University of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialization: MSc. Petroleum Engineering	Spring semester, 2020 Open
Author: Mauro Andres Encinas Quisbert	..... Mauro Andres Encinas Quisbert

Programme coordinator:
Øystein Arild

Supervisors:
UiS - Prof. Dan Sui
MHWirth - Ahmad Mirhaj

Title of master's thesis:
DATA DRIVEN ROP MODELING – ANALYSIS AND FEASIBILITY STUDY

Credits: 30

Keywords: Machine Learning, Rate of Penetration, Recurrent Neural Networks, Drilling.	Number of pages: 77 + supplemental material/other: 26 Stavanger, 15th July 2020

Mauro Andres Encinas Quisbert

Data Driven ROP Modeling Analysis and Feasibility Study

Master Thesis Project for the degree of
MSc in Petroleum Engineering

Stavanger, July 2020

University of Stavanger
Faculty of Science and Technology
Department of Energy and Petroleum Engineering

Universitetet i Stavanger

Abstract

Increasing the drilling speed in wells while maintaining the operational safety standards is a challenge that many Petroleum Engineers have undertaken. In recent years, high complexity wells (Horizontal, Extended Reach, HPHT, etc.) have been drilled increasingly, this forced the industry to continue investigating which parameters involved in the Rate Of Penetration (ROP) have the most influence on its behavior. This study integrates different concepts and methodologies from Petroleum Engineering, Drill String Mechanics, Data Analysis, and Machine Learning (ML). It aims to identify the most important parameters involved in ROP, using real well data to evaluate the influence of these parameters in different ML ROP predictive models.

The methodology includes the study of different physics-based ROP models, even though some of them were developed decades ago but remain relevant to this date. Improvement of these models accuracy came with the implementation of new technology and equipment on the drill site, such as Wired Drill Pipe, Precise Sensors, Top Drive Technology, Measurement While Drilling, Logging While Drilling, and many more. Those developments generated large quantities of data that companies used to store and now are proven to be relevant to understand and explain phenoms involved in drilling a well.

During the study, one parameter consistently appeared to be on top of all others Weight on Bit (WOB). All physics-based models projections are based on the accuracy of it, but in high complexity wells as the ones drilled nowadays, it is not easy to estimate, as Surface WOB (SWOB) and Downhole WOB (DWOB) values usually do not match. For this purpose, a complete well database was used to identify and extract relevant parameters and data that could allow this study to be carried. A Python code that predicts the DWOB value from surface measurements using a physics-based model, was successfully implemented.

Once the data was selected and prepared, different machine learning methods were implemented to identify the best ROP predictive model. Among them, we can mention Random Forest Regressor, K-Nearest Neighbors, Artificial Neural Networks, and Long Short Term Memory. When the best model was identified (LSTM), a sensitivity analysis was held using surface and a combination of surface - calculated parameters (DWOB) as input for the model, this was done to verify that machine learning models performance can be raised by improving the quality of input parameters using drilling engineering knowledge, instead of relying only on a data-science approach.

Acknowledgments

First, I would like to thank God, for providing me the strength needed to complete my studies satisfactorily.

To the University of Stavanger, especially my supervisor Prof. Dan Sui for the weekly meetings and continuous guidance. Also, to Andrzej Tunkiel for the productive discussions held during the elaboration of this study.

To Ahmad Mirhaj and everyone at MHWirth, for giving me the opportunity to work in this exciting collaborative study.

Finally, my deepest gratitude to my family and friends for their unconditional support, without you all, this would not have been possible.

Mauro Encinas

List of Abbreviations

ANN	Artificial Neural Network
BHA	Bottom Hole Assembly
CSV	Comma-Separated Value
DWOB	Downhole Weight on Bit
DTQ	Downhole Torque
ECD	Equivalen Circulating Density
HPHT	High Pressure High Temperature
ID	Inside Diameter
IQR	Interquartile Range
KNN	K Nearest Neighbors
LWD	Logging While Drilling
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi Layer Perceptron
MSE	Mechanical Specific Energy
MWD	Measuring While Drilling
NN	Neural Network
R^2	Coefficient of Determination
OD	Outside Diameter
RF	Random Forest
RNN	Recurrent Neural Network
ROP	Rate of Penetration
RPM	Revolutions per minute
SPP	Standpipe Pressure
STQ	Surface Torque
SWOB	Surface Weight on Bit
WOB	Weight on Bit
WDP	Wired Drill Pipe

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background, Motivation and Challenge

During the drilling process of a well several operations are held, to mention a few: Drilling, Tripping In/Out, Well Conditioning, Logging, Coring, Casing/Liner running, and many more. When we speak of Deepwater Offshore projects, the total cost of a well that includes such operations can be in the order of a $ 100 million [6]. Therefore, the optimization of each operation serves to reduce the overall cost of a project.

All operator companies expect to have a good performance during drilling operations, which is measured among other benchmarks by the ROP. Hence, these are some points to consider:

- All companies expect to have high values of ROP while maintaining appropriate operational safety standards.
- ROP is a combination of different parameters (WOB, RPM, Flow rate, etc.).
- It can give us an idea of the type of formation that is being drilled (usually, soft formations - High ROP and abrasive formations - low ROP).
- When it is lower than expected, even though different parameters have been applied, can be a signal for wear on the drilling bit.

- Vibration, poor weight transfer, and poor torque could reduce drilling performance by 40-50% [7]

Determining which parameters are relevant for a good ROP prediction and optimization, is a subject that has been theoretically and experimentally studied before. Even today is a complex issue to address, as different models can take a distinct number of variables for this calculation. One parameter that is consistently regarded as important in all models is the WOB, and since the development of high angle wells, it has been noted that there is, in some cases, a considerable difference between the SWOB and DWOB. Lately, as the interest in data science and machine learning increases, different alternatives for data-driven ROP prediction and optimization have been proposed by researchers [5] to find a suitable solution for this problem.

The motivation of this study is to implement a code to predict the DWOB value from surface measurements and generate a data-driven model for ROP prediction. This model should be easily applied in any type of well and improve drilling operations with low cost for the companies.

1.2 Objectives and Scope

The present study focuses on identifying the most relevant parameters involved in the ROP modeling process, as well as accurately predict its behavior. It, by no means, will develop an industrial solution but more of a concept that could be further investigated and developed by MHWirth, given the short time-frame for the thesis work. Therefore, the study will provide the first steps to further understand this exciting topic with a new approach given by machine learning. In order to accomplish the above stated, the following objectives are proposed:

- Understand all the components involved during the drilling phase of a well.
- Identify key parameters involved in ROP prediction.
- Choose an appropriate data set that contains relevant and consistent information.
- Effectively clean and prepare the data for further analysis.

- Apply a comprehensive and clear method to predict DWOB from surface measurements.
- Implement a machine learning algorithm that accurately predicts the ROP value for the chosen data.

The first objective is of utter relevance, understand how to measure and obtain the parameters involved in all ROP models, once this is accomplished, we can define the models and confidently select key parameters that are relevant. Then, it is possible to search and select an appropriate data set, that in the best scenario, will contain at least the necessary information.

In recent years, the use of Wired Drill Pipe (WDP) has shown that in some cases, especially in high angle wells, a considerable difference exists between the measured SWOB and DWOB. As all ROP models include the WOB parameter, assuming that this value is the real force applied to the bit at the bottom of the well, is important for the study to evaluate the possibility of implementing a code that can achieve a good prediction of the DWOB parameter using only surface measurements. As the last objective highlights, different machine learning techniques will be implemented for ROP modeling, to define the best algorithm based on the available information.

1.3 Methodology

The base of this study is coding and for such purpose Jupyter Notebook [8] will be the application of choice as it is user friendly, handles the selected programming language Python [9] and allows to set an appropriate environment to develop the study. Python [9] has gained a lot of adepts during these last years, as it is easy to learn and apply since previous programming experience is not required. Several packages were used to set the proper programming environment, the list of them is located in the Appendix A.

The most important item to successfully develop the study is to have an appropriate data set, the selected well or wells must contain among other things: sensor measurements of the

selected variables, a sufficient quantity of observations and consistent data along the trajectory of the well, to enable us to run Machine Learning models. Obtaining such data set is not an easy task, this is where data analysis techniques come into play, to fill data gaps, remove noises, and correct faulty measurements. These techniques will be explained in Chapter 3.

To solve one of the most important objectives of this study, a Python [9] code will be implemented to determine DWOB from surface measurements, based on the method proposed by Hareland et al. [10]. Once the well data is cleaned, the hook load measurement will be corrected taking into consideration some effects that will be further discussed. The corrected hook load represents the real weight of the string, based on T&D calculations (Johancsik et al. [11] model) a WOB value will be calculated for the corresponding hook load. To validate the results of such code, Halliburton's WellPlan [12] program will be used. This process will be fully explained in Chapters 4 and 5.

The final part of the study consists of the evaluation of different Machine Learning techniques, to identify the model that provides the best ROP predictions, the evaluated models and the inputs are presented in Chapter 6. The results should demonstrate the ability of the code to successfully recreate the DWOB values, comparing them to the sensor measured ones. The ML model that provides the best results will be evaluated separately, using different sets of data that include the SWOB and the calculated DWOB, the complete information about this process is located in Chapter 7.

Chapter 2

Literature Review

2.1 Drilling Operations (Equipment and Tools)

To properly understand the study, it is necessary to give a brief but clear explanation of the elements involved in drilling a well, but most importantly, how these elements are related to the calculations proposed in this work. For this purpose, this section will be divided into the following sub-sections:

- Drilling Rig Elements - Surface.
- Downhole Elements.

This section will not explain the drilling process, extensive literature can be found on this particular subject as the one presented by Mitchell et al. [13], but will focus on understanding how each element (Surface and Downhole) is connected to give us a value that will be used in calculations and also to make sense of the results of this study.

2.1.1 Drilling Rig Elements - Surface

The surface elements involved in drilling operations are, among others:

- Hoisting System.
- Rotating Equipment.

- Sand Control and Mud-Gas Separators.
- Blowout Preventers.
- Circulating System.
- Power System.

From the elements listed above we will focus on the Hoisting, Rotating Equipment, and the Circulating Systems; since the measurement of the parameters involved in them, will provide us numerical values necessary to determine the ROP.

Hoisting System

The Hoisting System supports the weight of the pipe during the different drilling operations, depending on the situation, it is necessary to lower or raise the pipe. The principal elements in this system are:

- Derrick
- Crown Block
- Traveling Block.
- Drilling Line.
- Drawworks.

During the different drilling phase operations, the drill string will be hanging from the traveling block, either by elevators or connected to the Top Drive. The weight of the drill string and the traveling block is transmitted to the derrick by the drilling line through a series of sheaves. To clarify this concept we refer to Figure 2.1, where the block and tackle scheme described by Bourgoyne et al. [1].

In the previously mentioned Figure 2.1, an element that will play an important role in the study is presented; the cell load. The cell's function is to record the weight of the drill string,

the position that is located (deadline anchor) is an industry-standard. As the wells drilled got more complicated and uncertainties about the real weight of the string were faced, sensors were placed closest to the source of the measurement; above the traveling block and below Top Drive's Saver Sub at the top of the drill string as the one present by Wu et al. [14]. A difference in measurements exists depending on the position of the sensors is a topic that will be explained in Chapter 4. The reason for such attempts to get better approximations of the drill string weight is related to the fact that the WOB is determined indirectly from the measured weight of the drill string.

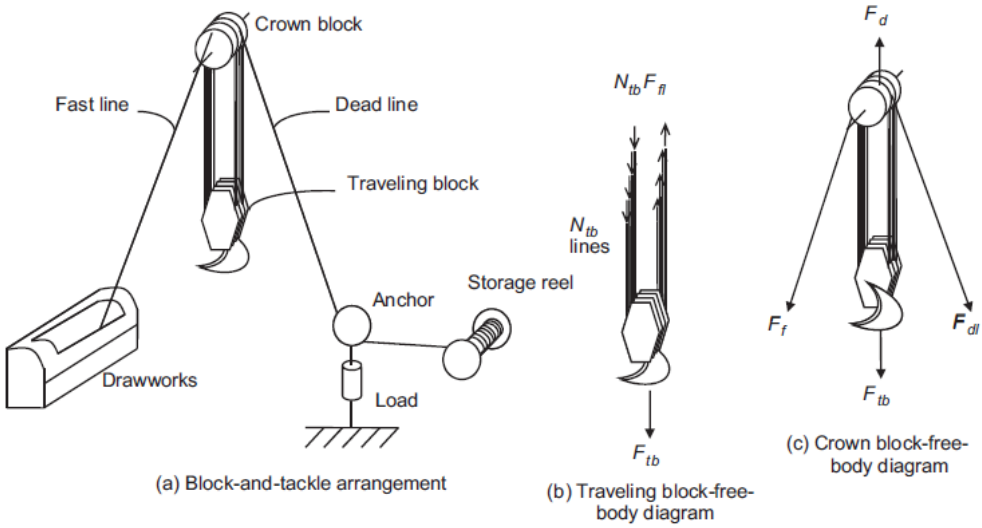

Figure 2.1: Block and Tackle Schematic, taken from [1]

Rotating System

Nowadays, the industry standard for this system in the drilling rigs is the use of Top Drive, which replaced the old Rotary Table technology. Major advantages came from the utilization of this technology, such as fewer connections required (drilling complete 3 joints stands), back-reaming, easier to connect to the drill string and resume circulation.

From the Rotating System, it is possible to obtain two measurements that are used in most of ROP models. The rate of rotation of the string per minute (RPM) and torque, which is defined by Johancsik et al. [11] as *"the moment required to rotate the pipe"*.

Circulating System

The mud circulating system is composed primarily by mud pumps, flow lines, standpipe, mud hoses, drill string, bit nozzles, mud pits, surface mud processing and preparation equipment.

This system provides us two measurements commonly used in ROP models: the flow rate and the standpipe pressure. The first measurement is determined indirectly by the number of strokes of the mud pump, its efficiency and piston diameter, or directly by flowmeters located before the standpipe. The second measurement is the total pressure loss inside surface installations, drill string (Drill Pipe, BHA, Bit), and the well annulus.

2.1.2 Downhole Elements

It is no secret the importance of the development of downhole measurement technology. Since then, it has been a reliable source of information to make decisions on the well site. Due to rate transmission limitations, mud pulse-based MWD systems decelerated the development of downhole drilling real-time applications. This changed after the introduction of Wired Drill Pipe (WDP), a technology that allowed an exponential increase of capability in data transmission rate.

According to Lesso et al. [15] *"WDP allows data to flow at approximately 10,000 times the rate of fast mud telemetry"*, this allowed much-needed improvements in real-time data analysis to be implemented in areas as petrophysical properties, drill string positioning, directional drilling control, and drilling mechanics and dynamics. For this study, we will turn our focus on the drilling mechanics and dynamics area.

Drilling mechanics, in this context, refers primarily to the study of torque & drag, drilling hydraulics and vibrations, and the ability of successfully transmitting downhole information with high transmission rates allowed measurements of DWOB, downhole torque (DTQ) and drill string vibrations measurements to be compared with models developed for such purpose, and evaluate the reasons associated with any possible discrepancy.

Pink et al. [16] developed a fully automated closed-loop drilling system that used WDP technology. The aim during the drilling phase is to increase the ROP as safely as possible, the use of WDP provides the possibility to retrieve information that could help to avoid damage to the drilling tools caused by downhole vibrations, bit whirl, and stick-slip; well control information (Equivalent Circulating Density - ECD) and drilling parameters DWOB and DTQ. As part of the DWOB controller system proposed in this paper, it is stated that there is a difference between the values of SWOB and DWOB and that a specific ROP is dependent on this DWOB, for this purpose once DWOB is measured this information is processed and a SWOB value is suggested to achieve the desired DWOB and therefore the aimed ROP.

2.2 ROP Traditional Models

The present section aims to introduce some of the most widely used ROP traditional models. Some of the parameters mentioned in the previous section and the way they interact in ROP estimation it is still partially unknown and has proven to be a complex problem [13]. Mathematical models have been developed based on industry knowledge and regression techniques. The principal objective of such work is to reduce the overall cost of drilling project by improving the drilling operations.

From this section, we will get an insight into the parameters that most of the ROP models consider relevant for such purpose. First major studies were developed in the 1950s and part of the 1960s, empirical relationships from ROP to WOB and RPM (R-W-N models) were developed [17]. It is important to mention that usually the models are designed to work either for Roller-Cone Bits or Fixed-Cutter Bits. The first model to be analyzed is the Equation 2.1 proposed by Bingham (1964) found in [1].

$$R = K \left(\frac{W}{d_b} \right)^{a_5} * N \tag{2.1}$$

Where:

K = Constant of proportionality.

W = Bit Weight or WOB.

d_b = Bit diameter.

a_5 = Bit weight exponent.

N = Rotary speed or RPM.

In this equation, the results are highly dependent on the value of a_5, but the determination of such exponent is not an easy task, as it requires relatively constant values of N and W for a certain lithology and a formation change could be experienced before the test is completed [1]. This model can work independently of the type of drilling bit chosen.

An ROP model widely accepted was presented by Bourgoyne et al. [18], it was an important step forward from the previously mentioned R-W-N models. It proposed the use of eight functions to model the effect of the most important phenomena during drilling, and it is defined by Equation 2.2 [1].

$$R = (f_1)\,(f_2)\,(f_3)\,...\,(f_n) \tag{2.2}$$

Where:

$$f_1 = e^{2.303 * a_1} = K_s$$

$$f_2 = e^{2.303 * a_2 * (10{,}000 - D)}$$

$$f_3 = e^{2.303 * a_3 * D^{0.69} * (g_p - 9.0)}$$

$$f_4 = e^{2.303 * a_4 * D * (g_p - \rho_c)}$$

$$f_5 = \left[\frac{\left(\frac{W}{d_b}\right) - \left(\frac{W}{d_b}\right)_t}{4 - \left(\frac{W}{d_b}\right)_t} \right]^{a_5}$$

$$f_6 = \left(\frac{N}{60}\right)^{a_6}$$

$$f_7 = e^{-a_7 * h}$$

$$f_8 = \left(\frac{F_j}{1,000}\right)^{a_8}$$

Each one of these functional variables is defined by specific values, such as: D true vertical depth (ft), g_p pore pressure gradient (lbm/gal), ρ_c equivalent circulating density(psi), $\left(\frac{W}{d_b}\right)_t$ threshold bit weight per inch of bit diameter at which the bit begins to drill ($1,000lbf/in$), h fractional tooth dullness, F_j hydraulic impact force beneath the bit (lbf) and eight constants a_1-a_8 that are to be chosen depending on local drilling conditions. Even if this model was designed for Roller-Cone Bits, in recent years it has been applied for wells drilled with PDC bits [19]. As observed this is a complete model, but for most cases, it is difficult to have available all the parameters required as inputs to successfully applying it.

The end of this section will cover a more recent development presented by Motahhari et al. [20], this ROP model is designed for PDC bits. The ROP for a PDC bit in perfect bit cleaning conditions is defined by Equation 2.3:

$$R = W_f \left(\frac{G * RPM_t^{\gamma} * WOB^{\alpha}}{D_b * S} \right) \tag{2.3}$$

Where:

W_f = Wear function.

G = Coefficient defined by the bit geometry, cutter size and design.

RPM_t = Rotary speed or RPM.

WOB = Bit Weight or WOB.

D_b = Bit diameter.

S = Confined rock strength.

α, γ = ROP model exponents.

$$W_f = k_w f \left(\frac{WOB}{N_c}\right)^{\rho} * \frac{1}{S^{\tau} * A_W^{\rho+1}} \tag{2.4}$$

Equation 2.4 presents the relation that estimates the wear function W_f. Where A_W defines PDC cutter characteristics which are a function of wear, it is important to note that wear of bit can only be measured after the arrangement has been pulled out of the whole using IADC (International Association of Drilling Contractors) dull grading, so for most cases, this value needs to be estimated by a constant degradation factor as a function of depth. N_c is the number of cutters on the bit face, $k_w f$ is the wear function constant, ρ and τ are wear function exponents. The application of this model is highly dependent on the value of W_f, which is difficult to implement as it has shown to introduce various fitting parameters [17].

2.3 ROP Data Driven Models

Data-driven modeling successful application in different industries have caused an increasing interest in the subject from the Oil & Gas Industry and is regarded as the future of the segment due to its potential for optimizing drilling operations [21].

Different Machine Learning models can be applied but it is important to understand that there is no universal solution, each well and its data should be analyzed independently. As observed in the previous section, optimizing ROP goes beyond determining the optimum combination of WOB and RPM it includes a complex correlation with a multitude of different attributes [22].

Barbosa et. al. [5] defined that after reviewing 53 papers related to ML for ROP prediction,

five methods are applied consistently. Based on this information, the two most widely used will be explained: Ensemble Methods and Artificial Neural Networks (ANN). Additionally, a branch of neural networks that takes advantage of the sequential essence of the input, called Recurrent Neural Networks (RNN) [4] will be analyzed as well, as per the literature review this kind of models has not been studied broadly by the industry for ROP modeling.

2.3.1 Ensemble Methods

As defined by Géron et al. [3] *"a group of predictors is called a ensemble... and an Ensemble Learning Algorithm is called an Ensemble method"*. The principle behind Ensemble Learning is that the aggregation of different low-performance predictors may end up with a better prediction that a single *"good"* predictor.

The type of base learners used will define the classification of the ensemble model. When all learners in the ensemble belong to the same type, it can be defined to be as homogeneous, otherwise, if there are different learners it is called a heterogeneous ensemble. One type of homogeneous ensemble can be based on Decision Trees (DT) used to get a single result from the different possibilities provided by each tree. This ensemble is popularly known as Random Forest (RF), which is a powerful Machine Learning algorithm, also in this category using the same base of learners are the Gradient Boosting Machines (GBM).

One of the unwritten ML *"best practices"*, is that there is no reason for using complex models when simple models can do the work. As mentioned RF is regarded as one of the most powerful ML algorithms and its implementation is not complicated. In this study, several different ML algorithms were tested and in some scenarios, the best results were obtained by RF and K-Nearest Neighbors (KNN, which falls out of the ensemble category).

Random Forest

As mentioned earlier the aggregation of the results of a group of Decision Trees is called a Random Forest, this method was introduced by Breiman [23]. Therefore, to understand a random

forest is necessary to understand how a DT works. Decision trees use a step-wise process to define to which category something belongs, in DT we start at the top of the tree and descend selecting a path based on characteristics or features of what we are analyzing, the decision is based between two options, as we get to the bottom of the DT we will have an outcome.

As interest in RF increased because they are easy to implement and have good performance, Biau [24] presented a complete mathematical analysis of the RF scheme proposed by Breiman [23], here the structure of such model is analyzed and explained. But, *what does all of this mathematical equations mean?*, as explained by Sagi et al. [25] *"randomness is injected into the decision tree inducers using two randomization processes... training each tree on a different sample of instances and... the inducer randomly samples a subset of the attributes and chooses the best split among them."*.

Han et al. [26] evaluated the performance of this architecture compared to the other that have been increasingly used such as ANNs and SVMs, he found that RF demonstrated excellent results for classification accuracy, stability, and robustness to features when there is not enough data available. To get the best results of this and all ML algorithms it is important to know which hyperparameters contain such a model, more on how the RF regressor model was implemented will be explained in Chapter 6 including hyperparameter tuning process.

2.3.2 Artificial Neural Networks (ANN)

As defined by Moolayil [2] *"a neural network is a hierarchical organization of neurons (similar to the neurons in the brain) with connections to other neurons"*, ANNs are the principal element used in Deep Learning, mainly because they are powerful, and ductile which makes them ideal to solve highly complex ML problems.

