Fig. 2.3) and extends along the ridge of a trending fault block in NE-SW direction (Gibbons et al., 2003).

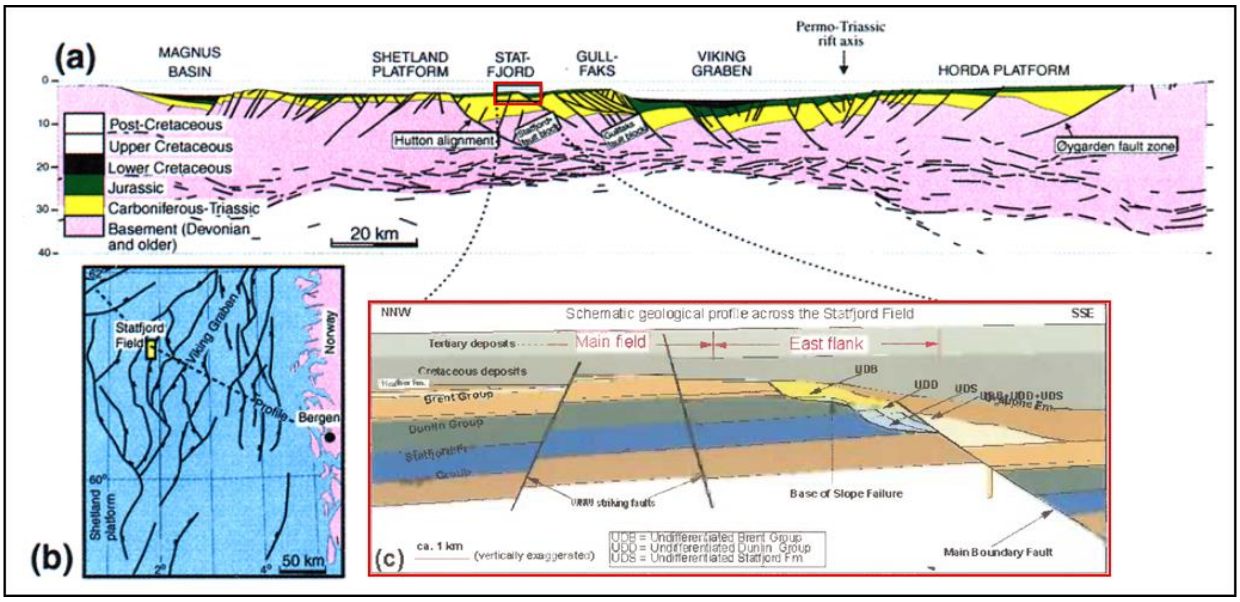


Figure 2.2: (a) Regional profile across northern North Sea and the Statfjord Field based on the work by Odinsen et al., in press (b), (b) fault map of the North Sea rift system around the Statfjord Field and (c) schematic cross-section of the Statfjord Field. (modified from Fossen et al., 1998; Hesthammer et al., 1999; and Gibbons et al., 2003)

There were at least two major rift events which occurred in the Statfjord field area after the Devonian thinning and regional stretching of the Caledonian crust (Hesthammer & Fossen, 1999). The Permo-Triassic rift, the first phase, formed the Viking Graben (Badley et al., 1984, 1988; Beach et al., 1987; Roberts et al., 1995). The second main rift phase (Brown, 1984; Thorne & Watts, 1989), which occurred in the latest middle Jurassic to earlier Cretaceous, developed a general extension in NW-SE direction (Roberts et al., 1990a,b). The Triassic and Jurassic reservoirs were deposited in a gradual rate as a result of relative sea level rise succeeding the second rift phase (Gibbons et al., 2003).

The structure of the Statfjord field dominantly consists of two sections (Fig. 2.2c). First, a relatively undistorted main field with dipping strata towards W-NW direction, and the second one, a highly distorted east flank area which underwent multiple phases of gravitational collapse towards east direction. The Cretaceous base is generally outweighed by multiple cross faults which dipping steeply in NW-SE direction over the main field area. The strike-slip deformation structures, which were formed in Tertiary, are then recognizable not only in the northern and central part of the field, but also in the hanging wall to the primary boundary fault (Gibbons et al., 2003).

The rotational block slides penetrate the reservoir layers and dominate the east flank area. Multiple phases of gravity block sliding occurred were found to be correlated with the middle to late Jurassic rift. These occurrences were linked to the tectonic activity on regional scale (Gibbons et al., 2003).

The rotated fault blocks along both margins of the Viking Graben, which formed during the rifting in Late Jurassic, are the most common trap for hydrocarbon in the Statfjord field area (Faleide et al., 2010).

2.2 Stratigraphy of the Statfjord field reservoir

The Brent group (Middle Jurassic) and the Statfjord formation (Upper Triassic-Lower Jurassic) (Fig. 2.3) are the fundamental reservoir formations in the Statfjord field with good to excellent reservoir properties (porosities range between 20-30%, and permeabilities in darcies) (Gibbons et al., 2003; Kirk, 1980). Hydrocarbon is also produced from the Cook formation from the Dunlin group (NPD, 2020). These reservoirs in general are found at a ranging depth from 2,500 to 3,000 meters and lie not only within an extensive fault blocks dipped westward, but also in some of the smaller blocks in the eastern flank area (NPD, 2020).

A general overview about the reservoir formations of the Statfjord field will be discussed in the section below.

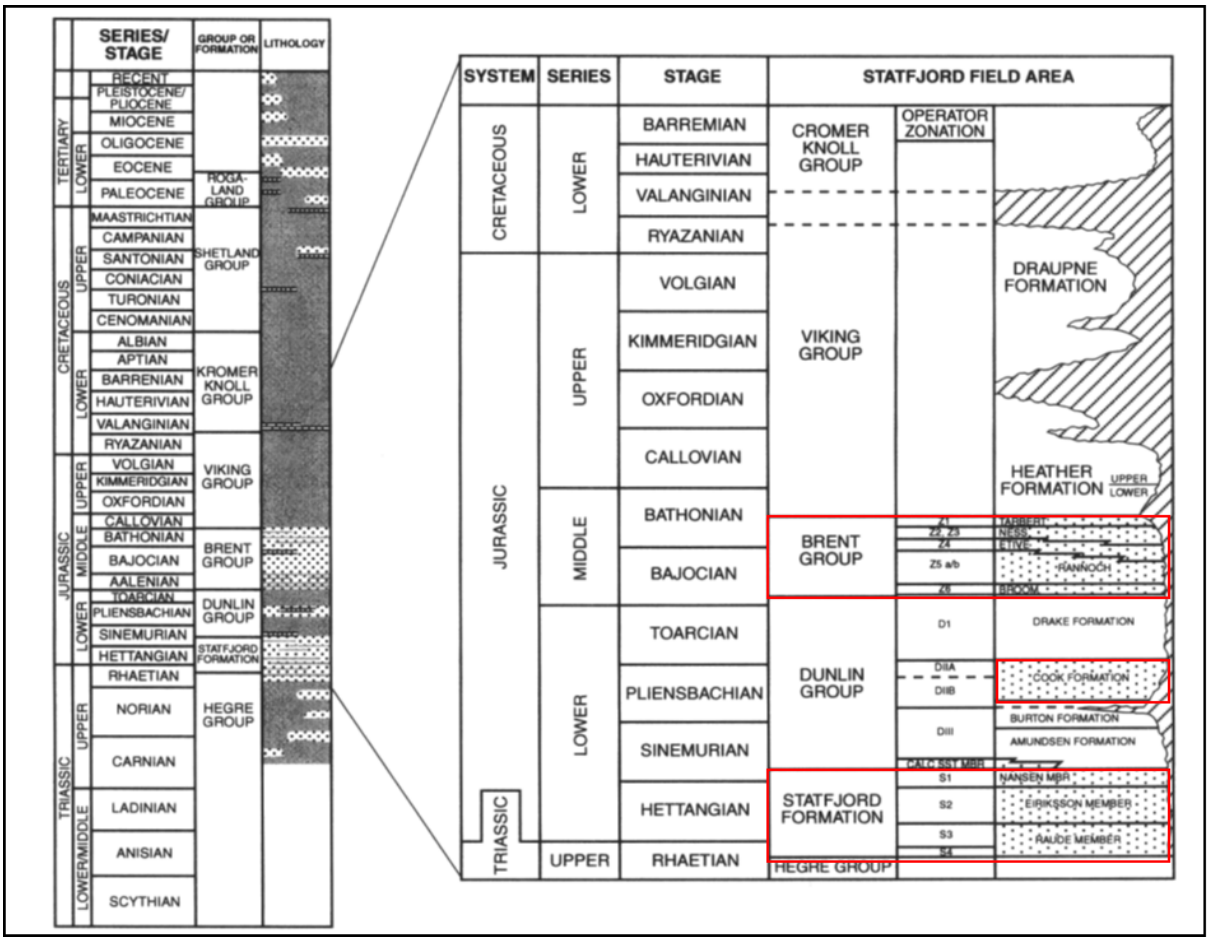


Figure 2.3: Stratigraphic column of the Statfjord Field (modified from Deegan & Scull, 1977; Vollset & Dore, 1984; and Hesthammer et al.,1999). The main reservoirs are highlighted (red).

2.2.1 Hegre group

The Hegre group was deposited during the Triassic and is confirmed to be the oldest strata drilled in the Statfjord field area at depth around 4,572 m (Kirk, 1980). The lithology of the group is characterized by interbedded intervals of sandstone, claystone, and shale correlated with primarily continental sandstone or shale/claystone sequences (Hesthammer et al., 1999). The calcaerous cement and clay matrix within the lithology lead to the generally poor reservoir quality of this group (Kirk, 1980).

2.2.2 Statfjord formation

The Statfjord formation was deposited from the upper Triassic (Rhaetian) to the lower Jurassic (Sinemurian) and appears to unconformably overlie the Triassic Hegre group on the regional scale (Kirk, 1980). The transition from the Hegre group and to the Statfjord formation is unclear and difficult to be interpreted (Kirk, 1980; Gibbons et al., 2003). This is caused by two aspects (Kirk, 1980). First, the insufficient occurrence of flora and fauna, and second, the poor log correlation due to indistinguishable changes in lithology.

The Statfjord formation contains interlayered sandstone/siltstone and shale, and has thickness range between 150 to 300 meters in the Statfjord field. Based on the depositional environment the formation is differentiated into three members, namely the Raude and Eirikkson member which represent the fluvial deposits, and the Nansen member which is interpreted as a transgressive marine sheet sand overlaying the alluvial flood basin (Hesthammer et al., 1999)

2.2.3 Dunlin group

The Dunlin group, which was deposited during the Lower Jurassic (late Sinemurian) to the Upper Jurassic (Bajocian), contains four formations. The Amundsen (oldest) and the Burton formation are characterized by shallow marine shale, claystone and siltstone. The Cook formation consists of silt and tidal-influenced shallow marine sandstones. The Drake formation (youngest) comprises shallow marine shale and siltstone. The thickness of the Dunlin group varies from 230 to 260 meters (Hesthammer et al., 1999).

2.2.4 Brent group

The Brent group has thickness between 180-250 meters. The formation was deposited during the Middle Jurassic (early Bajocian-mid-Bathonian), and is divided into five formations, namely the Broom, Rannoch, Etive, Ness, and Tarbert formations. The lithology of the group are mainly sandstone, siltstone, shale, and coal deposits from a prograding delta system toward the north direction. The Broom formation, the oldest unit, is characterized by storm deposits and small distal bar build-ups overlaying a shallow marine platform. The Rannoch formation is dominantly composed by sandstone deposits from pro-delta, delta front, and ebb-tidal environments. The Etive formation has coarser and cleaner sandstone. This is due to its depositional environment were in tidal inlet, upper shoreface foreshore, and lagoon barrier settings. The Ness formation was deposited in a delta plain setting. Consequently, this unit contains sandy channel deposits, shale, and coal. The Tarbert formation, the youngest unit in the Brent group, comprises shallow marine sands (Hesthammer et al., 1999).

3 Machine Learning

The terminology “machine learning” refers to the study that involves statistical approaches to give computer systems the ability to learn from data without being explicitly programmed. In another explanation, machine learning can also be defined as the study of software artifacts which utilizes past experience in order to generate future decisions. The main purpose of machine learning is to automate decision making processes by generalizing the prior experiences. Machine learning needs a collection of data, which often called ”training set”, to be trained into the algorithms in order to obtain experience. The performance of the “trained” algorithms are then evaluated using a set of data called “test set”. One basic example of machine learning application is spam filtering. The spam filters learn to differentiate and classify new messages by recognizing thousands of emails that have been previously marked as either spam or ham.

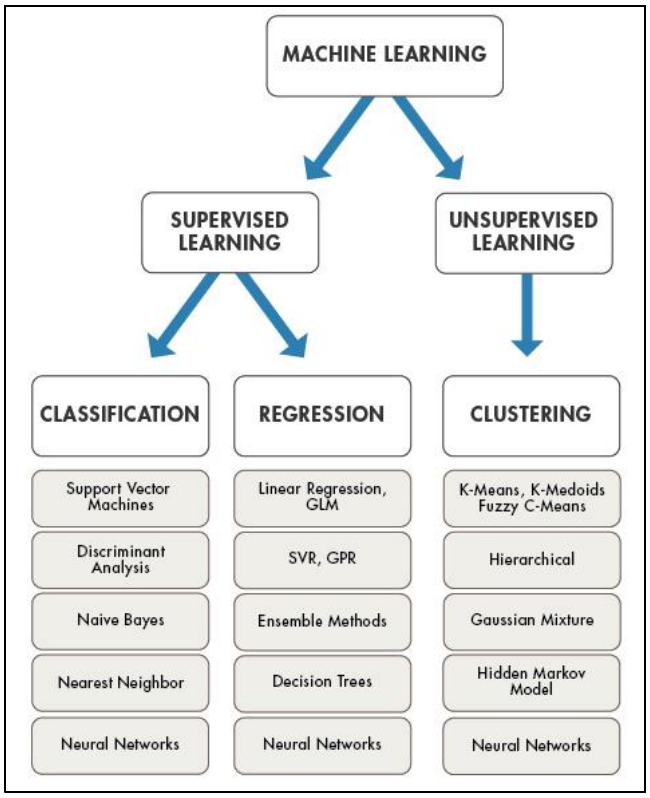


Figure 3.1: Machine learning types and algorithms (modified from www.mathworks.com)

According to the types of problems encountered, there are at least two types of learning are identified (Fig.3.1.):

<u>Supervised Learning</u>

In supervised learning problems, the user provides pairs of inputs as well as the desired outputs (labels) to an algorithm, and let the algorithm finds a way to generate the desired output given an example. There are two main tasks in supervised learning, namely classification and regression.

In classification problems the algorithms are required to predict discrete values for the outputs from one or more predictors. The algorithms must then classify the new inputs or observations into the most probable label or category. As for regression tasks, the algorithms are required to predict the value of a continuous output.

Examples of supervised learning algorithms are including:

1. Support Vector Machine (SVM)
2. Naïve Bayes
3. Nearest Neighbour
4. Decision Tree (DT)
5. Ensemble Methods
6. Neural Networks
7. Random Forest

Unsupervised Learning

In unsupervised learning, an algorithm does not learn from labelled data. In this setting, only the input data is known while there is no defined output. The unsupervised learning algorithms will attempt to infer patterns within the data.

The most common task for unsupervised learning algorithms is to discover groups within the training set based on their similarities among each other, or often called clustering.

Several algorithms in unsupervised learning:

1. K-means
2. Hierarchical Clustering
3. Fuzzy c-means
4. Self-organizing Maps (SOM)

3.1 Support Vector Machine/Regressor (SVM/SVR)

The Support Vector Machine (SVM) (Fig.3.2.) is an algorithm which originally was built to solve classification problem, while the counterpart, SVR, is basically a modified version of SVM that capable to predict continuous value for regression problem by using kernel functions (Vapnik, 1995). SVM classifies data maximizing the distance between the separating hyperlane (decision boundary), or so-called margin, and the training samples that are closest to this hyperlane or support vectors.

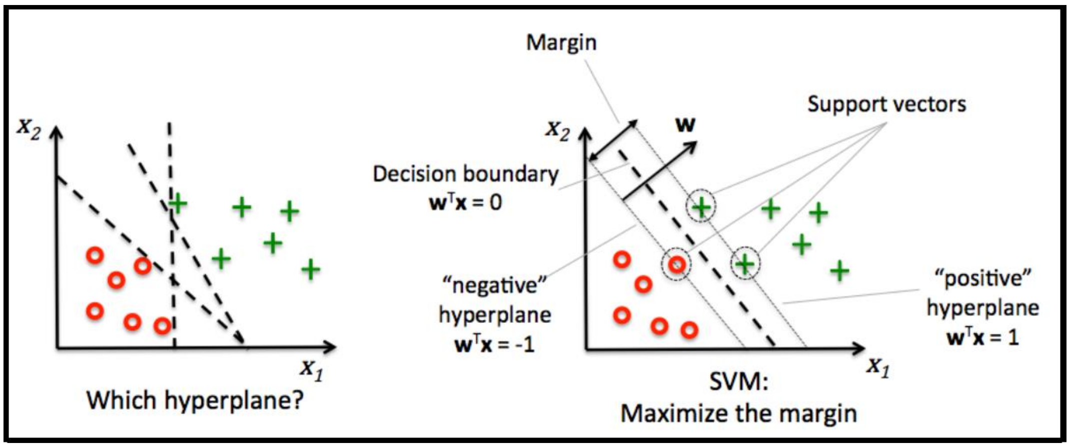


Figure 3.2: Support vector machine (SVM) classifier

Maximizing the margin in the decision boundaries will make the generalization error lower, however, models with small margins are more likely to overfitting. The following section explains about margin maximization process.

First, the positive and negative hyperlanes that are parallel to the decision boundary will be expressed as follows:

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{pos} = 1 \tag{1}$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{neg} = -1 \tag{2}$$

From the equations above w represents a set of weights, x are the input values, while $\boldsymbol{w}$ and $\boldsymbol{x}$ represent the dot products of w and x. If the linear equations (1) and (2) are subtracted from each other, we will obtain:

$$\boldsymbol{w}^T(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg}) = 2 \tag{3}$$

The equation (3) is the normalized by the length of the vector w, which is defined as:

$$||\boldsymbol{w}|| = \sqrt{\sum_{j=1}^{m} w_j^2} \tag{4}$$

In the end, we end up with the following equation:

$$\frac{\boldsymbol{w}^T(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg})}{||\boldsymbol{w}||} = \frac{2}{||\boldsymbol{w}||} \tag{5}$$

The left side of the equation (4) represents the distance between the positive and negative hyperplane or the margin we want to maximize. By maximizing $\frac{2}{||w||}$, the objective function of the SVM now becomes the maximization of this margin if the samples are classified correctly under these conditions:

$$\begin{gathered} w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \; if \; y^{(i)} = 1 \\ w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} < -1 \; if \; y^{(i)} = -1 \end{gathered} \tag{6}$$

The $x^{(i)}$ from the equations are the training samples, while $y^{(i)}$ are the predicted class labels. These constraints basically mean that all negative samples should be positioned on one side of the negative hyperplanes, while all positives samples should be positioned behind the positive hyperlane. The equations above can also simply be written as:

$$y^{(i)}\left(w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right) \geq 1 \forall_i \tag{7}$$

The SVM algorithm works best if the data is linearly separable. However, when dealing with nonlinearly separable case a parameter called the slack variable (ε) (Vapnik, 1995) needs to be included in the algorithm and which then leads to what it is called soft-margin classification. The argument for presenting the slack variable is because the linear constraints need to be adjusted for nonlinearly separable data to allow convergence of the optimization when the missclassifications are present. Consequently, the linear constraints will be conditioned as follows:

$$\begin{gathered} \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \; if \; y^{(i)} = 1\text{-}\ \varepsilon^{(i)} \\ \boldsymbol{w}^T \boldsymbol{x}^{(i)} < -1 \; if \; y^{(i)} = 1\text{+}\ \varepsilon^{(i)} \end{gathered} \tag{8}$$

As a result, the new objective to be minimized (subject to the prior constraints) is:

$$\frac{1}{2}\,||\boldsymbol{w}||^2 + C\left(\sum_i \varepsilon^{(i)}\right) \tag{9}$$

By using the variable C, the penalty for misclassification can now be controlled. The larger the values of C, the larger the error penalties will be, and so is the opposite. The parameter C is then used to control the width of the margin (Fig.3.3). The increasing values of C correspond to the increasing bias and decrease the variance of the model.

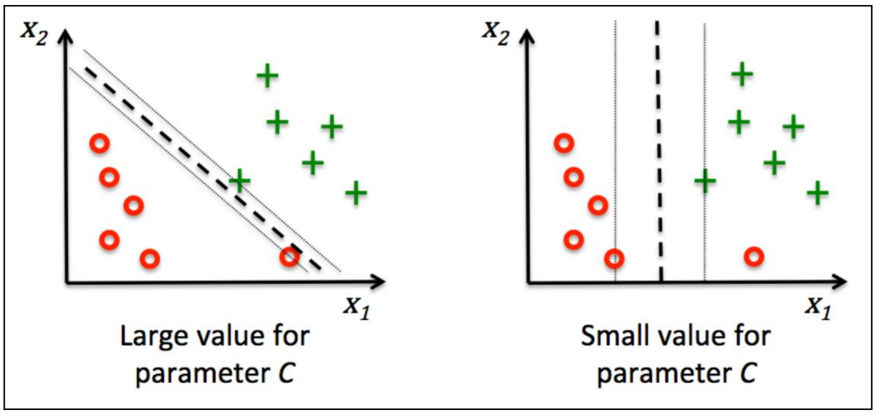


Figure 3.3: The influence of parameter C on SVM

3.2 Decision Tree

The Decision Tree (DT) belongs to the category of supervised learning algorithm which works for both continuous as well as categorical output variables. This algorithm is capable of handling classification and regression tasks. The DT builds classification and regression models in the structure of a tree (Fig.3.4) in order to either categorize (for classification) or to predict (for regression) data to produce meaningful outcome.

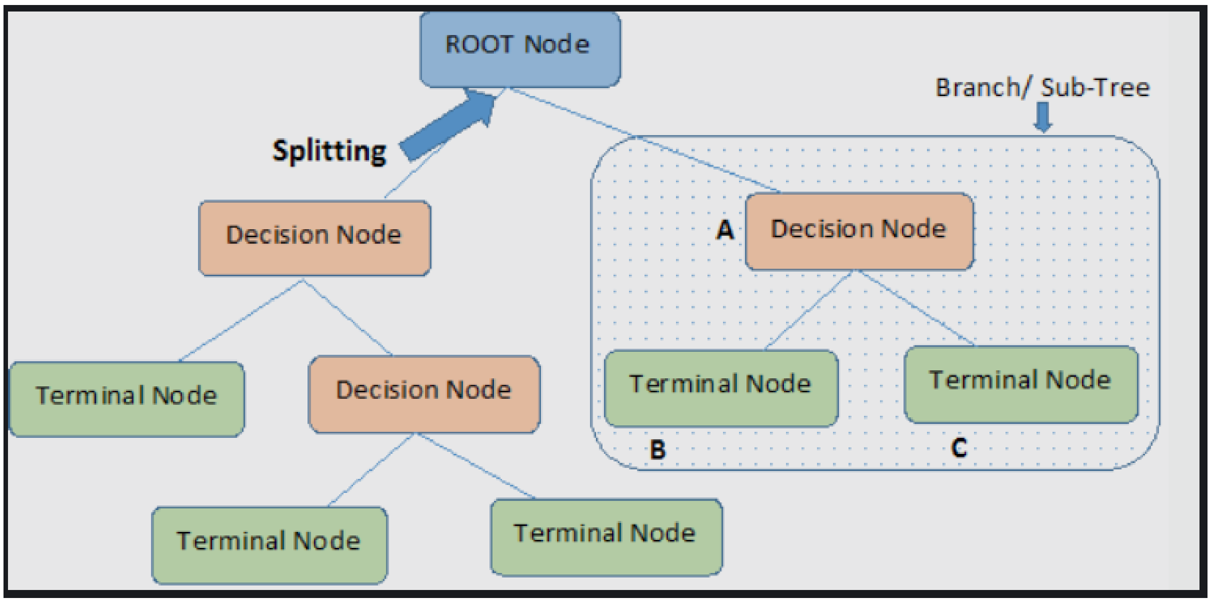


Figure 3.4: The Generic structure of DT

These are some basic terminology used in DT to be familiarized with:

- Root Node: A node which represents the total population or sample and later on will be divided into two or more homogeneous sets
- Splitting: A process of dividing a node into two or more sub-nodes
- Decision Node: A node which decides if a sub-node splits into further sub-nodes
- Leaf/Terminal Node: Nodes which do not split
- Pruning: A process of removing sub-nodes of a decision node, or often considered as the opposite process of splitting
- Branch/Sub-Tree: A sub section of the entire tree
- Parent and Child Node: Parent node is a node which is divided into sub-nodes whereas sub-nodes are the child of parent node

A decision tree generates estimation by basically asking a series of questions to the data which all are in a True/False form. Each True/False answer ends with separate branches and it will eventually lead to a prediction or leaf node no matter the answers to the questions.

In a simple manner, the steps to solve a problem using DT are mentioned as follows:

i. Put the best attribute of the dataset at the root node
ii. Split the dataset into subsets such that each subset contains the homogenous data, or in other words it contains having same value for an attribute
iii. Repeat step i and ii on each subset until leaf nodes are found in all the branches of the tree

In a regression problem, DT normally use mean squared error (MSE) to decide to split a node in two or more sub-nodes. To make it easier to understand, consider building a binary tree decision by:

i. Pick a variable and its value to split on such that two groups are as different from each other as possible
ii. For each group, the MSE will be calculated separately

iii. Calculate the average of the MSE between the two groups
iv. Repeat step i to iii for other variables
v. Compare the average MSE among the variables to determine the best split. The best split should be the one with the smallest MSE

Finally, the DT produces predictions by obtaining the average of the value of the dependant variable in the terminal or leaf node. This is done after running the dataset through the entire tree assessing all the questions until it reaches the leaf node. One thing to be noted is that the DT is unable to make accurate predictions if the 'test data' are unrelated with the trained data. Or in other word, it is unable to extrapolate to any kind of data it has not seen before.

Some advantages of DT are:

- Works well for non-linear dataset
- Easy to understand and interpret
- Less data preparation required

While some drawbacks are:

- Prone to overfit
- Cannot extrapolate
- Can be unstable when the data variance is big

3.3 Random Forests

The Random Forests (RF) algorithm is considered as one of ensemble learning methods and is confirmed to be effective for both classification and regression especially when dealing with large datasets. This algorithm utilizes decision trees (DT) as the building block. The rationale behind RF is because DT tends to overfit the training data.

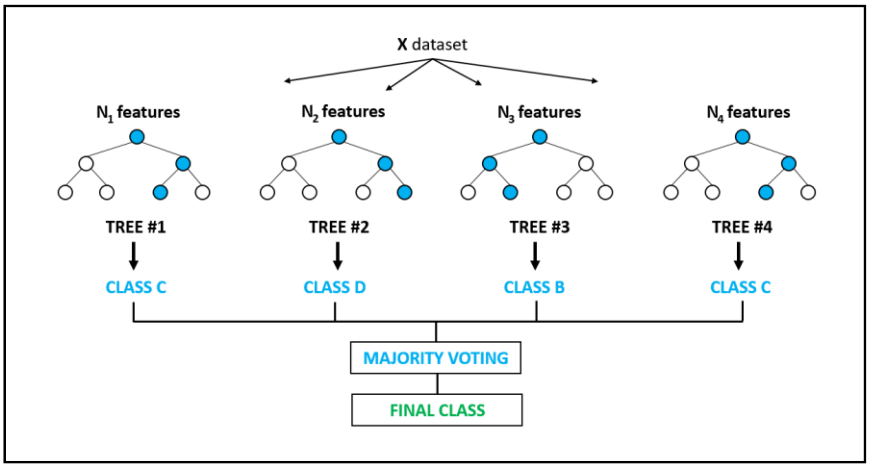


Figure 3.5: Illustration of RF algorithm structure

RF, which essentially consist of many trees (Fig. 3.5.), will overfit in many different ways on each tree. In order to solve this problem, RF reduce the amount of overfitting by making average of their results. To apply this procedure, each tree in RF should be able to make predictions decently and also not similar to other trees. The RF algorithm is simply comprised as follows:

i. Provide a random bootstrap sample of *n* size which should be selected randomly from the training set (with replacement)
ii. Generate a decision tree out of the bootstrap sample, and each node should:
    a. Select *d* number of features (without replacement)
    b. Separate the node based on the feature that has the best objective function (e.g. by maximizing the information gain)
iii. Do iterations for procedure i and ii for *k* times
iv. Aggregate the prediction by each tree to assign the class label by majority vote

The idea behind this algorithm as well as other ensemble methods is to merge weak predictors or learners to build a more robust model, a strong predictor or learner. The strong predictor will have a better generalization error and have lower tendency to overfit.

3.4 Previous work on Machine Learning application in Petroleum Geosciences

Both SVM and ANN are the two commonly used algorithms to solve geoscience problems. Some examples of their implementation are given below:

Naeeni et al. (2010) utilized the Feed-forward artificial neural networks (FF-ANN) with backpropagation to predict the permeability of reservoirs. The parameters that were used in this study comprised of depth, true conductivity (CT), sonic travel time (DT), neutron porosity (NPHI), bulk density (RHOB), spectral gamma ray (SGR), northing of well, easting of well, water saturation, and flow zone index (FZI). There were three hidden layers with 13, 10, and 1 neurons included in the networks. In this algorithm, the well log data and another parameter called rock quality index (RQI) were set as the input, while the permeability as the output. Prior to this procedure, different hydraulic flow units (HFU) were determined in order to determine the FZI values and, later on, permeability of various rock types. The final results showed the FF-ANN algorithm presented convincing performance in predicting permeability values of uncored wells. This was supported by the Pearson's correlation coefficient of 0.85 (from range between -1 to 1) in the validation phase.

Al-Anazi and Gates (2010a,b,c,d) applied SVM and compared its potential with back propagation neural network (BPPN) to predict the Poisson's Ratio and Young's Modulus of reservoir rock. In general, even though the neural network algorithms are capable of resolving nonlinear problem well, however, they need extensive training to enhance the network structure. One more issue when conducting the neural network algorithms is that the regression model results may overfit the unseen data. On the other hand, SVM successfully generalize and converge a global optimal solution. These studies included various parameters such as core-derived porosity, minimum horizontal stress, pore pressure, overburden stress, bulk density, compressional wave velocity (Vp), and shear wave velocity (Vs). To prevent overfitting, cross-validation was done ten times to obtain the optimal parameter to manage the trade-off between the model bias and variance. Eventually, the results demonstrated that SVM was better both in learning and prediction capabilities compared to BPPN. SVM produced a superior Poisson's ratio prediction, and also showed a faster decrease of error prediction as the training data developed.

4 Data

For this study, the data is mainly focused on multiple 3D seismic cubes and 172 wells which is provided by Equinor ASA. The details will be presented briefly below.

4.1 Seismic

The seismic data used in this study, which is named ST9703RZ16, was acquired in 1997 by WesternGeco and reprocessed in 2016-2017. This data covers the Statfjord main field and the North flank and contains multiple partial stacks (near, mid, and far stacks). The seismic survey adopted SEG reverse polarity, which means an increase in acoustic impedance corresponds to a negative amplitude (red-trough, whilst blue-peak indicates positive amplitude). The summary of the seismic data will be provided in Table 1 and Table 2 below.

Table 1. Provided seismic data summary

Seismic Survey	2D/3D	Acquisition Year	Coordinate Reference System	Latest processing	Other Notes
WG_ST9703	3D	1997	ST_ED50_UTM31N_P23031_T1133	2016	Single-component deghosthing, PSTM, PSDM, partial stacks, pre-stacks gathers

Table 2. Partial stacks of the seismic data

Seismic Polarity	Stack	Angle (˚)
SEG-Reverse Polarity	Near	13.5
	Mid	22.5
	Far	31.5
	Ultra Far	40.5

4.2 Well

Initially, there are hundreds of wells were provided, however only 172 wells are eligible to be used in this study due to several conditions which will be explained in Section 5.1. Wireline logs such as gamma ray (GR), spontaneous potential (SP), density (RHOB), neutron (NEU), resistivity, and sonic logs are also included.

4.3 Horizon

In addition to the seismic and well data, some of the key interpreted horizons are also provided at the reservoir interval in the study area such as:

- STATOIL+ST03M01_ffobc+Balder_Top+Time+2007+ob_despike_int_xyt.dat - Top Balder Fm. horizon
- evde_BCU_AMAP2018_time_structure_xyt.dat - BCU horizon

5 Thesis Workflow

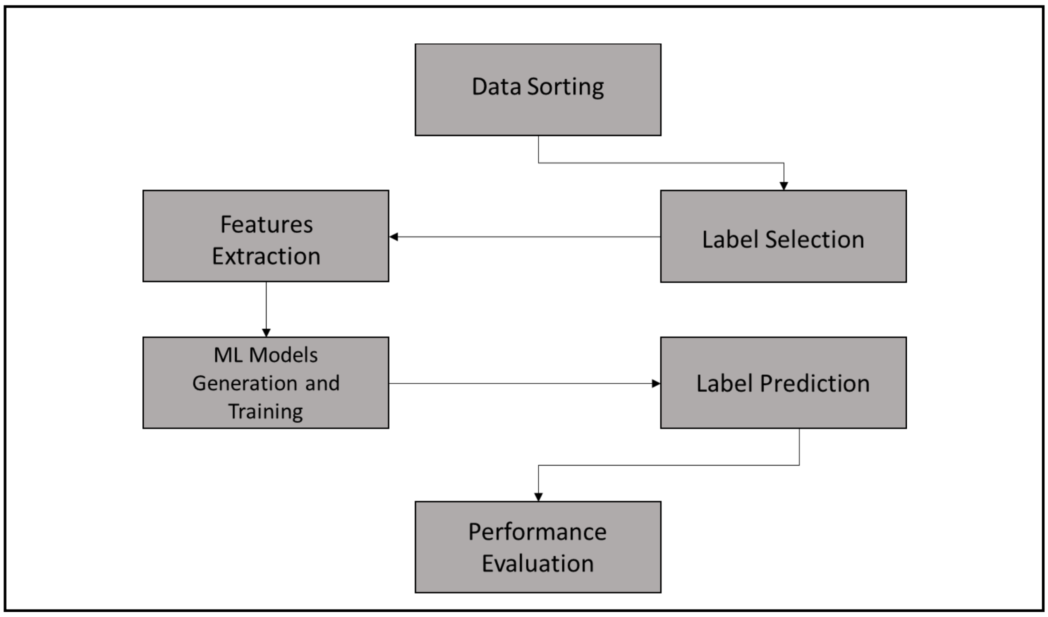


Figure 5.1: General workflow for this study

Several methods were carried out in this study. Figure 5.1 shows a generic workflow which contains an overview of the methods that were implemented in this study. The initial procedure was sorting out the horizons and wells to make them eligible for further steps in the study. Features were then extracted from the sorted data and later on followed by assigning label or target to be predicted. The results of the previous steps provided the optimal input for the next step, which was generating and training the Machine Learning models. Thereafter, the models were used to predict the assigned label or target, and finally the performance evaluation was measured.

5.1 Data Sorting

Data sorting is a fundamental step prior to performing ML algorithms. This procedure was meant to specify the scope of the study and to make the data fit for the features extraction.

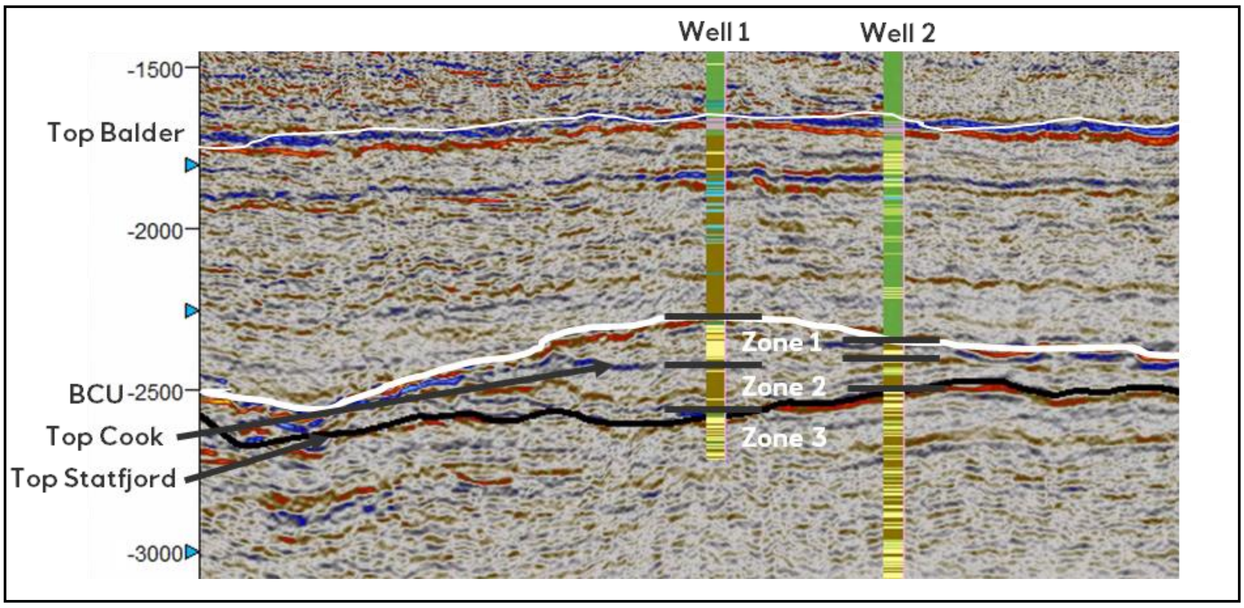


Figure 5.2: Key surface picks divided the three zones

Initially, in this study, the Statfjord field seismic data was divided into three zones (Fig.5.2) and only one selected zone that will continue to the next processes. The three zones were defined as follows:

- Zone 1: the area between the BCU and the Top Cook Fm.
- Zone 2: the area between the Top Cook Fm. and the Top Statfjord Fm.
- Zone 3: the area between the Top Statfjord Fm. and the end of the wells

Considering the amount of data needed for the next procedures, the selected zone should have the most penetrated wells among the others. Therefore, Zone 1 was chosen since it has 172 wells penetrated both the BCU and the Top Cook Fm.

5.2 Label Selection

In ML perspective, labels are simply defined as the variables that one is trying to predict or forecast. In this study, net-to-gross was selected as the label.

The net-to-gross has been a key factor when calculating original oil in place (OOIP) volumetrics. In other words, the net-to-gross indicates the producible hydrocarbon zones within the reservoir for further exploitation. The net-to-gross is simply explained as the total amount of sand divided by the total thickness of the reservoir interval. The outcome of the net-to-gross calculation is a fraction ranges from 0 to 1, which 0 represents non-producible reservoir and 1 represents potentially whole producible reservoir intervals. In this study, the practical steps of calculating the net-to-gross is shown in Figure 5.3. The whole processes of the net-to-gross calculation were computed in Python using several libraries such as Pandas (dataframe manipulation), NumPy (numerical operation), Glob (file and folder reader), and Lasio (well files reader).

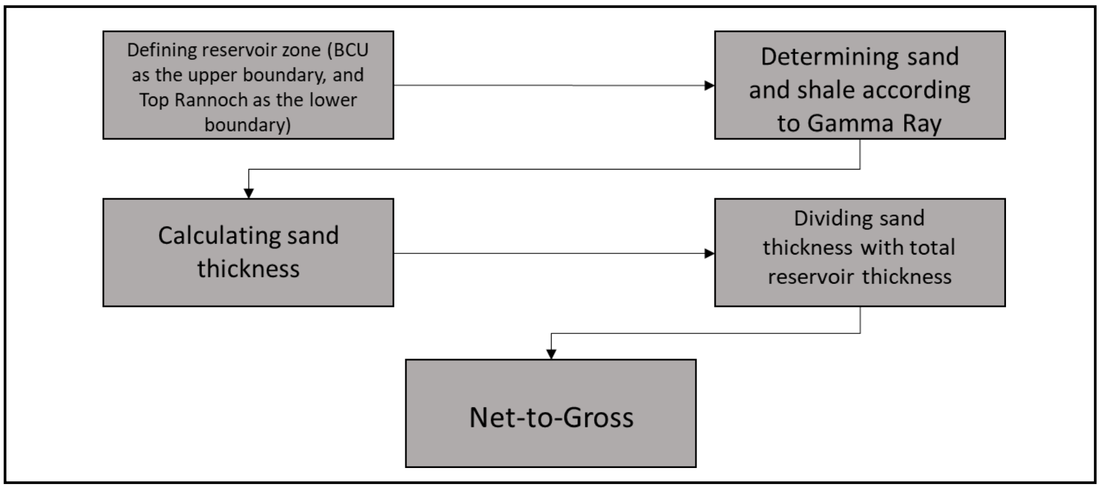

Defining reservoir zone (BCU as the upper boundary, and Top Rannoch as the lower boundary)
Determining sand and shale according to Gamma Ray
Calculating sand thickness
Dividing sand thickness with total reservoir thickness
Net-to-Gross

Figure 5.3: Net-to-gross calculation steps

The first step in calculating net-to-gross was defining the reservoir zone. In this case, the selected zone was Zone 1 which is the area between the BCU and the Top Cook Fm. The zone boundaries were defined by the surface picks obtained from Petrel. However, due to the unavailability of the Top Cook Fm surface picks in the data, the Top Rannoch was chosen instead.

The next following step was determining the ‘sand’ and ‘shale’ based on the Gamma Ray (GR) level. In this case, the sand was defined by lower GR level (GR<70) while shale was defined by higher GR level (GR>= 70). Then, the thickness of sand within the reservoir zone was calculated according to the prior definition. This was done for all wells.

Subsequently, the sand thickness was then divided by the total thickness of the reservoir zone to obtain the net-to-gross value. And this was also done for all the wells. This process yielded one single net-to-gross value for each well, and as a consequence, there were 172 net-to-gross values acquired for all the wells.

5.3 Features Extraction

In ML terminology, features are defined as measurable properties or variables of one object. Features are fundamental building blocks of the datasets which later on will be used as input in the system. Selecting and understanding the features are very important since they have a major impact on the quality of the insights one will gain when employing ML.

After the data sorting and the label selection, features extraction was performed to obtain the appropriate variables which contain useful information and also represent the condition of the real data. The overall workflow of this procedure is shown in Fig.5.4.

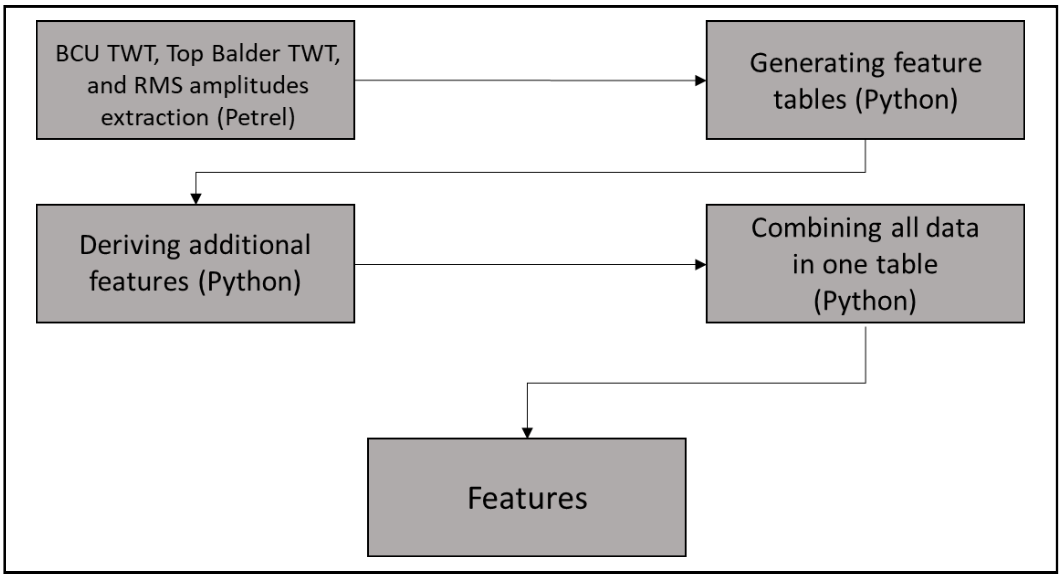

BCU TWT, Top Balder TWT, and RMS amplitudes extraction (Petrel)
Generating feature tables (Python)
Deriving additional features (Python)
Combining all data in one table (Python)
Features

Figure 5.4: Workflow for features extraction

There were four main features included for the initial round. These features are quantitative data which initially were extracted from Petrel and then were processed in Python using Pandas and NumPy libraries. The features which shown in Figure 5.5 are:

- Top Balder TWT (two-way time)
- BCU TWT (two-way time)
- BCU RMS near amplitude
- BCU RMS far amplitude

The seismic TWT for BCU and Top Balder Fm. were included in the features since they correspond to both the depth and velocity of the two formations. The root mean square (RMS) amplitude was also used due to its capability to produce hydrocarbon indicators by directly measure the reflectivity in zone of interest. Therefore, utilizing these variables as features will hopefully be a good combination when predicting the defined label.

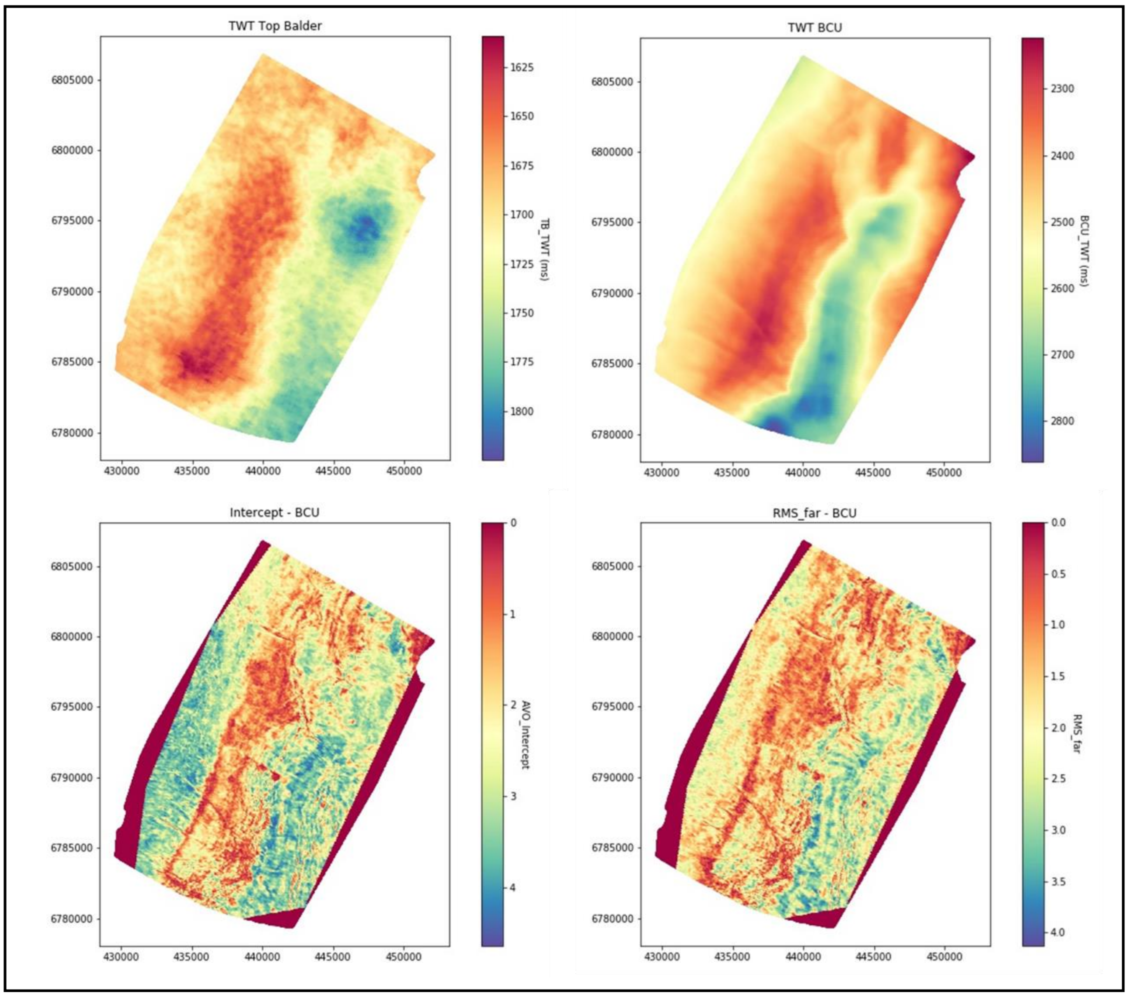
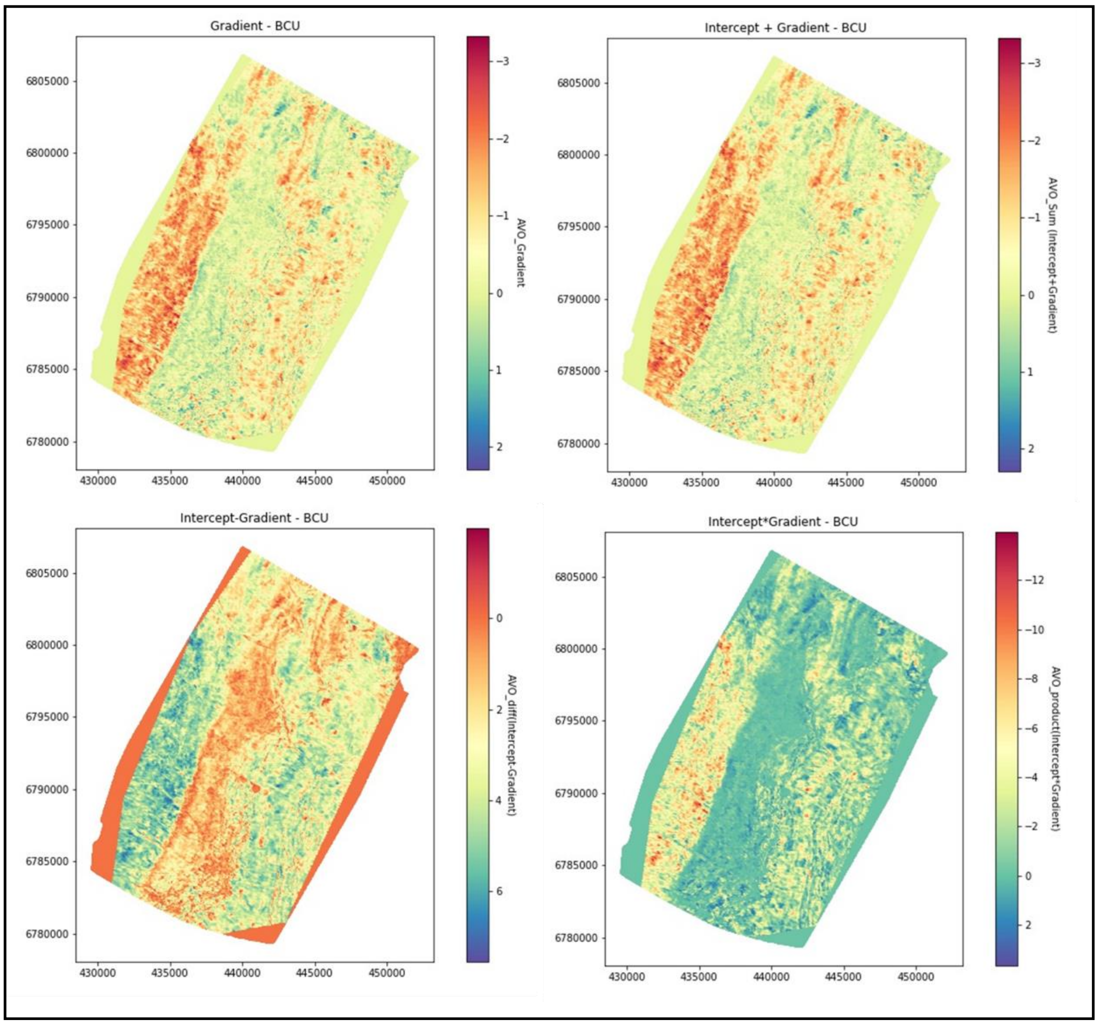


Figure 5.5: Maps of TWT and RMS amplitudes of the study area

As for the second round, there are four more features were added. These additional features were derived from the RMS amplitudes from the initial features using basic computation in Python. In ML, this process is called *feature engineering*. Feature engineering is a process of generating new features from the existing ones. The purpose feature engineering is mainly to improve the performance of ML models as well as to increase the predictive power of ML algorithms. The additional features are:

- Gradient: RMS far amplitude - RMS near amplitude
- AVO Product: RMS near amplitude * Gradient
- AVO Summation: RMS near amplitude + Gradient
- AVO Difference: RMS neat amplitude - Gradient

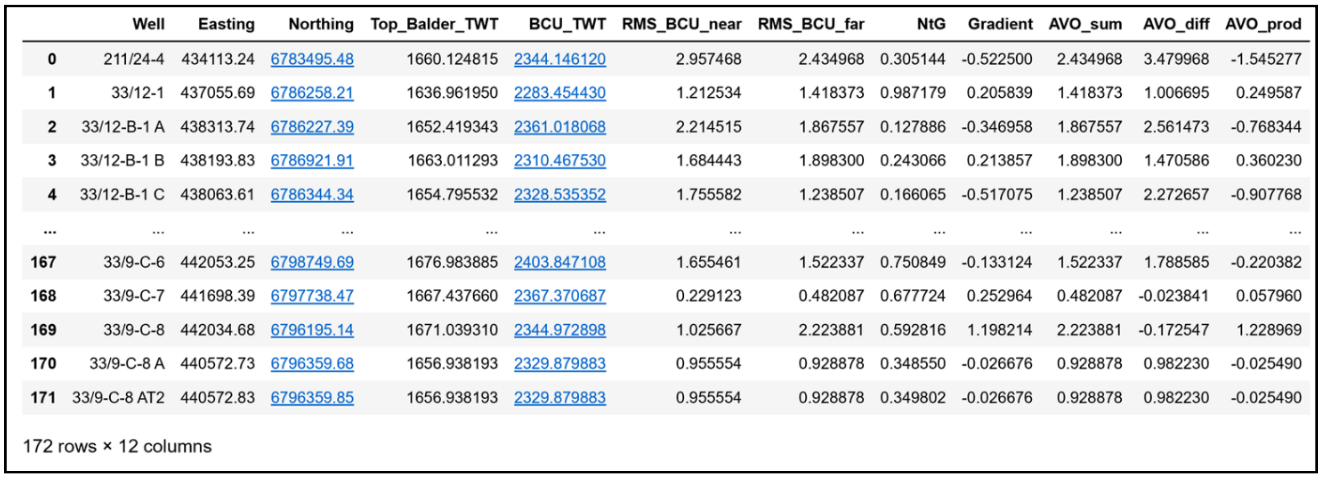
The AVO attributes of the study area are depicted in Figure 5.6. The uses of seismic amplitude variation with offset (AVO) attributes are widely spread among geoscientists. The AVO was included in the features since it has shown its capabilities in predicting and mapping hydrocarbons (Fatti et al., 1994; Ostrander, 1984). A detailed explanation about AVO will not be covered in this study.



Gradient - BCU
AVO_Gradient
Intercept + Gradient - BCU
AVO_Sum (Intercept+Gradient)
Intercept-Gradient - BCU
AVO_diff(Intercept-Gradient)
Intercept*Gradient - BCU
AVO_product(Intercept*Gradient)

Figure 5.6: Maps of AVO attributes in the study area

The final step in features extraction was combining all data needed including wells name, wells coordinates (Easting and Northing) and all features in one data table which was accomplished in Python (Fig.5.7). This table would then be the input for generating the ML models.