Contrary to what many believe, this is not a new concept, it was originated in the early 1940s and was proposed by Warren McCulloch [27]. This idea was explored until the 1960s, but due to lack of expected progress, it was abandoned. In the 1980s this concept was retaken, but due to

slow improvements and the appearance of Support Vector Machines (SVM) this technique was left behind. Even though it was abandoned before, nowadays, with large quantities of data and improved computational power, it has proven to be a reliable tool for solving different problems [3]. In Figure 2.2 we can observe a representation of a basic Neural Network proposed by Moolayil [2]:

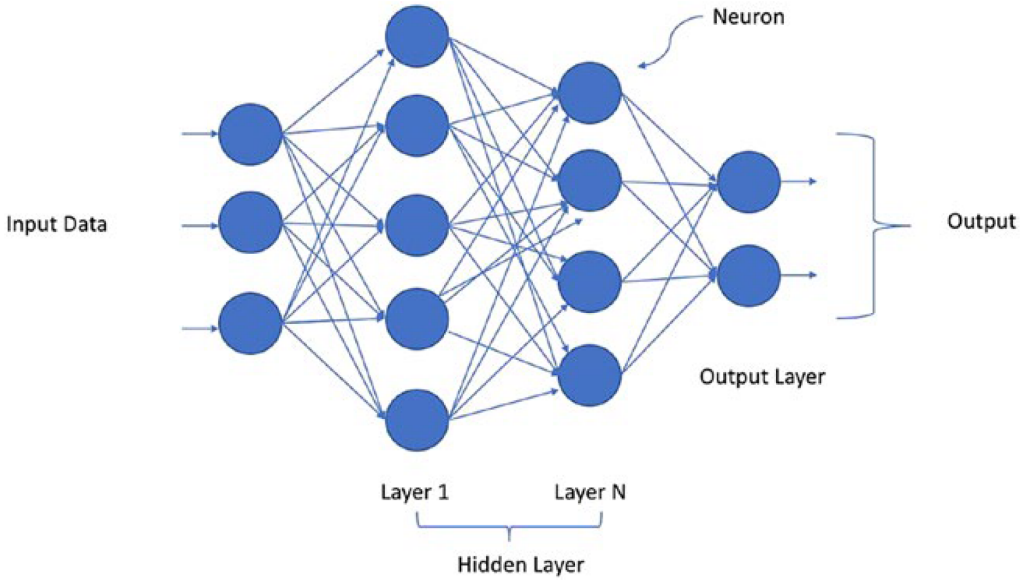

Figure 2.2: Basic Neural Network, taken from [2]

As observed in the previous image: the input data is fed to each neuron in the first hidden layer, the output of these neurons generates input for the neurons in the second hidden layer, and the result of each neuron goes to the output layer. When neurons in a layer have this type of behavior is called a *"fully connected"* or *"dense layer"* [3]. To further understand this subject Géron et al. [3] presents Equation 2.5 that calculates the output for an ANN layer:

$$h_{W,b}(X) = \phi(WX + b) \tag{2.5}$$

Where:

X = Represent the matrix of input features.

W = The weight matrix, contains the connections weights.

b = The bias vector, contains connection weights between the bias and the artificial neurons.

ϕ = Activation function.

Referring to the perceptrons training process; each one of them receives data for each data point at a given time, once they make a prediction, assuming that this one is wrong, will cause that the connection between this neuron and a previous one that would have had a better prediction in terms of accuracy strengthens. Equation 2.6 [3] present the mathematical expression behind this behavior.

$$w_{i,j}^{(nextstep)} = w_{i,j} + \eta \left(y_j - \hat{y}_j\right) x_i \tag{2.6}$$

Where:

$w_{i,j}$ = Connection between the input and output neurons.

x_i = Input value of current training instance.

$\hat{y}_j$ = Output of the output neuron for the current training instance.

y_j = Target output of the output neuron for the current training instance.

η = Learning rate.

Nevertheless, with this architecture, perceptrons do not behave in a way that could allow us to solve the complex problems that we need. It was observed that stacking up perceptrons delivered better results and it is known as a Multi-Layer Perceptron (MLP). In this type of architecture, all the layers (except the output) include a bias neuron (Figure 2.3).

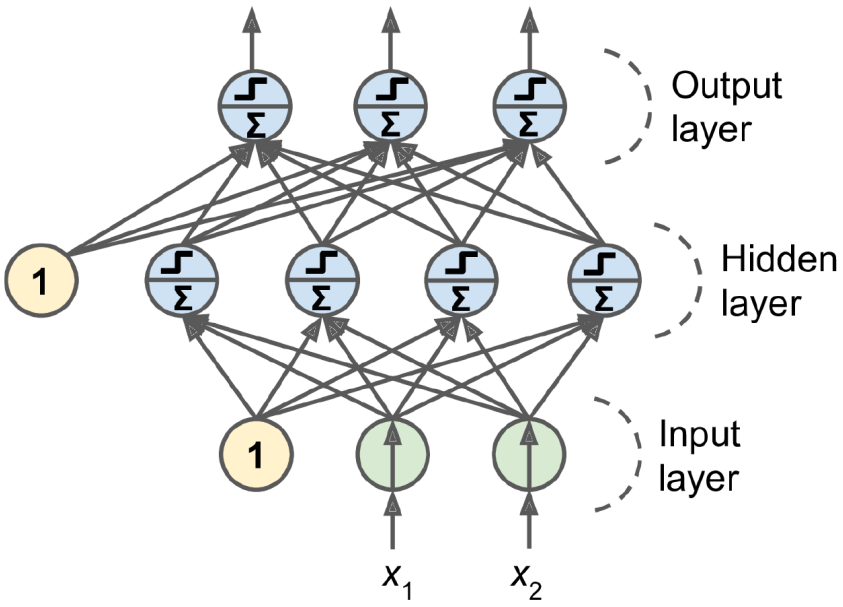

Figure 2.3: Multi-Layer Perceptron, taken from [3]

Such architecture is widely used for ROP prediction, Amer et al. [28] proposed the utilization of an MLP architecture in combination with a back-propagation (BP) training algorithm, this algorithm can identify how to change the weights and bias in each connection to reduce the error. Another example of the application of this architecture was presented by Moran et al. [29], in this case, the model was implemented to predict both ROP and Bit Wear, with good results.

In both cases the architecture consisted of only one layer but implemented with a different number of neurons (this depends on the number of inputs), meaning that we cannot say these models fall into the category of *"Deep Learning"*(DL), an example of this kind of DL model with BP applied for ROP prediction, was presented by Han et al. [30], in this case, the model consisted of three hidden layers with 56, 47 and 32 neurons, respectively.

The natural question after viewing these examples is, *Do more layers or neurons are translated to better results?*, the short answer is, not necessarily. Each well data is different and the implementation method depends highly on the person developing the model. For example, Amer et al. [28] reported results for correlation coefficient (R^2) between 0.78 and 0.99 for different wells, Moran et al.[29] reported R^2 value of 0.9872, and Han et al. [30] reported a percentage error of 14%. In all cases, the results could be considered acceptable, but as mentioned the architecture presented by the latter is considered to be more complex to design and apply. This will be further explored in Chapter 6.

2.3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN), are also known as "sequence models" because they use sequential information, the deep neural network previously analyzed assumes that there is no relationship between two training samples [2]. Another advantage of RNNs is that they can work on sequences of arbitrary lengths [3], while ANNs works with predetermined input size. Time series data (or for some drilling application, depth series data), have a dependence on past data.

As described by Gulli et al. [4] *"RNN cells incorporate this dependence by having a hidden state or memory, that holds the essence of what has been seen so far"*. The value of the hidden state (Equation 2.7) presented by Gulli et al. [4], defines the value of it at any point and shows that have a dependency on the value of the hidden state at a previous time, and also the input at the current time.

$$h_t = \phi\left(h_{t-1}, x_t\right) \tag{2.7}$$

Where, ϕ is the activation function of the cell, h_t and h_{t-1} are the values of the hidden states at present and previous time, x_t is the value of the input at present time. This equation can be modified depending on the size of the time-step desired for the model and is another advantage of RNNs since this means that this model can handle long sequences.

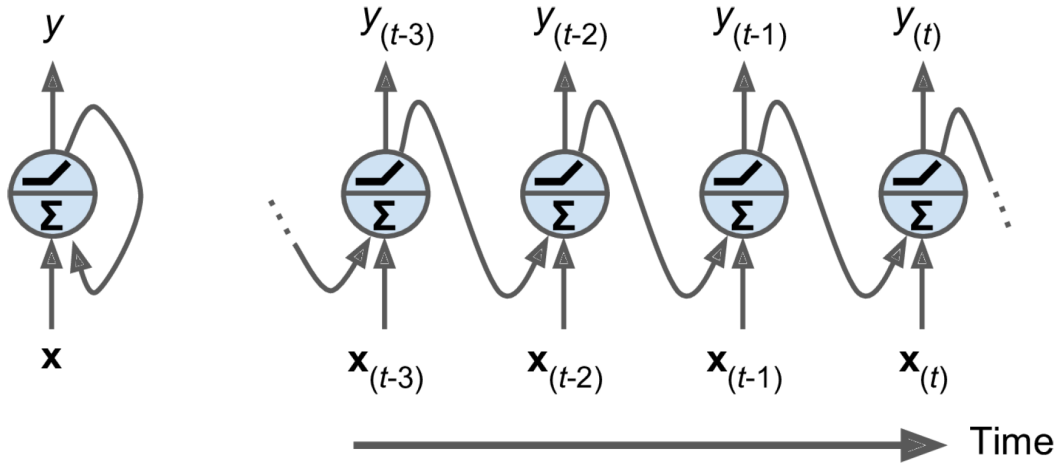

Figure 2.4: Recurrent neural neuron unrolled through time, taken from [3].

An RNN can be defined as a neural network with memory [2], to further explain this concept, if we look at the simplest RNN presented in Figure 2.4 which is a self-feeding neuron. This Figure is the graphical representation of what was presented in Equation 2.7, at each time step, the RNN receive an input for this specific time as well as the output of the previous time step.

The output of an RNN at a time (t) depends on the inputs for a time (t-1) up to the time (t-n), the part of a neural network that preserves the information from previous steps is called

a memory cell [3]. Based on this, a simple recurrent neuron is capable of learning only short patterns, there are more complex structures that will be analyzed further in this section.

Conventional RNNs have a problem of vanishing and exploding gradients, meaning that, when working with a deep RNN network, a Gradient Descent weight update may generate an increase in the outputs each time and therefore after some steps, it will increase considerably, and possibly explode. Another issue for regular RNNs is that after data goes through them, some information is lost and after some time the model may lose the sense of the original inputs [3]. To solve this problem long-term memory models have been developed, the most popular model is the Long Short Term Memory (LSTM), which is the one used in this study.

Long Short Term Memory Cells

LSTMs proposed by Hochreiter et al. [31] offer a solution for the previously discussed issues of regular RNN, this variant of RNN is capable of learning long term dependencies in the data [4]. Figure 2.5 presents the architecture of a LSTM cell. Regardless of looking like a regular cell, two vectors at the output represent the essence of this model, $c_{(t)}$ (long-term state) and $h_{(t)}$ (short-term state).

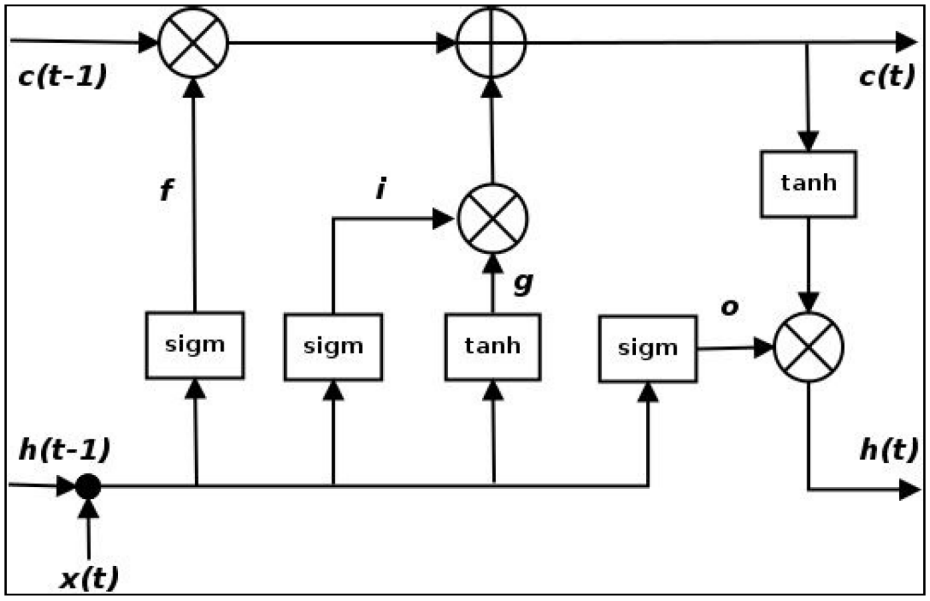

Figure 2.5: Long Short Term Memory cell, taken from [4].

The lines crossing $c_{(t)}$ and $h_{(t)}$, represent the cell and hidden state respectively, the i, f, o and g gates work with the mechanism that allows LSTM to solve the vanishing gradient problem. The four connected layers use $h_{(t-1)}$ and $x_{(t)}$ as inputs, which are the short term state at a previous time and input vector, respectively. The most important layer is the one that outputs $g_{(t)}$, as its processes both $x_{(t)}$ and $h_{(t-1)}$, the output goes straight to the results of this cell. The other three layers are gate controllers where; f is the forget gate, i the input gate and o the output gate.

As defined by Géron et al. [3] *"an LSTM cell can learn to recognize an important input... store it in the long-term state, preserve it for as long as it is needed (that's the role of the forget gate), and extract it whenever it is needed"*. Hence, the good performance of LSTM when working with long-term patterns.

In the O&G industry, more specifically in drilling, this kind of approach has not been studied in-depth, by the date of this publication only a handful of papers evaluating the use of this model for ROP prediction have been written. For example, Han et al. [30] used the LSTM model to increase the accuracy of an ROP predictive model that initially was developed using ANNs, the error of the model was reduced from 14% to 7% using the LSTM model.

2.3.4 Regression Metrics

Up until now, different ML methods have been discussed. But, *how to evaluate the performance of the algorithms?*, the regression models that we have mentioned learn to predict numeric values. In our context, they will predict the ROP value for a given combination of parameters based on previously learned information.

There are several *"metrics"* available to measure regression performance, in this study two metrics were used to evaluate the results: Coefficient of Determination (R^2) and Mean Absolute Error (MAE). Now we will provide a brief introduction about how these methods evaluate the performance of our regressors.

Coefficient of Determination (R^2)

As described by Smith [32], (R^2) Equation 2.8, compares the sum of squared prediction error to the sum of squared deviations of Y about its mean:

$$R^2 = \frac{\sum_{i=1}^{n_samples} (y_i - \hat{y})^2}{\sum_{i=1}^{n_samples} (y_i - \bar{y})^2} \tag{2.8}$$

Mean Absolute Error

The MAE is, as defined by Bonnin [33] *"a risk metric corresponding to the expected value of the absolute error loss, or l1-norm loss. If $\hat{y}_i$ is the predicted value of the ith sample, and y_i is the corresponding true value"*. The MAE Equation 2.9, estimated over a n number of samples is defined by:

$$MAE\,(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \mid y_i - \hat{y}_i \mid \tag{2.9}$$

2.4 The Importance of Selecting the Correct Inputs

Besides the Bingham ROP model (Equation 2.1) which requires only a few numbers of inputs, other ROP physics-based models like Bourgoyne and Young (Equation 2.2) and Motahhari et. al. (Equation 2.3) are dependent of a high number of inputs, much of them are hard to obtain and need to be estimated; this causes in some cases problems to accurately predict ROP. As the aim of this study is to successfully implement and ROP prediction ML algorithm, is important to ask, *how many inputs are necessary for an ML algorithm to work properly?*. This section will address this subject to get a good start-base for the selection of features for our ML algorithms.

Depending on the availability of sensors, the number of measurements, and the configuration of the surface/downhole equipment, a large number of measurements can be stored. This study will evaluate different ML algorithms and test their efficiency. To start this analysis, in the study presented by Barbosa et al. [5] (Figure 2.6) shows that from 53 different works analyzed, 10 reported the use of three or four inputs, 12 works used five or six inputs, also 12 works used

seven or eight inputs. Considering that 5 works did not report the number of input data for their models, we get that almost 70% of the studies worked with less than nine inputs for their respective models.

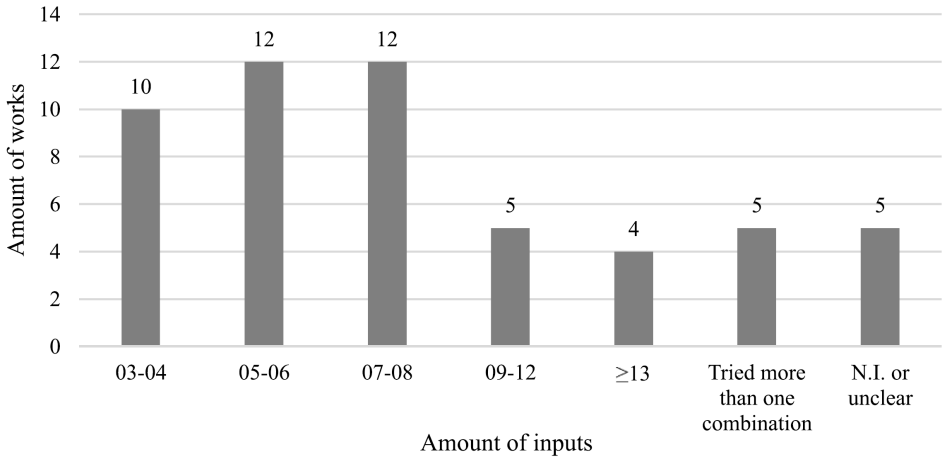

Figure 2.6: Amount of inputs employed to feed ROP data-driven models. Reprinted from Journal of Petroleum Science and Engineering, Vol 183, [5], Page 9, Copyright (2019), with permission from Elsevier.

This provides an interesting statistic to select the number of inputs, and to understand why even though a large number of possible inputs are available most researchers prefer to select just a number of them. For example, in the case of ANN's as described by Amer et al. [28] *"the performance of the neural network is negatively affected by increasing the number of input parameters"*.

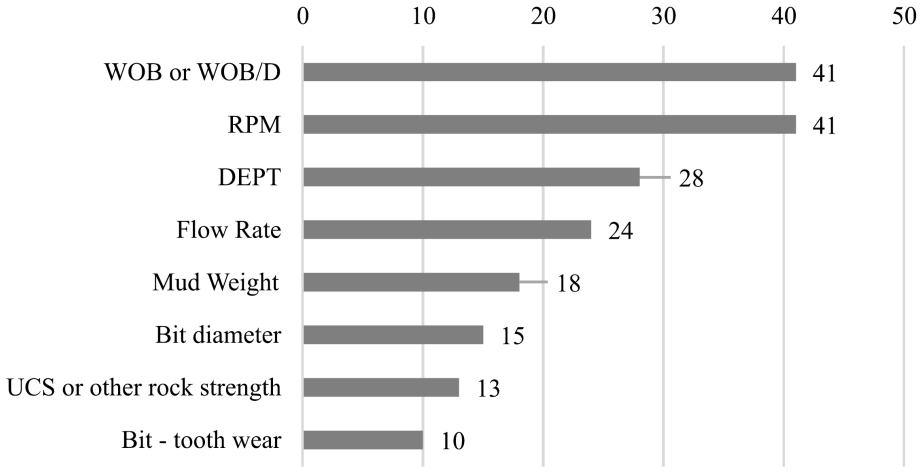

Figure 2.7: Frequency of inputs employed to feed ROP data-driven models, considering 43 from all 53 reviewed works. Reprinted from Journal of Petroleum Science and Engineering, Vol 183, [5], Page 10, Copyright (2019), with permission from Elsevier.

Now that an approximate number of the optimum inputs for an ML algorithm has been defined, it is necessary to identify which parameters are to be considered. For this, the study presented by Barbosa et al. [5], shows that, for the same 53 works analyzed, 43 were taken into account for the results presented in Figure 2.7.

The results show, that like most of the physics-based models WOB or DWOB and RPM are the most used parameters as they were inputs in 41 of the 43 reviewed works, we will further investigate if the use of DWOB represent an improvement in the results since this is the real value of the force applied by the bit at the bottom of the hole, or if that ML algorithms can achieve a good prediction of ROP using SWOB. For this study, we had available most of the information presented in Figure 2.7, except the UCS and bit tooth wear. Therefore, this available information will be used as inputs for our different ML models.

Chapter 3

Database Analysis

3.1 Volve Data set

In 2018 Equinor, together with the partners in the Volve field decided to make public all the downhole and production data from the field [34]. The information can be located at the Equinor's web page and is available for access for research purposes, we will briefly describe the Volve field and the work at the University of Stavanger (UiS) that pre-processed the information for the student's research works.

3.1.1 Volve Field

The Volve field is located in the central part of the North Sea, with water depths of around 80 meters. The field was discovered in 1993, but the development was approved in 2005. The field started production in 2008 [35] from the Maersk Inspirer jack-up rig. The oil produced from the field was stored and shipped to export from the Navion Saga FSO, the gas was piped to the Sleipner A platform. According to Equinor [36] *"Volve reached a recovery rate of 54% and in March 2016 the license decided to shut down its production permanently"*. The plateau production of Volve was around 56,000 barrels per day and the total recovered oil production was 63 million barrels.

3.1.2 UiS Work with Volve Data Set

The information used in this study is part of Equinors' Volve data set, but it was pre-processed by Andrzej Tunkiel from the University of Stavanger and can be located at Volve Field Data Link this page contains real-time drilling data, either in a set of time-based or depth-based data. The information was downloaded in a comma-separated value (csv) format archive, so the handling and selection of the most appropriate well to execute this study was easier.

Complementary information about the well selected (15/9-F-5) like bottom hole assemblies, fluid properties, well trajectory, and well geometry was also provided by the University and is available in Volve Wells App Link this web page was developed by Nagy [37]. Besides the possibility of extracting the information from the wells, there are interactive tools such as depth-days curves, well trajectory, geothermal gradient, and many more.

3.2 Data Analysis

As mentioned the data used in this study was pre-processed, but only to make it organized and accessible. The selected well contained raw data and different issues that will be assessed and explained in this Chapter, for this purpose we will first run an exploratory analysis to evaluate the quality and the quantity of the data, coding work was done using Python [9] and the Pandas library [38]. Then, it is necessary to select an appropriate section of the information and apply different data cleaning techniques to prepare it for normalization. The normalized data will be fed to different ML algorithms, therefore it needs to be clean and consistent.

3.2.1 Importing and Visualizing the Data

The first step into the exploratory analysis of data is importing and visualizing the data. In the data set for well 15/9-F-5, 201 different features were available, it is important to select the most relevant features. Based on the principles shown in the previous chapter, twelve features were selected, the next step is to verify if the selected parameters are stored in the correct data type (integer, float, Python object, etc.). Table 3.1 presents the features selected to work on this

study, the features have the appropriate units *"float64"* for the measurements and *"object"* for the time. Only two downhole parameters are selected for its relevance in this work, Downhole Weight on Bit (DWOB) and Downhole Torque (DTQ).