	Well	Easting	Northing	Top_Balder_TWT	BCU_TWT	RMS_BCU_near	RMS_BCU_far	NtG	Gradient	AVO_sum	AVO_diff	AVO_prod
0	211/24-4	434113.24	6783495.48	1660.124815	2344.146120	2.957468	2.434968	0.305144	-0.522500	2.434968	3.479968	-1.545277
1	33/12-1	437055.69	6786258.21	1636.961950	2283.454430	1.212534	1.418373	0.987179	0.205839	1.418373	1.006695	0.249587
2	33/12-B-1 A	438313.74	6786227.39	1652.419343	2361.018068	2.214515	1.867557	0.127886	-0.346958	1.867557	2.561473	-0.768344
3	33/12-B-1 B	438193.83	6786921.91	1663.011293	2310.467530	1.684443	1.898300	0.243066	0.213857	1.898300	1.470586	0.360230
4	33/12-B-1 C	438063.61	6786344.34	1654.795532	2328.535352	1.755582	1.238507	0.166065	-0.517075	1.238507	2.272657	-0.907768
...	...	...	...	...	...	...	...	...	...	...	...	...
167	33/9-C-6	442053.25	6798749.69	1676.983885	2403.847108	1.655461	1.522337	0.750849	-0.133124	1.522337	1.788585	-0.220382
168	33/9-C-7	441698.39	6797738.47	1667.437660	2367.370687	0.229123	0.482087	0.677724	0.252964	0.482087	-0.023841	0.057960
169	33/9-C-8	442034.68	6796195.14	1671.039310	2344.972898	1.025667	2.223881	0.592816	1.198214	2.223881	-0.172547	1.228969
170	33/9-C-8 A	440572.73	6796359.68	1656.938193	2329.879883	0.955554	0.928878	0.348550	-0.026676	0.928878	0.982230	-0.025490
171	33/9-C-8 AT2	440572.83	6796359.85	1656.938193	2329.879883	0.955554	0.928878	0.349802	-0.026676	0.928878	0.982230	-0.025490

172 rows × 12 columns

Figure 5.7: Table containing all data for generating the ML models

5.4 ML Models Generation and Training

Defining the type of task is important since this will determine which algorithms to use for generating the models as well as their performance evaluation techniques. As mentioned in Chapter 1, the objective of this study is to predict the determined label, which is the net-to-gross for the whole study area. Therefore, the approach of this task from ML perspective should be regression problem since the expected results are the predicted continuous net-to-gross values for the whole area. The whole process in this procedure was done in Python using *scikit-learn* library.

The general workflow of how the ML models were generated and trained are shown in Figure 5.8 below:

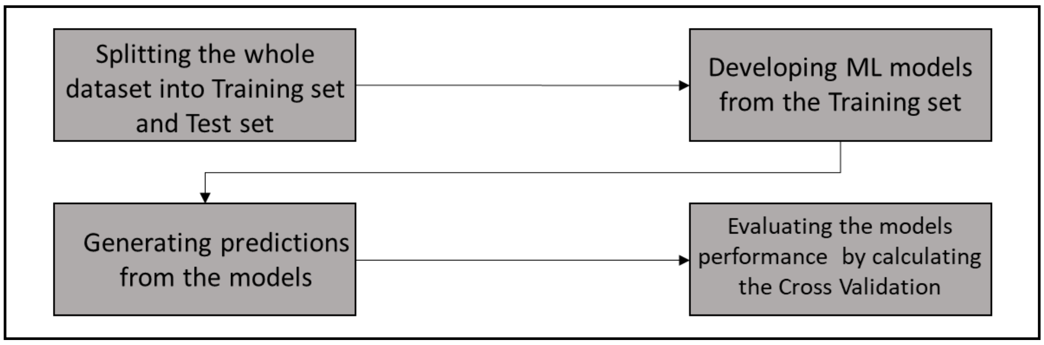

Splitting the whole dataset into Training set and Test set
Developing ML models from the Training set
Generating predictions from the models
Evaluating the models performance by calculating the Cross Validation

Figure 5.8: ML models generation and training processes

In this study, the whole dataset consists of 172 data points generated from the horizons and wells. The initial step in this procedure was to split the entire available dataset into two groups namely the training set and test set.

The next step was to develop ML models from the training set using various basic regression algorithms such as:

- Support Vector Regressor (SVR)
- Decision Tree Regressor (DTR)
- Random Forest Regressor (RFR)

And also some additional regressor such as:

- Linear Support Vector Regressor (Linear SVR)
- Gaussian Process Regressor (GPR)
- K-Nearest Neighbor Regressor (KNN)
- Stochastic Gradient Descent Regressor (SGDR)
- Gradient Boosting Regressor (GBR)

AutoML was also involved when developing the ML models. AutoML is basically an automatic process of applying ML to the dataset. The purpose of deploying AutoML is to allow non-experts to produce simpler and faster solutions and models. AutoML can automate several processes in ML including:

- Data preparation
- Feature selection

- Model selection
- Parameter optimization

In this study, this was done by using TPOT library which is built on top of *scikit-learn* library in Python. TPOT uses genetic programming in order to optimize ML pipelines by exploring thousands of possibilities to bring out the best one for the dataset. Figure 5.9 below illustrates how TPOT automates typical processes in ML.

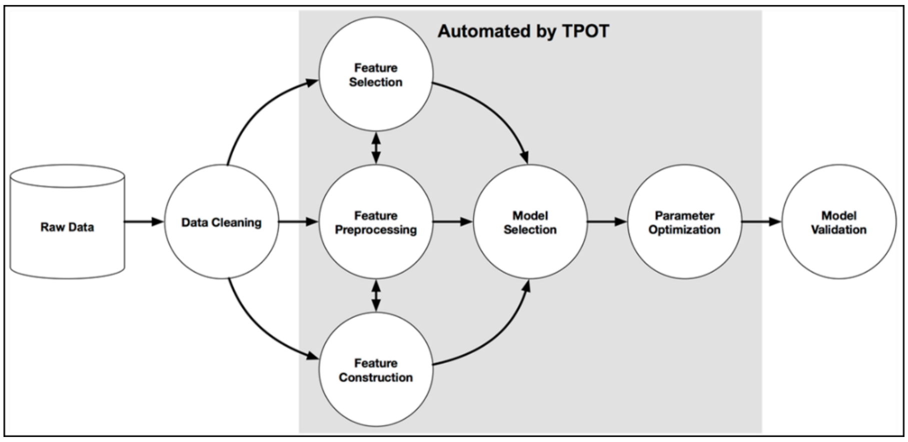


Figure 5.9: Automated stages in typical ML processes by TPOT

In order to evaluate the performance, the output models from each of the algorithms were then be validated with the test set by calculating the Cross Validation (CV). Basically, the purpose of CV is to evaluate the ability of the models to estimate new dataset in order to avoid overfitting or selection bias. Also, CV is expected to give an overview on how the models will generalize the unknown dataset, for instance from a real problem.

A basic CV technique that is commonly used is the *k*-fold CV (Fig 5.10) which was calculated with procedures defined as follows:

i. Split the training set into *k* smaller sets
ii. For each *k* folds, train the models using *k*-1 of the folds as training data and validate the resulting models on the remaining part of the data by calculating the performance measure (such as R2 for regression)
iii. Compute the average of the performance measure from all the *k*-folds as the final result of the CV

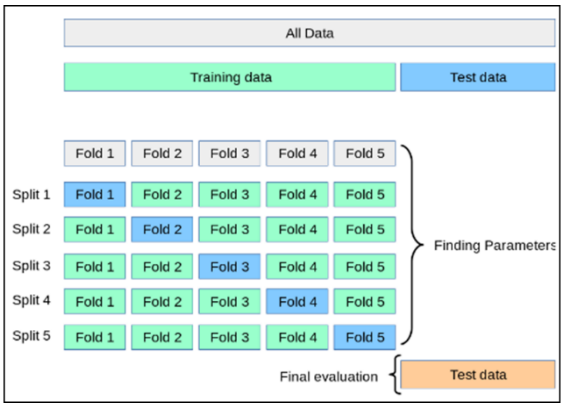


Figure 5.10: Illustration of k-fold cross validation (modified from scikit-learn.org)

5.5 Label Prediction

This procedure contains the execution of the net-to-gross prediction for the whole area by using the resulting models after they were both trained and evaluated. This step was done using *scikit-learn* library in Python.

5.6 Performance Evaluation

It is important to evaluate the performance of the ML models in order to know how close the prediction to the real values. In this study, the model performance is measured by correlation of determination or often denoted as R2, and statistical errors such as root mean squared error (RMSE) and mean absolute error (MAE).

1. **R2** – A direct indicator of correlation between predictions and true values. For the R2, the best possible value is 1.0 and it can also be negative if the model is arbitrarily worse. In general, the higher value of R2 indicates the better the model fits the real data. It can be computed by:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y}_i)^2} \quad (14)$$

   Where $\hat{y}_i$ is the predicted value of the i-th sample, y_i is the true value, and *n* is total number of samples

2. **RMSE** – It indicates on how much each predicted values deviate from its true value. The best value of RMSE is 0 which means a perfect estimation. It is expressed by:

$$RMSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad (15)$$

3. **MAE** – It presents a risk metric which corresponds to the predicted value of the absolute error loss. The best value of MAE is 0 which indicates a perfect prediction result. The MAE is defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \quad (16)$$

6 Results and Discussion

There are eight ML algorithms were deployed and compared in order to predict the NtG values for this study. They are Linear Support Vector Regressor (Linear SVR), Gaussian Process Regressor (GPR), Support Vector Regressor (SVR), K-Nearest Neighbor Regressor (KNN), Decision Trees Regressor (DTR), Random Forest Regressor (RFR), Stochastic Gradient Descent Regressor (SGDR), and Gradient Boosting Regressor (GBR). Also, there is one additional result which was generated from AutoML. The predicted net-to-gross values were then compared with the true net-to-gross which consist of 172 data points and were obtained from wells. The mean and the standard deviation of the true net-to-gross value is around 0.49 and 0.22 respectively with value distribution as depicted in Figure 6.1. In addition, in order to acquire more insight on how the ML approach performs in this study, the net-to-gross classification task and prediction of the sand thickness are also provided in this chapter to be compared with the main task (net-to-gross prediction). One important thing to note is that these two additional tasks were performed using eight features and the AutoML was not involved.

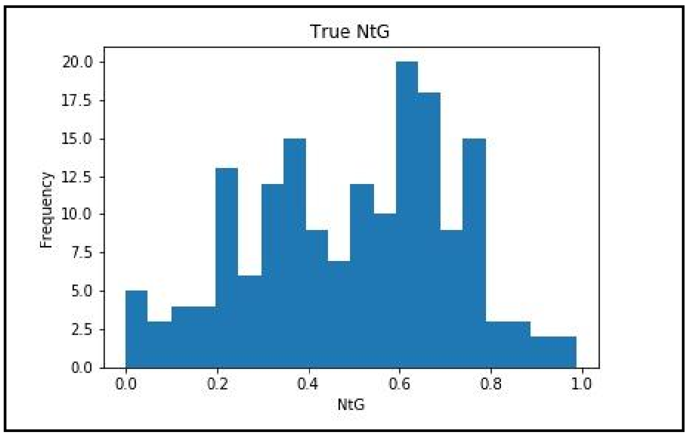


Figure 6.1: The distribution of the true value of net-to-gross from wells

6.1 Net-to-gross Prediction

6.1.1 Training and test set performance

Prior to applying the ML models into the real data (whole study area), the models were tested. The training set comprises 90% while the test set is 10% from the initial dataset (172 data points). Then *k*-fold cross validation (CV) was calculated to evaluate the performance of the models. Table 3 and Table 4 show the CV results when four and eight features were included in the models based on their R2 score.

Table 3. K-fold CV results of the models when using 4 features

	Linear SVR	GPR	SVR	KNN	DTR	RFR	SGDR	GBR
R2	0.14	0.06	0.34	0.09	0.70	0.69	0.12	0.13

Table 4. K-fold CV results of the models when using 8 features

	Linear SVR	GPR	SVR	KNN	DTR	RFR	SGDR	GBR
R2	0.15	0.07	0.28	0.12	0.88	0.88	0.14	0.12

In general, the CV results show very low R2 scores for most algorithms which range around 0.06-0.15 when four features were included, and the number added features does not have significant impact to the performance apparently. The SVR algorithm initially shows a moderately low index 0.34, however, the score decreases to 0.28 as more features were added. Among these algorithms, only DTR and RFR show an opposite trend. The performance indices of both algorithms are much higher compared to the rest, and these trends are improved when eight features were involved. In the end of the training and test stage, the R2 score of both algorithms reach 0.88.

6.1.2 Prediction results using four features

This section examines the results when applying four features namely Top Balder TWT, BCU TWT, BCU RMS near, and BCU RMS far into the algorithms. Figure 6.2 below presents the maps of predicted net-to-gross values for the whole area of study plotted by their respective coordinates (Eastings and Northings). Also Figure 6.3 depicts the distribution of predicted net-to-gross values in histograms for each algorithms.

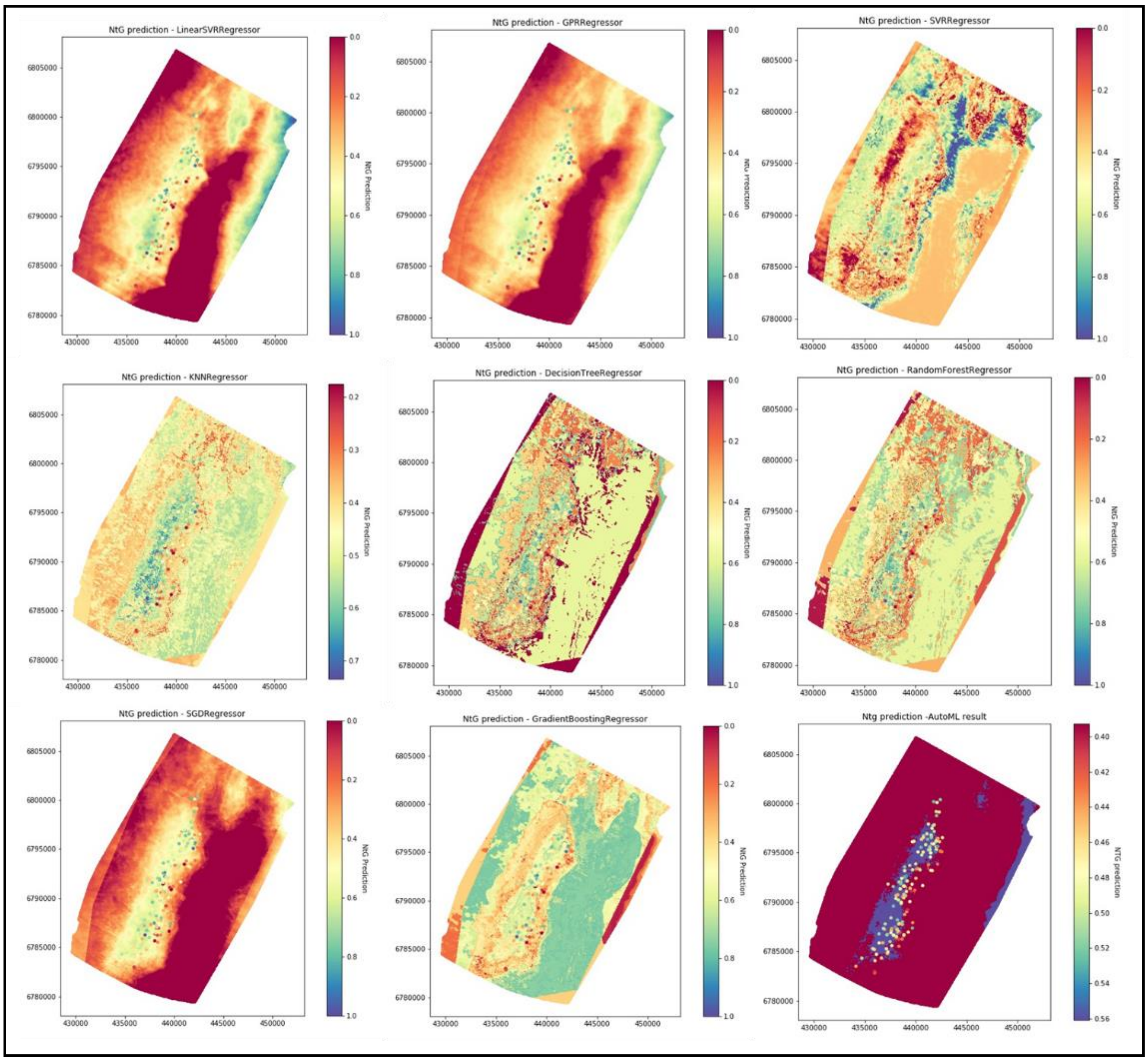

NtG prediction - LinearSVRRegressor
NtG prediction - GPRRegressor
NtG prediction - SVRRegressor
NtG prediction - KNNRegressor
NtG prediction - DecisionTreeRegressor
NtG prediction - RandomForestRegressor
NtG prediction - SGDRegressor
NtG prediction - GradientBoostingRegressor
Ntg prediction -AutoML result
NtG Prediction

Figure 6.2: Maps of the predicted net-to-gross values for the whole study area for each ML algorithms using 4 features. Wells location are represented by dots

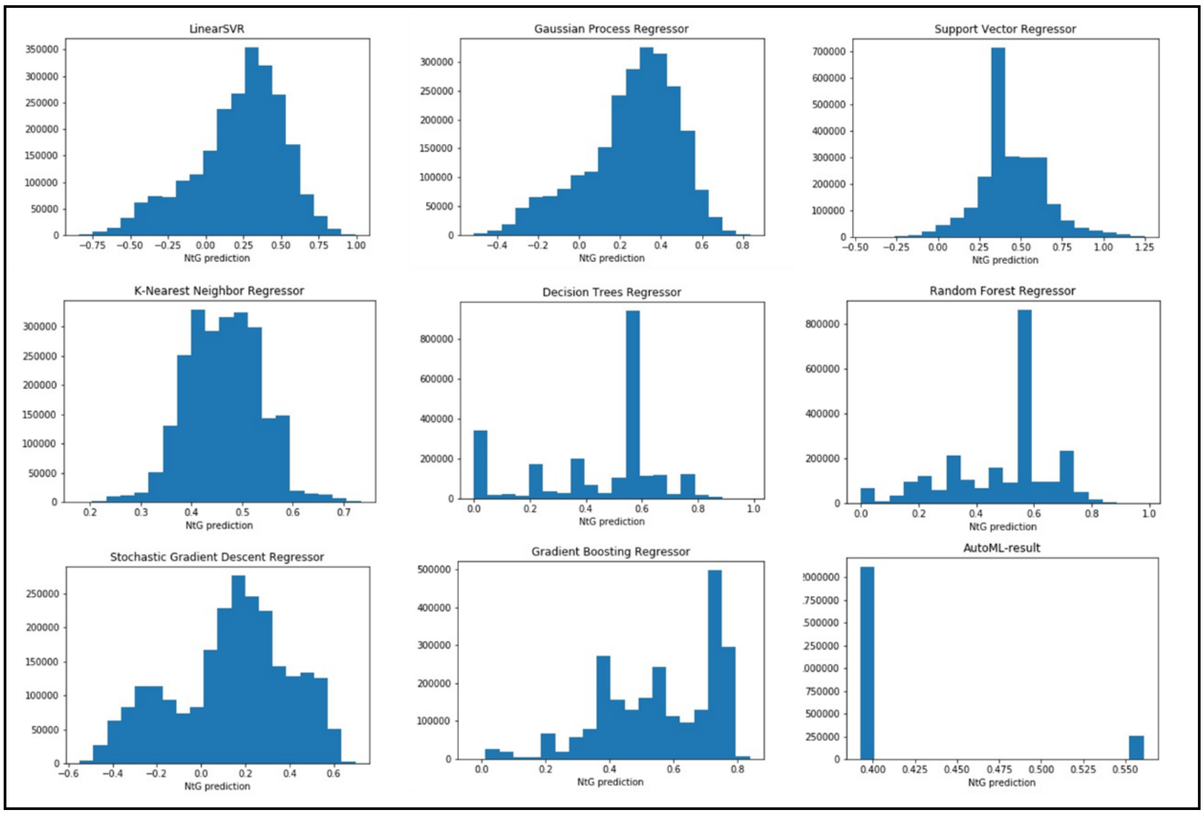


Figure 6.3: The distribution of the predicted net-to-gross values for each algorithm using 4 features

As shown in both figures above, the ML algorithms produced considerably different results of predicted net-to-gross among each other. In addition, Table 5 shows how the mean and the standard deviation of the prediction results differ for each algorithm. Most algorithms have the predicted net-to-gross mean around 0.4, however algorithms such as Linear SVR, and GPR have mean around 0.2. The SGDR algorithm has even lower predicted net-to-gross mean. The AutoML unexpectedly only yielded two values of the predicted net-to-gross. Another important thing to highlight is that apparently some algorithms such as Linear SVR, GPR, SVR, and SGDR have negative values on the predicted net-to-gross, which is impossible in geological manner (Fig.6.2 and Fig. 6.3).

Table 5. Mean and standard deviation of the predicted NtG using 4 features

	Linear SVR	GPR	SVR	KNN	DTR	RFR	SGDR	GBR	AutoML
Mean	0.22	0.26	0.44	0.46	0.45	0.49	0.14	0.56	0.41
St.Dev	0.29	0.23	0.19	0.07	0.23	0.18	0.26	0.18	0.05

6.1.3 Prediction results using eight features

This section presents the predicted net-to-gross results when four additional features were added into the algorithms. The additional four features are AVO gradient, AVO product (intercept * gradient), AVO summation (intercept + gradient), and AVO difference (intercept-gradient). Both Figure 6.4 and Figure 6.5 display the maps of the predicted net-to-gross values in the whole area and the distribution of the predicted net-to-gross values respectively.

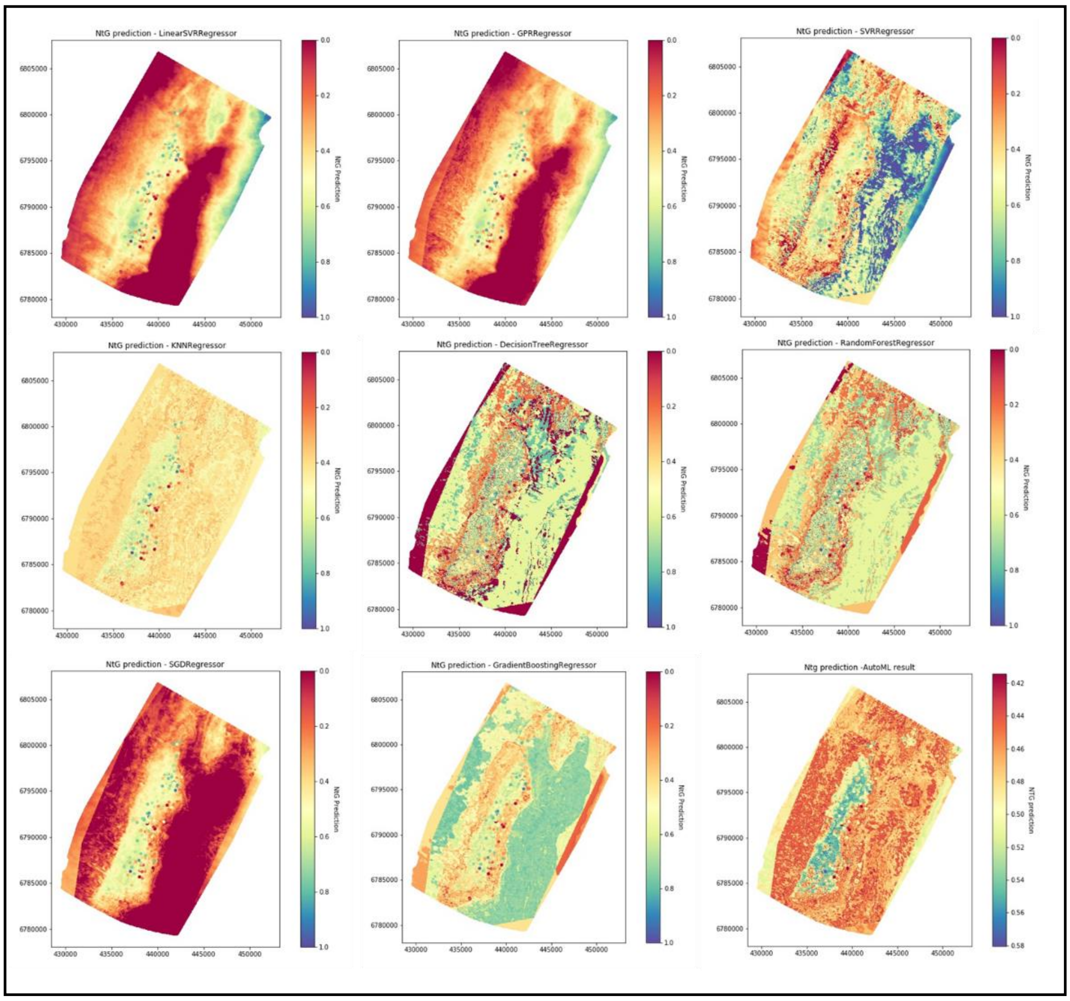

NtG prediction - LinearSVRRegressor
NtG prediction - GPRRegressor
NtG prediction - SVRRegressor
NtG prediction - KNNRegressor
NtG prediction - DecisionTreeRegressor
NtG prediction - RandomForestRegressor
NtG prediction - SGDRegressor
NtG prediction - GradientBoostingRegressor
Ntg prediction -AutoML result
NtG Prediction

Figure 6.4: Maps of the predicted net-to-gross values for the whole study area for each ML algorithms using 8 features. Wells location are represented by dots

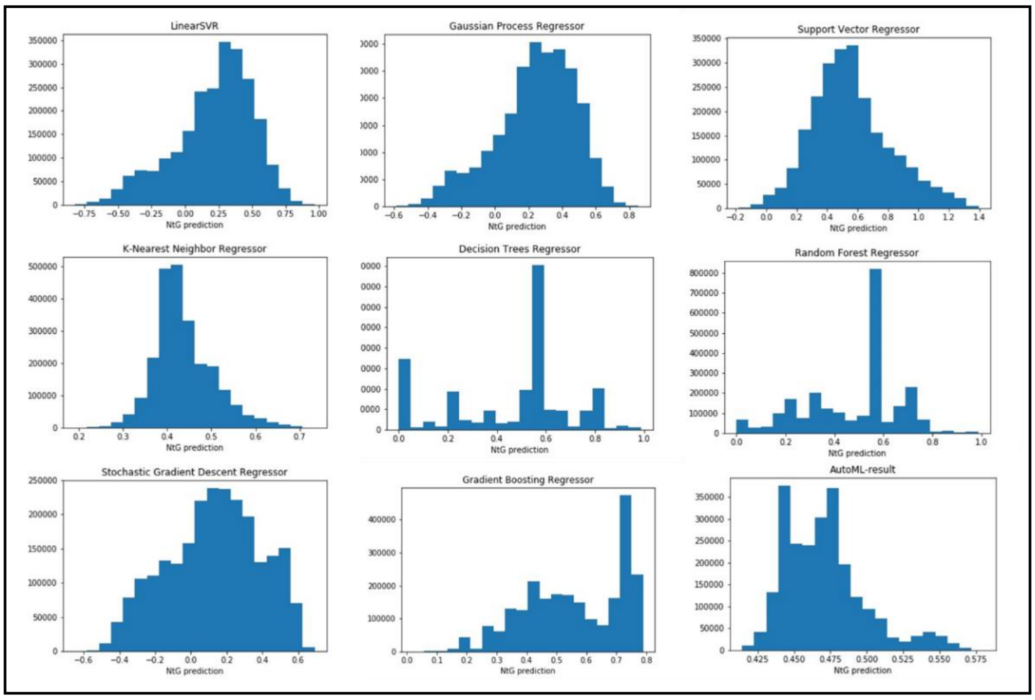


Figure 6.5: The distribution of the predicted net-to-gross values for each algorithm using 8 features

As seen in both Figure 6.4 and Figure 6.5 that the obtained results are almost identical with the previous section. The mean and the standard deviation shown in Table 6 are also quite similar with Table 5. The mean of the predicted net-to-gross is around 0.4 for the KNN, DTR, RFR, and AutoML. The Linear SVR, GPR, SVR, and SGDR are still having negative values on the prediction. However, there is one difference to note is that after including the additional features, the predicted values generated from AutoML have a better distribution.

Table 6. Mean and standard deviation of the predicted NtG using 8 features

	Linear SVR	GPR	SVR	KNN	DTR	RFR	SGDR	GBR	AutoML
Mean	0.21	0.24	0.56	0.44	0.47	0.49	0.13	0.56	0.47
St.Dev	0.29	0.24	0.26	0.07	0.25	0.19	0.26	0.16	0.03

6.1.4 Performance evaluation

These performance indices were basically calculated by the difference between the true net-to-gross values or the label and the predicted net-to-gross which were obtained from the ML models. Both Table 7 and Table 8 below present the performance indices for the prediction when using four and eight features respectively.

Table 7. Performance index of prediction when using 4 features

	Linear SVR	GPR	SVR	KNN	DTR	RFR	SGDR	GBR	AutoML
MAE	0.18	0.18	0.15	0.17	0.06	0.06	0.19	0.13	0.17
RMSE	0.22	0.21	0.18	0.19	0.14	0.14	0.22	0.16	0.21
R2	0.06	0.08	0.33	0.14	0.59	0.61	-0.004	0.47	0.12

Table 8. Performance index of prediction when using 8 features

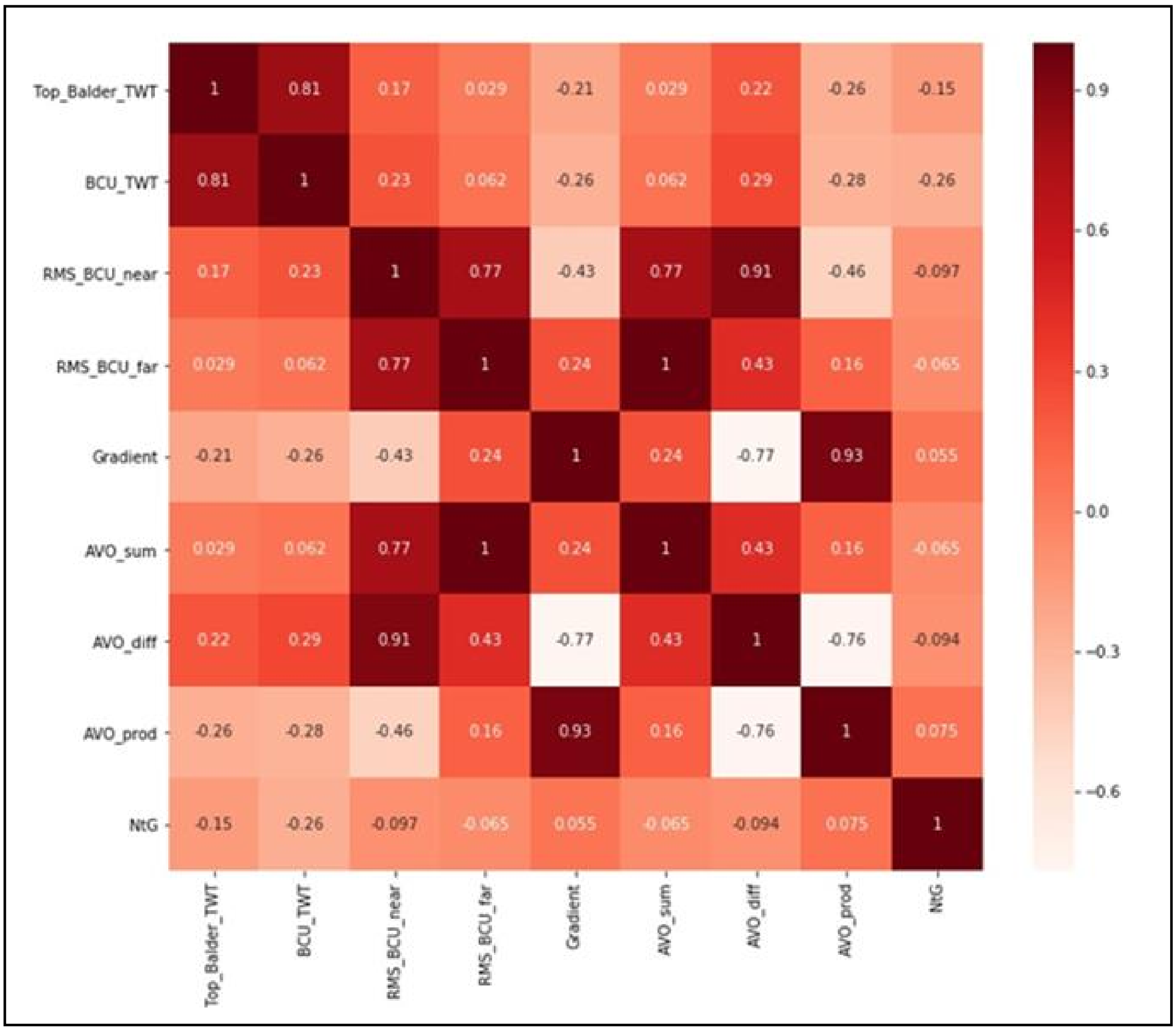
	Linear SVR	GPR	SVR	KNN	DTR	RFR	SGDR	GBR	AutoML
MAE	0.18	0.18	0.16	0.17	0.03	0.03	0.19	0.12	0.18
RMSE	0.22	0.21	0.19	0.19	0.09	0.08	0.22	0.16	0.21
R2	0.07	0.08	0.26	0.14	0.85	0.86	-0.015	0.49	0.08

According to both tables above, it can be seen that the values of MAE are relatively similar for most of the algorithms when using either four or eight features. The value of MAE ranges between 0.12-0.19 in most models. The RMSE also does not show significant changes for almost all models when the number of features were increased. However, the DTR and the RFR algorithms have different trends. Both algorithms have significantly lower MAE and RMSE errors among the others. And the errors are decreasing as the additional features were included.

Models generated by Linear SVR, GPR, KNN, SGDR and AutoML have considerably low R2 values either when four or eight features were involved. The models produced by SVR and GBR also have moderately low value of R2, and this does not change when more features were added. In contrast, the DTR and RFR models have particularly higher value of R2 among the others. And these numbers increase quite significantly as more features were included in the models. The R2 value of the DTR model rises from 0.59 to 0.85, while the R2 value of the RFR model increases from 0.61 to 0.86.

6.1.5 Features evaluation

The features need to be evaluated in order to find out which features are the most relevant to our models. One important thing to note is that irrelevant features can negatively impact the performance of the model. One way to do this is by showing the correlation matrix. The correlation matrix is based on Pearson's correlation coefficient. The values ranges between -1 to, where 1 is positive linear correlation, 0 is no linear correlation, and -1 is total negative linear correlation. Figure 6.5 shows how each features and label (net-to-gross) are correlated among each other. According to the correlation matrix map, it can be seen that most of the features have no linear correlation with the label which is the net-to-gross. This is shown by the correlation values are close to zero for most features. However, both the BCU TWT and the Top Balder TWT show a little negative correlation to the net-to-gross with -0.26 and -0.15 respectively.



	Top_Balder_TWT	BCU_TWT	RMS_BCU_near	RMS_BCU_far	Gradient	AVO_sum	AVO_diff	AVO_prod	NtG
Top_Balder_TWT	1	0.81	0.17	0.029	-0.21	0.029	0.22	-0.26	-0.15
BCU_TWT	0.81	1	0.23	0.062	-0.26	0.062	0.29	-0.28	-0.26
RMS_BCU_near	0.17	0.23	1	0.77	-0.43	0.77	0.91	-0.46	-0.097
RMS_BCU_far	0.029	0.062	0.77	1	0.24	1	0.43	0.16	-0.065
Gradient	-0.21	-0.26	-0.43	0.24	1	0.24	-0.77	0.93	0.055
AVO_sum	0.029	0.062	0.77	1	0.24	1	0.43	0.16	-0.065
AVO_diff	0.22	0.29	0.91	0.43	-0.77	0.43	1	-0.76	-0.094
AVO_prod	-0.26	-0.28	-0.46	0.16	0.93	0.16	-0.76	1	0.075
NtG	-0.15	-0.26	-0.097	-0.065	0.055	-0.065	-0.094	0.075	1

Figure 6.6: Features correlation matrix

Linear regression was also used to confirm the relation between the features and the label. This also gives a rough idea on how well the label when predicted with a simple regression line. Figure 6.7 presents the applied linear regression baseline on each AVO attributes – net-to-gross scatter plots. As shown in the picture above, the AVO attributes and the net-to-gross scatter plots have random values and so many outliers which cannot be fitted with a simple linear line. The regression index in Table 9 shows constant errors and significantly low values of R2 for all AVO attributes.

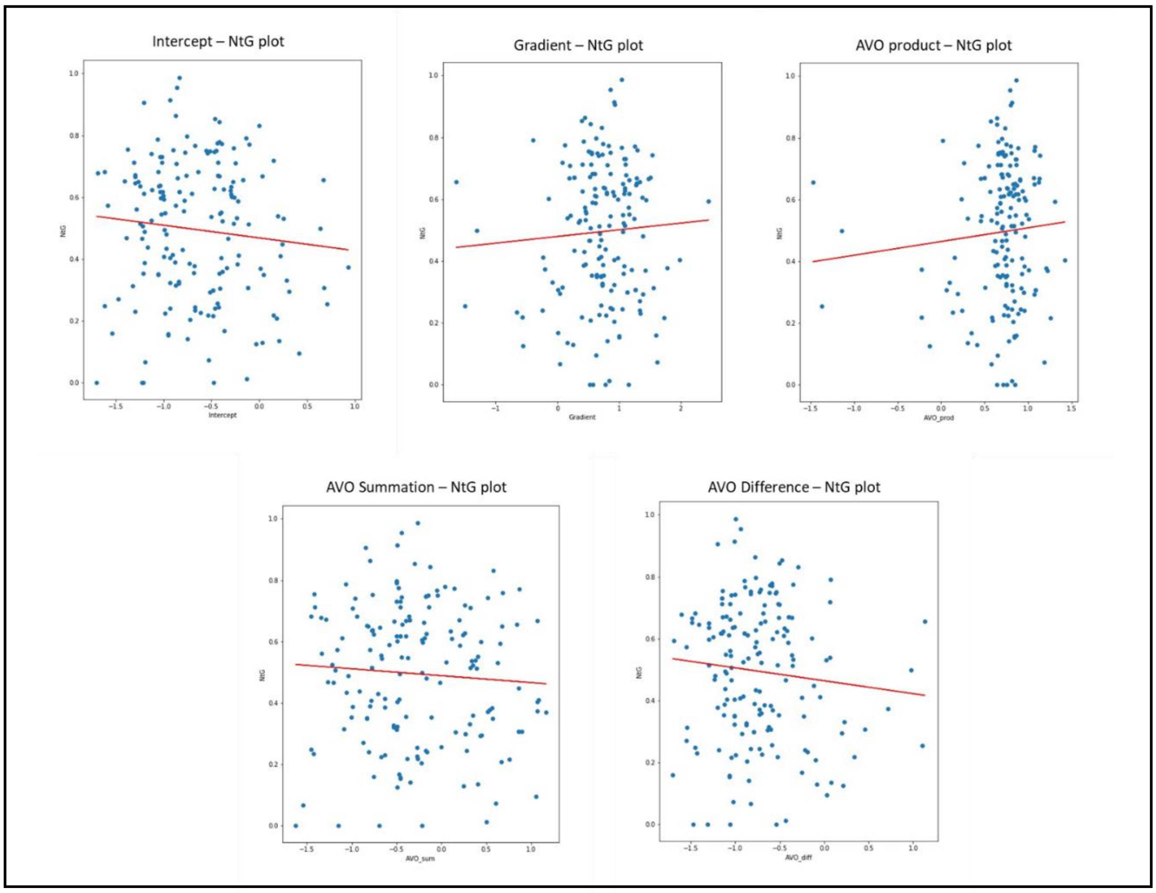


Figure 6.7: Linear regression line (red line) between AVO attributes and true net-to-gross

Table 9. Linear regression index of predicted net-to-gross using AVO attributes

	Intercept	Gradient	AVO product	AVO summation	AVO difference
MAE	0.19	0.19	0.19	0.19	0.19
RMSE	0.22	0.22	0.22	0.22	0.22
R2	0.003	0.003	0.006	0.004	0.009

6.2 Net-to-gross Classification

The classification task was performed using the same dataset, also with similar training/test set fraction (90% for training set, 10% for test set). In a simple manner, this task was carried out by following these steps:

i. Conditioning the label to a binary class of 1 and 0 according the net-to-gross value (i.e. NtG >= 0.5 is set as 1 (high net-to-gross); while NtG<0.5 is set as 0 (low net-to-gross))
ii. Applying the ML models to the training/test phase
iii. Applying the ML models to predict the whole study area

Basically, the models work in a similar way as in prediction, however the difference is instead of estimating continuous net-to-gross value as in the predicting task, the classification tries to classify the whole area into the two defined classes based on their features.

Classification has different algorithms and performance evaluation techniques compared to the prediction. Some of the classification algorithms were used in this study are:

- Logistic Regression (LR)
- Linear Discriminant Analysis (LDA)
- K-Nearest Neighbour Classifier (KNN)
- Decision Trees Classifier (DT/CART)
- Naive-Bayes Classifier (NB)
- Support Vector Machine (SVM)
- Random Forest Classifier (RF)

In addition, some of the classification performance index are:

- **Accuracy**: An index shows how the predicted value exactly match the true value
- **Precision**: The ratio of TP / (TP+FP), where TP is number of true positive values; FP is number of false positive values
- **Recall**: The ratio of TP / (TP+FN), where TP is number of true positive values; FN is number of false negative values
- **F1**: The weighted average of the precision and recall, or simply formulated as 2*(Precision * Recall) / (Precision * Recall)

The best value for all performance index mentioned above is 1, while the worst possible value is 0. The performance index in the training/test phase is provided in Table 10 below.

Table 10. Classification performance index in training/test phase

	LR	LDA	KNN	CART	NB	SVM	RF
Accuracy	0.54	0.53	0.59	0.59	0.58	0.59	0.57
Precision	0.55	0.54	0.60	0.65	0.58	0.58	0.60
Recall	0.77	0.69	0.70	0.52	0.76	0.88	0.58
F1	0.63	0.60	0.65	0.55	0.65	0.69	0.58

In terms of accuracy, all algorithms show moderately low index which is around 0.5. The precision also follows a relatively similar pattern with index range form 0.54-0.65. The highest recall value belongs to SVM, while the lowest belongs to DT and is followed by RF with 0.52 and 0.58 respectively. Most of the algorithms have quite similar precision score which range from 0.69-0.77. There are not much of differences in the F1 score where the index of all algorithms is around 0.6.

The results of the classification models and their performance index for the whole area is depicted in Figure 6.8 and Table 11 respectively.

Table 11. Classification performance index of the predicted results

	LR	LDA	KNN	CART	NB	SVM	RF
Accuracy	0.58	0.59	0.73	0.95	0.59	0.61	0.98
Precision	0.59	0.59	0.73	0.97	0.59	0.59	0.98
Recall	0.76	0.73	0.79	0.95	0.79	0.89	0.97
F1	0.66	0.66	0.76	0.96	0.68	0.71	0.98

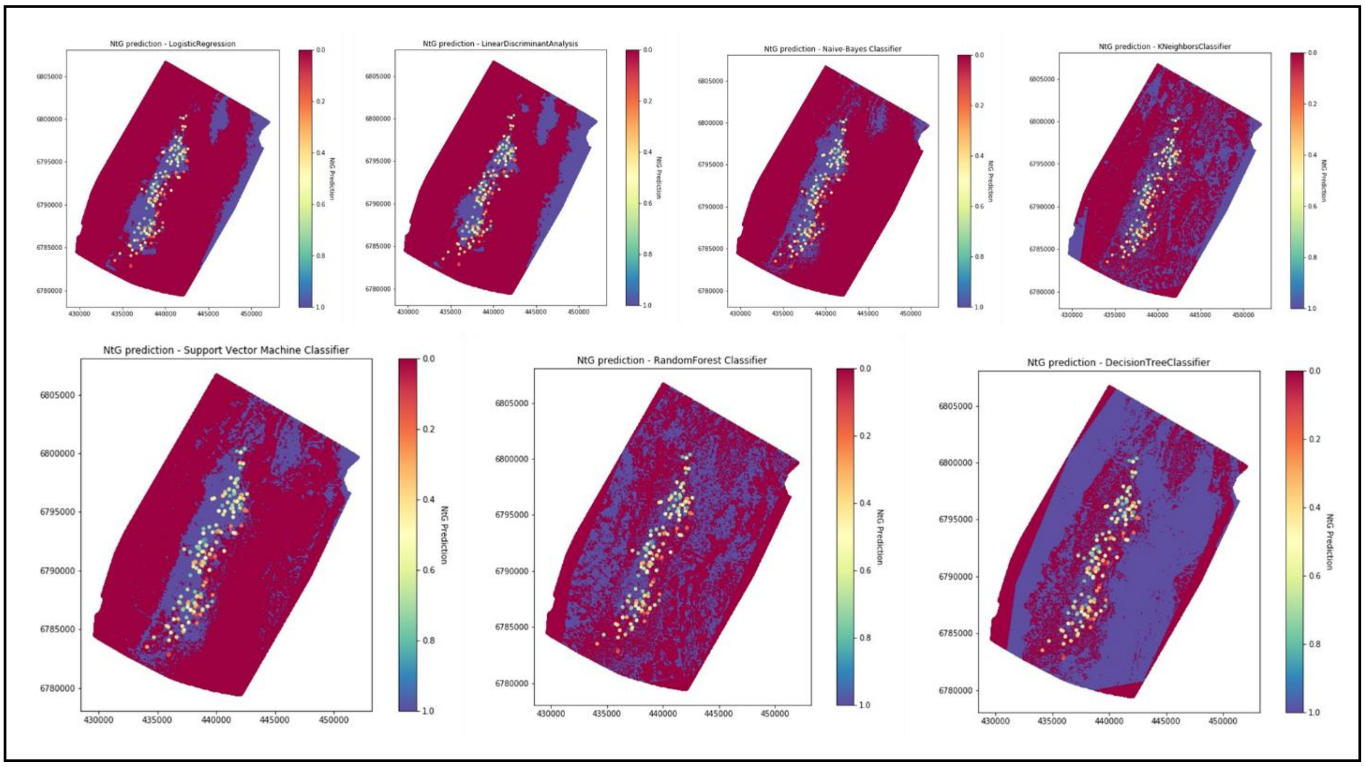

NtG prediction - LogisticRegression
NtG prediction - LinearDiscriminantAnalysis
NtG prediction - Naive-Bayes Classifier
NtG prediction - KNeighborsClassifier
NtG prediction - Support Vector Machine Classifier
NtG prediction - RandomForest Classifier
NtG prediction - DecisionTreeClassifier
NtG Prediction
6805000
6800000
6795000
6790000
6785000
6780000
430000
435000
440000
445000
450000
0.0
0.2
0.4
0.6
0.8
1.0

Figure 6.8: Maps of the predicted net-to-gross based on classification using ML algorithms. Wells location are represented by dots

The predicted maps generated from LR, LDA, NB, and SVM have pretty much similar pattern where the distribution of 'high' and 'low' net-to-gross were shown. However, the distribution is much more different in KNN, DT, and RF models. According to the performance evaluation index, the DT and RF models perform almost perfectly with both accuracy and F1 scores are very close to 1. Models from the other algorithms have shown moderately low performance which range from 0.58-0.73 for the accuracy, and 0.66-0.76 for the F1 score.

6.3 Sand Thickness Prediction

In this section, the thickness of sand (hSand) was defined as the label and was performed using the same dataset with the net-to-gross prediction task. The sand thickness was calculated by summing all lithology which has GR value less than 70 (GR<70) within the reservoir interval of each well and was then considered as 'true' sand thickness. So in total, there were 172 true sand thickness data points generated. The true sand thickness has distribution as shown in Figure 6.9 and has the average of 64.18 m.

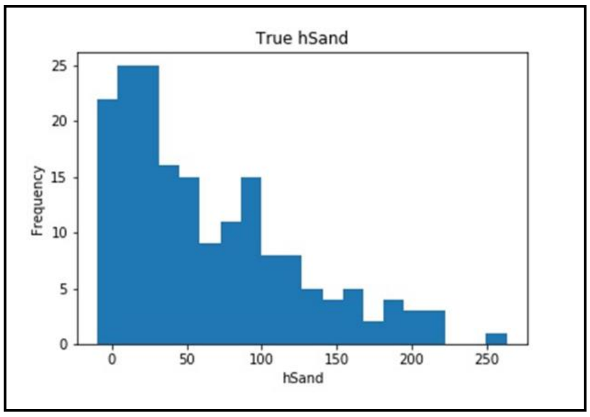


Figure 6.9: The distribution of true sand thickness value from wells

This prediction task deployed similar algorithms as in the net-to-gross prediction except without involving the AutoML. The training set comprised 90% of the whole data set while the test set was 10%. The *k*-fold CV results of the training/test phase are delivered in Table 12.

Table 12. K-fold CV results of the models in training/test phase

	Linear SVR	GPR	SVR	KNN	DTR	RFR	SGDR	GBR
R2	0.09	0.06	0.07	0.05	1.0	0.9	0.05	0.3

The performance of the models in training/test phase are extremely low for most algorithms. Although it still considered as moderately low, the GBR model shows a better performance index with R2 score of 0.3. On the other hand, the RFR and DTR present impressive performance in training/test stage. This is indicated by the R2 score of 0.9 for the RFR and a perfect 1.0 for DTR.

Figure 6.10 below shows how the predictive models perform when applied to the whole area, while the distribution, the mean and standard deviation of the predicted sand thickness are also delivered in Figure 6.11 and Table 13 respectively.

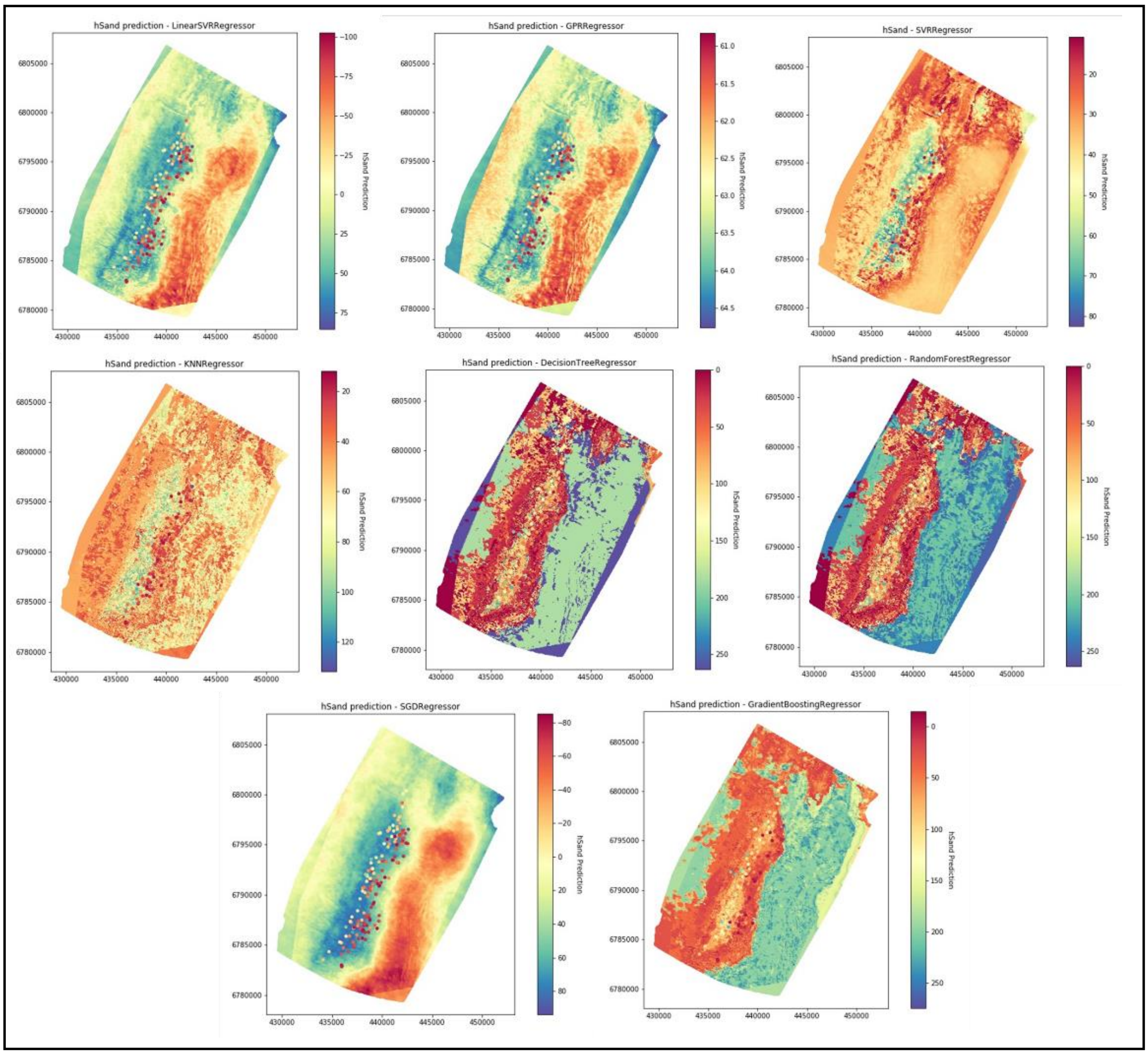

hSand prediction - LinearSVRRegressor
hSand prediction - GPRRegressor
hSand - SVRRegressor
hSand prediction - KNNRegressor
hSand prediction - DecisionTreeRegressor
hSand prediction - RandomForestRegressor
hSand prediction - SGDRegressor
hSand prediction - GradientBoostingRegressor
hSand Prediction

Figure 6.10: Maps of the predicted sand thickness for the whole area for each ML algorithms. Wells location are represented by dots

As occurred in the net-to-gross predictions, the ML models also yielded considerably different results of predicted sand thickness among each other. In addition, the sand thickness distribution has a wide range of values for each algorithm.

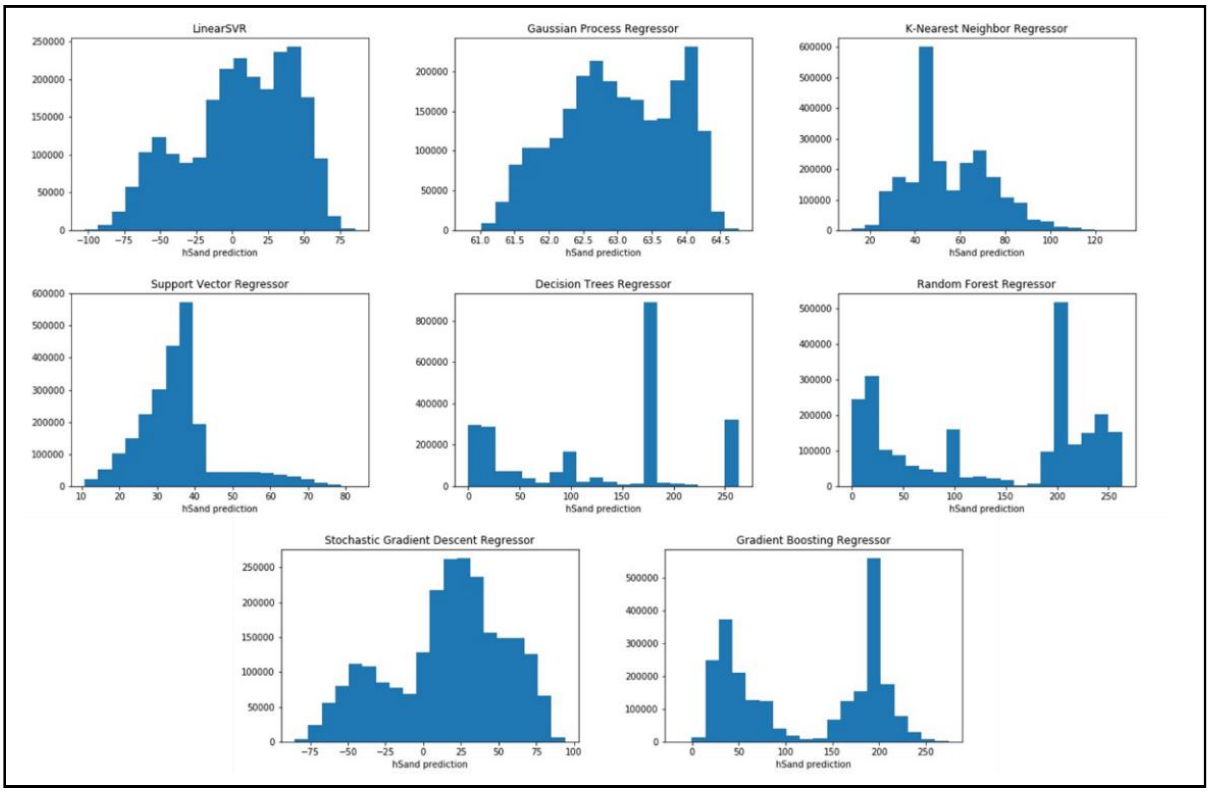


Figure 6.11: The distribution of the predicted sand thickness for each algorithm

Some models such as Linear SVR and SGDR even obtain negative values (Fig.6.10) and underestimate the predictions significantly (Table 13) compared to the true sand thickness mean. In contrast, the DTR and RFR models seem to overestimate the values around two times the true sand thickness average. The mean of predicted sand thickness of both models exceeds 120 meters.