Number	Feature	Units	Data Type
1	Time	seconds	object
2	Bit Depth (MD)	meters	float64
3	On Bottom Status	dimension-less	float64
4	Weight on Bit	tonnes	float64
5	Average Hookload	tonnes	float64
6	Average Surface Torque	kN-m	float64
7	MWD Downhole WOB	tonnes	float64
8	MWD Downhole Torque	kN-m	float64
9	Average Rotary Speed	rpm	float64
10	Mud Flow In	L/min	float64
11	Average Standpipe Pressure	kPa	float64
12	Rate of Penetration	m/h	float64

Table 3.1: Features selected from well 15/9-F-5

After the selection of the features an exploratory analysis of the information is needed, there are various techniques for this purpose; the first that we will analyze is the histogram plot presented in Figure 3.1, in the x axis is located the range of value of each feature and in the y axis the number of data points that have such value. From evaluating the plots presented in the raw data, the presence of outliers is evident and they need to be removed. For example, some negative value measurements in the Average Hookload, DWOB, DTQ, Standpipe Pressure (SPP), and Surface Torque.

Another widely used tool for data visualization is the heat map, which shows the correlation between variables. This is particularly useful in this case since it is necessary to analyze many variables and determine how they relate to the ROP. Figure 3.2 presents the heat map for the selected data, in this plot, the diagonal set of squares going from top left to bottom right are in the darkest color, this represents the intersection between same variables. Numerically, this will represent a Pearson correlation coefficient of one, which is the maximum value, the intensity of the color will decay depending on the correlation level between variables.

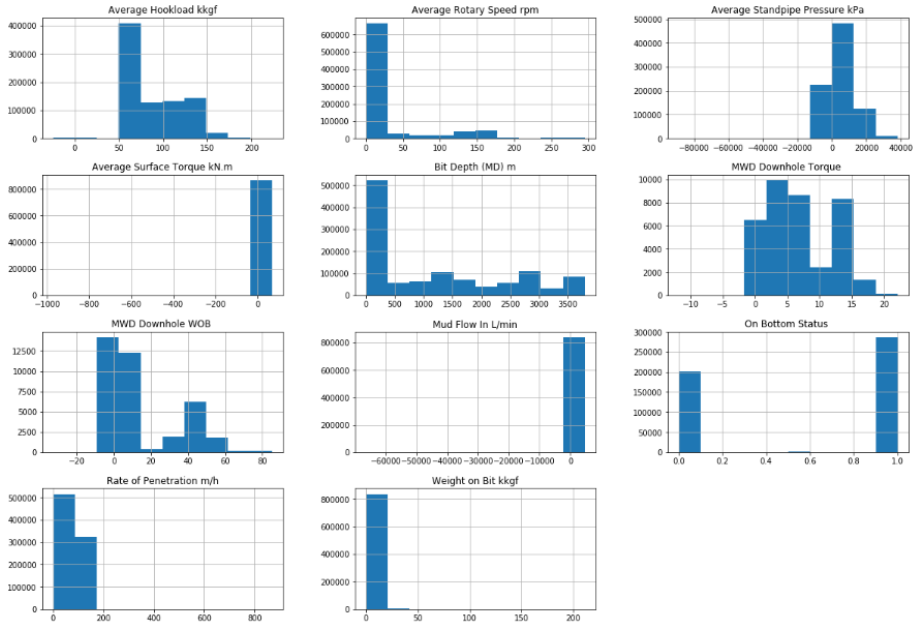

Figure 3.1: Histogram, raw data of features selected.

In this case, the complete data set of the well is analyzed, so it is required to identify appropriate sections of the well to work with. As observed in Figures 3.1 and 3.2, there is information from the parameters that need to be investigated, but it is imperative to ensure that the data is stored in an appropriate sequential manner with no missing values.

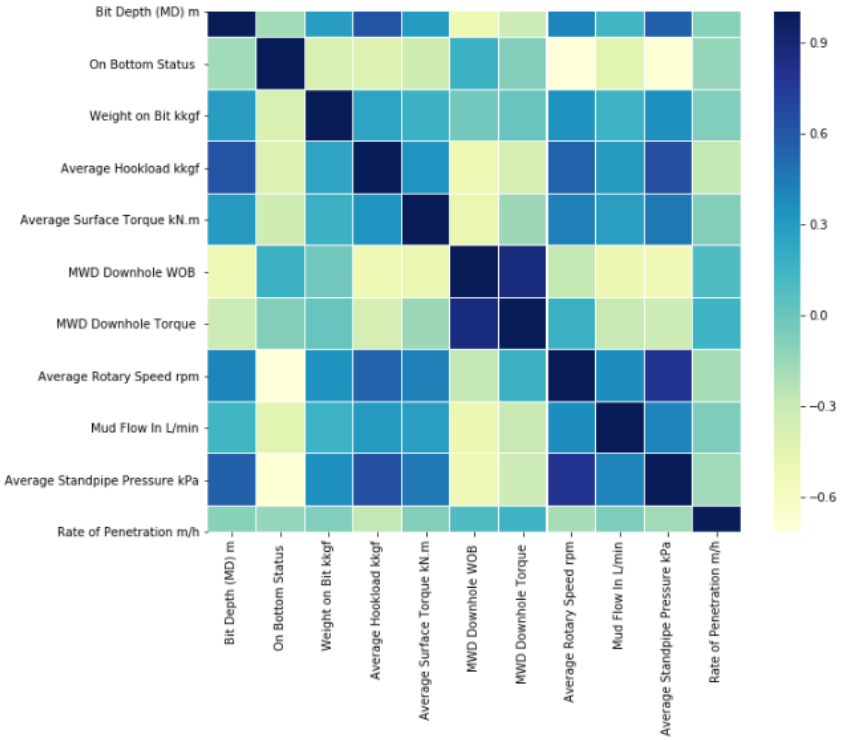

Figure 3.2: Heatmap, raw data of features selected.

3.2.2 Data Selection

It was observed that the data set contained relevant information for the development of this study. Now the job is to select parts from the data that contains (preferably) no missing values. First, by reading the daily drilling reports, it was identified that from the different well sections, the 12 1/4 inch section contained most of the relevant information for this study. After this step, an investigation about the data contained per depth was run and the results are presented in Figure 3.3.

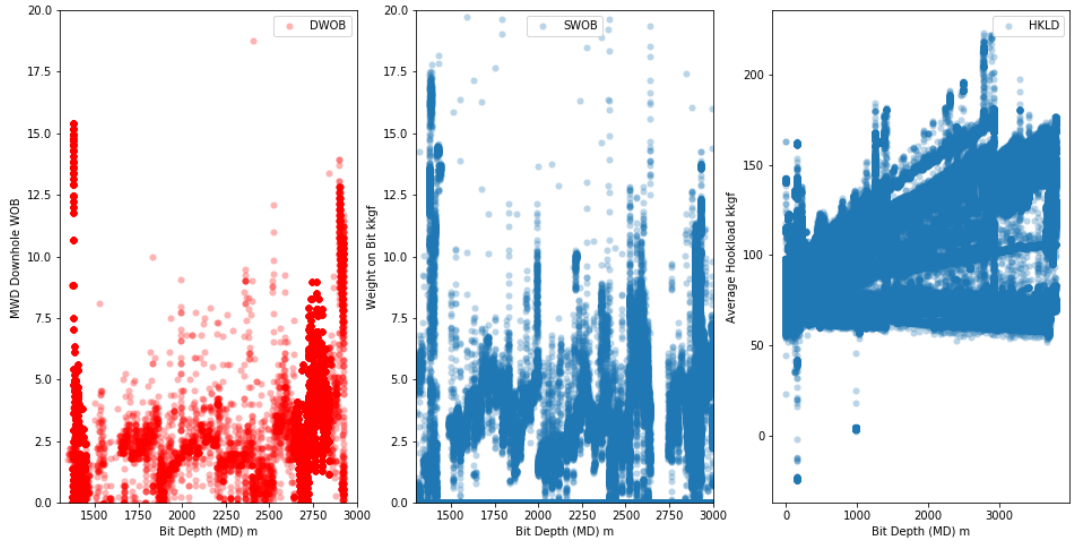

Figure 3.3: Comparative plot of DWOB, SWOB and Hookload data.

It is evident that there is a lot of information contained in this particular well section, the reason behind this is that all different drilling operations (e.g., drilling, tripping, casing running, etc.) are held within the same depths. Therefore, it is necessary to use other filtering techniques (besides depth-based selection) to extract information that could be used in this study.

3.2.3 Time Based Selection

After reviewing the daily drilling reports it was determined that the 12 1/4 inch section was drilled between the 14-07-2008 and 21-07-2018. As showed in Table 3.1 the first feature selected for this work was *"Time"*, this allowed to investigate the data contained in this period. Figure 3.4 shows the distribution of data points for the DWOB measurement stored during the mentioned time, and also the corresponding depth. *Why this is important?*, it is important be-

cause this allowed to see that the selected data that was generated during the drilling operations and removes most of the additional data points that correspond to complementary operations.

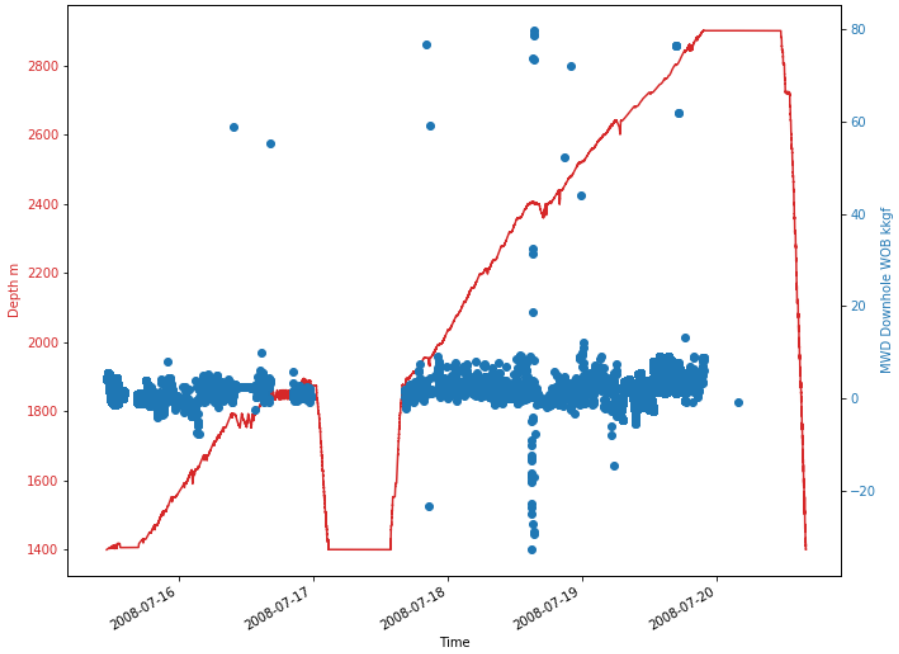

Figure 3.4: Time selected drilling data, Bith Depth and DWOB.

Besides the use of a time-based selection of information, that reduced the number of data points not related to drilling operations, another feature that serves this purpose was selected. The *"On Bottom Status"* feature that has two pre-set dimensionless values, '0' when the bit is on-bottom and '1' when the bit is off-bottom, as the aim of the study is to analyze only drilling-related data, an additional filtering process using this feature was applied. The results of this filtering process can be observed in Figure 3.5.

3.3 Data Cleaning

After the process of *"downsampling"* the information, it is necessary to improve the quality of the data to make it ready for the two analyses to be made, DWOB estimation from surface measurements and ML ROP prediction algorithm. In the following subsections, the three principal problems faced while working with this data set will be presented.

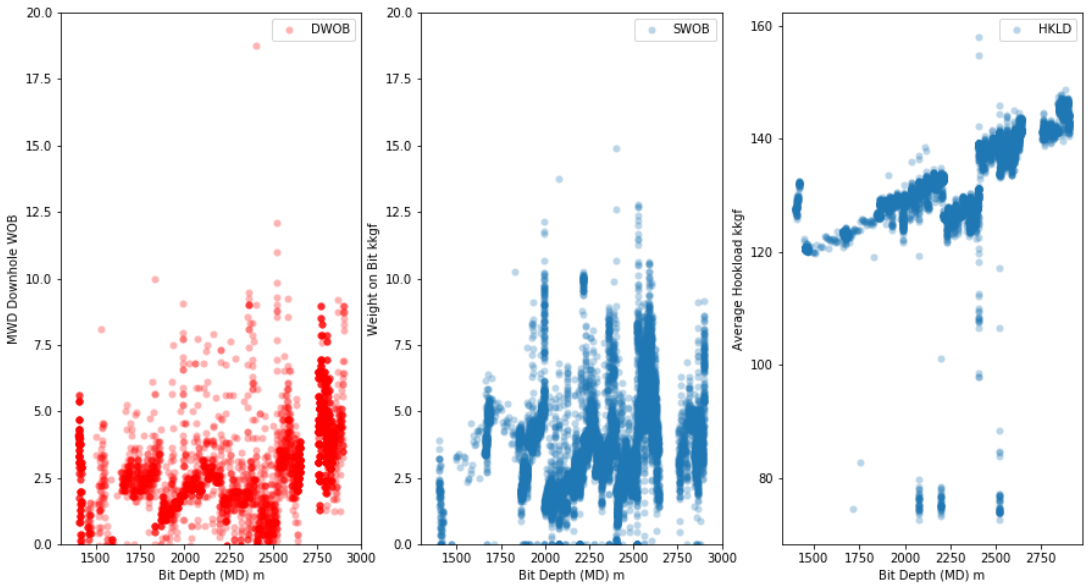

Figure 3.5: Comparative plot of DWOB, SWOB and Hookload data after *"Time"* and *"On Bottom Status"* filtering.

3.3.1 Missing Values Handling

The data did not contain duplicated values, but as a good practice, the eliminate duplicates command was executed before entering the missing value handling. Missing measurements in any cell are presented in the data set as a NaN (Not a Number) value. There are two ways of handling such cases: eliminate the whole row or fill the missing information. Depending on the size of the data set, one could argue whether to follow either option. For the data set used in this study, eliminating the whole row was not an option since it could potentially delete relevant information from other measurements. Therefore, it was necessary to fill the missing data; for this purpose, two common techniques can be used: interpolation or filling the missing value with the last known value. Both techniques were evaluated, and the interpolation method provided the best results due to the ever-changing behavior of the features studied.

Different interpolation techniques can be used for estimating missing values, for example, linear, quadratic, and cubic interpolation [39]. The linear interpolation is the simplest technique since it connects two data points with a straight line, the interpolation function is defined by Equation 3.1. Where b_0 is the intercept and b_1 is the slope of the line.

$$f_1(x) = b_0 + b_1(x - x_0) \tag{3.1}$$

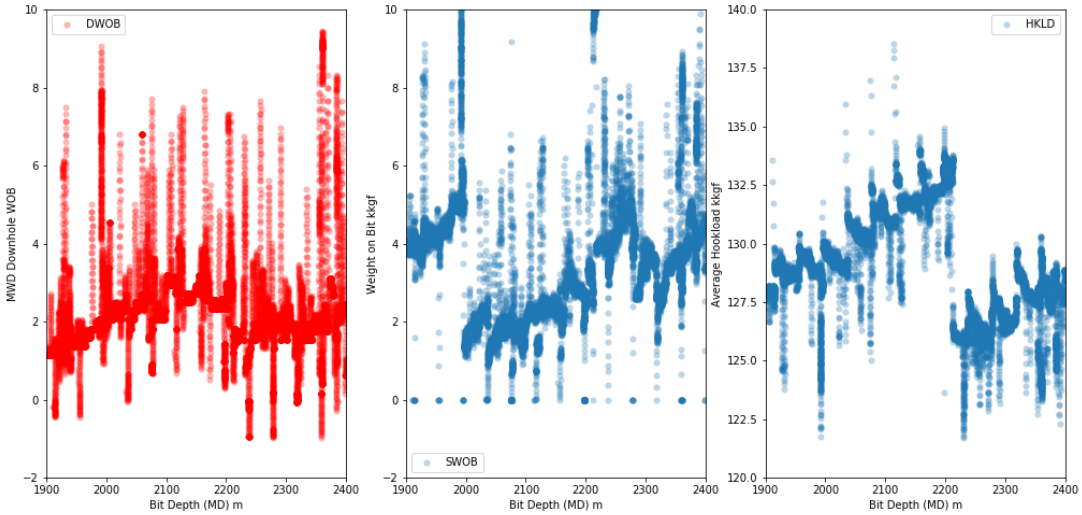

Figure 3.6: Comparative plot of DWOB, SWOB and Hookload data after interpolation.

It is also possible to use a quadratic or cubic interpolation, this depends on the distribution of the data to which it will be applied. In this case, after evaluating the three different possibilities, linear interpolation was selected as the best option, the results of this work are presented in Figure 3.6. It is also worth mentioning that the section of the well to be used for this study was reduced from 1400-2900m to 1900-2400m for various reasons, for example, gaps between data points in the neglected sections of the well were considerable; and as observed in the figure above for the hook load measurement after 2200m, inconsistent data was seen not only in this sensor but also in others less relevant for this work.

3.3.2 Faulty Measurements Handling

To visualize the *"faulty"* behavior of the hook load measurement previously described, Figure 3.7 presents a single plot of this sensor recordings. As mentioned, the hook load measurement after 2200m has an inconsistent behavior. *"Why is inconsistent?"*, because as the well goes deeper more pipe is introduced to it, and the weight of the drill string should increase, but in this particular case, there is a sudden drop in the weight of the string.

There are several possible reasons behind this behavior, e.g., change in density of the mud, change in flow rate, pack-off, sensor failure, etc. These possibilities were investigated, in the case of the mud density and flowrate, as found in the well reports, were constant throughout this

500 meters drilled section. The pack-off scenario was quickly discarded as it was not reported, and also the depth of the well increases as the operation continued seamlessly. This leaves us with the faulty sensor explanation for such behavior.

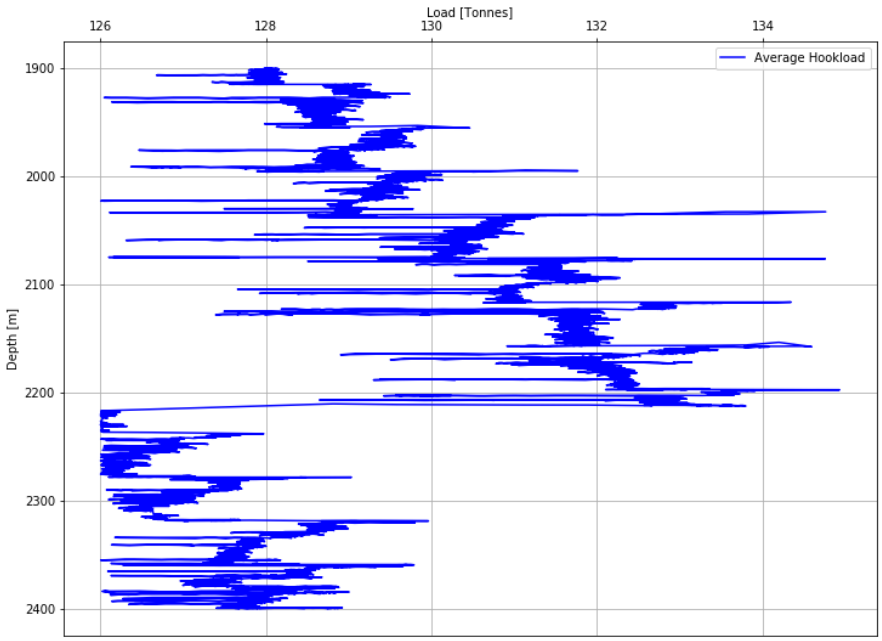

Figure 3.7: Hookload vs Depth plot before correction.

"How can we fix this error in the data?", for this kind of problem and in situations where two or more sensors are measuring one parameter, or whether this parameter can be both measured and calculated, *data assimilation* is a useful tool that could provide us a good estimation of the real value. The least-squares method assumes that if each data set shows the same distribution, it can be defined by its mean and standard deviation σ_1 and σ_2. If there is proven independence between the measurements that we will call x_1 and x_2 and estimator $\hat{x}$ [40], based on these measurements Equation 3.2 can be written as:

$$\hat{x} = a_1 x_1 + a_2 x_2 \tag{3.2}$$

Where,

$$a_1 + a_2 = 1$$

$$a_1 = \frac{1/\sigma_1^2}{1/\sigma_1^2 + 1/\sigma_2^2}$$

$$a_2 = \frac{1/\sigma_2^2}{1/\sigma_1^2 + 1/\sigma_2^2}$$

$$\frac{1}{\sigma_{\hat{x}}^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}$$

In this case, as the precision of the other measurements is added, higher precision than any single measurement is expected. But, *how can we apply this in our context?*, as shown in Figure 3.7 the behavior of the hook load is far from being constant, now we need to understand how this value is decomposed. In drilling operations the hook load value is equal to the weight of the drill string minus the WOB, therefore, to determine the weight of the drill string is necessary to add the value of the hook load plus the WOB, only after this a constant weight increment is visible. In this case, the available value is the WOB, and for calculating the missing hook load values, two independent signals were used. Halliburton's Wellplan [12] and an implemented Torque & Drag code based on Johancsik model [11].

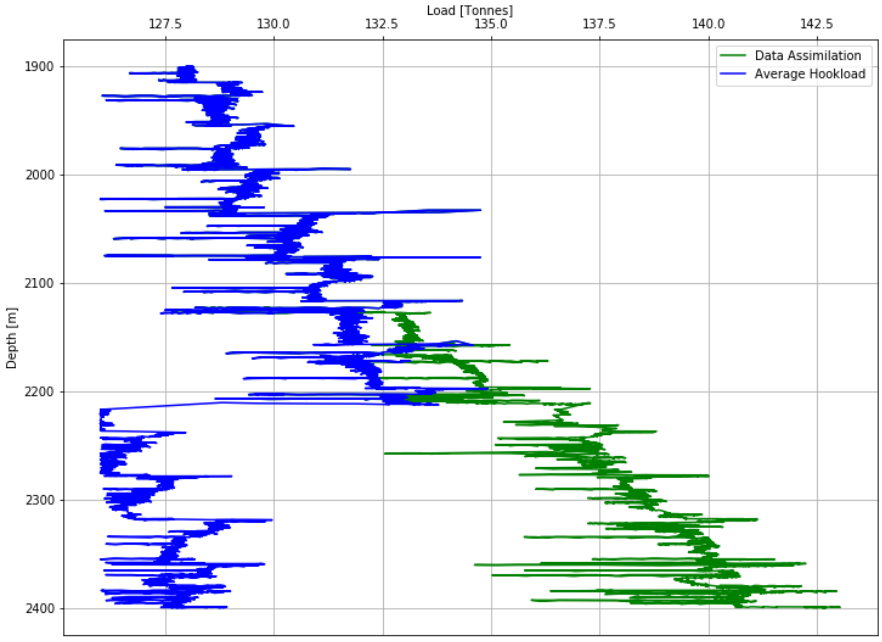

Figure 3.8: Hookload vs Depth comparison plot before-after correction.

The results of this process are presented in Figure 3.8 (green line), is evident that these outputs diverge from the original hook load values (blue line) starting at approximately 2120 meters. The corrected hook load values will replace the original values in the data set.

3.3.3 Outlier Removal

Before feeding any ML model, pre-processing and cleaning of the data is necessary. There is a direct correlation between the results of any model and the quality of the data. The final step for this process is to remove the outliers, initially, when they are easily identifiable it can be done manually after this step is necessary to use more advanced techniques. In this study, two different methods were evaluated: Interquartile Range (IQR) and Moving Average Filter.