Table 13. Mean and standard deviation of the predicted sand thickness

	Linear SVM	GPR	SVR	KNN	DTR	RFR	SGDR	GBR
Mean	5.28	62.99	35.69	55.56	129.23	136.79	16.04	122.54
St.Dev	37.28	0.83	10.90	18.16	87.61	91.74	38.29	76.56

The predicted results were then plotted against the true values in a scatter plot, and a linear regression was fitted between them. The purpose of fitting a linear line is to get an idea on how the predicted sand thickness match the true values. As shown in Figure 6.12, most of the predicted models do not match with the true sand thickness. There are a lot of values which located outside the regression line (outliers). Only results from DTR, RFR, and GBR models which fit the regression line quite well even though the outliers are still found.

The trends in the scatter plots are supported by the performance index shown in Table 14. In general, the MAE and the RMSE of the predicted results are in range of 40-47 meters and 52-60 meters respectively for most algorithms. However, the DTR and RFR models produce the lowest MAE and RMSE among the rest which is around 12 meters for both indices. The GBR has moderately low error values for both the MAE (~26 m) and RMSE (~39 m) compared to the others. In terms of the R2 score, the pattern is quite similar. The R2 score of the Linear SVR, GPR, SVR, KNN, and SGDR models are extremely low, while the DTR, RFR, GBR models bear the highest R2 score (0.64 and 0.54 respectively).

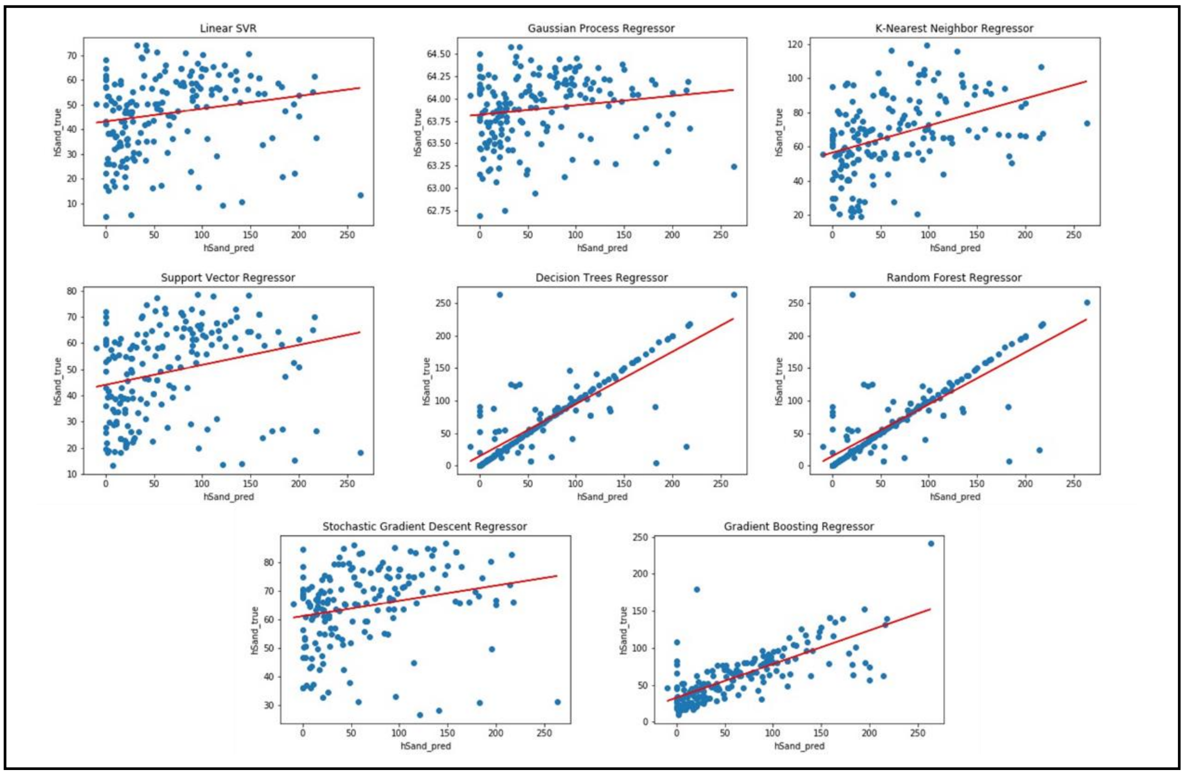


Figure 6.12: Linear regression between the predicted sand thickness vs the true sand thickness

Table 14. Performance index of the predicted sand thickness

	Linear SVR	GPR	SVR	KNN	DTR	RFR	SGDR	GBR
MAE	42.66	47.31	40.86	41.22	12.37	12.29	44.78	26.33
RMSE	59.75	57.94	58.21	52.95	34.65	34.32	56.49	39.15
R2	0.061	0.002	0.007	0.16	0.64	0.64	0.051	0.54

6.4 Discussion

6.4.1 ML simulation results

According to the feature correlation matrix from previous section, it can be inferred that the overall features used in this study almost do not have any correlation with the defined label, which is the net-to-gross. Even the additional features, the AVO attributes, do not directly correlate with the net-to-gross according to the baseline linear regression. This will directly correspond with the ML models performance, the R2. Most of the ML models have significantly low R2 values or even close to zero which basically mean that the dependent variable changes without any correlation to the independent variable.

One way to explain this is by first taking a look into the true net-to-gross values. As depicted in Figure 6.1, the mean and the standard deviation of the true net-to-gross are 0.49 and 0.22 respectively. Remember that the net-to-gross values are expressed in fraction. So, when the true net-to-gross has standard deviation of 0.22, it basically means that each point in the dataset deviates by 0.22 point from its mean, 0.49, on average scale. The standard deviation is almost half of the mean itself. This is a big difference. A high standard deviation indicates that the data points are spread out over a large range of values. Or in other words, the true net-to-gross values in this study are nearly random, hence have high bias. This explains why the RMSE and the MAE are significantly high for almost all ML models.

However, not all ML models performed poorly. The DTR and RFR algorithms are able to predict the net-to-gross values quite well. This is shown by significantly higher R2 values and lower errors. And those indices are improved as more features were added into the models. The final R2 value for the DTR is 0.85, while the RFR is 0.86 when all eight features were used. These values suggest that around 85% of the predicted models successfully fit the true value. The results obtained from DTR and RFR confirm that both algorithms can considerably handle non-linear data. The reason why these two algorithms work well is because the nodes/trees in both algorithms protect each other from their individual error by merging the predictors and generalize the errors. In this way, the groups of trees/nodes in both algorithms produce ensemble predictions which more accurate than any other individual predictions.

These trends are also appeared when predicting the sand thickness where most algorithms performed poorly but the DTR and RFR produced much better results. However, one important thing to note is that the R2 score of these two algorithms in the prediction are considerably lower than in the training/test phase. This may indicate overfitting where the trained model fails to generalize the unseen data. In this case, the overfitting was presumably caused due to the noise and inaccurate data entries in the dataset.

The negative values from some algorithms in both net-to-gross and sand thickness predictions were presumably caused by the outliers in the dataset. For instances, Figure 6.7 shows how the negative gradient and AVO product values affected the regression line in the scatter plot. Another example is shown in Figure 6.9 where the negative values of sand thickness have high frequency in the dataset. This could ruin the implicit pattern from the data which the algorithms tried to discover. These type of data should be removed in the data cleaning stage prior to generating the ML models.

The net-to-gross classification yielded different results compared to the predictions. In general, the performance index of the results is much higher than the predictions' in most algorithms. In addition, the accuracy and F1 score of classification results from DT and RF are around 0.9 or almost perfect. This suggests that the classification may be an easier task for the algorithms in this study since it was only 'asked' to predict binary class (1 and 0) instead of continuous number in predictions. However, what makes these results are still arguable is even though the performance indices of the DT and RF are remarkably high, the performance indices of the trained data are moderately low (~0.6). This makes the classification results are less reliable.

6.4.2 Seismic amplitudes

According to the literatures, seismic amplitude is considered as primary variable when generating the net-to-gross value from the seismic data. However, this could not be confirmed in this study since the correlation coefficients between the seismic amplitudes from the horizons and the obtained net-to-gross were very low.

The studies of the net-to-gross estimation from seismic data has been carried out by many (Brown et al., 1984, 1986; Connolly, 2007; Inichinbia et al., 2014; and Simm, 2009). Brown et al. (1984, 1986) proposed a technique using scaled seismic amplitude to remove the tuning effect when estimating the net-to-gross. Connolly (2005, 2007) introduced a technique to estimate the net pay using band limited impedance obtained from coloured inversion. According to these studies, the process of calculating the net-to-gross involved several scaling and calibrations of the amplitudes such as correcting seismic amplitude maps with the apparent thickness model, generating band-limited relative impedance using coloured inversion, and etc.

The AVO attributes are also commonly used when determining the net-to-gross. A study carried out by Simm (2009) highlighted the use of the AVO intercept and gradient cross plot to generate fluid factor in order to maximize the fluid content reflectivity and optimize the net pay estimation.

The studies mentioned above show how the seismic amplitude actually has significant impact in when determining the net-to-gross. However, the net-to-gross in this study case are roughly derived only from TWT of both top and bottom reservoir (BCU and Top Rannoch) as the zone boundaries, and the value of GR from wells as indicator of sand and shale. There are many processes were bypassed compared to the conventional procedures. This may explain why the true net-to-gross values are randomized which causes the extremely low correlation between the net-to-gross and the amplitudes, and hence, produces highly deviated prediction results.

6.4.3 Net-to-gross of the Brent Group

According to the reservoir studies of the Brent Group (Giles et al., 1992; Helland-Hansen et al., 1992; and Knag et al., 1995), the overall net-to-gross of this group ranges from 0.4-0.8. Meanwhile, the most promising prediction results (the ones with the lowest error and highest R2) from the DTR and RFR models show the average net-to-gross values around 0.45-0.49. Even though the predicted values are very close and limited to the minimum value from the literature, those values are still within the range. In addition, the predicted average net-to-gross from the DTR and RFR models are almost similar with the one obtained from the wells. This actually indicates that these algorithms perform quite well regardless the limited amount of data as well as its non-linearity.

6.4.4 Discussion remarks

Several constraints which affected the results of this study are defined as follows:

- **Limited amount of data**. As a reminder, only 172 data points utilized in training/test phase to predict around 2 million data points in the whole study area. This number obviously cannot be a good representation of the whole area. This also may be the cause of overfitting in the prediction model.
- **Poor data cleaning.** The dataset contains many outliers which will be the noise for the algorithms. This definitely will cause the algorithms to build less accurate models driven by noise.
- **High variance in the data.** This is actually a consequence of the previous constraint. High variance data generally will increase the complexity for the algorithms to generate proper models.
- **Oversimplified label determination.** As mentioned in the previous section, the net-to-gross were simply derived according to the GR level in the reservoir interval. As

consequence, this led to an inaccurate net-to-gross results and poor correlation between label and features.

In general, this study needs a lot of improvements, especially when defining both the features and the label. Ensuring good correlation between features and label is essential prior to applying ML models. In addition, establishing an appropriate label can also be challenging if the defined label is dependant variable. Therefore, ensuring the label is derived properly through decent procedures should be prioritized.

7 Conclusion

The main purpose of this study was to obtain a better understanding of how machine learning models perform prediction on net-to-gross value of the Brent Group in Statfjord field using parameters derived mainly from seismic horizons of the reservoir zone.

Several ML algorithms was implemented to reach this goal. In general, most of the prediction results exhibit high errors and low performance index. This is also confirmed by the performance index of the classification and sand thickness prediction tasks which were included as additional comparison for this study. However, not all models perform poorly. The Decision Tree and Random Forest seem to outperform other algorithms when completing the defined tasks. This is shown by the noticeably higher performance index and remarkably lower errors these two algorithms possess.

Despite the limitations and constraints in the dataset as described in the discussion, the ML models have potential to do this task. The fact that the net-to-gross prediction results are still within the range according to the literatures strengthens this claim. It is also supported by previous studies on the same domain which showed how ML successfully estimated petrophysical properties (Al-Anazi and Gates, 2010a,b,c,d), hydraulic pressure losses (Fruhwirth et al., 2006), and evaluated bottom hole pressure (Ashena et al., 2010) in drilling engineering. In order to enhance the models' performance, several improvements such as proper data cleaning, adding more relevant features, and establishing a solid label through appropriate procedures should be implemented in the subsequent studies. Keep in mind that the ultimate purpose of utilizing ML algorithms in this domain is to ensure the daily tasks in oil and gas industry to be accomplished more effectively compared to the conventional procedures.

8 Future Work Recommendations

There are some important aspects related to this study which were not included due to the restricted time and availability of the data. Hence, some recommendations to be considered in further study are:

- Increasing the number of data used in the training/test phase in order to build a more solid and representative model
- Adding more valuable information from the well logs (e.g. resistivity log, density log, etc.) and other horizons into the models
- A better data cleaning process such as filtering unwanted outliers, removing unwanted observation, and etc.
- Involving proper geophysical procedures, such as Coloured Inversion and AVO analysis, to define the net-to-gross before setting it as a label
- Utilization of more relevant features as well as ensuring the appropriate correlation between the label and features
- Utilization of more advanced algorithms such as Artificial Neural Networks (ANN), Ensemble methods, Dimensionality Reduction algorithms, and even Deep Learning

References

Al-Anazi A, Gates ID. (2010a) On the capability of support vector machines to classify lithology from well logs. Nat Resour Res 19(2):125–139. doi:10.1007/s11053-010-9118-9

Al-Anazi A, Gates ID. (2010b) Support vector regression for permeability prediction in a heterogeneous reservoir: a comparative study. SPE Res Eval Eng 13(3):485–495. SPE-126339 PA. doi:10.2118/126339-PA

Al-Anazi A, Gates ID. (2010c) A support vector machine algorithm to classify lithofacies and model permeability in heterogeneous reservoirs. Eng Geol 114:267–277. doi:10.1016/j.enggeo. 2010.05.005

Al-Anazi A, Gates ID. (2010d) Support vector regression for porosity prediction in a heterogeneous reservoir: a comparative study. Comput Geosci 36(12):1494–1503

Ashena, R., Moghadasi, J., Ghalambor, A., Bataee, M., Ashena, R., & Feghhi, A. (2010, January 1). Neural Networks in BHCP Prediction Performed Much Better Than Mechanistic Models. International Oil and Gas Conference and Exhibition in China. https://doi.org/10.2118/130095-MS

Badley, M. E., Egeberg, T. & Nipen, O. (1984). Development of rift basins illustrated by the structural evolution of the Oseberg structure, Block 30/6, offshore Norway. Journal of the Geological Society, London, 141, 639-649.

Badley, M. E., Price, J. D., Rambech Dahl, C. & Abdestein, T. (1988). The structural evolution of the northern Viking Graben and its bearing upon extensional modes of graben formation. Journal of the Geological Society, London, 145, 455-472.

Beach, A., Bird, T. & Gibbs, A. (1987). Extensional tectonics and crustal structure: deep seismic reflection data from the northern North Sea Viking Graben. In: Coward, M. P., Dewey, J. F. & Hancock, P. L. (eds) Continental extensional tectonics. Geological Society, London, Special Publications, 28, 467 476.

Brown, A.R., Wright, R.M., Burkart, K.D., Abriel, W.L. and McBeath, R.G. (1986). Tuning effects, lithological effects and depositional effects in the seismic response of gas reservoirs. Geophysical Prospecting, **32**, 623-647.

Brown, A.R., Wright, R.M., Burkart, K.D. and Abriel, W.L. (1984). Interactive seismic mapping of net producible gas sand in the Gulf of Mexico. Geophysics, **49**, 686-714.

Brown, G. (1984). Jurassic. In: Glennie, K. W. (ed.) Introduction to the petroleum geology of the North Sea. Blackwell Science Publications, Oxford, 103-131.

Connolly, P. (2005). Net pay estimation from seismic attributes. 67th EAGE Conference & Exhibition, Extended Abstracts, F016.

Connolly, P. (2007). A simple, robust algorithm for seismic net pay estimation. The Leading Edge, **26**, 1278-1282.

Connolly, P. and Kemper, M. (2007). Statistical uncertainty of seismic net pay estimations. The Leading Edge, **26**, 1284-1289.

Connolly, P., Wilkins, S., Allen, T., Schurter, G., & Rose-Innes, N. (2005). Fluid and lithology identification using high-resolution 3D seismic data. Petroleum Geology: North-West Europe and Global Perspectives—Proceedings of the 6th Petroleum Geology Conference. In A. G. Doré & B. A. Vining (Eds.), (Vol. 6, pp. 1425-1433): Geological Society of London.

Cranganu, C., Luchian, H., & Breaban, M. E. (Eds.). (2015). Artificial Intelligent Approaches in Petroleum Geosciences. Springer International Publishing. https://doi.org/10.1007/978-3-319-16531-8

Deegan, C. E. & Scull, B. J. (1977). A proposed standard lithostratigraphic nomenclature For the Central and Northern North Sea. Institute of Geological Sciences Report, 77/25.

Elahifar, B., Thonhauser, G., Fruhwirth, R. K., & Esmaeili, A. (2012, January 1). ROP Modeling using NeuralNetwork and Drill String Vibration Data. SPE Kuwait International Petroleum Conference and Exhibition. https://doi.org/10.2118/163330-MS

Faleide, J. I., Bjørlykke, K., & Gabrielsen, R. H. (2010). Geology of the Norwegian Continental Shelf. In K. Bjorlykke, Petroleum Geoscience (pp. 467–499). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-02332-3_22

Fatti, J., Smith, G., Vail, P., Strauss, P., & Levitt, P. (1994). Detection Of Gas In Sandstone Reservoirs Using Avo Analysis - A 3-D Seismic Case-History Using The Geostack Technique. Geophysics, 59(9), 1362-1376.

Fossen, H., & Hesthammer, J. (1998). Structural geology of the Gullfaks Field. Structural geology in reservoir characterization. In M. P. Coward, H. Johnson & T. S. Daltaban (Eds.), Geological Society of London Special Publications, 127, 231-261.

Fruhwirth, R. K., Thonhauser, G., & Mathis, W. (2006, January 1). Hybrid Simulation Using Neural Networks To Predict Drilling Hydraulics in Real Time. SPE Annual Technical Conference and Exhibition. https://doi.org/10.2118/103217-MS

Gabrielsen, R. H. (1986). Structural elements in graben systems and their influence on hydrocarbon trap types. Habitat of Hydrocarbons on the Norvegian Continental Shelf. International Conference, 55–60.

Gabrielsen, R. H., Føerseth, R. B., Steel, R. J., Idil, S., & Klovjan, O. S. (1990). Architectural styles of basin fill in the northern Viking Graben. Unknown Journal, 158–179.

Gibbons, K. A., Jourdan, C. A., & Hesthammer, J. (2003). The Statfjord Field, Blocks 33/9, 33/12 Norwegian sector, Blocks 211/24, 211/25 UK sector, Northern North Sea. Geological Society, London, Memoirs, 20(1), 335–353. https://doi.org/10.1144/GSL.MEM.2003.020.01.29

Giles, M.R., Stevenson, S., Martin, S.V, Cannan, S.J.C, Hamilton, PJ., Marshall, J.D. and Samways, G.M. (1992). The reservoir properties and diagenesis of the Brent Group: a regional perspective. In: A.C. Morton, R.S. Hazeldine, M.R. Giles and S. Brown (Editors), Geology of the Brent Group. Geol. Soc. Spec. Publ., 61: 289-327.

Helland-Hansen, W, Ashton, M., Lomo, L. and Steel, R. (1992). Advance and retreat of the Brent delta: recent contributions to the depositional model. In: A.C. Morton, R.S. Hazeldine, M.R. Giles and S. Brown (Editors), Geology of the Brent Group. Geol. Soc. London, Spec. Publ, 61: 109-128.

Hesthammer, J., & Fossen, H. (1999). Evolution and geometries of gravitational collapse structures with examples from the StatfJord Field, northern North Sea. Marine and Petroleum Geology, 16(3), 259–281. https://doi.org/10.1016/S0264-8172(98)00071-3

Hesthammer, J., Jourdan, C. A., Nielsen, P. E., Ekern, T. E. & Gibbons, K. A. (1999). A tectonostratigraphic framework for the Statfjord Field, northern North Sea. Petroleum Geoscience, 5, 241-256 Johnson, A, & Eyssautier, M. 1987. Alwyn North Field

Inichinbia S, Sule PO, Hamza H, and Ahmed AL. (2014). Estimation of net-to-gross of among hydrocarbon field using well log and 3D seismic data. IOSR Jour. of Appl. Geol. and Geophys., 2014; 2(2): 18-26.

Kirk, R. H. (1980). Statfjord Field—A North Sea Giant. 12, 95–116.

Knag, Ø. G., South, D., and Spencer, A. M. (1995). Exploration trends in the nortliern North Sea (60-62"N). Petroleum Exploration and Exploitation in Norway edited by S. HansHen NPF Special Publication 4, pp. 115-134, Elsevier, Amsterdam. © Norwegian Petroleum Society (NPF), 1995.

Lary, D. J., Alavi, A. H., Gandomi, A. H., & Walker, A. L. (2016). Machine learning in geosciences and remote sensing. Geoscience Frontiers, 7(1), 3–10. https://doi.org/10.1016/j.gsf.2015.07.003

Naeeni, M. N., Zargari, H., Ashena, R., Ashena, R., & Kharrat, R. (2010, January 1). Permeability Prediction of Un-cored Intervals Using New IMLR Method and Artificial Neural Networks: A Case Study of Bangestan Field, Iran. Nigeria Annual International Conference and Exhibition. https://doi.org/10.2118/140682-MS

Nwachukwu, C. (2018). Machine learning solutions for reservoir characterization, management, and optimization. PhD thesis.

Odinsen. T., Reemst, P., van der Beck. P., Faleide, J.I., & Gabrielsen R. H. (in press b). Permo Triassic and Jurassic extension in the northern North Sea: results from tectonostratigraphic forward modelhng. NGT Special Publication.

Ostrander, W. (1984). Plane-wave reection coe_cients for gas sands at nonnormal angles of incidence. Geophysics, 49(10), 1637{1648.

Roberts, A. M., Price, J. D. & Olsen, T. S. (1990a). Late Jurassic halfgraben control on the siting and structure of hydrocarbon accumulations: UK/Norwegian Central Graben. In: Hardman, R. F. P. & Brooks, J. (eds) Tectonic Events Responsible for Britain's" Oil and Gas Reserves. Geological Society, London, Special Publications, 55,229-257.

Roberts, A. M., Yielding, G. & Badley, M. E. (1990b). A kinematic model for the orthogonal opening of the Late Jurassic North Sea rift system, Denmark-Mid Norway. In: Blundell, D. J. & Gibbs, A. D. (eds) Tectonic Evolution of the North Sea Rifts. Clarendon Press, Oxford, 180-199.

Roberts, A. M., Yielding, G., Kusznir, N. J., Walker, I. U. & Dornlopez, D. (1995). Quantitative analysis of Triassic extension in the northern Viking Graben. Journal of the Geological Society, London, 152, 15 26.

Roberts, J. D., Mathieson, A. S. & Hampson, J. M. (1987). Statfjord. In: Spencer, A. M. et al. (eds) Geology of the Norwegian Oil and Gas Fields. Norwegian Petroleum Society. Graham & Trotman, London, 319-340.

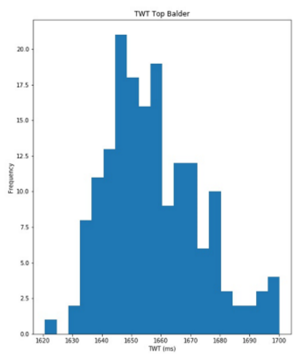
Simm, Rob. (2009). Simple net pay estimation from seismic: A modelling study. First Break. 27. 10.3997/1365-2397.2009014.

Thorne, J. A. & Watts, A. B. (1989). Quantitative analysis of North Sea subsidence. American Association of Petroleum Geologists, Bulletin, 73, 88-116.

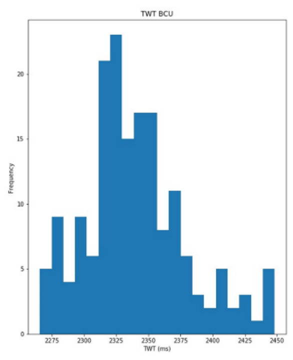
Vapnik, V. (1995) The nature of statistical learning theory. Springer, New York

Vollset, J. & Dore, A. G. (1984). A revised Triassic and Jurassic lithostratigraphic nomenclature for the Norwegian North Sea. Norwegian Petroleum Directorate Bulletin, Stavanger, 3.