Interquartile Range (IQR)

This method is particularly good to remove outliers that are located far away from the observation point, one issue to consider when working with IQR (Equation 3.3) is that is very aggressive when removing outliers, for this cause sometimes relevant information could be eliminated. As described by Deep [41], the way to calculate and therefore remove outliers is given by:

$$IQR = P_{75} - P_{25} \tag{3.3}$$

Where,

$$LowerRange = P_{25} - 1.5 * IQR \tag{3.4}$$

$$UpperRange = P_{75} + 1.5 * IQR \tag{3.5}$$

Any data value that lies outside the range defined by Equations 3.4 and 3.5, which are a function of P_{25} and P_{75} (that represent the 25th and 75th percentile of data points, respectively) is identified as an outlier. The results of IQR outlier removal from the data set are presented in Figure 3.9, where is visible that some data points have been eliminated and sharp "interpolation" lines connected the data. This data still needs further treatment, as it contains visible noise.

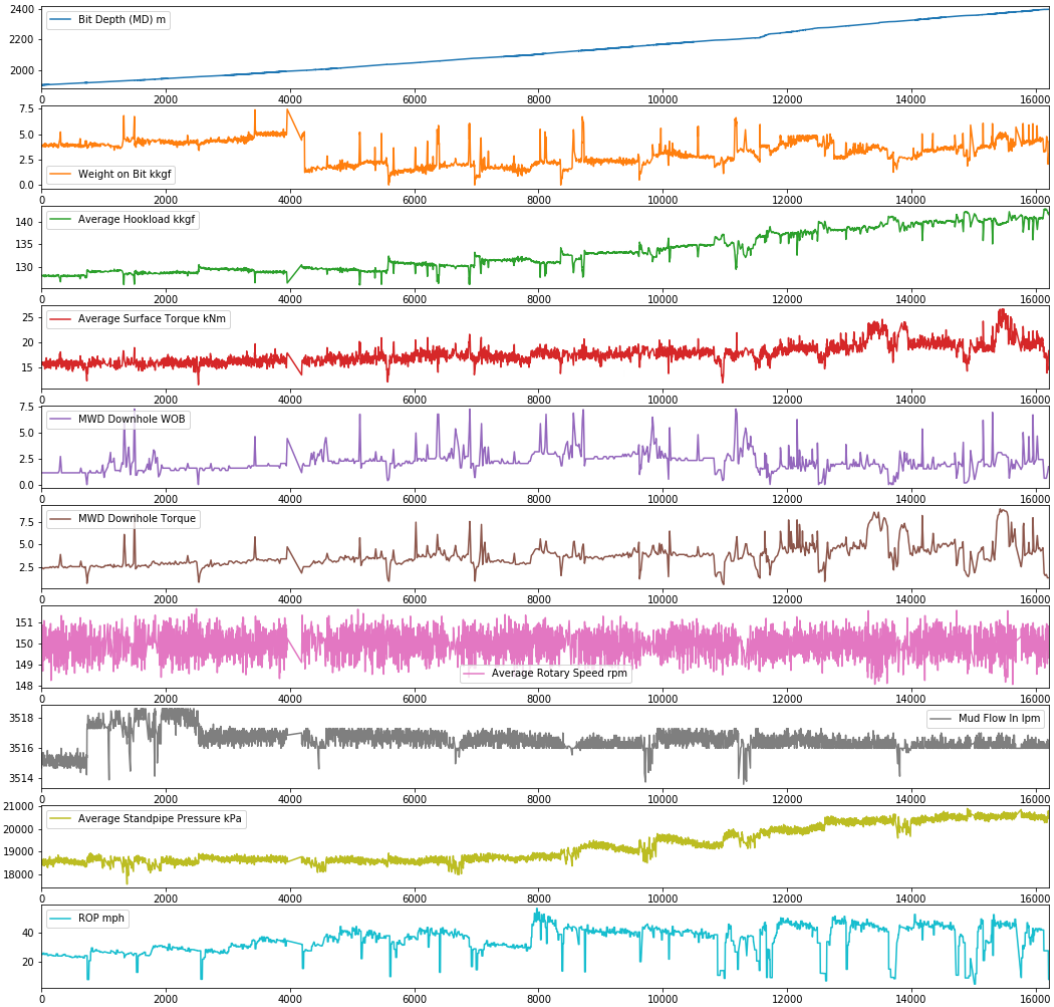

Figure 3.9: Plot of parameters after IQR.

Moving Average Filter

The moving average filter is a low-pass filter that is commonly used for smoothing data signals. It is highly dependent on the number of samples that will be fed at a given time since it takes the average of the number of samples and produces a single output point. As defined by Sui [42], the model is given by

$$y\left(t\right) = \frac{1}{n}\left(x_{(t-n+1)} + x_{(t-n+2)} + ... + x_{(t)}\right) \tag{3.6}$$

Where, $x_{(t)}$ is the sample at the t time and n the number of samples. With the Fourier transform:

$$x\left(t-k\right) \underset{\longleftrightarrow}{F} X\left(j\omega\right) e^{-jk\omega} \tag{3.7}$$

the previous equation is converted to the one in frequency domain as:

$$Y(j\omega) = \frac{1}{n} X(j\omega) \left(e^{-j\omega(n-1)} + ... + e^{-j\omega} + 1\right) \tag{3.8}$$

Finally the transfer function becomes:

$$H(j\omega) = \frac{1 - e^{-j\omega n}}{n(1 - e^{-j\omega})} \tag{3.9}$$

An important concept to understand is that with a higher value of n, more noises with high frequencies will be stopped, but also, the delay effect becomes larger. The data cleaned with the IQR was used as an input for this filter, different values of n were used evaluating the effects of delay and also smoothing the curves, as with the IQR the objective is to eliminate the outliers but at the same time avoid missing relevant data. The results of the use of this filter in the data are presented in Figure 3.10.

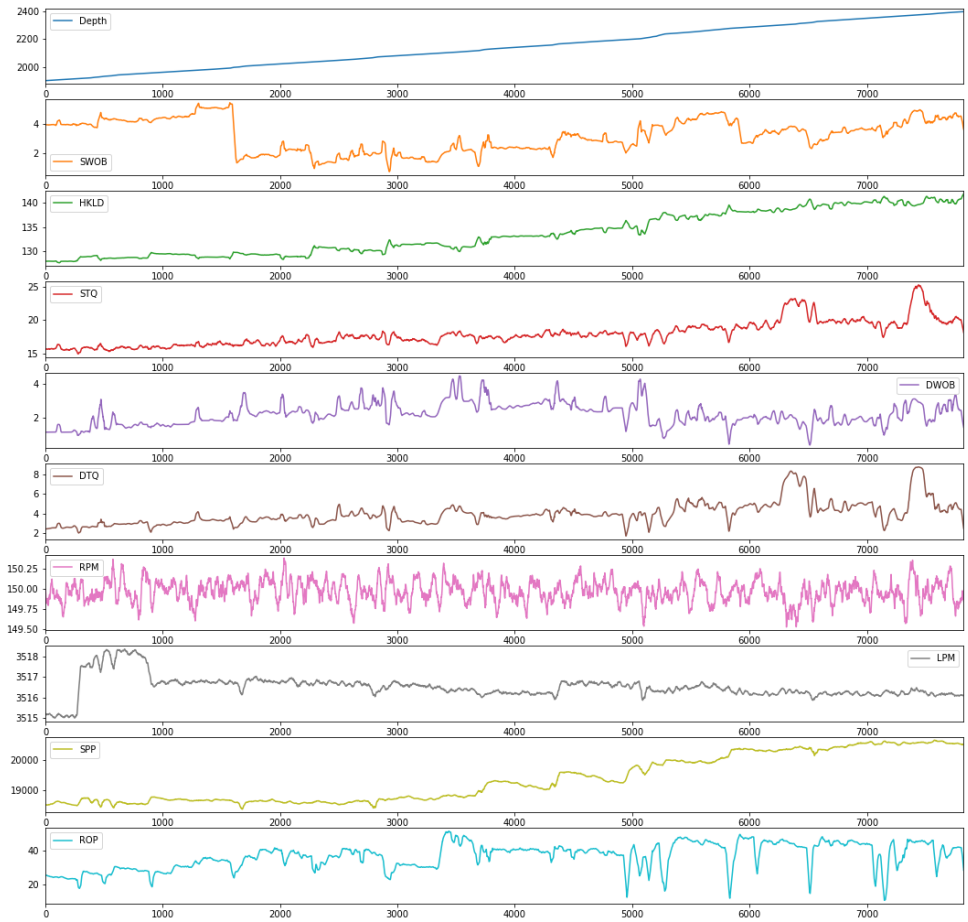

Figure 3.10: Plot of parameters after moving average filter.

3.4 Feature Scaling

Feature scaling is regarded as one of the most important steps, as it is known that some ML algorithms do not perform properly when the inputs have different scales. To better describe this issue with relation to the drilling operations, we work with different scales and units when measuring parameters, e.g., hook load measurements vary from 126 to 137 Tonnes while WOB measurements from 0 to 8 Tonnes, depth is measured in meters with values varying in the analyzed section from 1900 to 2400 and torque measurements recorded in Kn-m with values of 15-25.

Based on the ML algorithms used in this study, the *min-max* scaling (often called normalization) methodology was used. This method scales the values in a range [0,1]. To normalize a data point, first, is necessary to subtract the minimum value in the data set and then divide this value by the difference between the maximum and minimum value (Equation 3.10) [43]. For this purpose Scikit-Learn's [44] transformer MinMaxScaler was used, this method is particularly useful in ANN and LSTM algorithms [3].

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}} \tag{3.10}$$

Chapter 4

Well Effects on Hook Load

4.1 Friction in Drilling Operations

To model drilling operations, first, it is necessary to fully understand all the phenomena involved in the drilling scenario to be analyzed. One of the most important analysis is torque and drag, as these models aim to predict the weight of the drill string in different scenarios during operation. Most of the models predict the weight of the string, but when compared to field measurements often a discrepancy in results is visible, the reason behind this is in one part due to friction on the well and the other due to load measurement issues that can be referred as Surface Friction.

As it was stated at the start of this study, one of the objectives is to evaluate the feasibility of predicting downhole measurements, in this case, the DWOB from surface measurements, to further use this calculated parameter in the different ML models. For this purpose, an extensive literature review was held, and from various possibilities, considering our dataset limitations, the model proposed by Hareland et al. [10] was identified as the best alternative. The implementation of this model will be explained in the next chapter, as we will discuss the effects that are mentioned in such study and use different points of view from other authors as well.

4.1.1 Friction in the Well

Friction in the well is caused by the interaction between the drill string and the wellbore. As described by Samuel [45] *"friction is an important source of wear and energy loss in the tubular system, is a resistive force that retards the motion of an object."*, this resistive force is seen in the well as drag. The configuration and quality of the well play an important role in the drag force which is also dependent on the kind of operation held if the pipe is rotating, the direction of movement and many other factors.

When trying to predict the behavior of a drill string in the well, the accuracy of the model relies on the use of an appropriate coefficient of friction. There are uncertainties regarding wellbore quality and the possibility of not considering all the contributing parameters involved when determining the coefficient of friction, this could potentially result in over or underestimation of the calculated values. Some of these contributing factors are tight hole conditions, keyseats, aggressive doglegs, cavings, poor hole cleaning, among others, that are associated with problem conditions in the wellbore [11].

4.1.2 Surface Friction - Sheave Effect

As observed in Figure 2.1 (the block and tackle system), the drilling line goes from the reel to the anchor, continues through the crown block and traveling block, and eventually ends up in the drawworks. The weight of the drill string in this image is measured in the cell load located on the anchor. Consequently, the weight must be transmitted through all the sheaves before reaching this point. This system grants an easier operation of heavy loads, as it relies on the principle of *mechanical advantage* [1], but it is necessary to know that there is always some friction present in each sheave. In the past it was assumed that the whole friction loss was caused by the fast line sheave (the sheave closest to the drawworks) [46], further developments on the analysis of this phenomena have been presented over the years, from those, the most relevant will be covered.

The first approximation was presented by Bourgoyne et al. [1], which considers the use of the principle of *ideal mechanical advantage*, that neglects the effect of friction present in the sheaves. In this scenario, the tension in the drilling line is constant throughout the system, Equation 4.1 presents this relation.

$$W = nF_{dl} \tag{4.1}$$

Where n is the number of lines that pass through the traveling block, F_{dl} is the measured tension in the sensor and W the hook load. This approximation does not consider the effects caused by sheave efficiency or block-movement direction.

Luke et al. [46] used another approach for this subject, as it considers different scenarios, for example, when the hook load is lowered, the maximum tension is experienced by the deadline, and the lowest at the fast line, this phenomenon is inversely proportional when the hook load is hoisted. This paper presents the effects of the sheave efficiency effect on hook load and also the possibility of either considering the deadline sheave as active or inactive, this meaning that the friction in this last sheave is considered to be present or not. Finally, the analysis showed that the inactive dead-line sheave model (Equation 4.2) performed the best result in predicting the hook and derrick loads.

$$W = F_{dl}\frac{(1-e^n)}{(1-e)} \tag{4.2}$$

Where n is the number of lines that pass through the traveling block, F_{dl} is the measured tension in the sensor, e is the individual sheave efficiency, and W the hook load.

4.2 Pressure Effects

The Top Drive contains all the necessary machinery for the rotation of the drill string. Therefore, there are several elements connected to it, starting from hydraulic, electrical lines, and the mud hose that connects the surface circulation circuit to the well. All these elements result in an additional force on the Top Drive, that will be recorded by the hook load sensor.

Cayeux et al. [47] presented a complete analysis of how to calculate the effect generated by these lines, but to apply it is necessary to know the dimensions of the drilling mast, the longitude of the mud hose and much more information, not available in this study. Nevertheless, a reference value of how much the impact of this load generates on the recorded hook load is presented, the approach was to analyze the necessary WOB to drill a certain formation when the Top Drive is at its highest position against the lowest. The results presented that, when the Top Drive finished drilling one stand and therefore it was at its lowest position, an additional 2 tonnes, were necessary to drill the formation, compared to, when the drilling was resumed, and the Top Drive was at its highest position.

4.2.1 Stand Pipe Pressure Effect

Hareland et al. [10] presented Equation 4.3 that estimates the additional hook load generated due to the standpipe pressure. The result of this equation will theoretically include the previously mentioned effects (weight of electric and hydraulic lines and mud hose), due to data availability this equation was selected to estimate these effects.

$$HL_{a3} = 5.095 * 10^{-5} * SPP * ID^2 \tag{4.3}$$

Where HL_{a3} is the load effect due to stan pipe pressure in kdaN, SPP is the standpipe pressure in psi and ID the inner diameter of the mud hose in inches. To be consistent with the units used in this study, is necessary to convert the result from kdaN to tonnes.

4.3 Combined Effect

Generally, the hook load is measured indirectly as tension in the deadline or close to the traveling equipment. The position of this sensor is relevant as it will include, or not, additional loads in the system. According to Cayeux [47] et al. some of the forces included are *"weight of the mud hose and umbilicals attached to the top drive, imperfect tension transmission... and static friction in sheave ball-bearings and gravitational and inertia forces"*.

The true weight of the drill string can be recorded with the use of special subs like the one presented by Wu et al. [14], which was connected directly to the top of the drill string, in a Top Drive that was able to host such tool. Most of the drilling rigs do not have with such capability, hence, the hook load measurement comes from indirect measurements in the deadline or by a load cell placed at the hanging point of the Top Drive, in those cases, the measurement is subject to loads that are not related to the weight of the drill string [47].

The platform used to drill the well analyzed in this study, measured the hook load indirectly, with a tension sensor placed in the deadline. This means that based on the available data it is necessary to consider some of the loads that could distort the real from the recorded weight of the drill string, in this case, the sheave friction effect, weight of the traveling block, and standpipe pressure effect. For the first two effects, the Luke et al. [46] inactive deadline sheave equation will be used as presented by Hareland et al. [10] as well as the standpipe pressure effect. Finally, the corrected hook load can be described by Equation 4.4.

$$HL_{corrected} = HL_{a1} - HL_{a2} - HL_{a3} \tag{4.4}$$

Where $HL_{corrected}$ is the corrected hookload (tonnes), HL_{a1} total deadline measurement corrected for a e sheave efficiency (tonnes), HL_{a2} is the weight of the traveling block corrected for the same e sheave efficiency and finally (tonnes), HL_{a3} is the standpipe pressure effect (tonnes).

Chapter 5

Downhole Weight On Bit Calculation

5.1 Measured Hook Load Correction

This chapter will cover the implementation of the hook load correction methodology proposed by Hareland et al. [10] and explained in the previous Chapter. For such purpose, the coding was written in Python [9], and the following calculations were implemented using Jupyter notebook [8]. The input parameters for this calculus were the cleaned data presented in Chapter 3. Due to data unavailability concerning the drilling rig characteristics, specially sheave efficiency, a sensibility analysis was effectuated. According to literature, the individual sheave efficiency ranges from 96% to 99% [46]. The necessary information to run this analysis is:

- Hook load measurement (tension sensor at the dead line).
- Traveling Block Weight.
- Stand Pipe Pressure.

5.1.1 Sheave Effect- Inactive Dead Line Sheave

As mentioned in Section 4.1.2 sheaves generate a difference between the real hook load and the one measured after them, from the different models analyzed in such section the *inactive dead line* model was the most precise and the one that we will use for this calculation. Figure 5.1 present the result of the application of Equation 4.2 (the y axis presents the hook load and

the x axis the number of data points) for different sheave efficiencies, this plot shows that if the sheave efficiency increases the weight of the drill string increases as well.

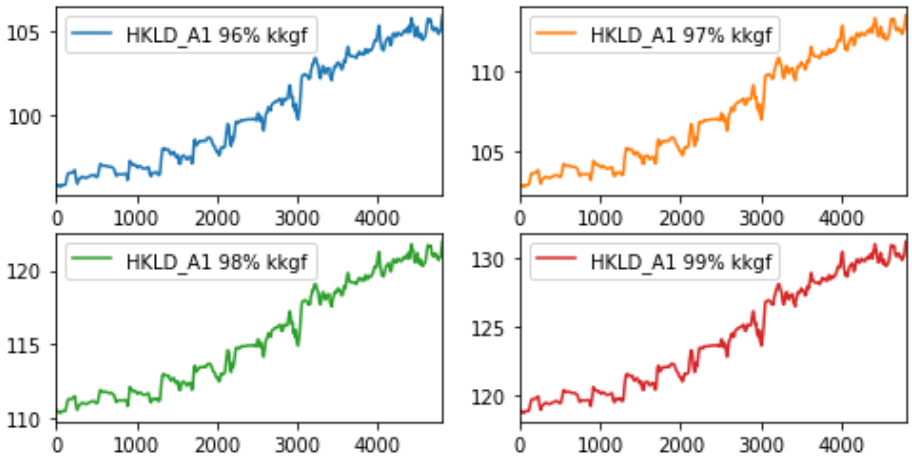

Figure 5.1: Hook load corrected for different sheave efficiencies.

5.1.2 Static Hook Load

To subtract the weight of the traveling block or the *static hook load*, information about the rig is needed, which was not available in this case. Therefore, it was necessary to estimate this parameter with the information contained on the data set.

To determine the weight of the traveling block, without knowing the characteristics of the rig, relying solely on data, requires an understanding of how the drilling processes are held. Once one stand is drilled, it is a good practice to reciprocate the drill string allowing the cuttings generated to circulate above the BHA, depending on the operator policies this step might be skipped, but regardless of this, to add a new stand to drill, the top drive is lowered, pumps shut down and the top drive disconnects from the drill pipe. After this, the *"hook load"* registered by the sensor is only the weight of the traveling block.

Figure 5.2 shows that the lowest values registered for the hook load are between 50 and 60 tonnes, to identify which is the value that is most repeated between this two values, data analysis techniques were used and the result is that the traveling block weight is approximately 55 tonnes.

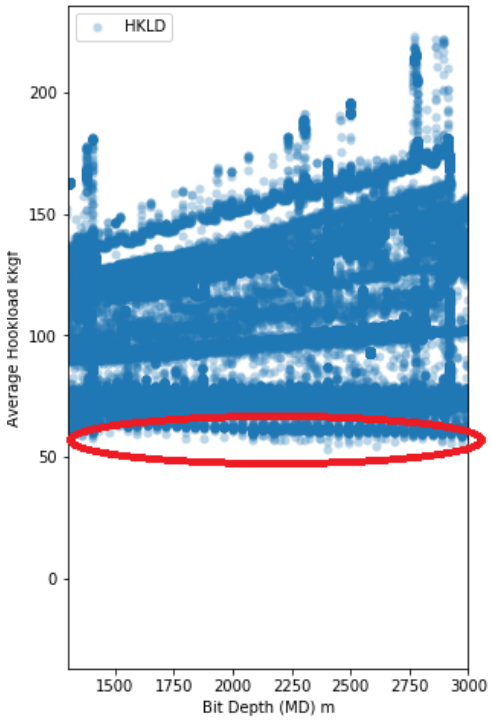

Figure 5.2: Traveling block weight determination.

This value is included in the hook load recorded by the tension sensor present in the dead-line. Therefore, it is necessary to apply the same procedure as in the previous case and take into consideration different sheave efficiencies (Figure 5.3). As seen in Equation 4.4 the value calculated will be subtracted from the one defined previously in Section 5.1.1.

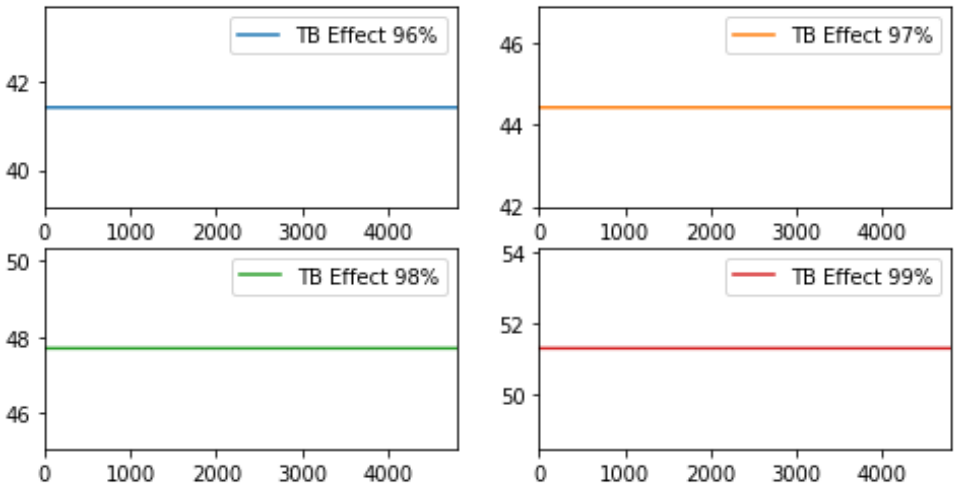

Figure 5.3: Traveling block weight corrected for different sheave efficiencies.

5.1.3 Stand Pipe Pressure Effect

This is another effect taken into consideration when correcting the hook load measurement, and as its name state is dependant on the standpipe pressure. All calculations were based on Equation 4.2, Figure 5.4 presents the results of such calculations for the data points contained in the analyzed section, already converted to Tonnes, this value is directly subtracted from the previous ones as described by the methodology proposed by Hareland et al. [10].