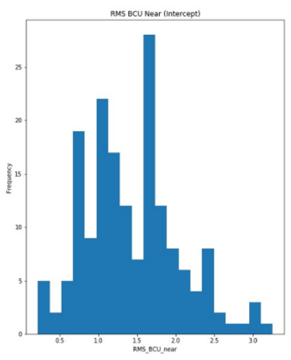
Appendix 1: Statistical details of each features



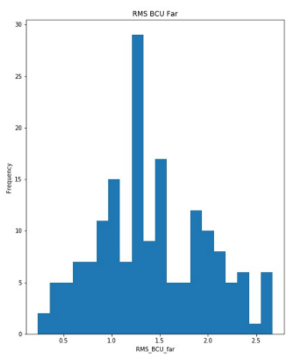
```
count     172.000000
mean     1657.685588
std        15.804070
min      1620.684447
25%      1646.030214
50%      1655.246243
75%      1668.007537
max      1700.156005
Name: Top_Balder_TWT, dtype: float64
```



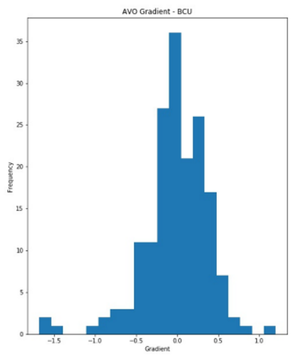
```
count     172.000000
mean     2339.905334
std        39.574086
min      2265.527246
25%      2314.650344
50%      2334.813690
75%      2358.855698
max      2448.202090
Name: BCU_TWT, dtype: float64
```



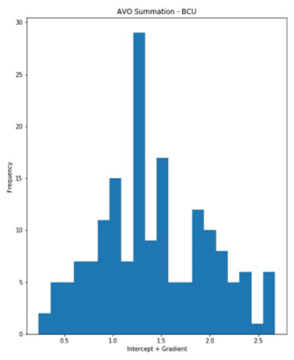
```
count    172.000000
mean       1.427015
std        0.606544
min        0.215142
25%        0.986025
50%        1.374334
75%        1.788085
max        3.253640
Name: RMS_BCU_near, dtype: float64
```



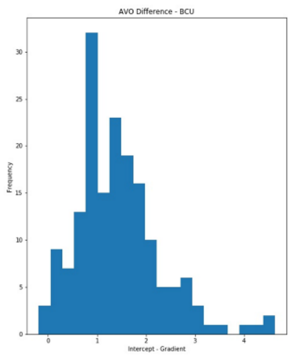
```
count    172.000000
mean       1.412228
std        0.565110
min        0.234508
25%        0.980981
50%        1.312440
75%        1.877980
max        2.670810
Name: RMS_BCU_far, dtype: float64
```



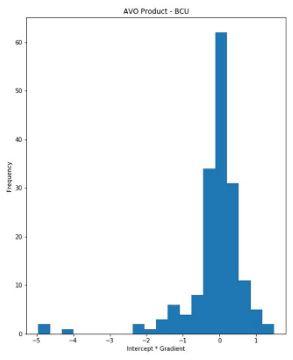
```
count    172.000000
mean      -0.014788
std        0.397409
min       -1.680412
25%       -0.179426
50%       -0.004471
75%        0.236351
max        1.198214
Name: Gradient, dtype: float64
```



```
count    172.000000
mean       1.412228
std        0.565110
min        0.234508
25%        0.980981
50%        1.312440
75%        1.877980
max        2.670810
Name: AVO_sum, dtype: float64
```

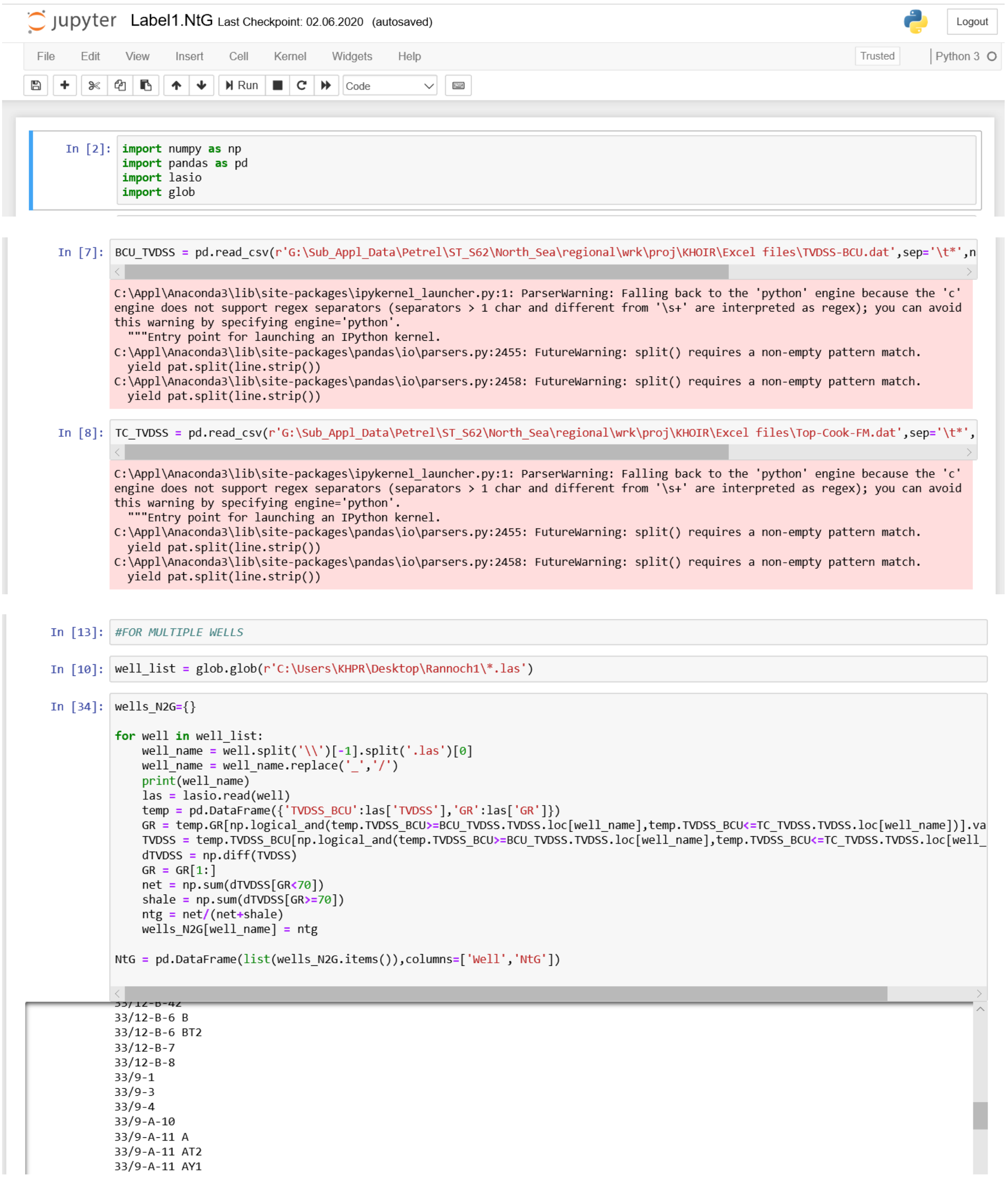


```
count    172.000000
mean       1.441803
std        0.855750
min       -0.192541
25%        0.896283
50%        1.362261
75%        1.831438
max        4.631719
Name: AVO_diff, dtype: float64
```



```
count    172.000000
mean      -0.123742
std        0.836091
min       -4.959412
25%       -0.243996
50%       -0.003954
75%        0.232982
max        1.492398
Name: AVO_prod, dtype: float64
```

Appendix 2: Net-to-gross calculation for all wells input code in Python



```
In [2]: import numpy as np
        import pandas as pd
        import lasio
        import glob
```

```
In [7]: BCU_TVDSS = pd.read_csv(r'G:\Sub_Appl_Data\Petrel\ST_S62\North_Sea\regional\wrk\proj\KHOIR\Excel files\TVDSS-BCU.dat',sep='\t*',n
```

```
C:\Appl\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine because the 'c'
engine does not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); you can avoid
this warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
C:\Appl\Anaconda3\lib\site-packages\pandas\io\parsers.py:2455: FutureWarning: split() requires a non-empty pattern match.
  yield pat.split(line.strip())
C:\Appl\Anaconda3\lib\site-packages\pandas\io\parsers.py:2458: FutureWarning: split() requires a non-empty pattern match.
  yield pat.split(line.strip())
```

```
In [8]: TC_TVDSS = pd.read_csv(r'G:\Sub_Appl_Data\Petrel\ST_S62\North_Sea\regional\wrk\proj\KHOIR\Excel files\Top-Cook-FM.dat',sep='\t*',
```

```
C:\Appl\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine because the 'c'
engine does not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); you can avoid
this warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
C:\Appl\Anaconda3\lib\site-packages\pandas\io\parsers.py:2455: FutureWarning: split() requires a non-empty pattern match.
  yield pat.split(line.strip())
C:\Appl\Anaconda3\lib\site-packages\pandas\io\parsers.py:2458: FutureWarning: split() requires a non-empty pattern match.
  yield pat.split(line.strip())
```

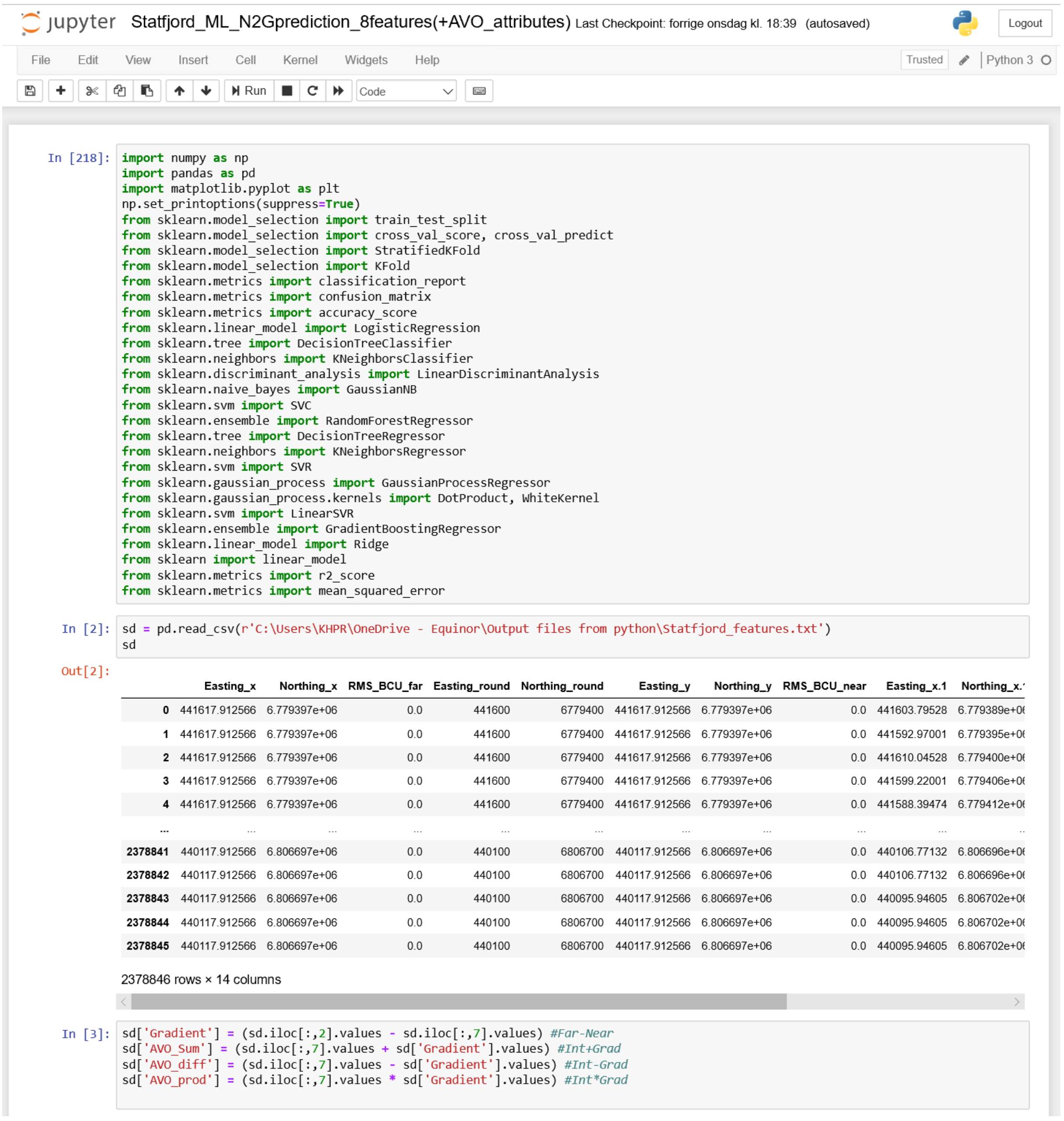
```
In [13]: #FOR MULTIPLE WELLS
```

```
In [10]: well_list = glob.glob(r'C:\Users\KHPR\Desktop\Rannoch1\*.las')
```

```
In [34]: wells_N2G={}

         for well in well_list:
             well_name = well.split('\\')[-1].split('.las')[0]
             well_name = well_name.replace('_','/')
             print(well_name)
             las = lasio.read(well)
             temp = pd.DataFrame({'TVDSS_BCU':las['TVDSS'],'GR':las['GR']})
             GR = temp.GR[np.logical_and(temp.TVDSS_BCU>=BCU_TVDSS.TVDSS.loc[well_name],temp.TVDSS_BCU<=TC_TVDSS.TVDSS.loc[well_name])].va
             TVDSS = temp.TVDSS_BCU[np.logical_and(temp.TVDSS_BCU>=BCU_TVDSS.TVDSS.loc[well_name],temp.TVDSS_BCU<=TC_TVDSS.TVDSS.loc[well_
             dTVDSS = np.diff(TVDSS)
             GR = GR[1:]
             net = np.sum(dTVDSS[GR<70])
             shale = np.sum(dTVDSS[GR>=70])
             ntg = net/(net+shale)
             wells_N2G[well_name] = ntg

         NtG = pd.DataFrame(list(wells_N2G.items()),columns=['Well','NtG'])
```

```
33/12-B-42
33/12-B-6 B
33/12-B-6 BT2
33/12-B-7
33/12-B-8
33/9-1
33/9-3
33/9-4
33/9-A-10
33/9-A-11 A
33/9-A-11 AT2
33/9-A-11 AY1
```

Appendix 3: Net-to-gross prediction input code in Python


jupyter
Statfjord_ML_N2Gprediction_8features(+AVO_attributes) Last Checkpoint: forrige onsdag kl. 18:39 (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

In [218]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
np.set_printoptions(suppress=True)
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel
from sklearn.svm import LinearSVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import Ridge
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

In [2]:
```
sd = pd.read_csv(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\Statfjord_features.txt')
sd
```

Out[2]:

	Easting_x	Northing_x	RMS_BCU_far	Easting_round	Northing_round	Easting_y	Northing_y	RMS_BCU_near	Easting_x.1	Northing_x.
0	441617.912566	6.779397e+06	0.0	441600	6779400	441617.912566	6.779397e+06	0.0	441603.79528	6.779389e+0
1	441617.912566	6.779397e+06	0.0	441600	6779400	441617.912566	6.779397e+06	0.0	441592.97001	6.779395e+0
2	441617.912566	6.779397e+06	0.0	441600	6779400	441617.912566	6.779397e+06	0.0	441610.04528	6.779400e+0
3	441617.912566	6.779397e+06	0.0	441600	6779400	441617.912566	6.779397e+06	0.0	441599.22001	6.779406e+0
4	441617.912566	6.779397e+06	0.0	441600	6779400	441617.912566	6.779397e+06	0.0	441588.39474	6.779412e+0
...	...	...	...	...	...	...	...	...	...	..
2378841	440117.912566	6.806697e+06	0.0	440100	6806700	440117.912566	6.806697e+06	0.0	440106.77132	6.806696e+0
2378842	440117.912566	6.806697e+06	0.0	440100	6806700	440117.912566	6.806697e+06	0.0	440106.77132	6.806696e+0
2378843	440117.912566	6.806697e+06	0.0	440100	6806700	440117.912566	6.806697e+06	0.0	440095.94605	6.806702e+0
2378844	440117.912566	6.806697e+06	0.0	440100	6806700	440117.912566	6.806697e+06	0.0	440095.94605	6.806702e+0
2378845	440117.912566	6.806697e+06	0.0	440100	6806700	440117.912566	6.806697e+06	0.0	440095.94605	6.806702e+0

2378846 rows × 14 columns

In [3]:
```
sd['Gradient'] = (sd.iloc[:,2].values - sd.iloc[:,7].values) #Far-Near
sd['AVO_Sum'] = (sd.iloc[:,7].values + sd['Gradient'].values) #Int+Grad
sd['AVO_diff'] = (sd.iloc[:,7].values - sd['Gradient'].values) #Int-Grad
sd['AVO_prod'] = (sd.iloc[:,7].values * sd['Gradient'].values) #Int*Grad
```

```
In [ ]: ##############################
```

```
In [63]: wells = pd.read_csv (r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\Stat_features_label1_selectedWells.txt')
         wells
```

Out[63]:

	Well	Easting	Northing	Top_Balder_TWT	BCU_TWT	RMS_BCU_near	RMS_BCU_far	NtG
0	211/24-4	434113.24	6783495.48	1660.124815	2344.146120	2.957468	2.434968	0.305144
1	33/12-1	437055.69	6786258.21	1636.961950	2283.454430	1.212534	1.418373	0.987179
2	33/12-B-1 A	438313.74	6786227.39	1652.419343	2361.018068	2.214515	1.867557	0.127886
3	33/12-B-1 B	438193.83	6786921.91	1663.011293	2310.467530	1.684443	1.898300	0.243066
4	33/12-B-1 C	438063.61	6786344.34	1654.795532	2328.535352	1.755582	1.238507	0.166065
...	...	...	...	...	...	...	...	...
167	33/9-C-6	442053.25	6798749.69	1676.983885	2403.847108	1.655461	1.522337	0.750849
168	33/9-C-7	441698.39	6797738.47	1667.437660	2367.370687	0.229123	0.482087	0.677724
169	33/9-C-8	442034.68	6796195.14	1671.039310	2344.972898	1.025667	2.223881	0.592816
170	33/9-C-8 A	440572.73	6796359.68	1656.938193	2329.879883	0.955554	0.928878	0.348550
171	33/9-C-8 AT2	440572.83	6796359.85	1656.938193	2329.879883	0.955554	0.928878	0.349802

172 rows × 8 columns

```
In [64]: wells['Gradient'] = (wells.iloc[:,6] - wells.iloc[:,5])
         wells['AVO_sum'] = (wells.iloc[:,5] + wells['Gradient'])
         wells['AVO_diff'] = (wells.iloc[:,5] - wells['Gradient'])
         wells['AVO_prod'] = (wells.iloc[:,5] * wells['Gradient'])

         wells
```

Out[64]:

	Well	Easting	Northing	Top_Balder_TWT	BCU_TWT	RMS_BCU_near	RMS_BCU_far	NtG	Gradient	AVO_sum	AVO_diff	AVO_prod
0	211/24-4	434113.24	6783495.48	1660.124815	2344.146120	2.957468	2.434968	0.305144	-0.522500	2.434968	3.479968	-1.545277
1	33/12-1	437055.69	6786258.21	1636.961950	2283.454430	1.212534	1.418373	0.987179	0.205839	1.418373	1.006695	0.249587
2	33/12-B-1 A	438313.74	6786227.39	1652.419343	2361.018068	2.214515	1.867557	0.127886	-0.346958	1.867557	2.561473	-0.768344
3	33/12-B-1 B	438193.83	6786921.91	1663.011293	2310.467530	1.684443	1.898300	0.243066	0.213857	1.898300	1.470586	0.360230
4	33/12-B-1 C	438063.61	6786344.34	1654.795532	2328.535352	1.755582	1.238507	0.166065	-0.517075	1.238507	2.272657	-0.907768
...	...	...	...	...	...	...	...	...	...	...	...	...
167	33/9-C-6	442053.25	6798749.69	1676.983885	2403.847108	1.655461	1.522337	0.750849	-0.133124	1.522337	1.788585	-0.220382
168	33/9-C-7	441698.39	6797738.47	1667.437660	2367.370687	0.229123	0.482087	0.677724	0.252964	0.482087	-0.023841	0.057960
169	33/9-C-8	442034.68	6796195.14	1671.039310	2344.972898	1.025667	2.223881	0.592816	1.198214	2.223881	-0.172547	1.228969
170	33/9-C-8 A	440572.73	6796359.68	1656.938193	2329.879883	0.955554	0.928878	0.348550	-0.026676	0.928878	0.982230	-0.025490
171	33/9-C-8 AT2	440572.83	6796359.85	1656.938193	2329.879883	0.955554	0.928878	0.349802	-0.026676	0.928878	0.982230	-0.025490

172 rows × 12 columns

```
In [7]: X = wells.iloc[:,[3,4,5,6,8,9,10,11]].values
        X[0,:]
        X
```

```
In [8]: y = wells.iloc[:,7].values
        y
```

```
In [9]: X_sd = sd.iloc[:,[13,10,7,2,14,15,16,17]].values
        X_sd
```

In [10]:
```
# let us try a little standardizing
from sklearn.preprocessing import StandardScaler

# let us actually fit to the entire area of interest and not just the places with wells!
scalar = StandardScaler().fit(X_sd)
X_transformed = scalar.transform(X)
X_sd_transformed = scalar.transform(X_sd)
```

In [246]:
```
# Spot Check Algorithms
models = []

models.append(('RFR', RandomForestRegressor(n_estimators=50, random_state=42, bootstrap=False)))
models.append(('LinearSVR', LinearSVR(max_iter=5000)))
models.append(('GPR', GaussianProcessRegressor(kernel = DotProduct() + WhiteKernel())))
models.append(('SVR', SVR(C=10, epsilon=0.1)))
models.append(('KNN', KNeighborsRegressor(n_neighbors=10)))
models.append(('CART', DecisionTreeRegressor(random_state=42)))
models.append(('SGDR', linear_model.SGDRegressor()))
models.append(('GBR', GradientBoostingRegressor(loss='huber', n_estimators=50)))


# evaluate each model in turn
results = []
names = []
for name, model in models:
    #kfold = StratifiedKFold(n_splits=10, random_state=42)
    kfold = KFold(n_splits=10, random_state=42, shuffle=True)
    cv_results = cross_val_score(model, X_transformed, y, cv=kfold, scoring='r2')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, abs(cv_results.mean()), cv_results.std()))
```

```
RFR: 0.877452 (0.640102)
LinearSVR: 0.151046 (0.201697)
GPR: 0.072982 (0.156716)
SVR: 0.278009 (0.268332)
KNN: 0.128249 (0.212094)
CART: 0.883515 (0.716798)
SGDR: 0.141696 (0.298149)
GBR: 0.124958 (0.192895)
```

In [13]:
```
#####
model_LinearSVR = LinearSVR(max_iter=5000)
model_LinearSVR.fit(X_transformed, y)
LinearSVR_prediction = model_LinearSVR.predict(X_sd_transformed)
LinearSVR_prediction
```

Out[13]:
```
array([-0.0994826 , -0.10055925, -0.09827005, ..., -0.23043981,
       -0.22484084, -0.22601215])
```

In [203]:
```
plt.hist(LinearSVR_prediction, bins = 20)
plt.title('LinearSVR')
plt.xlabel('NtG prediction')
plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\linearSVRtable.jpg')
print('Mean:', np.mean(LinearSVR_prediction))
print('St.Dev:', np.std(LinearSVR_prediction))
plt.show()
```

In [204]:
```
plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=LinearSVR_prediction, vmin=0, vmax=1)
clb = plt.colorbar()
clb.set_label('NtG Prediction', rotation=270, labelpad=15)
clb.ax.invert_yaxis()
plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
plt.title('NtG prediction - LinearSVRRegressor')
plt.gca().set_aspect('equal')
plt.gcf().set_size_inches(12, 8)
plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\LinearSVR.jpg')
```

In [54]:
```
###
model_GPR = GaussianProcessRegressor(kernel = DotProduct() + WhiteKernel())
model_GPR.fit(X_transformed, y)
GPR_prediction = model_GPR.predict(X_sd_transformed)
GPR_prediction
```

Out[54]:
```
array([ 0.02564194,  0.02483444,  0.02655136, ..., -0.05267248,
       -0.04887911, -0.04967268])
```

In [17]:
```
np.mean(GPR_prediction)
```

Out[17]:
```
0.2413478540699206
```

In [184]:
```
plt.hist(GPR_prediction, bins = 20)
plt.title('Gaussian Process Regressor')
plt.xlabel('NtG prediction')
plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\GPRtable.jpg')
print('Mean:', np.mean(GPR_prediction))
print('St.Dev:', np.std(GPR_prediction))
plt.show()
```

```
In [185]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=GPR_prediction, vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - GPRRegressor')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\GPR.jpg')
```

```
In [145]: ###
          model_SVR = SVR(C=10, epsilon=0.1)
          model_SVR.fit(X_transformed, y)
          SVR_prediction = model_SVR.predict(X_sd_transformed)
          SVR_prediction

Out[145]: array([0.42836543, 0.428255  , 0.42849028, ..., 0.09693634, 0.10993901,
                 0.10723125])

In [146]: np.mean(SVR_prediction)

Out[146]: 0.5585675287632528

In [201]: plt.hist(SVR_prediction, bins = 20)
          plt.title('Support Vector Regressor')
          plt.xlabel('NtG prediction')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\SVRtable.jpg')
          print('Mean:', np.mean(SVR_prediction))
          print('St.Dev:', np.std(SVR_prediction))
          plt.show()
```

```
In [202]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=SVR_prediction, vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - SVRRegressor')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\SVR.jpg')
```

```
In [25]: ###
         model_KNN = KNeighborsRegressor(n_neighbors=10)
         model_KNN.fit(X_transformed, y)
         KNN_prediction = model_KNN.predict(X_sd_transformed)
         KNN_prediction

Out[25]: array([0.36103366, 0.36103366, 0.36103366, ..., 0.40574605, 0.40574605,
                0.40574605])

In [26]: np.mean(KNN_prediction)

Out[26]: 0.43610725015039165

In [186]: plt.hist(KNN_prediction, bins = 20)
          plt.title('K-Nearest Neighbor Regressor')
          plt.xlabel('NtG prediction')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\KNNtable.jpg')
          print('Mean:', np.mean(KNN_prediction))
          print('St.Dev:', np.std(KNN_prediction))
          plt.show()
```

```
In [161]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=KNN_prediction)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - KNNRegressor')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
```

```
In [30]: ###
         model_CART = DecisionTreeRegressor(random_state=42)
         model_CART.fit(X_transformed, y)
         CART_prediction = model_CART.predict(X_sd_transformed)
         CART_prediction

Out[30]: array([0., 0., 0., ..., 0., 0., 0.])

In [31]: np.mean(CART_prediction)

Out[31]: 0.4660546146263803

In [188]: plt.hist(CART_prediction, bins = 20)
          plt.title('Decision Trees Regressor')
          plt.xlabel('NtG prediction')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\DTRtable.jpg')
          print('Mean:', np.mean(CART_prediction))
          print('St.Dev:', np.std(CART_prediction))
          plt.show()
```

```
In [189]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=CART_prediction, vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - DecisionTreeRegressor')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\DTR.jpg')
```

```
In [34]: #####
         model_RFR = RandomForestRegressor(n_estimators=50, random_state=42, bootstrap=False)
         model_RFR.fit(X_transformed, y)
         RFR_prediction = model_RFR.predict(X_sd_transformed)
         RFR_prediction

Out[34]: array([0.34143453, 0.34143453, 0.34143453, ..., 0.34143453, 0.34143453,
                0.34143453])

In [35]: np.mean(RFR_prediction)

Out[35]: 0.4860298235343122

In [190]: plt.hist(RFR_prediction, bins = 20)
          plt.title('Random Forest Regressor')
          plt.xlabel('NtG prediction')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\RFRtable.jpg')
          print('Mean:', np.mean(RFR_prediction))
          print('St.Dev:', np.std(RFR_prediction))
          plt.show()
```

```
In [191]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=RFR_prediction, vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - RandomForestRegressor')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\RFR.jpg')
```

```
In [38]: model_SGDR = linear_model.SGDRegressor()
         model_SGDR.fit(X_transformed, y)
         SGDR_prediction = model_SGDR.predict(X_sd_transformed)
         SGDR_prediction

Out[38]: array([-0.11061243, -0.1109848 , -0.11019306, ...,  0.14322476,
                 0.13906127,  0.13993228])

In [39]: np.mean(SGDR_prediction)

Out[39]: 0.1303577195815378

In [192]: plt.hist(SGDR_prediction, bins = 20)
          plt.title('Stochastic Gradient Descent Regressor')
          plt.xlabel('NtG prediction')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\SGDRtable.jpg')
          print('Mean:', np.mean(SGDR_prediction))
          print('St.Dev:', np.std(SGDR_prediction))
          plt.show()
```

```
In [194]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=SGDR_prediction, vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - SGDRegressor')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\SGDR.jpg')
```

```
In [42]: model_GBR = GradientBoostingRegressor(loss='huber', n_estimators=50)
         model_GBR.fit(X_transformed, y)
         GBR_prediction = model_GBR.predict(X_sd_transformed)
         GBR_prediction

Out[42]: array([0.41772345, 0.41772345, 0.41772345, ..., 0.41772345, 0.41772345,
                0.41772345])

In [43]: np.mean(GBR_prediction)

Out[43]: 0.5568627223767593

In [195]: plt.hist(GBR_prediction, bins = 20)
          plt.title('Gradient Boosting Regressor')
          plt.xlabel('NtG prediction')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\GBRtable.jpg')
          print('Mean:', np.mean(GBR_prediction))
          print('St.Dev:', np.std(GBR_prediction))
          plt.show()
```

```
In [196]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=GBR_prediction, vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - GradientBoostingRegressor')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\GBR.jpg')
```

```
In [ ]: ##########################
```

```
In [46]: import tpot
```

```
C:\Appl\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143: FutureWarning: The sklearn.metrics.scorer module is  depr
ecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported fr
om sklearn.metrics. Anything that cannot be imported from sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
C:\Appl\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143: FutureWarning: The sklearn.feature_selection.base module
is  deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imp
orted from sklearn.feature_selection. Anything that cannot be imported from sklearn.feature_selection is now part of the privat
e API.
  warnings.warn(message, FutureWarning)
```

```
In [47]: pipeline_optimizer = tpot.TPOTRegressor()
         pipeline_optimizer = tpot.TPOTRegressor(n_jobs=7, generations=100, population_size=100, cv=10,
                                                 random_state=42, verbosity=2)
```

```
In [48]: pipeline_optimizer.fit(X_transformed, y)
```

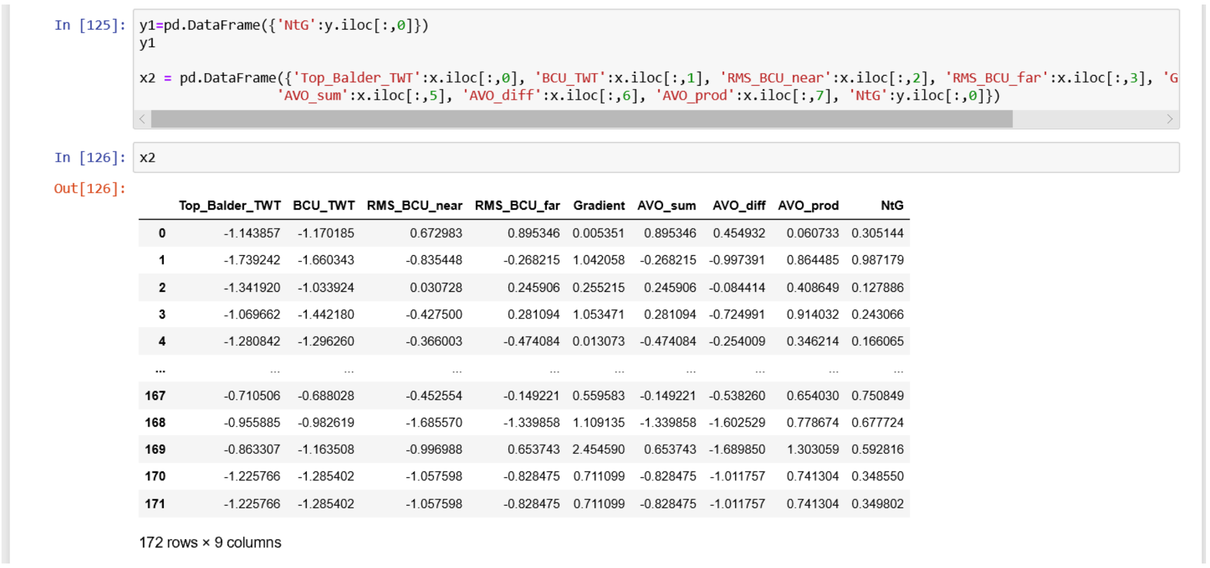
```
[illegible]
Generation 2 - Current best internal CV score: [illegible]
[illegible]
```

```
In [49]: tpot_prediction_001 = pipeline_optimizer.predict(X_sd_transformed)
         tpot_prediction_001

Out[49]: array([0.47142139, 0.47142139, 0.46156933, ..., 0.4960154 , 0.4960154 ,
                0.4960154 ])
```

```
In [197]: plt.hist(tpot_prediction_001, bins = 20)
          plt.title('AutoML-result')
          plt.xlabel('NtG prediction')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\AutoMLtable.jpg')
          print('Mean:', np.mean(tpot_prediction_001))
          print('St.Dev:', np.std(tpot_prediction_001))
          plt.show()
```

```
In [198]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=tpot_prediction_001)
          clb = plt.colorbar()
          clb.set_label('NTG prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG,vmin=0,vmax=1)
          plt.title('Ntg prediction -AutoML result')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\AutoML.jpg')
```