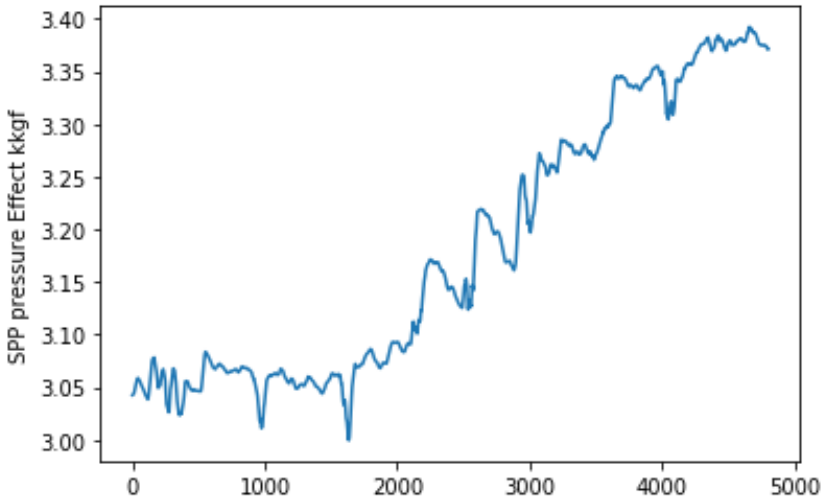

Figure 5.4: Standpipe pressure effect.

5.1.4 Corrected Hook Load

Since all the effects on the hook load have been defined, based on Equation 4.4, it is possible to determine the *True Weight* of the drill string. For this purpose, a sensitivity analysis of different sheave efficiencies was held.

Figure 5.5 shows the result of such analysis (the y axis present the depth of the well and the x axis the hook load measured in tonnes), with this plot is evident that as the efficiency of the sheaves increases, the value of the hook load increases as well, meaning that, each sheave transmits more efficiently the weight of the string. To identify which sheave efficiency describes the real weight of the string, these values will be compared with the one estimated by a Torque and Drag model (T&D) that will be explained in the next section.

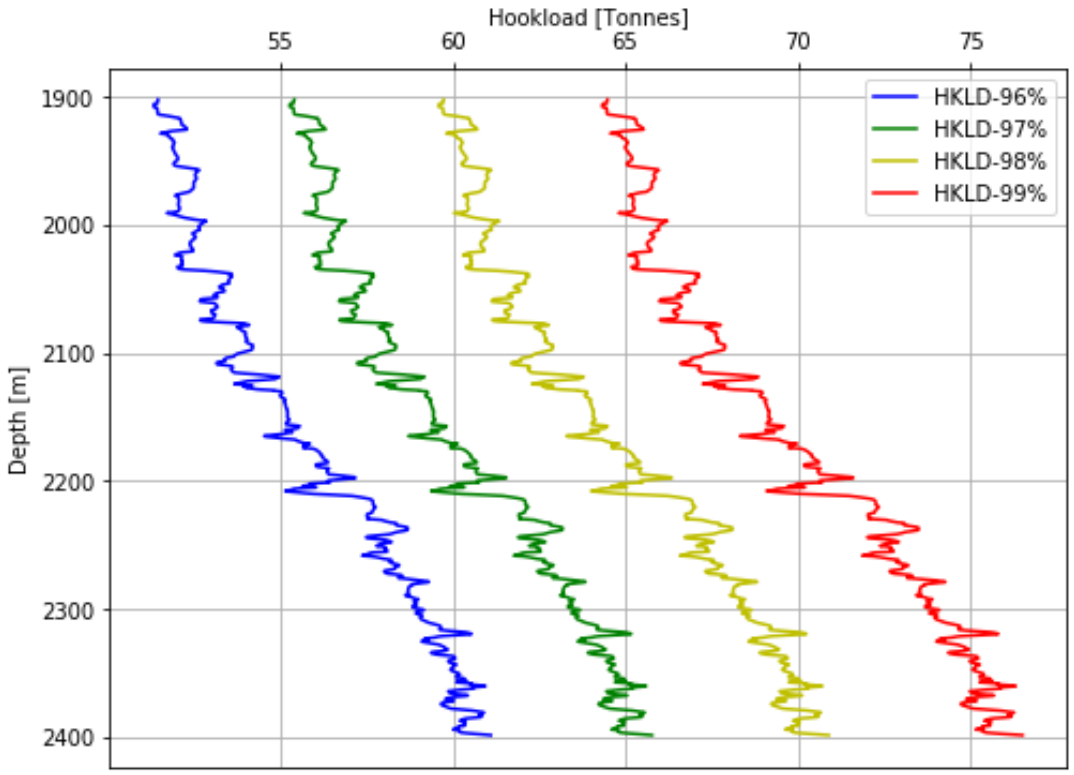

Figure 5.5: Corrected hook load (all effects), sensitivity analysis for different sheave efficiencies.

5.2 DWOB Calculation via T&D model

The determination of the DWOB from surface parameters is one of the most important objectives of this study, as it aims to refine current data-driven models by using improved input parameters based on petroleum engineering knowledge. To achieve this, the main step in this process is to implement a T&D model that can successfully calculate the parameters needed for this purpose, to give further validation, the results of such model and the previous corrected hook load will be compared with Halliburton's WellPlan [12] software, one of the most used industrial software for well design.

5.2.1 Torque and Drag Model

After reviewing different possibilities, the Johancsik et al. [11] T&D model was selected to be implemented in this study. Because up to this date this model is still regarded as one of the most precise ones, it relies on the principle that the torque and drag forces in directional wells are primarily caused by sliding friction.

To determine the normal force is necessary to estimate the weight increments for an element of the drill string. As described by Johancsik et al. [11] *"The net normal force, F_n, is the negative vector sum of normal components from the weight, W, and the two tension forces, F_t and $F_t + \Delta F_t$"*. The magnitude of the normal force is defined by Equation 5.1:

$$F_n = \left[\left(F_t \Delta\alpha sin\bar{\theta}\right)^2 + \left(F_t \Delta\theta + W sin\bar{\theta}\right)^2\right]^{1/2} \tag{5.1}$$

Where:

F_n = Net normal force acting on the element (N).

F_t = Axial tension acting at lower end of the element (N).

$\Delta\alpha$ = Increase in azimuth over the length of the element (rad).

$\bar{\theta}$ = Average inclination angle of the element (rad).

$\Delta\theta$ = Increase in inclination angle over the length of the element (rad).

W = Buoyed weight of drillstring elemenet (N).

This equation leads us to the equation that describes the tension increment and is defined by Equation 5.2:

$$\Delta F_t = W cos\bar{\theta} \pm \mu F_n \tag{5.2}$$

Where μ is the sliding friction coefficient between the drill string and the wellbore (dimensionless), in case of a rotating pipe, the second part of this equation becomes zero. Finally, for the torsion increments can be defined by Equation 5.3:

$$\Delta M = \mu F_n r \tag{5.3}$$

Where ΔM is the increase in torsion over length of element (Nm), r is the characteristic radius of drill string element (m).

5.2.2 Obtaining DWOB Values

One principle stated by Johancsik et al. [11] is that in case the sliding friction factor parameter is unknown, it is possible to back-calculate it, assuming a friction coefficient and iterating to match the data, for this, the drill string characteristics and wellbore trajectory are required. For the case analyzed in this study, it is aimed to determine the DWOB using the same approach, which was also addressed by Hareland et al. [10]. To achieve this several steps are necessary and will be further explained.

Split the Well in Sections

According to Johancsik et al. [11] the T&D calculations in the paper presented were elaborated for a well divided into 30.5-meters elements, in some cases this could provide an acceptable precision, but since the model aims to predict the DWOB with a small difference concerning the real value, this approach may not be suitable for our case.

To further explain the reason behind this, is necessary to analyze the BHA case, for example, if we have two drill collars weighing 238 kilograms per meter and one heavyweight drill pipe weighing 78 kilograms per meter in a 30-meter section, *which weight should we use?*, an average weight per meter would give us a lower weight (approx. 4740kg) than the real one (approx. 5540 kg) and also if the weight of the drill collar is assumed for the whole 30-meter section, the result would be higher than the real one. To solve this issue, it was decided to split the BHA into 5-meter length sections, as this was accurate enough for the smallest element in the BHA. The other sections of the well that contained only drill pipes with the same linear weight were analyzed every 20 meters.

Define the Trajectory of the Well for Each Data Point

The well trajectory information had survey measurements separated every 40 meters, this is bigger than the smallest section to be analyzed (5 meters), to solve this issue was necessary to interpolate within the surveys to have the same separation (5 meters) between survey points. The method selected to interpolate between survey points was taken from Mitchell et al. [13].

Once the interpolation process was completed, another issue was faced. The data were re-sampled to 0.1 meters, this means that it was necessary to calculate the DWOB every 0.1 meters to have the same number of data points in the T&D model as in the data set from the well. To solve this issue, an algorithm that determines the position of the bit and defines the inclination and azimuth for each cell, based on the nearest survey point, was implemented. As it was observed that the inclination and azimuth did not change significantly every 5 meters.

Calculate DWOB for Each Data Point

As proposed by Hareland et al. [10] and Johancsik et al. [11] once the well has been divided into sections, and the trajectory is known for each data point, the approach for determining the DWOB is based on the iterative use of the "shooting method" [48] designed to solve boundary value problems. An initial guess of the DWOB must be submitted to the implemented algorithm, e.g., the first guess could be to assume that the DWOB and the SWOB are equal, with this lower boundary condition, the algorithm back-calculates the tension across the well until the surface is reached and returns the weight of the drill string. This hook load value is compared to the corrected hook load previously defined (96-99% sheave efficiency) if the difference is higher than a predefined tolerance error another DWOB guess is set as the lower boundary condition (increased or decreased based on the difference of hook loads). This process is repeated until the difference between the hook loads is within the predefined tolerance (0.25 Tonnes).

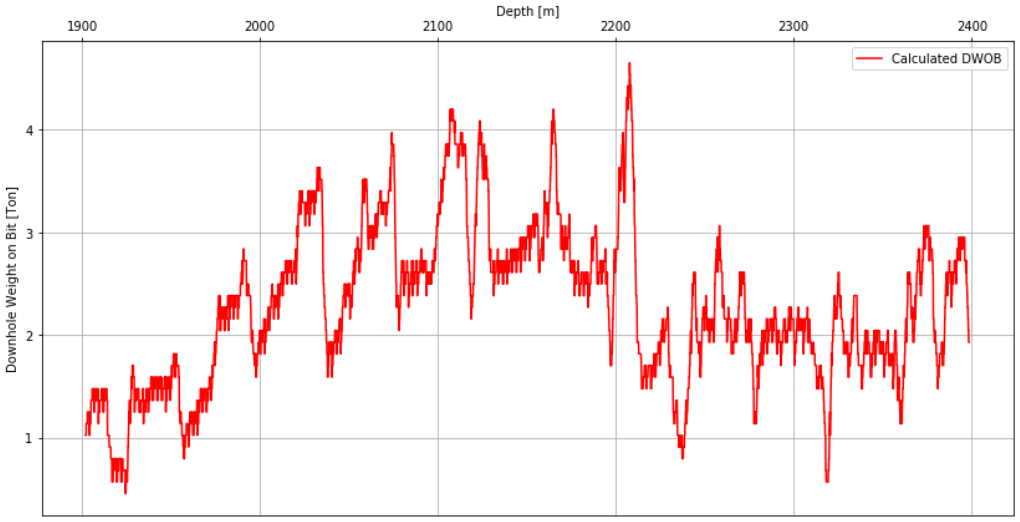

Figure 5.6: Calculated DWOB values.

Figure 5.6 presents the results of the calculated DWOB, the procedure mentioned previously must be repeated for each of the more than four thousand data points. To achieve this, the code includes a series of loops and conditional statements to automate this process.

Chapter 6

Machine Learning

6.1 Machine Learning Implementation

This chapter will cover the implementation of different machine learning techniques using Python [9], but more than this, evaluate with the available data set, which model provides the best results in a simulated real drilling scenario. For this, the most widely used machine learning algorithms will be used, and evaluated in two different scenarios provided by the kind of data splitting selection.

Number	Feature	Units	Data Type
1	Bit Depth (MD)	meters	float64
2	Downhole Weight on Bit	tonnes	float64
3	Average Hookload	tonnes	float64
4	Downhole Torque	kN-m	float64
5	Average Rotary Speed	rpm	float64
6	Average Standpipe Pressure	kPa	float64

Table 6.1: Parameters used in model comparison.

To implement the Random Forest (RF) and K-Nearest Neighbors (KNN) regressor models, Scikit Learn [44], a Python [9] module that contains these machine learning algorithms was used. For the Artificial Neural Networks (ANN) and Long Short Term Memory (LSTM) models, Keras [49] a deep learning API written in Python [9], that runs on top of TensorFlow [50] was used. Relevant parts of the code for each algorithm are located in Appendix A. The parameters used as inputs to compare the models' performance are presented in Table 6.1.

These parameters had the best correlation with the ROP, a more in-depth analysis that will include the use of surface parameters (SWOB and surface torque) and calculated parameters (calculated DWOB) will be held in Chapter 7, as this study aims to evaluate if the calculated parameters improve the performance of ML models compared to only using the given surface parameters.

6.1.1 Splitting Data

How to properly split the data?, is something that is not usually written when working with drilling data, as per the literature review, most of the papers discussing this topic are not clear about how they split their data. First, we need to mention that when working with machine learning models, the data should be divided into three parts to avoid overfitting and model bias, they are called:

- Training set (largest one).
- Validation set.
- Test set.

There are some cases when the data is split into two parts and the validation set is called a test set, which is also acceptable. Depending on the quantity of data available the corresponding percentages to each element can vary. The training set is the data used to fit the model, and as its name state, is used to train the model, it must optimize the parameters while observes and learns from this particular data. The test set is used to evaluate how a model will generalize to new cases, it should be used only after the model has passed through the training and validation set (if this one was created). A sign that the model is overfitting the training data, is given by a high testing error but low training error [3].

Another topic to be considered is the ratio of splitting the data, usually, this depends on the quantity of data available, for the data set used in this study more than 4000 data points were available, a 60-40 percent split was used for the training and testing sets, respectively.

Sampling Random or Sequential

Once the percentages of information to be assigned to the training and testing sets are defined, how the data points will be selected for each of them is another important matter that will define the performance of the model. Two options, among many, were identified as suitable for the study:

- Random Sampling.
- Sequential Sampling.

The first option, *Random Sampling* consists of, randomly select 60% of the available data points and assign them for the training set, leaving the residual 40% for testing. There are certainly advantages regarding this option, as the data will be trained with a bigger span of different data, e.g., the last meters drilled could have had a higher ROP than the first meters, and such values are only encountered at these depths. Therefore, the data would have been *trained* with data that has in some way previously seen and increase the possibility of having a better performance. The drawback of this approach is that, it is hard to determine if the model has *learned* to find correlations, allowing it to make predictions or it has just *memorized* points. Meaning that, when presented with new data, different than the one that has been trained on, the prediction accuracy would decay.

The other option, *Sequential Sampling*, splits the data in the requested percentages but avoids randomizing it. For example, when using sequential sampling the first 60% of the data would be used for training and the rest for testing, this, in a way is similar to a real drilling scenario and assures that when testing, the data is *out of sample*. This methodology ensures that the model creates the necessary correlations to properly predict results, a comparison between the two possibilities was executed with different machine learning algorithms.

6.1.2 Random Forest Regressor

The algorithm was implemented using Scikit Learn [44], to obtain the best performance for any ML algorithm, hyperparameter tuning is one of the most important steps, for this purpose

many possibilities are available starting by manual tuning that is a very time-consuming activity, another possibility and the one chosen for this study is to use Scikit Learn's based commands such as GridSearchCV. To run such a search is necessary to input some base-line parameters that will be evaluated and compared using a cross-validation procedure that is described more in-depth by Hastie et al. [51]. In the case of the random forest regressor, the parameters evaluated were:

- Number of estimators: 10, 30, 50 or 70.
- Maximum depth of the tree: 10, 20 or 30.

The search uses these suggested parameters, but also includes other hyperparameters of the model, the result of this search was implemented in the model. After this important step, the data is introduced into the model, and the two options presented in the previous section were analyzed. First, using the *random sampling*, figure 6.1 shows the results of RF regressor when randomly selecting the data, the coefficient of correlation obtained is of 0.986 which means that this model has a high prediction accuracy.

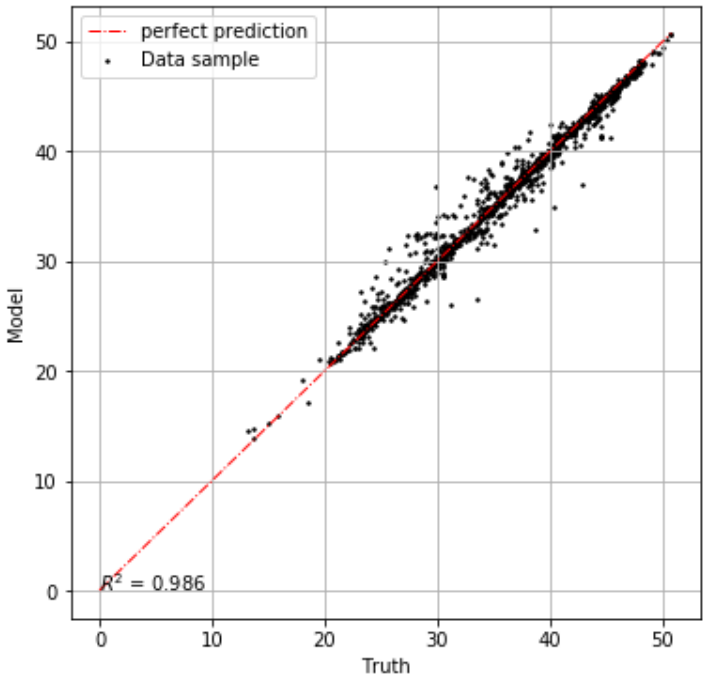

Figure 6.1: Random Forest Regressor results using random sampling.

The high prediction accuracy provides confidence that the model would perform well in any given circumstance. Therefore, *sequential sampling* was tested with the same percentages.

Figure 6.2 presents the results for the RF regressor for this case, as it is noticeable, the accuracy of the prediction is lowered to half compared to the one obtained using random sampling. This behavior raised questions about the model applicability in the desired scenario.

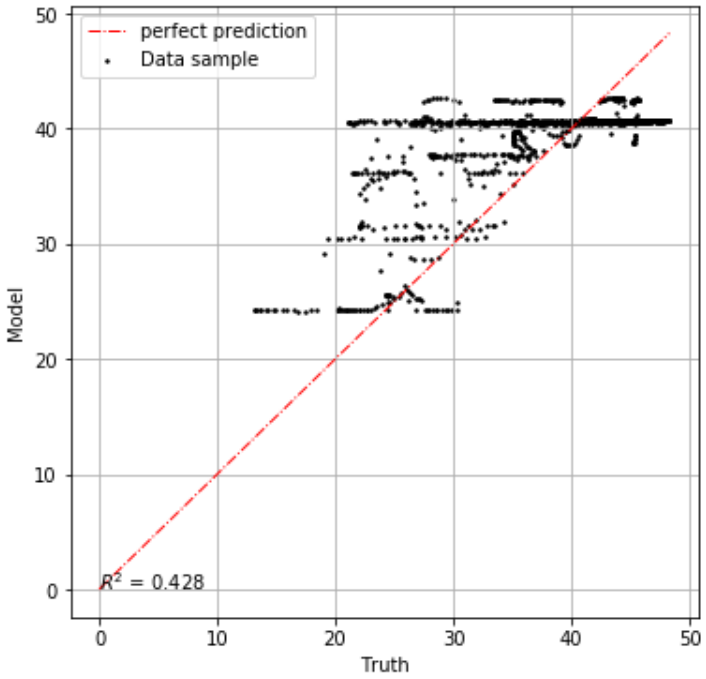

Figure 6.2: Random Forest Regressor results using sequential sampling.

6.1.3 K-Nearest Neighbors

As with the RF, the KNN algorithm was implemented using Scikit-Learn [44], to obtain the best performance the algorithms hyperparameters were tuned using GridSearchCV, in this case, the parameters evaluated were:

- Number of neighbors: 3, 5 or 11.
- Weights: distance or uniform.
- Algorithm: auto, ball-tree, kd-tree or brute.

After this search concluded and the best parameters were found. The algorithm was tested with *random sampled* and *sequential sampled* data as the RF model. Figure 6.3 shows the results for *random sampling*, the coefficient of determination in this case (R^2) is 0.99 meaning that we have a perfect fit for our predicted points.

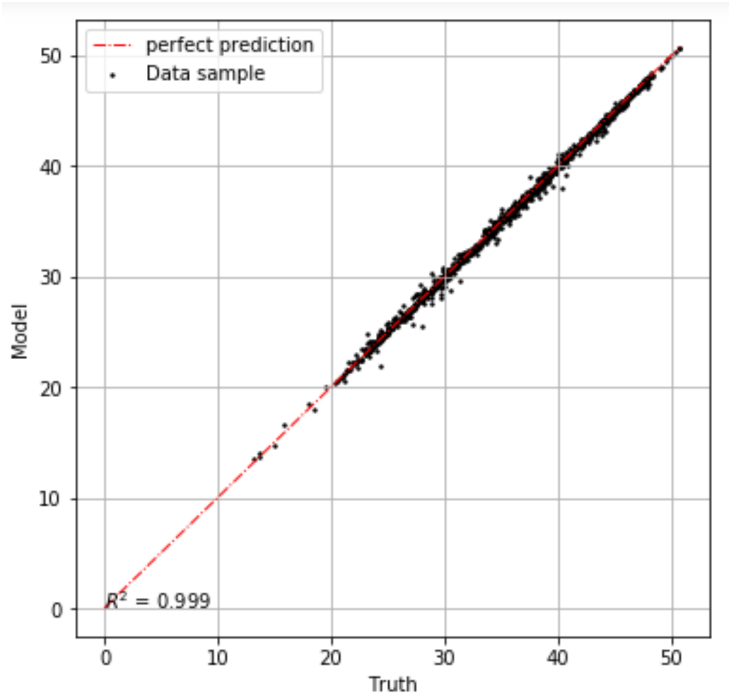

Figure 6.3: K- Nearest Neighbors Regressor results using random sampling.

Figure 6.4 presents the results for the model using the *sequential sampling* methodology, in this case, the R^2 value is equal to 0, meaning that the predictions are not accurate at all.

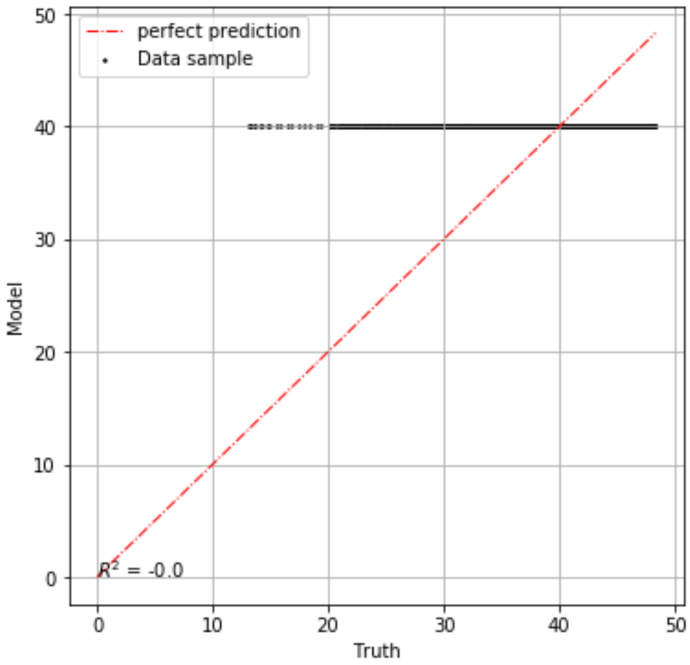

Figure 6.4: K- Nearest Neighbors Regressor results using sequential sampling.

6.1.4 Artificial Neural Networks

The ANN model was implemented using Keras [49]. As explained previously in Chapter 2, to build an ANN is necessary to determine the number of layers and neurons in each layer. There are no clear rules when building ANNs, according to Amer et al. [28] *"defining the optimal network that simulated the data sets is not an easy task. Although it is an iterative process, some rules of thumb were followed as guides."*. These rules of thumb include defining the number of neurons and layers based on the number of inputs for the model, but other parameters need to be considered:

- Activation function.
- Learning rate.
- Dropout layers.
- Callbacks.
- Optimizer.
- Loss function.
- Metrics to evaluate the performance of the model.

As with the RF and KNN models, to obtain the best possible performance of the ANN model is necessary to optimize the hyperparameters, this task can be carried out manually but this is a time-consuming task, especially if we consider that the model can have more than one layer. There are some options to automate this process, the first option evaluated was Keras Tuner [52] a library that helps to determine the optimum hyperparameters for an hypermodel defined, the other option is to develop a functional model that we consider a good approximation and use Scikit-Learn's function RandomizedSearchCV to run a search for the best possible combination of only selected hyperparameters.

Once the model is defined, it is necessary to ensure that the model does not overfit the data. To prevent this issue, additional *Dropout* layers were added, the function of these layers is to

randomly deactivate a predefined percentage of the neurons during the training phase of the model. As an additional measure to prevent overfitting, an *EarlyStopping* callback was introduced to the model. *Which is the purpose of this clause?*, when defining the number of *epochs* (an epoch is completed when the data has passed both forward and backward through the model) that the model will complete, a low number of epochs can potentially reduce the accuracy of the model, whereas too many epochs can cause overfitting, this clause stops the training once the improvement of the model on the validation set stops.

As with the RF and KNN, the model was executed first using *random sampling*. To improve the visualization of how this approach is related to the data, Figure 6.5, contrary to the previously presented cross-plots, shows how the model predictions compare to the actual data. The R^2 value for this model is 0.85, which is lower than the one obtained with the RF and KNN regressors.

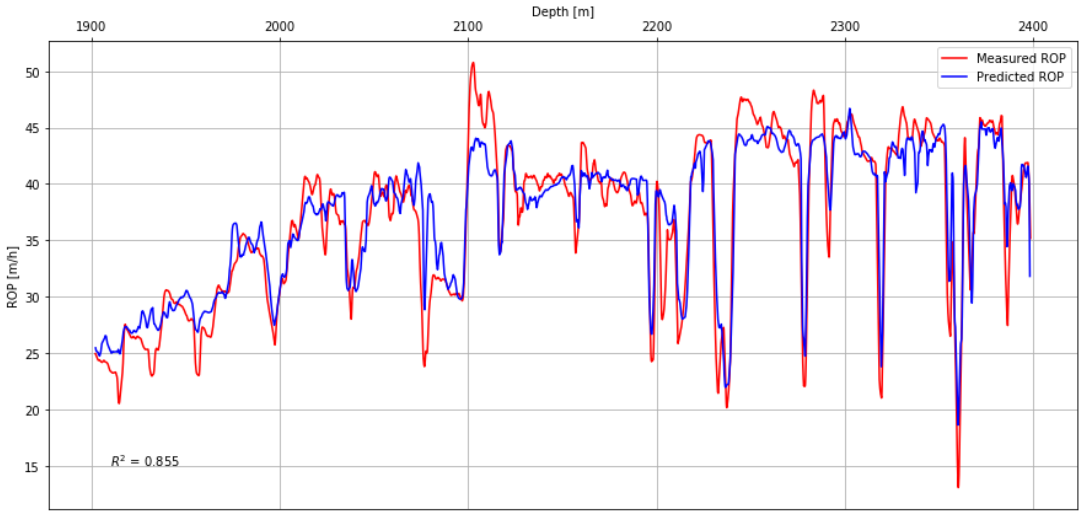

Figure 6.5: Artificial Neural Network results using random sampling.

Now is turn to test the model with *sequential sampling*, Figure 6.6 presents the results for this approach, it is visible that the first 300 meters of the data were used for training and the rest for testing, in this case, the value of R^2 is 0.198 which means that the model is not accurate enough for this purpose.

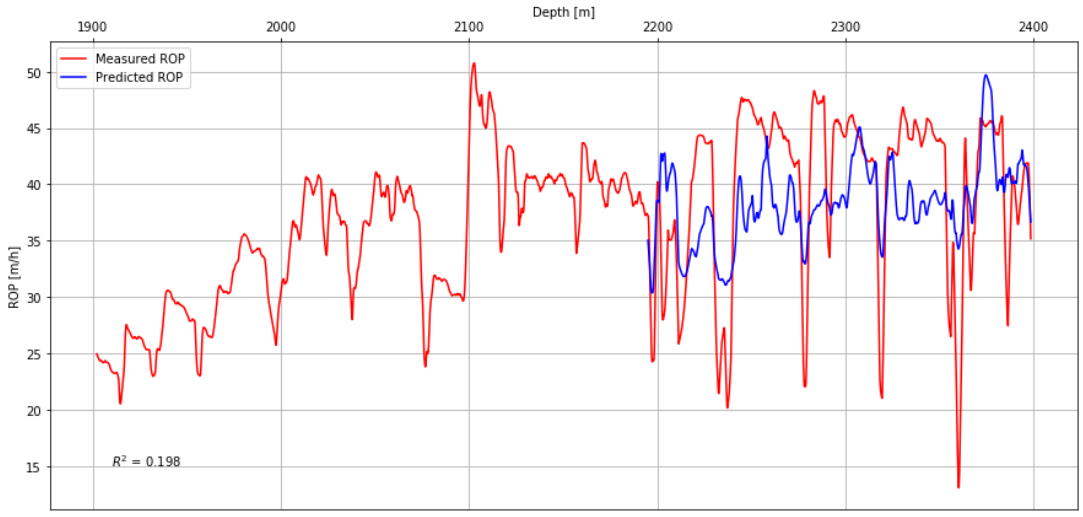

Figure 6.6: Artificial Neural Network results using sequential sampling.

6.1.5 Long Short Term Memory (LSTM)

The concept of LSTM [31] models is not recent, but its application in the oil and gas industry, especially in drilling, is relatively new with few publications surrounding ROP modeling with such structures available to this date. As was seen with the first three models evaluated, the performance when using *random sampling* was better than for the *sequential sampling* with the available data. To solve this type of sequence prediction problem Recurrent Neural Networks (RNN) were developed and long short term memory (LSTM) models are perhaps the most successful RNN. LSTMs can be used to work either as time or preferably for our application in a depth base sequence. One of the main improvements from this approach concerning the previous models is that it considers time sequence effects [30], e.g., interactions between rock and drill bit, changes in pressure, torque, rpm, etc.

The process of designing an LSTM model is similar to the ANN, for this case a manual search was conducted to identify the best hyperparameters for the model, it is worth mentioning that some of the features for this model were left as they came by default, activation (tanh) and recurrent activation (sigmoid). To avoid overfitting as with the ANN model, *droput* layers and *earlystopping* callback clauses were used in the model. The structure of this model is not suitable for the use of *random sampling*, it works better when solving problems that use *sequential sampling*, therefore, in this case, this is the only option analyzed.

The difference between these two models (LSTM - ANN) relies on how it handles the features used for predicting a value, one of the advantages of the LSTM (as mentioned in Chapter 2), is the ability of LSTM cells to remember some information, in other words, having *"memory"*. *"How can we use this in our context?"*, to predict the ROP value at a time (t), the input parameters showed in table 6.1 can be stored for several time steps, e.g., for the WOB parameter we could have stored in memory WOB at a time (t-1), up to time (t-n), this depending on how much information is considered relevant to be stored. Not only this, but the ROP value from previous times can be used as an input parameter for predicting the following ROP values. Therefore, besides the parameters shown in table 6.1, the ROP value from previous times was included as an input parameter for the model, the logic behind this is that, in a normal drilling operation, this parameter will be known at the same time as the other parameters and it can help us to predict the future value of ROP.

The implemented LSTM model makes an ROP prediction based on three previous time steps, starting at a time (t-1) up to time (t-3), this provides the model enough stability to make predictions with a fairly good amount of precision. In other words, the model makes a prediction based on 21 inputs, the shape of the input is a three-dimensional array, e.g., for one particular point the input for the model have the shape (1,3,7), which means that is one row, three time-steps, and 7 columns (one for each input parameter).

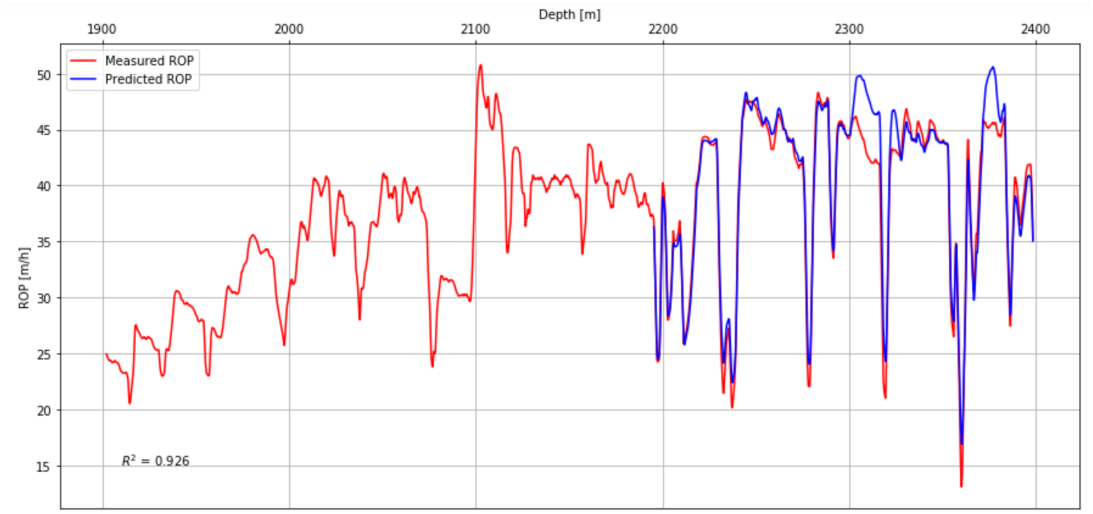

Figure 6.7: Long Short Term Memory results using sequential sampling.

Figure 6.7 presents the results for the LSTM model, the first 300 meters of the data were used for training, and the rest for testing, obtaining a value for R^2 equal to 0.926 and an MAE of 1.44 meaning that the model accurately predicts the behavior of ROP for this well. It is important to mention that these results are based on the hypothesis that the input parameters are known for the times (t-1) up to (t-3), a more in-depth evaluation of this model performance for different scenarios will be presented in Chapter 7.

Chapter 7

Results and Discussion

7.1 DWOB Calculation Results

From Figure 5.5 all efficiencies were tested and compared with the implemented Johanscik et al. [11] T&D model, but the 99% sheave efficiency provided the best results compared to the calculated ones, this was validated using Halliburton's WellPlan [12]. The maximum allowable difference in hook load was set at 0.25 Tonnes. Figure 7.1 shows the results of the hook load comparison between the implemented T&D, the corrected hook load at 99% sheave efficiency, and WellPlan results.

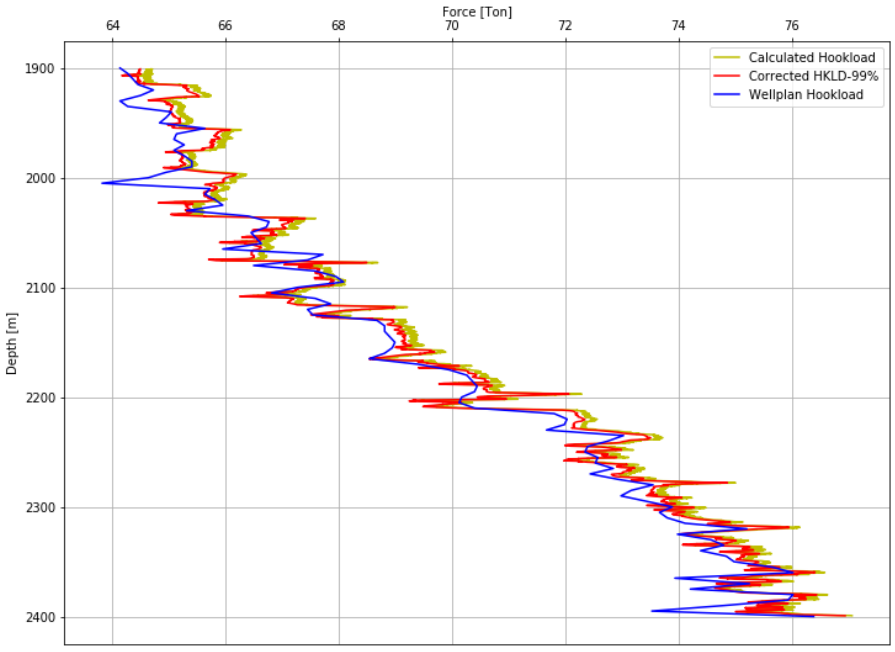

Figure 7.1: Comparison between measured, calculated and WellPlan hook load.

Figure 7.2 presents a comparison between the measured and the calculated DWOB, it is worth mentioning that to obtain such a parameter, usually, it is necessary to use wired drill pipe [16], this translates into higher expenses for a drilling project. But, the objective of such calculation, in this study, is to verify if this parameter can lead to improved accuracy in ROP prediction.

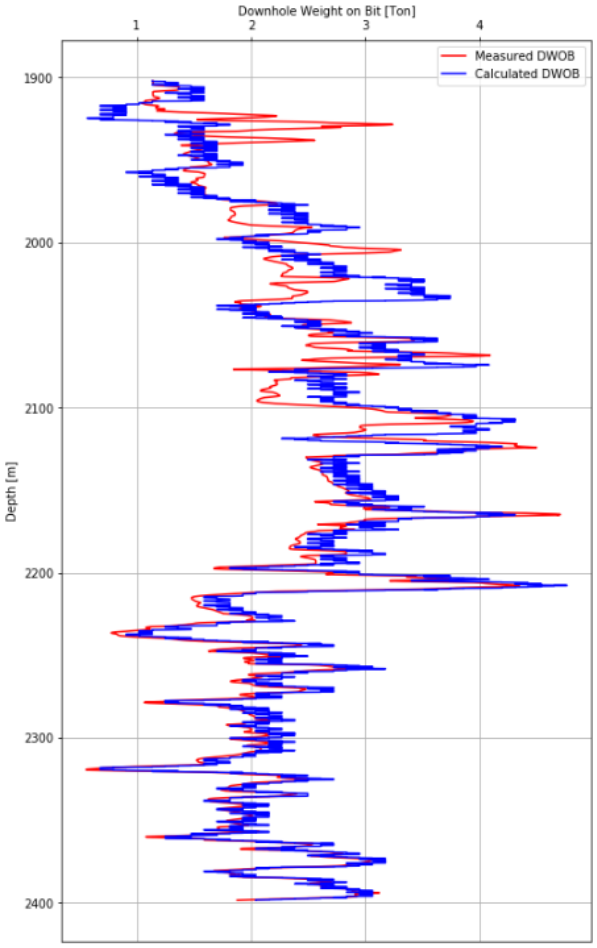

Figure 7.2: Comparison between measured and calculated DWOB.

The implemented model was able to predict the DWOB from surface parameters with only a 10% Mean Absolute Percentage Error (MAPE). But why it is important to determine the value of the DWOB, as mentioned, the WOB parameter is used in all of ROP models, physics-based or ML. Usually, as with the data for this well, there is a difference between the surface and downhole WOB, some of the reasons behind it have been explained in this work, but, to visualize this difference Figure 7.3 shows the values for the SWOB in the data set.

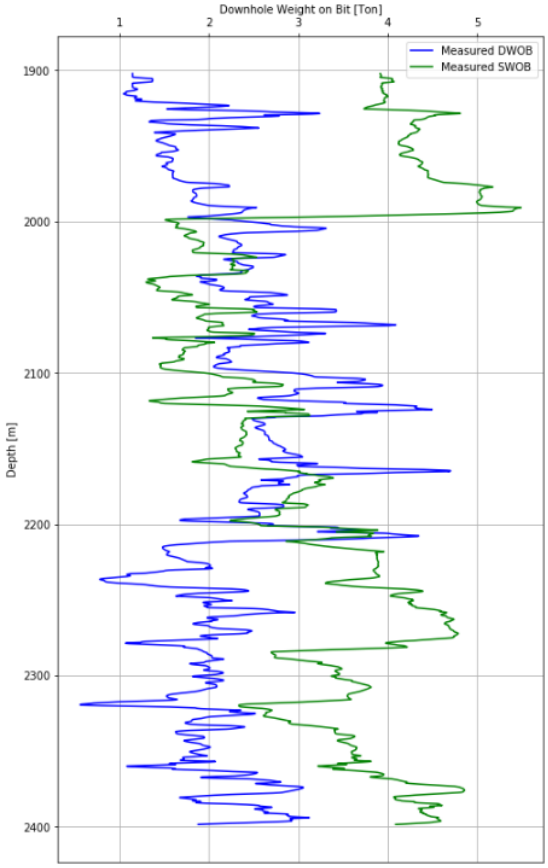

Figure 7.3: Comparison between measured SWOB vs DWOB values.

If we compare the values of SWOB and DWOB, it is evident that there is a difference throughout the whole analyzed section, in some cases, the SWOB is twice the value of the DWOB. If for some reason these values (SWOB) were set as an input for a physics-based model, the accuracy of the prediction would have reduced considerably, as all models assume that the value of SWOB is the measured value of the weight applied to the bit at the bottom of the well. How this affects a ML model will be explained in the next section, as all parameters: surface, downhole and calculated will be fed to the ML models.

7.2 Machine Learning Results

As mentioned in Chapter 6, the models were evaluated using the parameters that would ensure the best performance of each one of them, to determine the model that performs better and that could be used in a wider scope than the one presented in this work.

Table 7.1 presents the summary of such work, the results show that for the data used in this study (using downhole measurements as input parameters for the model), the first three models evaluated, RF, KNN, and ANN performed well when using *random sampling* but not performed properly when the data was sampled *sequentially*.

Number	**Model**	**Random Sampling** R^2	**Sequential Sampling** R^2	**Random Sampling MAE**	**Sequential Sampling MAE**
1	Random Forest Regressor	0.986	0.428	0.415	4.514
2	K-Nearest Neighbors Regressor	0.99	0	0.136	5.904
3	Artificial Neural Networks	0.85	0.198	2.234	5.589
4	Long Short Term Memory	N/A	0.926	N/A	1.44

Table 7.1: Machine learning models evaluation summary.

Which of these models should we select as the best model for the study?, to answer this is necessary to make sense of the results presented in Table 7.1. *What could be understood from the models that performed well using random sampling*?, these models (RF, KNN, and ANN), could have *memorized* points instead of created correlations. As described by Gulli et al. [4] *"traditional multilayer perceptron neural networks make the assumption that all inputs are independent of each other. This assumption breaks down in the case of sequence data"*.

It is important to consider that if this methodology is used as a base for a real-time drilling application, the data is fed to the model sequentially, and not all of the information is available for the model as the data is generated while drilling. Hence, the LSTM model that had the best performance when *sequential sampling* was used, and is the one chosen to be the best model for predicting ROP on a real drilling scenario.

7.2.1 Surface Measurements as Input Parameters

First, the model was evaluated using only surface measurements, this data is available in most of the drilling rigs. The input parameters are presented in Table 7.2. It is important to mention that to work with LSTM models the data must be scaled in a range between 0 and 1 (See Chapter 3 Section 4).

Number	Feature	Units	Data Type
1	Bit Depth (MD)	meters	float64
2	Weight on Bit	tonnes	float64
3	Average Hookload	tonnes	float64
4	Average Surface Torque	kN-m	float64
5	Average Rotary Speed	rpm	float64
6	Average Standpipe Pressure	kPa	float64
7	Rate of Penetration	mph	float64

Table 7.2: Surface measurements used in LSTM model.

Figure 7.4 presents the results obtained when using the surface measurements as inputs for the model, the coefficient of determination (R^2) value is equal to 0.894 and the MAE is equal to 2.14. These values may not be considered accurate enough for an ROP predictive model.

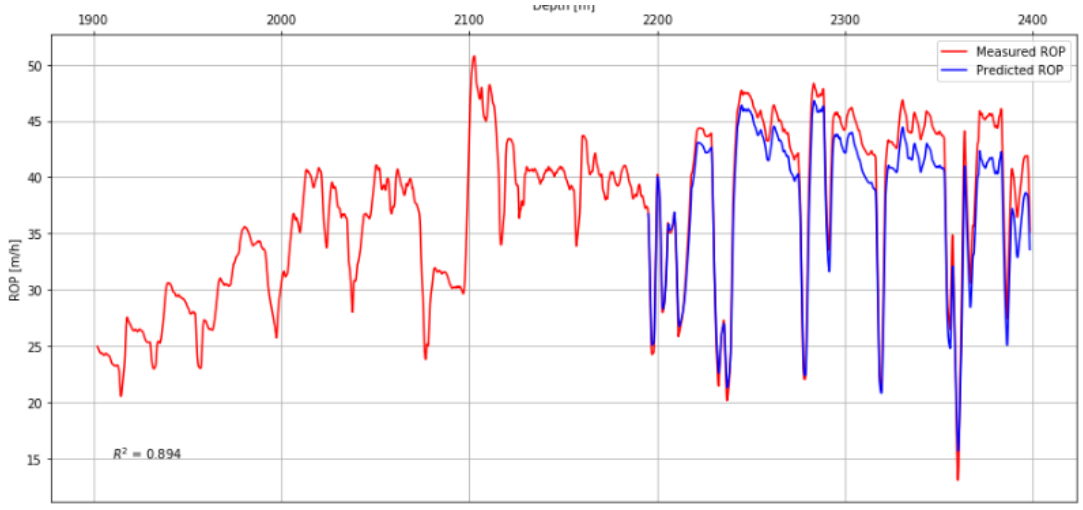

Figure 7.4: LSTM model results using surface measurements as input parameters.

7.2.2 Surface Measurements with Calculated DWOB as Input Parameters

For this evaluation, the inputs for the model consist of surface measurements, but the value of SWOB was replaced by the calculated DWOB. Table 7.3 presents all the parameters that were used as input for the LSTM model.

Number	Feature	Units	Data Type
1	Bit Depth (MD)	meters	float64
2	Downhole Weight on Bit (calculated)	tonnes	float64
3	Average Hookload	tonnes	float64
4	Average Surface Torque	kN-m	float64
5	Average Rotary Speed	rpm	float64
6	Average Standpipe Pressure	kPa	float64
7	Rate of Penetration	mph	float64

Table 7.3: Surface measurements with calculated DWOB used in LSTM model.

Figure 7.5 presents the results of the model when using the previously shown parameters. In this case, the model performed better than when using only surface measurements as input for the model. The value of R^2 is equal to 0.95 and the MAE is equal to 1.41.