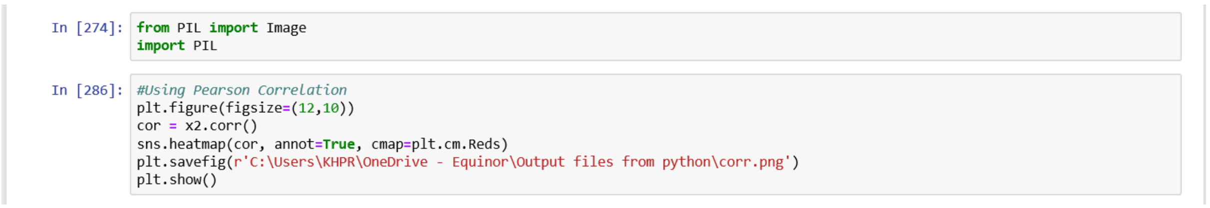
Appendix 4: Net-to-gross prediction validation input code in Python



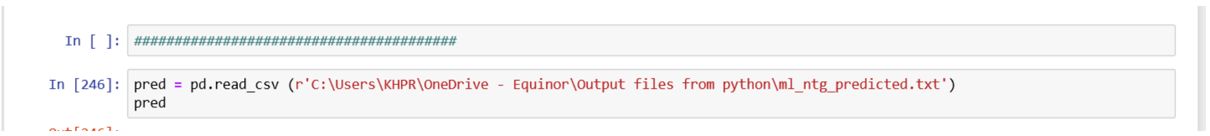
```
In [125]: y1=pd.DataFrame({'NtG':y.iloc[:,0]})
          y1

          x2 = pd.DataFrame({'Top_Balder_TWT':x.iloc[:,0], 'BCU_TWT':x.iloc[:,1], 'RMS_BCU_near':x.iloc[:,2], 'RMS_BCU_far':x.iloc[:,3], 'G
                             'AVO_sum':x.iloc[:,5], 'AVO_diff':x.iloc[:,6], 'AVO_prod':x.iloc[:,7], 'NtG':y.iloc[:,0]})

In [126]: x2

Out[126]:
```

	Top_Balder_TWT	BCU_TWT	RMS_BCU_near	RMS_BCU_far	Gradient	AVO_sum	AVO_diff	AVO_prod	NtG
0	-1.143857	-1.170185	0.672983	0.895346	0.005351	0.895346	0.454932	0.060733	0.305144
1	-1.739242	-1.660343	-0.835448	-0.268215	1.042058	-0.268215	-0.997391	0.864485	0.987179
2	-1.341920	-1.033924	0.030728	0.245906	0.255215	0.245906	-0.084414	0.408649	0.127886
3	-1.069662	-1.442180	-0.427500	0.281094	1.053471	0.281094	-0.724991	0.914032	0.243066
4	-1.280842	-1.296260	-0.366003	-0.474084	0.013073	-0.474084	-0.254009	0.346214	0.166065
...	...	...	...	...	...	...	...	...	...
167	-0.710506	-0.688028	-0.452554	-0.149221	0.559583	-0.149221	-0.538260	0.654030	0.750849
168	-0.955885	-0.982619	-1.685570	-1.339858	1.109135	-1.339858	-1.602529	0.778674	0.677724
169	-0.863307	-1.163508	-0.996988	0.653743	2.454590	0.653743	-1.689850	1.303059	0.592816
170	-1.225766	-1.285402	-1.057598	-0.828475	0.711099	-0.828475	-1.011757	0.741304	0.348550
171	-1.225766	-1.285402	-1.057598	-0.828475	0.711099	-0.828475	-1.011757	0.741304	0.349802

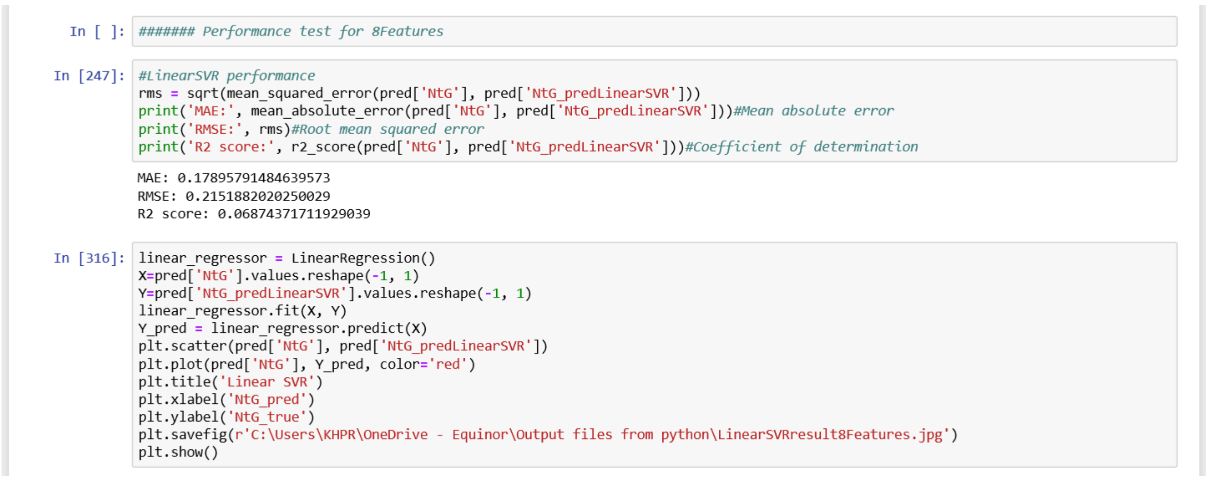
172 rows × 9 columns



```
In [274]: from PIL import Image
          import PIL

In [286]: #Using Pearson Correlation
          plt.figure(figsize=(12,10))
          cor = x2.corr()
          sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\corr.png')
          plt.show()
```



```
In [ ]: ##########################################

In [246]: pred = pd.read_csv (r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\ml_ntg_predicted.txt')
          pred
```



```
In [ ]: ####### Performance test for 8Features

In [247]: #LinearSVR performance
          rms = sqrt(mean_squared_error(pred['NtG'], pred['NtG_predLinearSVR']))
          print('MAE:', mean_absolute_error(pred['NtG'], pred['NtG_predLinearSVR']))#Mean absolute error
          print('RMSE:', rms)#Root mean squared error
          print('R2 score:', r2_score(pred['NtG'], pred['NtG_predLinearSVR']))#Coefficient of determination

          MAE: 0.17895791484639573
          RMSE: 0.2151882020250029
          R2 score: 0.06874371711929039

In [316]: linear_regressor = LinearRegression()
          X=pred['NtG'].values.reshape(-1, 1)
          Y=pred['NtG_predLinearSVR'].values.reshape(-1, 1)
          linear_regressor.fit(X, Y)
          Y_pred = linear_regressor.predict(X)
          plt.scatter(pred['NtG'], pred['NtG_predLinearSVR'])
          plt.plot(pred['NtG'], Y_pred, color='red')
          plt.title('Linear SVR')
          plt.xlabel('NtG_pred')
          plt.ylabel('NtG_true')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\LinearSVRresult8Features.jpg')
          plt.show()
```

```
In [248]: #GPR performance
          rms = sqrt(mean_squared_error(pred['NtG'], pred['NtG_predGPR']))
          print('MAE:', mean_absolute_error(pred['NtG'], pred['NtG_predGPR']))#Mean absolute error
          print('RMSE:', rms)#Root mean squared error
          print('R2 score:', r2_score(pred['NtG'], pred['NtG_predGPR']))#Coefficient of determination

          MAE: 0.18017519637173207
          RMSE: 0.21401431660879258
          R2 score: 0.07887630247897248

In [317]: linear_regressor = LinearRegression()
          X=pred['NtG'].values.reshape(-1, 1)
          Y=pred['NtG_predGPR'].values.reshape(-1, 1)
          linear_regressor.fit(X, Y)
          Y_pred = linear_regressor.predict(X)
          plt.scatter(pred['NtG'], pred['NtG_predGPR'])
          plt.plot(pred['NtG'], Y_pred, color='red')
          plt.title('Gaussian Process Regressor')
          plt.xlabel('NtG_pred')
          plt.ylabel('NtG_true')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\GPRresult8Features.jpg')
          plt.show()
```

```
In [249]: #SVR performance
          rms = sqrt(mean_squared_error(pred['NtG'], pred['NtG_predSVR']))
          print('MAE:', mean_absolute_error(pred['NtG'], pred['NtG_predSVR']))#Mean absolute error
          print('RMSE:', rms)#Root mean squared error
          print('R2 score:', r2_score(pred['NtG'], pred['NtG_predSVR']))#Coefficient of determination

          MAE: 0.1578972828928246
          RMSE: 0.19179614816467616
          R2 score: 0.26020385479147434

In [318]: linear_regressor = LinearRegression()
          X=pred['NtG'].values.reshape(-1, 1)
          Y=pred['NtG_predSVR'].values.reshape(-1, 1)
          linear_regressor.fit(X, Y)
          Y_pred = linear_regressor.predict(X)
          plt.scatter(pred['NtG'], pred['NtG_predSVR'])
          plt.plot(pred['NtG'], Y_pred, color='red')
          plt.title('Support Vector Regressor')
          plt.xlabel('NtG_pred')
          plt.ylabel('NtG_true')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\SVRresult8Features.jpg')
          plt.show()
```

```
In [250]: #KNN performance
          rms = sqrt(mean_squared_error(pred['NtG'], pred['NtG_predKNN']))
          print('MAE:', mean_absolute_error(pred['NtG'], pred['NtG_predKNN']))#Mean absolute error
          print('RMSE:', rms)#Root mean squared error
          print('R2 score:', r2_score(pred['NtG'], pred['NtG_predKNN']))#Coefficient of determination

          MAE: 0.16970770311971917
          RMSE: 0.20638889425136958
          R2 score: 0.14334697934054685

In [315]: linear_regressor = LinearRegression()
          X=pred['NtG'].values.reshape(-1, 1)
          Y=pred['NtG_predKNN'].values.reshape(-1, 1)
          linear_regressor.fit(X, Y)
          Y_pred = linear_regressor.predict(X)
          plt.scatter(pred['NtG'], pred['NtG_predKNN'])
          plt.plot(pred['NtG'], Y_pred, color='red')
          plt.title('K-Nearest Neighbour')
          plt.xlabel('NtG_pred')
          plt.ylabel('NtG_true')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\KNNresult8Features.jpg')
          plt.show()
```

```
In [251]: #DTR performance
          rms = sqrt(mean_squared_error(pred['NtG'], pred['NtG_predDTR']))
          print('MAE:', mean_absolute_error(pred['NtG'], pred['NtG_predDTR']))#Mean absolute error
          print('RMSE:', rms)#Root mean squared error
          print('R2 score:', r2_score(pred['NtG'], pred['NtG_predDTR']))#Coefficient of determination

          MAE: 0.02920276920238602
          RMSE: 0.08537643635906635
          R2 score: 0.853408648601155

In [319]: linear_regressor = LinearRegression()
          X=pred['NtG'].values.reshape(-1, 1)
          Y=pred['NtG_predDTR'].values.reshape(-1, 1)
          linear_regressor.fit(X, Y)
          Y_pred = linear_regressor.predict(X)
          plt.scatter(pred['NtG'], pred['NtG_predDTR'])
          plt.plot(pred['NtG'], Y_pred, color='red')
          plt.title('Decision Trees Regressor')
          plt.xlabel('NtG_pred')
          plt.ylabel('NtG_true')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\DTRresult8Features.jpg')
          plt.show()
```

```
In [252]: #RFR performance
          rms = sqrt(mean_squared_error(pred['NtG'], pred['NtG_predRFR']))
          print('MAE:', mean_absolute_error(pred['NtG'], pred['NtG_predRFR']))#Mean absolute error
          print('RMSE:', rms)#Root mean squared error
          print('R2 score:', r2_score(pred['NtG'], pred['NtG_predRFR']))#Coefficient of determination

          MAE: 0.02932807305032732
          RMSE: 0.08101402573611907
          R2 score: 0.8680064439932874
```

```
In [320]: linear_regressor = LinearRegression()
          X=pred['NtG'].values.reshape(-1, 1)
          Y=pred['NtG_predRFR'].values.reshape(-1, 1)
          linear_regressor.fit(X, Y)
          Y_pred = linear_regressor.predict(X)
          plt.scatter(pred['NtG'], pred['NtG_predRFR'])
          plt.plot(pred['NtG'], Y_pred, color='red')
          plt.title('Random Forest Regressor')
          plt.xlabel('NtG_pred')
          plt.ylabel('NtG_true')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\RFRresult8Features.jpg')
          plt.show()
```

```
In [253]: #SGDR performance
          rms = sqrt(mean_squared_error(pred['NtG'], pred['NtG_predSGDR']))
          print('MAE:', mean_absolute_error(pred['NtG'], pred['NtG_predSGDR']))#Mean absolute error
          print('RMSE:', rms)#Root mean squared error
          print('R2 score:', r2_score(pred['NtG'], pred['NtG_predSGDR']))#Coefficient of determination

          MAE: 0.1868782498489886
          RMSE: 0.22466792002255365
          R2 score: -0.0151130974776863
```

```
In [321]: linear_regressor = LinearRegression()
          X=pred['NtG'].values.reshape(-1, 1)
          Y=pred['NtG_predSGDR'].values.reshape(-1, 1)
          linear_regressor.fit(X, Y)
          Y_pred = linear_regressor.predict(X)
          plt.scatter(pred['NtG'], pred['NtG_predSGDR'])
          plt.plot(pred['NtG'], Y_pred, color='red')
          plt.title('Stochastic Gradient Descent Regressor')
          plt.xlabel('NtG_pred')
          plt.ylabel('NtG_true')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\SGDRresult8Features.jpg')
          plt.show()
```

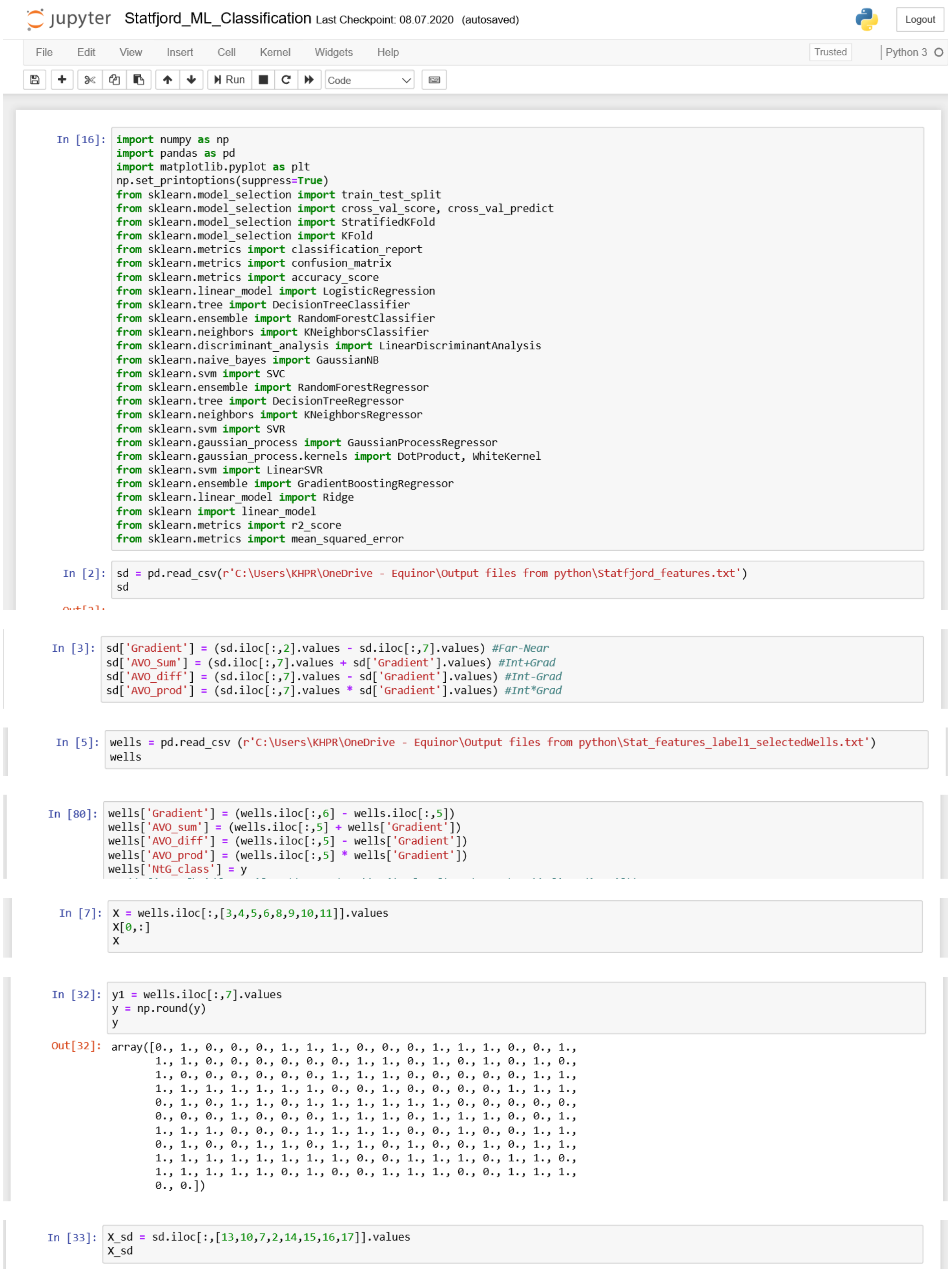
```
In [254]: #GBR performance
          rms = sqrt(mean_squared_error(pred['NtG'], pred['NtG_predGBR']))
          print('MAE:', mean_absolute_error(pred['NtG'], pred['NtG_predGBR']))#Mean absolute error
          print('RMSE:', rms)#Root mean squared error
          print('R2 score:', r2_score(pred['NtG'], pred['NtG_predGBR']))#Coefficient of determination

          MAE: 0.12340088061246313
          RMSE: 0.1569951453873776
          R2 score: 0.5043161734646342
```

```
In [322]: linear_regressor = LinearRegression()
          X=pred['NtG'].values.reshape(-1, 1)
          Y=pred['NtG_predGBR'].values.reshape(-1, 1)
          linear_regressor.fit(X, Y)
          Y_pred = linear_regressor.predict(X)
          plt.scatter(pred['NtG'], pred['NtG_predGBR'])
          plt.plot(pred['NtG'], Y_pred, color='red')
          plt.title('Gradient Boosting Regressor')
          plt.xlabel('NtG_pred')
          plt.ylabel('NtG_true')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\GBRresult8Features.jpg')
          plt.show()
```

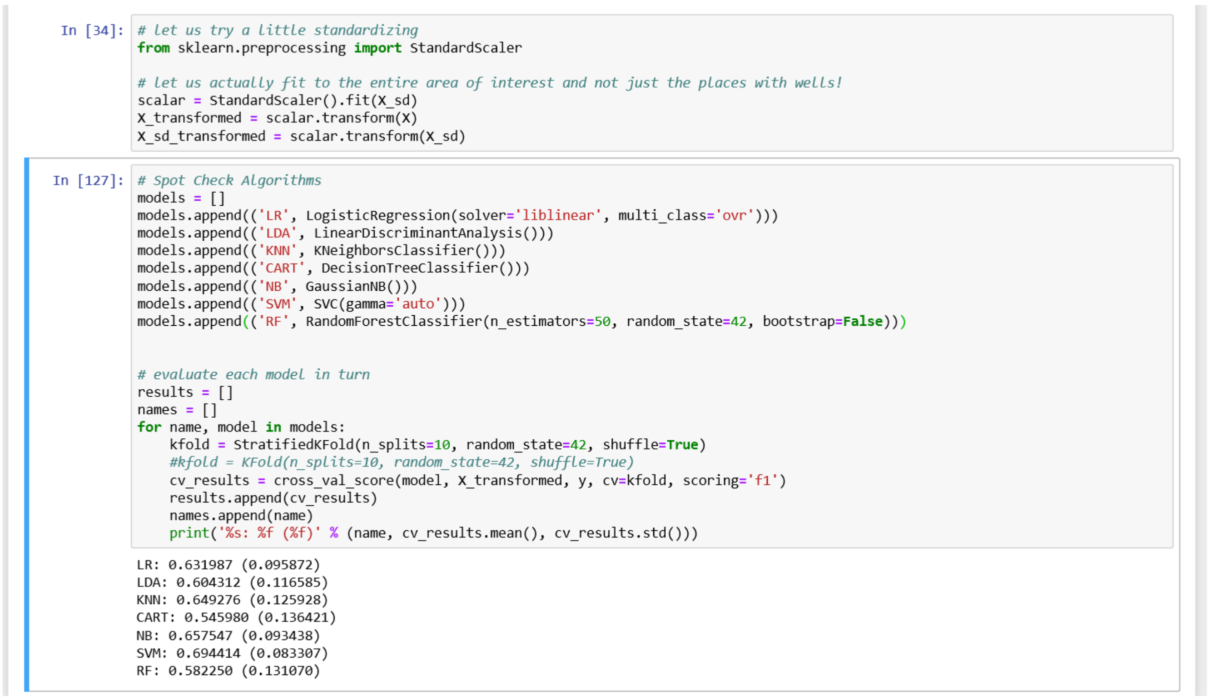
```
In [255]: #AutoML performance
          rms = sqrt(mean_squared_error(pred['NtG'], pred['NtG_predAutoML']))
          print('MAE:', mean_absolute_error(pred['NtG'], pred['NtG_predAutoML']))#Mean absolute error
          print('RMSE:', rms)#Root mean squared error
          print('R2 score:', r2_score(pred['NtG'], pred['NtG_predAutoML']))#Coefficient of determination

          MAE: 0.1801065162988866
          RMSE: 0.2137888748122647
          R2 score: 0.08081589615200091
```

```
In [323]: linear_regressor = LinearRegression()
          X=pred['NtG'].values.reshape(-1, 1)
          Y=pred['NtG_predAutoML'].values.reshape(-1, 1)
          linear_regressor.fit(X, Y)
          Y_pred = linear_regressor.predict(X)
          plt.scatter(pred['NtG'], pred['NtG_predAutoML'])
          plt.plot(pred['NtG'], Y_pred, color='red')
          plt.title('AutoML')
          plt.xlabel('NtG_pred')
          plt.ylabel('NtG_true')
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\AutoMLresult8Features.jpg')
          plt.show()
```

Appendix 5: Net-to-gross classification input code in Python

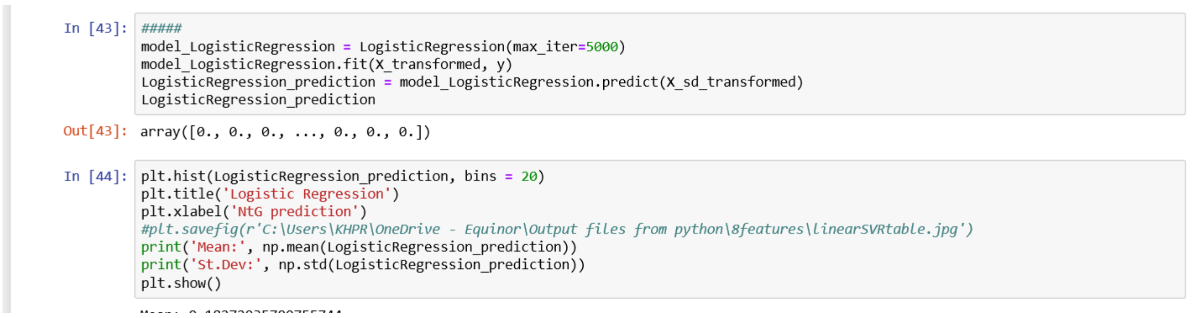


```
In [16]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         np.set_printoptions(suppress=True)
         from sklearn.model_selection import train_test_split
         from sklearn.model_selection import cross_val_score, cross_val_predict
         from sklearn.model_selection import StratifiedKFold
         from sklearn.model_selection import KFold
         from sklearn.metrics import classification_report
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import accuracy_score
         from sklearn.linear_model import LogisticRegression
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         from sklearn.naive_bayes import GaussianNB
         from sklearn.svm import SVC
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.neighbors import KNeighborsRegressor
         from sklearn.svm import SVR
         from sklearn.gaussian_process import GaussianProcessRegressor
         from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel
         from sklearn.svm import LinearSVR
         from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.linear_model import Ridge
         from sklearn import linear_model
         from sklearn.metrics import r2_score
         from sklearn.metrics import mean_squared_error

 In [2]: sd = pd.read_csv(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\Statfjord_features.txt')
         sd

 In [3]: sd['Gradient'] = (sd.iloc[:,2].values - sd.iloc[:,7].values) #Far-Near
         sd['AVO_Sum'] = (sd.iloc[:,7].values + sd['Gradient'].values) #Int+Grad
         sd['AVO_diff'] = (sd.iloc[:,7].values - sd['Gradient'].values) #Int-Grad
         sd['AVO_prod'] = (sd.iloc[:,7].values * sd['Gradient'].values) #Int*Grad

 In [5]: wells = pd.read_csv (r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\Stat_features_label1_selectedWells.txt')
         wells

In [80]: wells['Gradient'] = (wells.iloc[:,6] - wells.iloc[:,5])
         wells['AVO_sum'] = (wells.iloc[:,5] + wells['Gradient'])
         wells['AVO_diff'] = (wells.iloc[:,5] - wells['Gradient'])
         wells['AVO_prod'] = (wells.iloc[:,5] * wells['Gradient'])
         wells['NtG_class'] = y

 In [7]: X = wells.iloc[:,[3,4,5,6,8,9,10,11]].values
         X[0,:]
         X

In [32]: y1 = wells.iloc[:,7].values
         y = np.round(y)
         y

Out[32]: array([0., 1., 0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 1., 1., 0., 0., 1.,
                1., 1., 0., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0.,
                1., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 1., 1.,
                1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1.,
                0., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0.,
                0., 0., 0., 1., 0., 0., 0., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1.,
                1., 1., 1., 0., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1.,
                0., 1., 0., 0., 1., 1., 0., 1., 1., 0., 1., 0., 0., 1., 0., 1., 1.,
                1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 1., 1., 0., 1., 1., 0.,
                1., 1., 1., 1., 1., 0., 1., 0., 0., 1., 1., 1., 0., 0., 1., 1., 1.,
                0., 0.])

In [33]: X_sd = sd.iloc[:,[13,10,7,2,14,15,16,17]].values
         X_sd
```