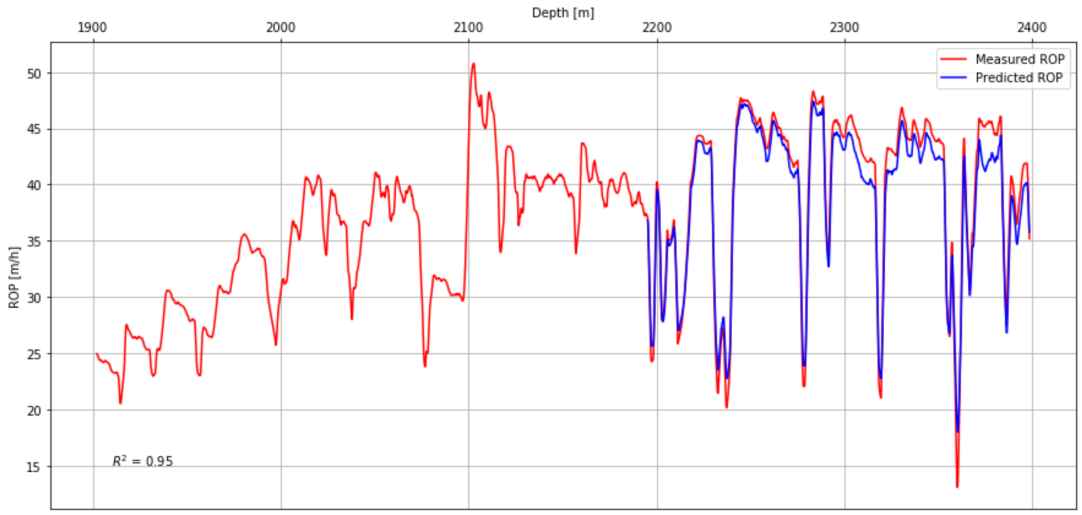

Figure 7.5: LSTM model results using surface measurements with calculated DWOB as input parameters.

7.3 Discussion

Once the algorithm that calculates the DWOB from surface measurements was successfully implemented and also identified the best ML model for ROP prediction. The model was evaluated with three sets of data, using: downhole measurements, surface measurements, and a combination of surface measurements with the calculated DWOB as input parameters. To visualize the values obtained during this search Table 7.4 presents the results for the different cases.

Number	Data used as Input for LSTM model	R^2	MAE	Figure
1	Downhole Measurements	0.926	1.44	Figure 6.7
2	Surface Measurements	0.894	2.14	Figure 7.4
3	Surface Measurements w/ Calculated DWOB	0.95	1.41	Figure 7.5

Table 7.4: LSTM model evaluation summary.

The best result for this model was obtained by the last option evaluated based on the MAE metrics, the expected average error in the prediction is of 1.41 mph, which is the lowest value of all the options evaluated, corresponding to the surface measurements with calculated DWOB. Also, if we compare directly the difference of the MAE values obtained using option 2 or 3, it is hard to imagine that the reason behind this is only that the SWOB value was replaced by the calculated DWOB.

As seen in Figure 7.3, the values of SWOB in most parts of the section are not near the true values of DWOB, and it was mentioned that if used in a physics-based ROP model this would have produced wrong predictions of ROP, this behavior was also replicated by the ML models. Therefore, the question, *Does improved data translates into better modeling results?*, was answered. As the results showed, replacing a parameter (SWOB) that was not providing relevant information to the model, with another parameter (DWOB), that did, the model successfully found the correlations needed to predict the aimed value (ROP) with good accuracy.

7.3.1 LSTM Performance Analysis

As mentioned in Chapter 6, the model was built as a *"rolling"* LSTM architecture able of keeping in the memory many previous time steps, for the results presented in Table 7.4 the model predicted the ROP for the next time step (t), fed with information from all of the previous time steps (t-1) up to (t-3). The performance of the model was evaluated in different scenarios using the input parameters from Table 7.3, with the predicted ROP replacing the value of known ROP value from previous time steps (the other parameters are known at all times), the results are presented in Table 7.5.

Test	Predicted Value	Known ROP value used as Input	Predicted ROP value used as Input	R^2	MAE
1	ROP (t)	$(t-2)$ $(t-3)$	$(t-1)$	0.92	1.78
2	ROP (t)	$(t-3)$	$(t-1)$ $(t-2)$	0.77	3.07
3	ROP (t)	None	$(t-1)$ $(t-2)$ $(t-3)$	-3.17	13.83

Table 7.5: LSTM performance analysis.

First, the model was evaluated using the predicted ROP value at the time (t) to continuously replace the value of (t-1) for the next time steps, but keeping the information for the times (t-2) and (t-3) as provided in the data set. The results show that the R^2 value decayed from 0.95 to 0.92 and the MAE increased from 1.41 to 1.78. Because the model is self-feeding itself with generated predictions for the previous time step (t-1), the result is more than acceptable.

For the second case, the model was evaluated using the predicted ROP value at the time (t), replacing the value for the time (t-1) in the next time step, but keeping the information for the times (t-2) and (t-3) as provided in the data set. The prediction generated at this point was used to replace the ROP value at the time (t-1), and the previous prediction will replace the value at the time (t-2). Therefore, the only value that remained as in the original data set for all evaluations is ROP at the time (t-3). This process is repeated continuously for the rest of the data. In this case, the model R^2 dropped to 0.77 and the MAE increased to 3.07. This result, while far from ideal, still shows a relatively good performance, as it delivers better predictions than the RF, KNN and ANN models.

For the final test, the same logic used in the second case was applied, but this time all known ROP values were replaced progressively by predicted ROP values. Therefore, the model is predicting ROP and feeding itself with generated values continuously. The R^2 value for this case is equal to -3.17 which means that the predicted values are nowhere near the real values, the MAE result is 13.83, hence, on average a prediction can give a plus or minus 13.83 mph error, this is not acceptable for any ROP predictive model. The plotted results for all cases are located in Appendix B and they show how the model accumulates the errors of each prediction making the last predictions of the ROP model the least accurate ones.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

During the planning and execution stages of this study, many hypotheses regarding the possibility of using a different approach to improve the performance of ML models were analyzed, and objectives set to achieve this result, the conclusions of such work are:

- The most important parameters for the ML predictive models were identified: depth, WOB (surface-downhole), hook load, torque (surface-downhole), rpm, and standpipe pressure.
- A complete well data set was used, the parameters used in this study were selected from over 200 available features, the data was cleaned and prepared for further use.
- An algorithm that calculates the DWOB using surface measurements was successfully implemented with an acceptable mean absolute percentage error.
- Different ML algorithms were tested, but the LSTM model provided the best results predicting ROP in a scenario closer to the real drilling operations.
- The algorithm was tested with three different combinations of the available data, using, surface and downhole measurements, but also surface measurements with the calculated DWOB replacing the SWOB. It was observed that the model provided the best performance with the combination of inputs that used the calculated DWOB.

- It was proved that the performance of a model can be improved by using petroleum engineering knowledge to enhance the quality of the model inputs, instead of treating this just as a purely data-science task.

8.2 Future Work

This study can be used as a stepping stone to develop drilling optimization solutions, but before this, it is advisable to follow the next steps:

- Extensive testing, more wells with different configurations.
- Evaluate and add more effects to corrected the hook load, such as mud hose weight effect, sheave angle, and others not evaluated in this study due to lack of data. This can help to reduce the MAPE for the predicted DWOB.
- Evaluate using the Principal Component Analysis (PCA)[53], against traditional input selection based on petroleum engineering knowledge, and evaluate to determine the best methodology.

With a good and reliable ROP predictive algorithm, it is possible to develop an ROP optimization model. The ROP can be optimized based on different parameters or variables, e.g., consider just WOB, different combinations of WOB-RPM, analyze mechanical specific energy (MSE) [54] which include both parameters, and many other possibilities.

The idea is to let the model predict ROP using different combinations of parameters, and select the combination that provides the best ROP, for this, different boundaries need to be defined, e.g., maximum WOB, maximum RPM, etc. To improve the decision making and additional benchmark could be introduced to determine the best ROP, e.g., via minimizing the MSE. Analyzing a case in which it is desired to optimize ROP using only WOB; after the model is trained for the data at a time (t-1), a range of different WOB values to be introduced into the data set the model will generate different ROP predictions, that can be evaluated using the MSE equation. To illustrate this idea, Figure 8.1 present a *"draft"* of the aimed result.

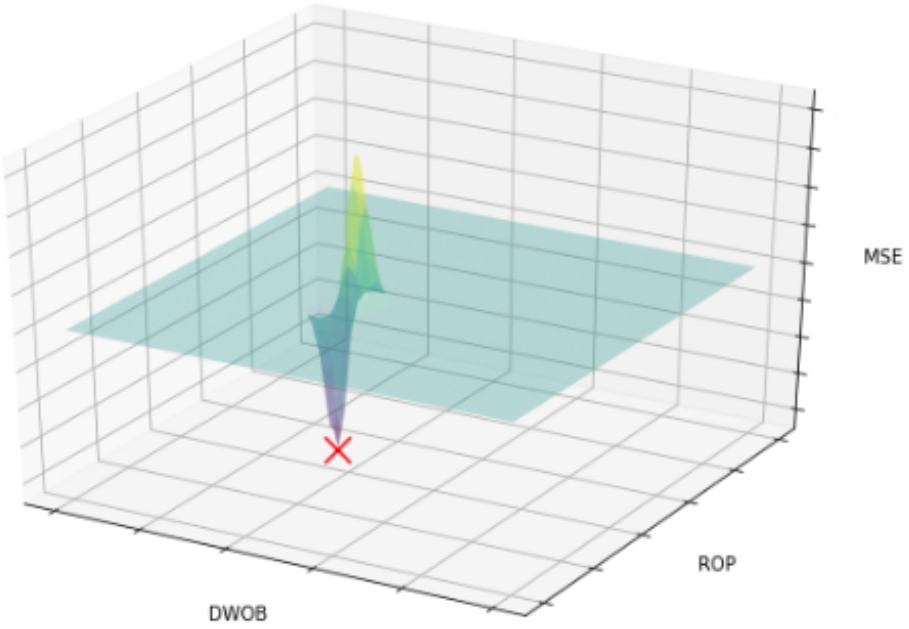

Figure 8.1: Future work, ROP optimization.

This figure locates the DWOB in the x axis, the ROP in the y axis and MSE in the z axis. Theoretically, the model should aim to identify a DWOB value that maximizes ROP and produce the lowest MSE (e.g., the "x" mark on the plot).

References

[1] A.T. Bourgoyne, K.K. Millheim, M.E. Chenevert, and F.S. Young. *Applied Drilling Engineering*. SPE Textbook Series, 1991.

[2] Jojo John Moolayil. *Learn Keras for Deep Neural Networks. A Fast-Track Approach to Modern Deep Learning with Python*. Apress, 2019.

[3] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc., 2017.

[4] Antonio Gulli and Sujit Pal. *Deep Learning with Keras*. Packt Publishing, 2017.

[5] Luís Felipe F.M. Barbosa, Andreas Nascimento, Mauro Hugo Mathias, and João Andrade de Carvalho. Machine learning methods applied to drilling rate of penetration prediction and optimization - a review. *Journal of Petroleum Science and Engineering*, 183, Dec 2019.

[6] David A. Elley, Norbert Meierhoefer, and Michael S. Strathman. Time-based real time drilling operations excellence delivered, Jan 2007. SPE.

[7] Anthony Paul Pink, Lisa Lynch Harrell, Daniel Perez Garcia, John Carrico, and Paul Mackinnon. Integrated drilling optimization and downhole drilling dynamics enables pdc bits to drill in the colorado region of the san juan unconventional gas basin, Jan 2011. SPE.

[8] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[9] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[10] Geir Hareland, Andrew Wu, and Lingyun Lei. The field tests for measurement of downhole weight on bit(dwob) and the calibration of a real-time dwob model, Jan 2014. IPTC.

[11] C.A. Johancsik, D.B. Friesen, and Rapier Dawson. Torque and drag in directional wells-prediction and measurement. 1984.

[12] Halliburton. Wellplan engineering software. https://www.landmark.solutions/WellPlan-Well-Engineering-Software, 2020.

[13] Robert F. Mitchell and Stefan Z. Miska. *Fundamentals of Drilling Engineering*. SPE Textbook Series, 2011.

[14] Z. Wu, G. Hareland, S. M. Loggins, S. Lai, A. Eddy, and L. Olesen. A new method of calculating rig parameters and its application in downhole weight on bit automation in horizontal drilling, Apr 2017. SPE.

[15] W. G. Lesso, H. Laastad, A. Newton, and T. S. Olberg. The utilization of the massive amount of real time data acquired in wired-drillpipe operations. 2008.

[16] T. P. Pink, W. L. Koederitz, A. Barrie, D. R. Bert, and D. C. Overgaard. Closed loop automation of downhole weight on bit improves sliding performance and reduces conservatism in unconventional horizontal well development. 2013.

[17] Cesar Soares and Kenneth Gray. Real-time predictive capabilities of analytical and machine learning rate of penetration (rop) models. *Journal of Petroleum Science and Engineering*, 2019.

[18] A.T. Bourgoyne and F.S. Young. A multiple regression approach to optimal drilling and abnormal pressure detection. 1974.

[19] Ekaterina Wiktorski, Artem Kuznetcov, and Dan Sui. Rop optimization and modeling in directional drilling process. 2017.

[20] H.R. Motahhari, G. Hareland, and J.A. James. Improved drilling efficiency technique using integrated pdm and pdc bit parameters. *Journal of Canadian Petroleum Technology*, 2010.

[21] B. Mantha and R. Samuel. Rop optimization using artificial intelligence techniques with statistical regression coupling. 2016.

[22] C.I. Noshi and J.J. Schubert. Application of data science and machine learning algorithms for rop prediction: Turning data into knowledge. 2019.

[23] L. Breiman. Random forest. pages 5–32, 2001.

[24] Gerard Biau. Analysis of a random forests model. *Journal of Machine Learning Research*, pages 1063–1095, 2012.

[25] O. Sagi and L. Rokach. Ensemble learning: A survey. *WIREs Data Mining and Knowledge Discovery*, 2012.

[26] T. Han, D. Jiang, Q. Zhao, L. Wang, and K. Yin. Comparison of random forest, artificial neural networks and support vector machine for intelligent diagnosis of rotating machinery. *Transactions of the Institute of Measurement and Control*, pages 2681—-2693, 2018.

[27] W.S. McCulloch and W. A Pitts. A logical calculus of the ideas immanent in nervous activity. 1943.

[28] M. M. Amer, A. S. Dahab, and A.-A. H. El-Sayed. An rop predictive model in nile delta area using artificial neural networks. 2017.

[29] D. P. Moran, H. F. Ibrahim, A. Purwanto, and J. Osmond. Sophisticated rop prediction technology based on neural network delivers accurate results. 2010.

[30] J. Han, Y. Sun, and S. Zhang. A data driven approach of rop prediction and drilling performance estimation. 2019.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[32] Gary Smith. *10 - Multiple Regression*, pages 301–337. Academic Press, Boston, Jan 2015.

[33] Rodolfo Bonnin. *Machine Learning for Developers*. Packt Publishing, 2017.

[34] Equinor. Disclose all volve data. https://www.equinor.com/en/news/14jun2018-disclosing-volve-data.html, 2018. Accessed: 2020-05-19.

[35] NPD. Volve field fact pages. https://factpages.npd.no/en/field/pageview/all/3420717, 2020. Accessed: 2020-05-19.

[36] Equinor. Volve field. https://www.equinor.com/en/what-we-do/norwegian-continental-shelf-platforms/volve.html, 2020. Accessed: 2020-05-19.

[37] Attila Nagy. Availability and quality of drilling data in the volve dataset, 2019.

[38] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[39] Mohd Mustafa Al Bakri Abdullah. Filling missing data using interpolation methods: Study on the effect of fitting distribution. *Key Engineering Materials*, 594-595:889–895, 01 2014.

[40] Suranga C. H. Geekiyanage, Dan Sui, and Bernt S. Aadnoy. Drilling Data Quality Management: Case Study With a Laboratory Scale Drilling Rig. Volume 8: Polar and Arctic Sciences and Technology; Petroleum Technology, 06 2018. V008T11A010.

[41] Ronald Deep. *Probability and Statistics with Integrated Software Routines*, page 290. Academic Press, 2005.

[42] Dan Sui. *Drilling Automation and Modeling Compendium*, pages 171–172. 2019.

[43] Sebastian Raschka, David Julian, and John Hearty. *Python: Deeper Insights into Machine Learning*, pages 112–113. Packt Publishing, 2016.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[45] Robello Samuel. Friction factors: What are they for torque, drag, vibration, bottom hole assembly and transient surge/swab analyses? *Journal of Petroleum Science and Engineering*, 73(3):258–266, Sep 2010.

[46] G. R. Luke and H. C. Juvkam-Wold. The determination of true hook-and-line tension under dynamic conditions. *SPE Drilling & Completion*, 8(04):259–264, Dec 1993. SPE.

[47] Eric Cayeux, Hans Joakim Skadsem, and Roald Kluge. Accuracy and correction of hook load measurements during drilling operations, Mar 2015. SPE.

[48] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, USA, 3 edition, 2007.

[49] François Chollet et al. Keras. https://keras.io, 2015.

[50] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[51] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2017.

[52] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Keras Tuner. https://github.com/keras-team/keras-tuner, 2019.

[53] Hervé Abdi and Lynne J. Williams. Principal component analysis. *WIREs Computational Statistics*, 2(4):433–459, 2010.

[54] R. Teale. The concept of specific energy in rock drilling. *International Journal of Rock Mechanics and Mining Sciences Geomechanics Abstracts*, 2(1):57 – 73, 1965.

[55] Ahmed A. Elgibaly, Mohammed Shehata Farhat, Eric W. Trant, and Mohammed Kelany. A study of friction factor model for directional wells. *Egyptian Journal of Petroleum*, 26(2):489–504, Jun 2017.

Appendices

Appendix A

Python Code

A.1 Installed Packages

Name	Version	Name	Version
_tflow_select	2.1.0	oauthlib	3.1.0
absl-py	0.9.0	openpyxl	3.0.4
alabaster	0.7.12	openssl	1.1.1g
astor	0.8.0	opt-einsum	3.1.0
attrs	19.3.0	opt_einsum	3.1.0
babel	2.8.0	packaging	20.4
backcall	0.2.0	pandas	1.0.3
blas	1.0	pandoc	2.9.2.1
bleach	2.1.3	pandocfilters	1.4.2
blinker	1.4.0	parso	0.7.0
brotlipy	0.7.0	pickleshare	0.7.5
ca-certificates	2020.6.24	pip	20.0.2
cachetools	4.1.0	prometheus-client	0.8.0
certifi	2020.6.20	prompt-toolkit	3.0.5
cffi	1.14.0	prompt-toolkit	3.0.5
chardet	3.0.4	protobuf	3.12.3
click	7.1.2	pyasn1	0.4.8
colorama	0.4.3	pyasn1-modules	0.2.7
cryptography	2.9.2	pycparser	2.20
cudatoolkit	10.1.243	pygments	2.6.1
cycler	0.10.0	pyjwt	1.7.1
decorator	4.4.2	pyopenssl	19.1.0
defusedxml	0.6.0	pyparsing	2.4.6
docutils	0.16	pyqt	5.9.2
entrypoints	0.3	pyreadline	2.1
et-xmlfile	1.0.1	pyrsistent	0.16.0
et_xmlfile	1.0.1	pysocks	1.7.1
freetype	2.9.1	python	3.7.7
gast	0.2.2	python-dateutil	2.8.1
google-auth	1.17.2	pytz	2019.3
google-auth-oauthlib	0.4.1	pywin32	227
google-pasta	0.2.0	pywinpty	0.5.7
grpcio	1.27.2	pyyaml	5.3.1

h5py	2.10.0	pyzmq	19.0.1
hdf5	1.10.4	qt	5.9.7
icc_rt	2019.0.0	requests	2.24.0
icu	58.2	requests-oauthlib	1.3.0
idna	2.10	rsa	4.0
imagesize	1.2.0	scikit-learn	0.23.1
importlib-metadata	1.7.0	scipy	1.5.0
intel-openmp	2020.0	seaborn	0.10.1
ipykernel	5.3.0	send2trash	1.5.0
ipython	7.16.1	setuptools	46.1.3
ipython-genutils	0.2.0	sip	4.19.8
jdcal	1.4.1	six	1.14.0
jedi	0.17.1	snowballstemmer	2.0.0
jinja2	2.11.2	sphinx	3.1.2
joblib	0.16.0	sphinxcontrib-applehelp	1.0.2
jpeg	9b	sphinxcontrib-devhelp	1.0.2
jsonschema	3.2.0	sphinxcontrib-htmlhelp	1.0.3
jupyter-client	6.1.5	sphinxcontrib-jsmath	1.0.1
jupyter-core	4.6.3	sphinxcontrib-qthelp	1.0.3
jupyter_client	6.1.5	sphinxcontrib-serializinghtml	1.1.4
jupyter_core	4.6.3	sqlite	3.31.1
keras	2.3.1	tensorboard	2.2.1
keras-applications	1.0.8	tensorboard-plugin-wit	1.6.0
keras-base	2.3.1	tensorflow	2.1.0
keras-gpu	2.3.1	tensorflow-base	2.1.0
keras-preprocessing	1.1.0	tensorflow-estimator	2.1.0
kiwisolver	1.1.0	terminado	0.8.3
libpng	1.6.37	testpath	0.4.4
libsodium	1.0.18	threadpoolctl	2.1.0
m2w64-gcc-libgfortran	5.3.0	tornado	6.0.4
m2w64-gcc-libs	5.3.0	traitlets	4.3.3
m2w64-gcc-libs-core	5.3.0	urllib3	1.25.9
m2w64-gmp	6.1.0	vc	14.1
m2w64-libwinpthread-git	5.0.0.4634.697f757	vs2015_runtime	14.16.27012
markdown	3.1.1	wcwidth	0.2.5
markupsafe	1.1.1	webencodings	0.5.1
matplotlib	3.1.3	werkzeug	0.16.1
matplotlib-base	3.1.3	wheel	0.34.2
mistune	0.8.4	win-inet-pton	1.1.0
mkl	2020.0	wincertstore	0.2
mkl-fft	1.0.15	winpty	0.4.3
mkl-random	1.1.0	wrapt	1.12.1
mkl-service	2.3.0	xlrd	1.2.0
mkl_fft	1.0.15	yaml	0.2.5
mkl_random	1.1.0	zeromq	4.3.2
msgpack	1.0.0	zipp	3.1.0
msgpack-numpy	0.4.4.3	zlib	1.2.11
msgpack-python	1.0.0		
msys2-conda-epoch	20160418		
nbconvert	5.6.1		
nbformat	5.0.7		
notebook	6.0.3		
numpy	1.18.5		
numpy-base	1.18.5		
numpy-devel	1.18.5		
numpydoc	1.1.0		

A.2 Data Cleaning Code

```
#Import parameters to be used in the project

df_time = df[['Time s','Bit Depth (MD) m','On Bottom Status ','Bit on Bottom ','Weight on Bit kkgf', 'Average Hookload kkgf','Average Surface Torque kN.m','MWD Downhole WOB ', 'MWD Downhole Torque ', 'Average Rotary Speed rpm','Mud Flow In L/min','Average Standpipe Pressure kPa','Rate of Penetration m/h']]

df_time.describe() #Evaluate quantity and quality of data

df_time.dtypes #check if the data type is correct for each feature

df_time.corr() #check correlation between parameters
```

Listing A.1: Selecting Parameters (only the relevant part of the code is presented)

```
df_time['Time s'] = pd.to_datetime(df_time['Time s']) #Transform time type from "object" to "datetime"

df_time.sort_values(by=['Time s'])

df_time_s = df_time

df_time_s['Time s'] = df_time_s['Time s'].dt.tz_localize(None)

df_time_s = df_time_s[df_time_s['Bit Depth (MD) m'].between(1400,2900)]

start_date = '2008-07-14'

end_date = '2008-07-21'

mask = (df_time_s['Time s']> start_date) & (df_time_s['Time s'] <= end_date)
df_time_s = df_time_s.loc[mask]
```