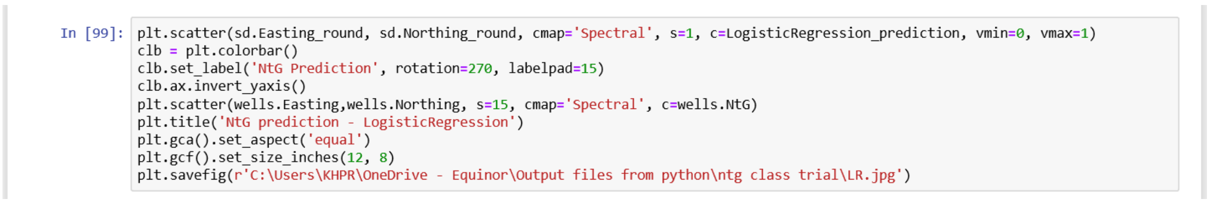
```
In [34]:  # let us try a little standardizing
          from sklearn.preprocessing import StandardScaler

          # let us actually fit to the entire area of interest and not just the places with wells!
          scalar = StandardScaler().fit(X_sd)
          X_transformed = scalar.transform(X)
          X_sd_transformed = scalar.transform(X_sd)
```

```
In [127]: # Spot Check Algorithms
          models = []
          models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
          models.append(('LDA', LinearDiscriminantAnalysis()))
          models.append(('KNN', KNeighborsClassifier()))
          models.append(('CART', DecisionTreeClassifier()))
          models.append(('NB', GaussianNB()))
          models.append(('SVM', SVC(gamma='auto')))
          models.append(('RF', RandomForestClassifier(n_estimators=50, random_state=42, bootstrap=False)))


          # evaluate each model in turn
          results = []
          names = []
          for name, model in models:
              kfold = StratifiedKFold(n_splits=10, random_state=42, shuffle=True)
              #kfold = KFold(n_splits=10, random_state=42, shuffle=True)
              cv_results = cross_val_score(model, X_transformed, y, cv=kfold, scoring='f1')
              results.append(cv_results)
              names.append(name)
              print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

          LR: 0.631987 (0.095872)
          LDA: 0.604312 (0.116585)
          KNN: 0.649276 (0.125928)
          CART: 0.545980 (0.136421)
          NB: 0.657547 (0.093438)
          SVM: 0.694414 (0.083307)
          RF: 0.582250 (0.131070)
```

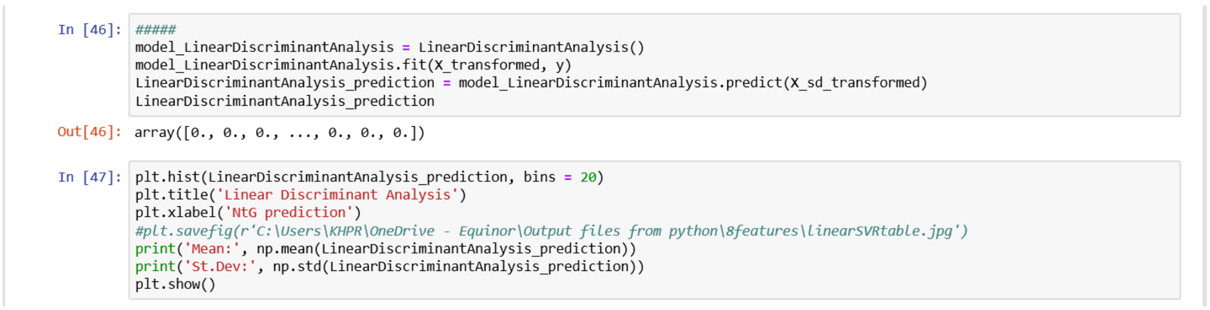


```
In [43]:  #####
          model_LogisticRegression = LogisticRegression(max_iter=5000)
          model_LogisticRegression.fit(X_transformed, y)
          LogisticRegression_prediction = model_LogisticRegression.predict(X_sd_transformed)
          LogisticRegression_prediction

Out[43]:  array([0., 0., 0., ..., 0., 0., 0.])
```

```
In [44]:  plt.hist(LogisticRegression_prediction, bins = 20)
          plt.title('Logistic Regression')
          plt.xlabel('NtG prediction')
          #plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\linearSVRtable.jpg')
          print('Mean:', np.mean(LogisticRegression_prediction))
          print('St.Dev:', np.std(LogisticRegression_prediction))
          plt.show()
```

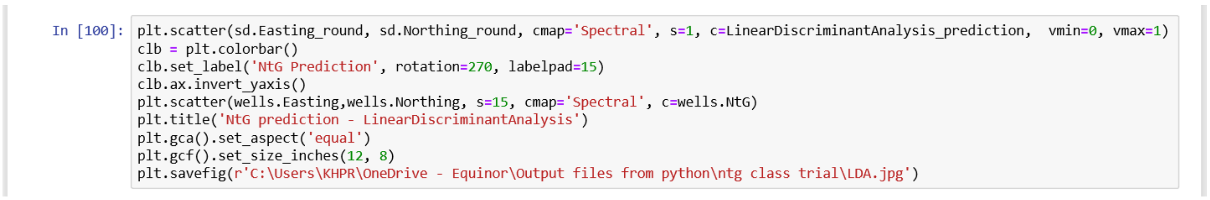


```
In [99]:  plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=LogisticRegression_prediction, vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - LogisticRegression')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\ntg class trial\LR.jpg')
```



```
In [46]:  #####
          model_LinearDiscriminantAnalysis = LinearDiscriminantAnalysis()
          model_LinearDiscriminantAnalysis.fit(X_transformed, y)
          LinearDiscriminantAnalysis_prediction = model_LinearDiscriminantAnalysis.predict(X_sd_transformed)
          LinearDiscriminantAnalysis_prediction

Out[46]:  array([0., 0., 0., ..., 0., 0., 0.])
```

```
In [47]:  plt.hist(LinearDiscriminantAnalysis_prediction, bins = 20)
          plt.title('Linear Discriminant Analysis')
          plt.xlabel('NtG prediction')
          #plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\linearSVRtable.jpg')
          print('Mean:', np.mean(LinearDiscriminantAnalysis_prediction))
          print('St.Dev:', np.std(LinearDiscriminantAnalysis_prediction))
          plt.show()
```



```
In [100]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=LinearDiscriminantAnalysis_prediction,  vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - LinearDiscriminantAnalysis')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\ntg class trial\LDA.jpg')
```

```
In [49]:  #####
          model_KNeighborsClassifier = KNeighborsClassifier()
          model_KNeighborsClassifier.fit(X_transformed, y)
          KNeighborsClassifier_prediction = model_KNeighborsClassifier.predict(X_sd_transformed)
          KNeighborsClassifier_prediction

Out[49]:  array([0., 0., 0., ..., 0., 0., 0.])

In [50]:  plt.hist(KNeighborsClassifier_prediction, bins = 20)
          plt.title('KNeighborsClassifier')
          plt.xlabel('NtG prediction')
          #plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\linearSVRtable.jpg')
          print('Mean:', np.mean(KNeighborsClassifier_prediction))
          print('St.Dev:', np.std(KNeighborsClassifier_prediction))
          plt.show()
```

```
In [101]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=KNeighborsClassifier_prediction, vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - KNeighborsClassifier')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\ntg class trial\KNN.jpg')
```

```
In [52]:  #####
          model_DecisionTreeClassifier = DecisionTreeClassifier()
          model_DecisionTreeClassifier.fit(X_transformed, y)
          DecisionTreeClassifier_prediction = model_DecisionTreeClassifier.predict(X_sd_transformed)
          DecisionTreeClassifier_prediction

Out[52]:  array([0., 0., 0., ..., 0., 0., 0.])

In [53]:  plt.hist(DecisionTreeClassifier_prediction, bins = 20)
          plt.title('DecisionTreeClassifier')
          plt.xlabel('NtG prediction')
          #plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\linearSVRtable.jpg')
          print('Mean:', np.mean(DecisionTreeClassifier_prediction))
          print('St.Dev:', np.std(DecisionTreeClassifier_prediction))
          plt.show()
```

```
In [108]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=DecisionTreeClassifier_prediction.round(), vmin=0, vmax=
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - DecisionTreeClassifier')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\ntg class trial\DTR.jpg')
```

```
In [55]:  #####
          model_GaussianNB = GaussianNB()
          model_GaussianNB.fit(X_transformed, y)
          GaussianNB_prediction = model_GaussianNB.predict(X_sd_transformed)
          GaussianNB_prediction

Out[55]:  array([0., 0., 0., ..., 0., 0., 0.])

In [56]:  plt.hist(GaussianNB_prediction, bins = 20)
          plt.title('Naive-Bayes Classifier')
          plt.xlabel('NtG prediction')
          #plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\linearSVRtable.jpg')
          print('Mean:', np.mean(GaussianNB_prediction))
          print('St.Dev:', np.std(GaussianNB_prediction))
          plt.show()
```

```
In [103]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=GaussianNB_prediction, vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - Naive-Bayes Classifier')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\ntg class trial\NB.jpg')
```

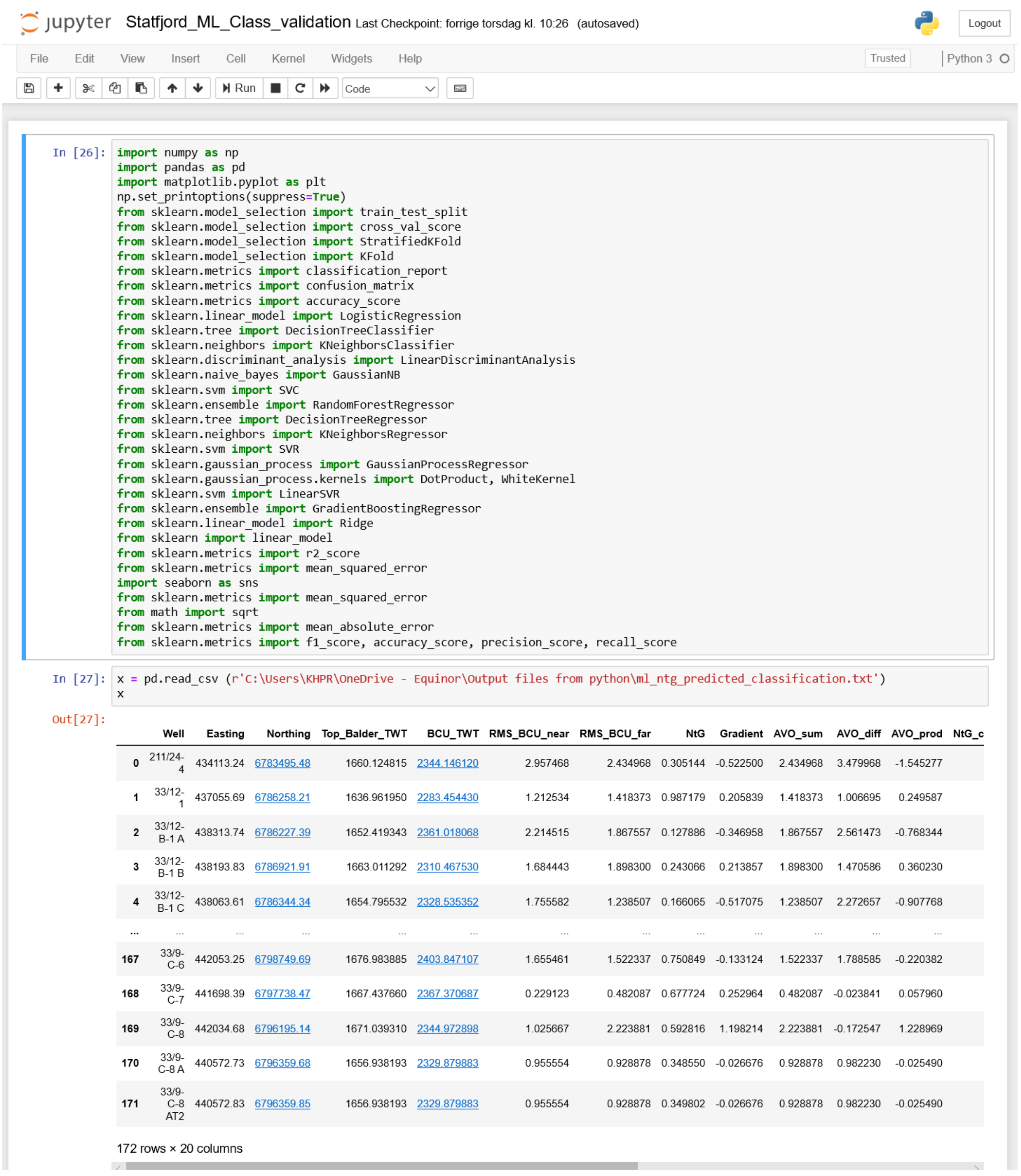
```
In [58]: #####
         model_SVC = SVC(gamma='auto')
         model_SVC.fit(X_transformed, y)
         SVC_prediction = model_SVC.predict(X_sd_transformed)
         SVC_prediction

Out[58]: array([0., 0., 0., ..., 0., 0., 0.])

In [59]: plt.hist(SVC_prediction, bins = 20)
         plt.title('Support Vector Machine Classifier')
         plt.xlabel('NtG prediction')
         #plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\linearSVRtable.jpg')
         print('Mean:', np.mean(SVC_prediction))
         print('St.Dev:', np.std(SVC_prediction))
         plt.show()
```

```
In [104]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=SVC_prediction,  vmin=0, vmax=1)
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - Support Vector Machine Classifier')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\ntg class trial\SVM.jpg')
```

```
In [62]: #####
         model_RandomForestClassifier = RandomForestClassifier(n_estimators=50, random_state=42, bootstrap=False)
         model_RandomForestClassifier.fit(X_transformed, y)
         RandomForestClassifier_prediction = model_RandomForestClassifier.predict(X_sd_transformed)
         RandomForestClassifier_prediction

Out[62]: array([0., 0., 0., ..., 0., 0., 0.])

In [63]: plt.hist(RandomForestClassifier_prediction, bins = 20)
         plt.title('RandomForest Classifier')
         plt.xlabel('NtG prediction')
         #plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\8features\linearSVRtable.jpg')
         print('Mean:', np.mean(RandomForestClassifier_prediction))
         print('St.Dev:', np.std(RandomForestClassifier_prediction))
         plt.show()
```

```
In [124]: plt.scatter(sd.Easting_round, sd.Northing_round, cmap='Spectral', s=1, c=RandomForestClassifier_prediction.round(),  vmin=0, vmax
          clb = plt.colorbar()
          clb.set_label('NtG Prediction', rotation=270, labelpad=15)
          clb.ax.invert_yaxis()
          plt.scatter(wells.Easting,wells.Northing, s=15, cmap='Spectral', c=wells.NtG)
          plt.title('NtG prediction - RandomForest Classifier')
          plt.gca().set_aspect('equal')
          plt.gcf().set_size_inches(12, 8)
          plt.savefig(r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\ntg class trial\RFR.jpg')
```

Appendix 6: Net-to-gross classification validation input code in Python



Statfjord_ML_Class_validation Last Checkpoint: forrige torsdag kl. 10:26 (autosaved)

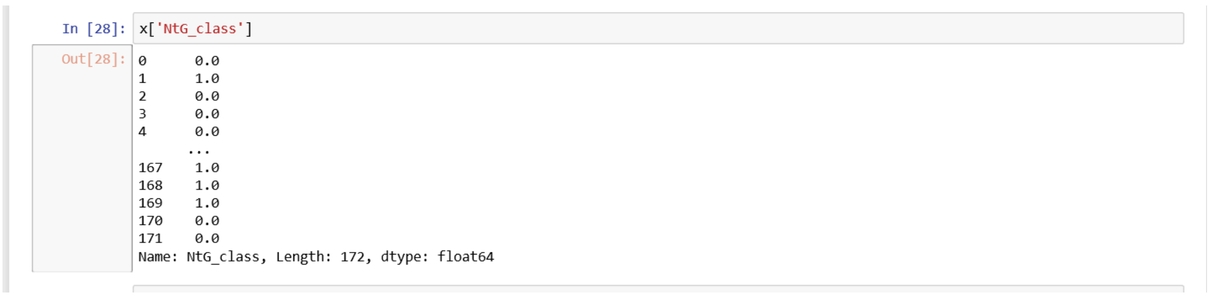
```
In [26]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         np.set_printoptions(suppress=True)
         from sklearn.model_selection import train_test_split
         from sklearn.model_selection import cross_val_score
         from sklearn.model_selection import StratifiedKFold
         from sklearn.model_selection import KFold
         from sklearn.metrics import classification_report
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import accuracy_score
         from sklearn.linear_model import LogisticRegression
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         from sklearn.naive_bayes import GaussianNB
         from sklearn.svm import SVC
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.neighbors import KNeighborsRegressor
         from sklearn.svm import SVR
         from sklearn.gaussian_process import GaussianProcessRegressor
         from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel
         from sklearn.svm import LinearSVR
         from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.linear_model import Ridge
         from sklearn import linear_model
         from sklearn.metrics import r2_score
         from sklearn.metrics import mean_squared_error
         import seaborn as sns
         from sklearn.metrics import mean_squared_error
         from math import sqrt
         from sklearn.metrics import mean_absolute_error
         from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score
```

```
In [27]: x = pd.read_csv (r'C:\Users\KHPR\OneDrive - Equinor\Output files from python\ml_ntg_predicted_classification.txt')
         x
```

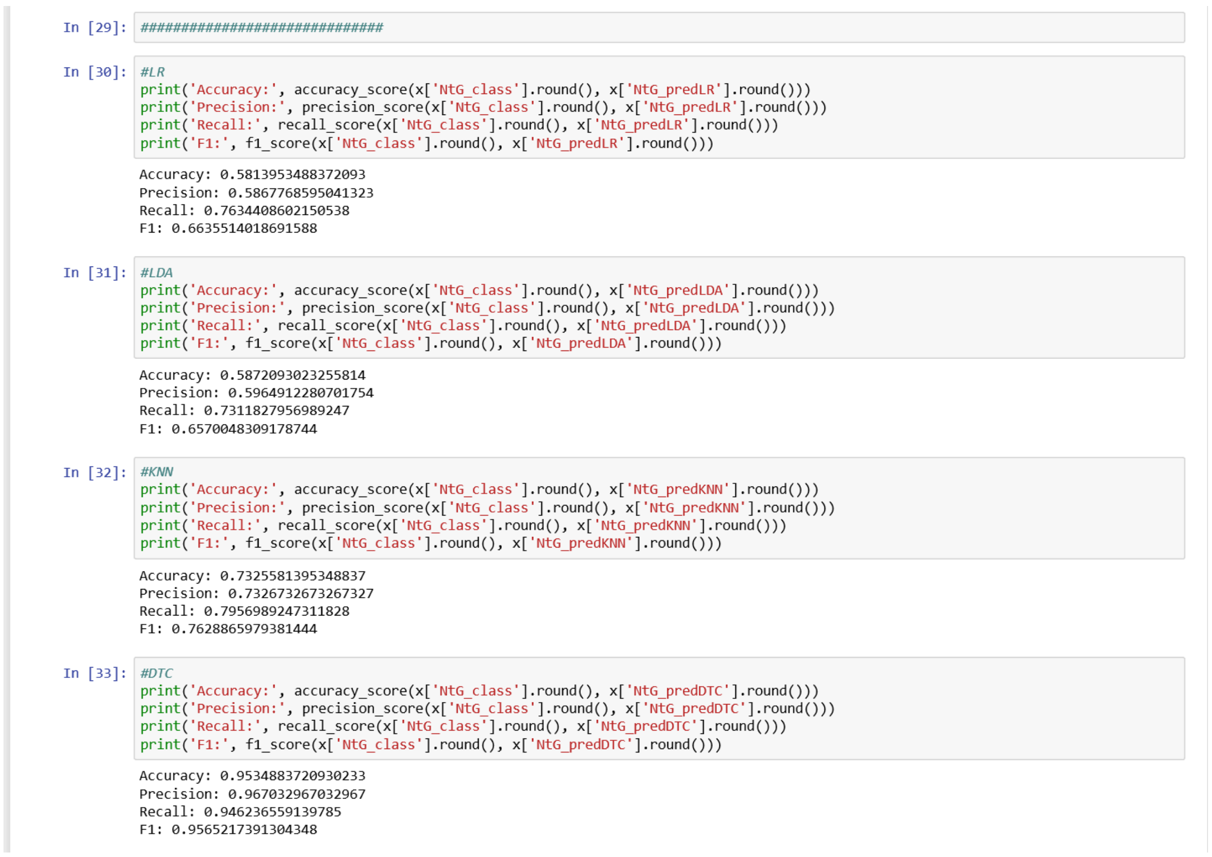
Out[27]:

	Well	Easting	Northing	Top_Balder_TWT	BCU_TWT	RMS_BCU_near	RMS_BCU_far	NtG	Gradient	AVO_sum	AVO_diff	AVO_prod	NtG_c
0	211/24-4	434113.24	6783495.48	1660.124815	2344.146120	2.957468	2.434968	0.305144	-0.522500	2.434968	3.479968	-1.545277	
1	33/12-1	437055.69	6786258.21	1636.961950	2283.454430	1.212534	1.418373	0.987179	0.205839	1.418373	1.006695	0.249587	
2	33/12-B-1 A	438313.74	6786227.39	1652.419343	2361.018068	2.214515	1.867557	0.127886	-0.346958	1.867557	2.561473	-0.768344	
3	33/12-B-1 B	438193.83	6786921.91	1663.011292	2310.467530	1.684443	1.898300	0.243066	0.213857	1.898300	1.470586	0.360230	
4	33/12-B-1 C	438063.61	6786344.34	1654.795532	2328.535352	1.755582	1.238507	0.166065	-0.517075	1.238507	2.272657	-0.907768	
...	...	...	...	...	...	...	...	...	...	...	...	...	
167	33/9-C-6	442053.25	6798749.69	1676.983885	2403.847107	1.655461	1.522337	0.750849	-0.133124	1.522337	1.788585	-0.220382	
168	33/9-C-7	441698.39	6797738.47	1667.437660	2367.370687	0.229123	0.482087	0.677724	0.252964	0.482087	-0.023841	0.057960	
169	33/9-C-8	442034.68	6796195.14	1671.039310	2344.972898	1.025667	2.223881	0.592816	1.198214	2.223881	-0.172547	1.228969	
170	33/9-C-8 A	440572.73	6796359.68	1656.938193	2329.879883	0.955554	0.928878	0.348550	-0.026676	0.928878	0.982230	-0.025490	
171	33/9-C-8 AT2	440572.83	6796359.85	1656.938193	2329.879883	0.955554	0.928878	0.349802	-0.026676	0.928878	0.982230	-0.025490	

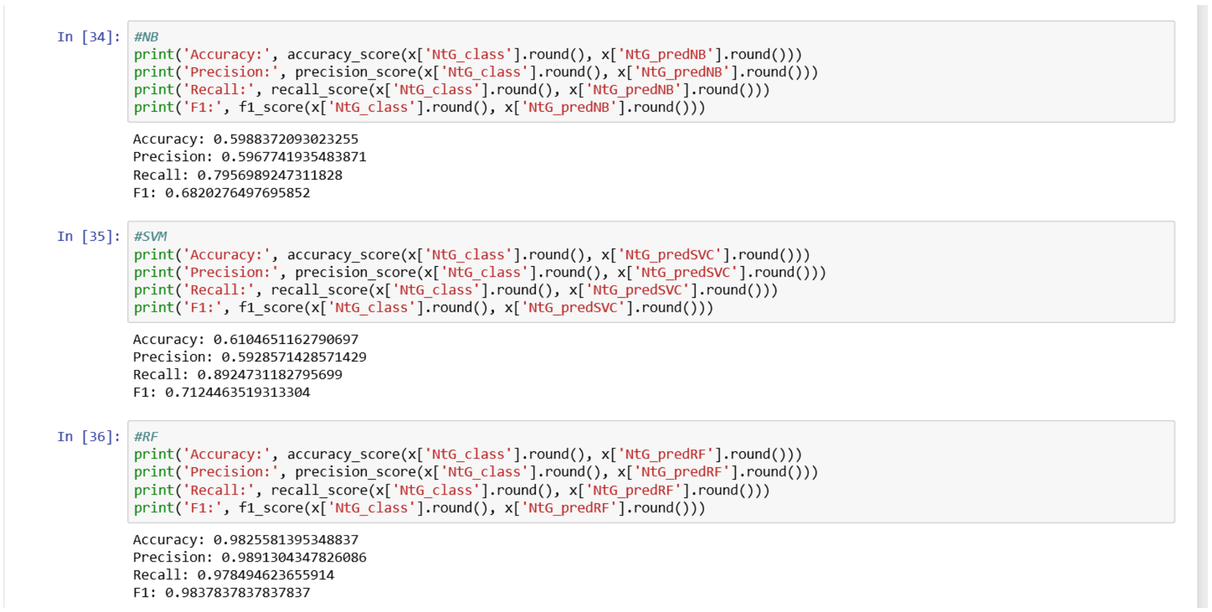
172 rows × 20 columns



```
In [28]: x['NtG_class']

Out[28]: 0      0.0
         1      1.0
         2      0.0
         3      0.0
         4      0.0
               ...
         167    1.0
         168    1.0
         169    1.0
         170    0.0
         171    0.0
         Name: NtG_class, Length: 172, dtype: float64
```



```
In [29]: ##############################

In [30]: #LR
         print('Accuracy:', accuracy_score(x['NtG_class'].round(), x['NtG_predLR'].round()))
         print('Precision:', precision_score(x['NtG_class'].round(), x['NtG_predLR'].round()))
         print('Recall:', recall_score(x['NtG_class'].round(), x['NtG_predLR'].round()))
         print('F1:', f1_score(x['NtG_class'].round(), x['NtG_predLR'].round()))

         Accuracy: 0.5813953488372093
         Precision: 0.5867768595041323
         Recall: 0.7634408602150538
         F1: 0.6635514018691588

In [31]: #LDA
         print('Accuracy:', accuracy_score(x['NtG_class'].round(), x['NtG_predLDA'].round()))
         print('Precision:', precision_score(x['NtG_class'].round(), x['NtG_predLDA'].round()))
         print('Recall:', recall_score(x['NtG_class'].round(), x['NtG_predLDA'].round()))
         print('F1:', f1_score(x['NtG_class'].round(), x['NtG_predLDA'].round()))

         Accuracy: 0.5872093023255814
         Precision: 0.5964912280701754
         Recall: 0.7311827956989247
         F1: 0.6570048309178744

In [32]: #KNN
         print('Accuracy:', accuracy_score(x['NtG_class'].round(), x['NtG_predKNN'].round()))
         print('Precision:', precision_score(x['NtG_class'].round(), x['NtG_predKNN'].round()))
         print('Recall:', recall_score(x['NtG_class'].round(), x['NtG_predKNN'].round()))
         print('F1:', f1_score(x['NtG_class'].round(), x['NtG_predKNN'].round()))

         Accuracy: 0.7325581395348837
         Precision: 0.7326732673267327
         Recall: 0.7956989247311828
         F1: 0.7628865979381444

In [33]: #DTC
         print('Accuracy:', accuracy_score(x['NtG_class'].round(), x['NtG_predDTC'].round()))
         print('Precision:', precision_score(x['NtG_class'].round(), x['NtG_predDTC'].round()))
         print('Recall:', recall_score(x['NtG_class'].round(), x['NtG_predDTC'].round()))
         print('F1:', f1_score(x['NtG_class'].round(), x['NtG_predDTC'].round()))

         Accuracy: 0.9534883720930233
         Precision: 0.967032967032967
         Recall: 0.946236559139785
         F1: 0.9565217391304348
```



```
In [34]: #NB
         print('Accuracy:', accuracy_score(x['NtG_class'].round(), x['NtG_predNB'].round()))
         print('Precision:', precision_score(x['NtG_class'].round(), x['NtG_predNB'].round()))
         print('Recall:', recall_score(x['NtG_class'].round(), x['NtG_predNB'].round()))
         print('F1:', f1_score(x['NtG_class'].round(), x['NtG_predNB'].round()))

         Accuracy: 0.5988372093023255
         Precision: 0.5967741935483871
         Recall: 0.7956989247311828
         F1: 0.6820276497695852

In [35]: #SVM
         print('Accuracy:', accuracy_score(x['NtG_class'].round(), x['NtG_predSVC'].round()))
         print('Precision:', precision_score(x['NtG_class'].round(), x['NtG_predSVC'].round()))
         print('Recall:', recall_score(x['NtG_class'].round(), x['NtG_predSVC'].round()))
         print('F1:', f1_score(x['NtG_class'].round(), x['NtG_predSVC'].round()))

         Accuracy: 0.6104651162790697
         Precision: 0.5928571428571429
         Recall: 0.8924731182795699
         F1: 0.7124463519313304

In [36]: #RF
         print('Accuracy:', accuracy_score(x['NtG_class'].round(), x['NtG_predRF'].round()))
         print('Precision:', precision_score(x['NtG_class'].round(), x['NtG_predRF'].round()))
         print('Recall:', recall_score(x['NtG_class'].round(), x['NtG_predRF'].round()))
         print('F1:', f1_score(x['NtG_class'].round(), x['NtG_predRF'].round()))

         Accuracy: 0.9825581395348837
         Precision: 0.9891304347826086
         Recall: 0.978494623655914
         F1: 0.9837837837837837
```