Listing A.2: Selection of time-frame used for this study (only the relevant part of the code is presented)

```
df_time_1 = df_time_s

df_time_1.dropna(subset=['On Bottom Status'],inplace=True)

df_time_1 = df_time_1[df_time_1['On Bottom Status'] < 0.2]
```

```
    df_time_1 = df_time_1[df_time_1['Bit on Bottom'] < 0.2]

    df_time_1.drop_duplicates(inplace=True)
```

Listing A.3: Using 'On bottom status' as flag for selecting drilling operations only.(only the relevant part of the code is presented)

```
    df_int = df_time_1

    df_int = df_int[df_int['Bit Depth (MD) m'].between(1900,2400)]

    df_int = df_int.interpolate(method= 'linear')

    df_int.drop_duplicates(inplace=True)

    df_int.dropna(inplace=True)

    fig, axs = plt.subplots(ncols=3,figsize=(17,8)) #checking plots
of three parameters to see interpolation results

    sns.scatterplot(df_int['Bit Depth (MD) m'],df_int['MWD Downhole
WOB'], linewidth=0, alpha=0.3, label="DWOB", ax=axs[0], color='r')
.set(xlim=(1900,2400), ylim=(-2, 10))

    sns.scatterplot(df_int['Bit Depth (MD) m'],df_int['Weight on Bit
kkgf'], linewidth=0, alpha=0.3, label="SWOB",ax=axs[1]).set(xlim
=(1900,2400), ylim=(-2, 10))

    sns.scatterplot(df_int['Bit Depth (MD) m'],df_int['Average
Hookload kkgf'], linewidth=0, alpha=0.3, label="HKLD",ax=axs[2]).
set(xlim=(1900,2400), ylim=(120, 140))

    plt.show()
    plt.close()
```

Listing A.4: Interpolating missing values.(only the relevant part of the code is presented)

```
    Q1_DWOB = df_test['MWD Downhole WOB'].quantile(0.25)
    Q3_DWOB = df_test['MWD Downhole WOB'].quantile(0.95)
    IQR_DWOB = Q3_DWOB - Q1_DWOB

    Q1_SWOB = df_test['Weight on Bit kkgf'].quantile(0.25)
    Q3_SWOB = df_test['Weight on Bit kkgf'].quantile(0.95)
    IQR_SWOB = Q3_SWOB - Q1_SWOB

    Q1_HKLD = df_test['Average Hookload kkgf'].quantile(0.25)
    Q3_HKLD = df_test['Average Hookload kkgf'].quantile(0.90)
```

```
    IQR_HKLD = Q3_HKLD - Q1_HKLD

    Q1_STQ = df_test['Average Surface Torque kNm'].quantile(0.25)
    Q3_STQ = df_test['Average Surface Torque kNm'].quantile(0.95)
    IQR_STQ = Q3_STQ - Q1_STQ

    Q1_DTQ = df_test['MWD Downhole Torque'].quantile(0.25)
    Q3_DTQ = df_test['MWD Downhole Torque'].quantile(0.95)
    IQR_DTQ = Q3_DTQ - Q1_DTQ

    Q1_ROP = df_test['ROP mph'].quantile(0.25)
    Q3_ROP = df_test['ROP mph'].quantile(0.95)
    IQR_ROP = Q3_ROP - Q1_ROP

    Q1_RPM = df_test['Average Rotary Speed rpm'].quantile(0.25)
    Q3_RPM = df_test['Average Rotary Speed rpm'].quantile(0.95)
    IQR_RPM = Q3_RPM - Q1_RPM

    Q1_LXM = df_test['Mud Flow In lpm'].quantile(0.25)
    Q3_LXM = df_test['Mud Flow In lpm'].quantile(0.95)
    IQR_LXM = Q3_LXM - Q1_LXM

    Q1_PPR = df_test['Average Standpipe Pressure kPa'].quantile(0.25)
    Q3_PPR = df_test['Average Standpipe Pressure kPa'].quantile(0.95)
    IQR_PPR = Q3_PPR - Q1_PPR

    df_test = df_test.query('(@Q1_DWOB - 1.5 * @IQR_DWOB) <= `MWD
Downhole WOB` <= (@Q3_DWOB + 1.5 * @IQR_DWOB)')

    df_test = df_test.query('(@Q1_SWOB - 1.5 * @IQR_SWOB) <= `Weight
on Bit kkgf` <= (@Q3_SWOB + 1.5 * @IQR_SWOB)')

    df_test = df_test.query('(@Q1_HKLD - 1.5 * @IQR_HKLD) <= `Average
 Hookload kkgf` <= (@Q3_HKLD + 1.5 * @IQR_HKLD)')

    df_test = df_test.query('(@Q1_STQ - 1.5 * @IQR_STQ) <= `Average
Surface Torque kNm` <= (@Q3_STQ + 1.5 * @IQR_STQ)')

    df_test = df_test.query('(@Q1_DTQ - 1.5 * @IQR_DTQ) <= `MWD
Downhole Torque` <= (@Q3_DTQ + 1.5 * @IQR_DTQ)')

    df_test = df_test.query('(@Q1_ROP - 1.5 * @IQR_ROP) <= `ROP mph`
<= (@Q3_ROP + 1.5 * @IQR_ROP)')

    df_test = df_test.query('(@Q1_RPM - 1.5 * @IQR_RPM) <= `Average
Rotary Speed rpm` <= (@Q3_RPM + 1.5 * @IQR_RPM)')

    df_test = df_test.query('(@Q1_LXM - 1.5 * @IQR_LXM) <= `Mud Flow
In lpm` <= (@Q3_LXM + 1.5 * @IQR_LXM)')

    df_test = df_test.query('(@Q1_PPR - 1.5 * @IQR_PPR) <= `Average
```

```
    Standpipe Pressure kPa` <= (@Q3_PPR + 1.5 * @IQR_PPR)')
55
```

Listing A.5: IQR outlier removal.(only the relevant part of the code is presented)

```
1
2   df_rolling['Depth'] = df_rolling[0].rolling(window=30,center=True
    ).mean()
3
4   df_rolling['SWOB'] = df_rolling[1].rolling(window=30,center=True)
    .mean()
5
6   df_rolling['HKLD'] = df_rolling[2].rolling(window=30,center=True)
    .mean()
7
8   df_rolling['STQ'] = df_rolling[3].rolling(window=30,center=True).
    mean()
9
10  df_rolling['DWOB'] = df_rolling[4].rolling(window=30,center=True)
    .mean()
11
12  df_rolling['DTQ'] = df_rolling[5].rolling(window=30,center=True).
    mean()
13
14  df_rolling['RPM'] = df_rolling[6].rolling(window=30,center=True).
    mean()
15
16  df_rolling['LPM'] = df_rolling[7].rolling(window=30,center=True).
    mean()
17
18  df_rolling['SPP'] = df_rolling[8].rolling(window=30,center=True).
    mean()
19
20  df_rolling['ROP'] = df_rolling[9].rolling(window=30,center=True).
    mean()
21
22  df_rolling.dropna(inplace = True)
23
```

Listing A.6: Moving average noise reduction.(only the relevant part of the code is presented)

A.3 Corrected Field Hookload Calculation

```
1
2   df_hook = df_int[['Depth', 'HKLD','SWOB','DWOB','SPP']]
3
4   df_hook.dropna(inplace = True)
5
6   df_hook.reset_index(drop=True,inplace=True)
7
```

```
8
9     def Wcorr(Fdl,e):
10
11        n=16
12
13        return (Fdl/n)*(1-e**n)/(1-e)
14
15
16    W_corr_1 = Wcorr(df_hook[['HKLD']],0.96)
17
18    W_corr_2 = Wcorr(df_hook[['HKLD']],0.97)
19
20    W_corr_3 = Wcorr(df_hook[['HKLD']],0.98)
21
22    W_corr_4 = Wcorr(df_hook[['HKLD']],0.99)
23
24    HL_a_1 = pd.concat([W_corr_1, W_corr_2, W_corr_3, W_corr_4],axis
      =1)
25
```

Listing A.7: Sheave Effect - Inactive Dead-Line Sheave.(only the relevant part of the code is presented)

```
1
2     df_slips = df_slips[['Slips stat (1=Out,0=In) ', 'Average
      Hookload kkgf']]
3
4     df_slips = df_slips.rename(columns = {'Average Hookload kkgf':'
      Traveling Block Weight kkgf'})
5
6     df_slips.dropna(inplace=True)
7
8     df_slips = df_slips[df_slips['Slips stat (1=Out,0=In) ']<=0]
9
10    df_slips = df_slips[df_slips['Traveling Block Weight kkgf'].
      between(50,100)]
11
12
13    def BLKW(hkld, e):
14
15        n=16
16
17        return (hkld/n)*(1-e**n)/(1-e)
18
19
20    BLKW_corr_1 = BLKW(df_slips[['Traveling Block Weight kkgf']].min
      (),0.96)
21
22    BLKW_corr_2 = BLKW(df_slips[['Traveling Block Weight kkgf']].min
      (),0.97)
23
24    BLKW_corr_3 = BLKW(df_slips[['Traveling Block Weight kkgf']].min
      (),0.98)
```

```
25
26  BLKW_corr_4 = BLKW(df_slips[['Traveling Block Weight kkgf']].min
    ),0.99)
27
28  HL_a2 = pd.concat([BLKW_corr_1, BLKW_corr_2, BLKW_corr_3,
    BLKW_corr_4],axis=1)
29
```

Listing A.8: Static hook load weight.(only the relevant part of the code is presented)

```
1
2  df_pump = df_hook[['Depth', 'SPP']] / 6.89475 # Turn KPa to psi
3
4
5  def Press(P1):
6  Id=4.67
7  return (5.095*10**(-5))*P1*(Id**2)*1.019716 ## To convert from
   KdaN to kkgf
8
9  HL_a_3 = Press(df_pump[['SPP']])
10
```

Listing A.9: Stand pipe pressure effect.(only the relevant part of the code is presented)

```
1
2  Real_HKLD = df_hook[['Depth','HKLD']]# df_hook = df_int[['Depth',
   'HKLD','DWOB','SPP']]
3
4  Real_HKLD['SWOB kips'] = (df_hook[['SWOB']])*2.2046 #conversion
   from tons to kip
5
6  Real_HKLD['DWOB kips'] = (df_hook[['DWOB']])*2.2046 #conversion
   from tons to kip
7
8  Real_HKLD['HKLD-96%'] = (HL_a_1['HKLD_A1 96% kkgf'] - HL_a_2['TB
   Effect 96%'] - HL_a_3['HKLD_A3 kkgf'])*2.2046
9
10 Real_HKLD['HKLD-97%'] = (HL_a_1['HKLD_A1 97% kkgf'] - HL_a_2['TB
   Effect 97%'] - HL_a_3['HKLD_A3 kkgf'])*2.2046
11
12 Real_HKLD['HKLD-98%'] = (HL_a_1['HKLD_A1 98% kkgf'] - HL_a_2['TB
   Effect 98%'] - HL_a_3['HKLD_A3 kkgf'])*2.2046
13
14 Real_HKLD['HKLD-99%'] = (HL_a_1['HKLD_A1 99% kkgf'] - HL_a_2['TB
   Effect 99%'] - HL_a_3['HKLD_A3 kkgf'])*2.2046
15
```

Listing A.10: Real hook load sensitivity analysis.(only the relevant part of the code is presented)

A.4 DWOB Calculation via T&D

```
1
2   def real_hookload(Surveys, BHA, Buoyancy, WOB):
3
4       T_D = np.zeros((len(Surveys),12))
5
6       for i in reversed(range(1,len(T_D))):
7
8           T_D[i,0] = Surveys[i,0] #Depth(m)
9
10          T_D[i,1] = Surveys[i,1] #Inclination[rad]
11
12          T_D[i,2]= Surveys[i,2] #Azimuth[rad]
13
14          T_D[i-1,3] = (Surveys[i,1] + Surveys[i-1,1])/2 #Average
    Inclination ((I1+I2)/2) [rad]
15
16          T_D[i-1,4] = Surveys[i,1] - Surveys[i-1,1] #Inclination
    Diff (I2-I1) [rad]
17
18          T_D[i-1,5]= Surveys[i,2] - Surveys[i-1,2] #Azimuth Diff (
    A2-A1) [rad]
19
20          T_D[i-1,6]= Surveys[i,0] - Surveys[i-1,0] #Depth
    Difference (D2-D1) [m]
21
22          if Surveys[i,0] > (Surveys[-1,0]-BHA[0,0]): #Determine
    the Weight of the String in each cell [N/m]
23
24              T_D[i-1,7]= BHA[0,3] #DD
25          elif Surveys[i,0] > (Surveys[-1,0]-BHA[0,0]-BHA[1,0]):
26
27              T_D[i-1,7]= BHA[1,3] #LWB
28
29          elif Surveys[i,0] > (Surveys[-1,0]-BHA[0,0]-BHA[1,0]-BHA
    [2,0]):
30
31              T_D[i-1,7]= BHA[2,3] #NMDC
32
33          elif Surveys[i,0] > (Surveys[-1,0]-BHA[0,0]-BHA[1,0]-BHA
    [2,0]-BHA[3,0]):
34
35              T_D[i-1,7]= BHA[3,3] #STB
36
37          elif Surveys[i,0] > (Surveys[-1,0]-BHA[0,0]-BHA[1,0]-BHA
    [2,0]-BHA[3,0]-BHA[4,0]):
38
39              T_D[i-1,7]= BHA[4,3] #DC
40
```

```
41            elif Surveys[i,0] > (Surveys[-1,0]-BHA[0,0]-BHA[1,0]-BHA
    [2,0]-BHA[3,0]-BHA[4,0]-BHA[5,0]):
42
43                T_D[i-1,7]= BHA[5,3] #DC2
44
45            elif Surveys[i,0] > (Surveys[-1,0]-BHA[0,0]-BHA[1,0]-BHA
    [2,0]-BHA[3,0]-BHA[4,0]-BHA[5,0]-BHA[6,0      ]):
46
47                T_D[i-1,7]= BHA[6,3] #Jar
48
49            elif Surveys[i,0] > (Surveys[-1,0]-BHA[0,0]-BHA[1,0]-BHA
    [2,0]-BHA[3,0]-BHA[4,0]-BHA[5,0]-BHA[6,0]-BHA[7,0]):
50
51                T_D[i-1,7]= BHA[7,3] #DC3
52
53            elif Surveys[i,0] > (Surveys[-1,0]-BHA[0,0]-BHA[1,0]-BHA
    [2,0]-BHA[3,0]-BHA[4,0]-BHA[5,0]-BHA[6,0]-BHA[7,0]-BHA[8,0]):
54
55                T_D[i-1,7]= BHA[8,3] #HWDP
56
57            else:
58
59                T_D[i-1,7]= BHA[9,3] #DP
60
61            if Surveys[i,0] > 1405:
62
63                T_D[i-1,8]= Friction_factor[0] #OpenHole
64
65            else:
66
67                T_D[i-1,8]= Friction_factor[1] #CasedHole
68
69     Incremental_down = -WOB #Set lower boundary condition [N]
70
71     Incremental_up = -WOB #Set lower boundary condition [N]
72
73     Rotating = -WOB #Set lower boundary condition [N]
74
75     for i in reversed(range(len(T_D))):
76
77         T_D[i,9] = Incremental_down #Incremental force downwards [N]
78         Incremental_down += Buoyancy * T_D[i-1,7] * T_D[i-1,6] * cos(
    T_D[i-1,3]) - T_D[i-1,8]*(((Incremental_down*T_D[i-1,5]*sin(T_D[i
    -1,3]))**2+(Incremental_down*T_D[i-1,4]+Buoyancy * T_D[i-1,7] *
    T_D[i-1,6]*sin(T_D[i-1,3]))**2)**.5)
79
80         T_D[i,10] = Incremental_up #Incremental force upwards [N]
81         Incremental_up += Buoyancy * T_D[i-1,7] * T_D[i-1,6] * cos(
    T_D[i-1,3]) + T_D[i-1,8]*(((Incremental_up*T_D[i-1,5]*sin(T_D[i
    -1,3]))**2+(Incremental_up*T_D[i-1,4]+Buoyancy * T_D[i-1,7] * T_D[
    i-1,6]*sin(T_D[i-1,3]))**2)**.5)
```

```
82
83          T_D[i,11] = Rotating #Hookload when Rotating [N]
84          Rotating += Buoyancy * T_D[i-1,7] * T_D[i-1,6] * cos(T_D[i
    -1,3])
85
86      return Rotating/4448.22162 #stores result in [kips]
87
```

Listing A.11: TD Johancsik model for Hookload calculation.(only the relevant part of the code is presented)

```
1
2       def int_svy(Y, Z):
3
4       A = np.zeros((len(Y),3)) #creating a table with the dimension of
    the surveys
5
6       for i in reversed(range(len(Y))):
7
8           if  Y[i-1,0] <= Z <= Y[i,0]: # finds the position of the bit
    between the surveys
9
10              x = Y[i,0] - Z
11
12              y = Z - Y[i-1,0]
13
14              if x < y: #choose the closest value (upper-lower) and
    equates the inclination and azimuth [rad]
15
16                  A[i,0] = Z
17
18                  A[i,1] = Y[i,1]
19
20                  A[i,2] = Y[i,2]
21
22              else:
23
24                  A[i,0] = Z
25
26                  A[i,1] = Y[i-1,1]
27
28                  A[i,2] = Y[i-1,2]
29
30          elif Z <= Y[i,0]: #if the survey depth is bigger than the bit
     position, the row is eliminated
31
32              A = np.delete(A, A[i,0], axis=0)
33
34          else: #if the bit depth is bigger than the other surveys
    depth, copy the survey depth, inclination and azimuth
35
36              A[i,0] = Y[i,0]
37
```

```
38            A[i,1] = Y[i,1]
39
40            A[i,2] = Y[i,2]
41
42        return A
43
```

Listing A.12: Trajectory of the well depending on the bit position.(only the relevant part of the code is presented)

```
1    D = np.zeros((len(N),3))
2
3    for i in reversed(range(len(D))): #iterates within the table in
     reverse order
4
5        D[i,0] = N[i,0] #Set the value of the bit depth
6
7        D[i,1] = N[i,1] #Initially set the value of the WOB with the
     SWOB, which will be changed, if we don't meet the evaluation
     criteria
8        Traject = int_svy(Y,D[i,0]) #Run the trajectory function for
     each bit depth
9
10       Rot = real_hookload(Traject, BHA, Buoyancy, D[i,1]) #Run the
     T&D function for each bit depth, returns a Hooklad
11
12       D[i,2] = Rot #Stores the value of the Hook load for the
     condition stated above [kips]
13
14       while abs(D[i,2]-C[i,1])> .5 : #While the difference between
     the calculated and corrected value is bigger than .5 [kips]
15
16           if (D[i,2]-C[i,1]) > 0: #Calculated value by T&D is
     bigger than the corrected value
17
18               D[i,1] += 4448.22162/4 # Increase WOB 0.25 [kips]
19           else:
20
21               D[i,1] -= 4448.22162/4 # Decrease WOB 0.25 [kips]
22       Rot = real_hookload(Traject, BHA, Buoyancy, D[i,1])
23
24       D[i,2] = Rot
25
```

Listing A.13: Iterate and obtain values for various depths and different DWOB's.(only the relevant part of the code is presented)

A.5 Random Forest and K-Nearest Neighbors Methods

```
    samples = np.zeros((len(df2),6))

    results = np.zeros((len(df2)))

    for i in range(len(samples1)):

        samples1[i,0] = df2[i,0] #Depth

        samples1[i,1] = df2[i,4] #DWOB

        samples1[i,2] = df2[i,2] #HKLD

        samples1[i,3] = df2[i,5] #DTQ

        samples1[i,4] = df2[i,6] #RPM

        samples1[i,5] = df2[i,8] #SPP

        results1[i] = df2[i,9] #ROP

```

Listing A.14: Parameters selected for ensemble model and KNN.(only the relevant part of the code is presented)

```
    X1 = samples

    y1 = results

    param_list1 = {"n_estimators": [10,30,50,70], "max_depth":
[10,20,30]}

    RFR_model = RandomForestRegressor(random_state = 0)

    grid_search1 = GridSearchCV(RFR_model, param_grid=param_list1 ,
cv=10, scoring='r2' , n_jobs=4).fit(X1,y1)

    print("Best parameters RandomForest:", grid_search1.
best_estimator_)

    print("Best score RFR:", grid_search1.best_score_)

```

Listing A.15: Hyperparameter tuning RF.(only the relevant part of the code is presented)

```
    param_list2 = {"n_neighbors":[3,5,11],"weights":['distance','
uniform'],"algorithm":['auto', 'ball_tree', 'kd_tree', 'brute']}

```

```
4    KNR_model = KNeighborsRegressor()
5
6    grid_search2 = GridSearchCV(KNR_model, param_grid=param_list2 ,
     cv=10, scoring='r2' , n_jobs=4).fit(X1,y1)
7
8    print("Best parameters KNR:", grid_search2.best_estimator_)
9
10   print("Best score KNR:", grid_search2.best_score_)
11
```

Listing A.16: Hyperparameter tuning KNN.(only the relevant part of the code is presented)

```
1
2    #Create train and test dataset with an 60:40 split
3
4    x_train, x_test, y_train, y_test = train_test_split(samples,
     results,test_size=0.4,random_state = 0, shuffle = True)
5
6    #Further divide training dataset into train and validation
     dataset with an 80:20 split
7    x_train, x_val, y_train, y_val = train_test_split(x_train,y_train
     ,test_size=0.2,random_state = 0, shuffle = True)
8
```

Listing A.17: Data "random sampling".(only the relevant part of the code is presented)

```
1
2    #Create train and test dataset with an 60:40 split
3
4    x_train, x_test, y_train, y_test = train_test_split(samples,
     results,test_size=0.4,random_state = 0, shuffle = False)
5
6    #Further divide training dataset into train and validation
     dataset with an 80:20 split
7    x_train, x_val, y_train, y_val = train_test_split(x_train,y_train
     ,test_size=0.2,random_state = 0, shuffle = False)
8
```

Listing A.18: Data "sequential sampling".(only the relevant part of the code is presented)

```
1
2    def evaluate_model(rf, x_train, x_val, x_test, y_train, y_val,
     y_test):
3
4        rf.fit(x_train, y_train)
5
6        validation = rf.predict(x_val)
7
8        predictions = rf.predict(x_test)
9
10       plt.figure(figsize=(6,6))
11
```

```
12        e_val = np.abs(validation - y_val)
13
14        errors = np.abs(predictions - y_test)
15
16        print('Mean Absolute Error_ val:', round(np.mean(e_val), 10),
    'units')
17
18        print('Mean Absolute Error:', round(np.mean(errors), 10), '
    units')
19
20        print(f'MSE_val = {mean_squared_error(y_val,validation)}')
21
22        print(f'MSE = {mean_squared_error(y_test,predictions)}')
23
24        print(f'R2_val = {r2_score(y_val,validation)}')
25
26        print(f'R2 = {r2_score(y_test,predictions)}')
27
28        plt.scatter(y_test,predictions, s=2, c="black",label="Data
    sample", alpha=1)
29
30        plt.plot([0,np.max([y_test,predictions])],[0,np.max([y_test,
    predictions])],c="red", linewidth=1, linestyle="-.", label="
    perfect prediction")
31
32
33        plt.xlabel("Truth")
34
35        plt.ylabel("Model")
36
37        plt.legend()
38
39        plt.grid()
40
41        plt.text(0.061, 0.001, f'$R^2$ = {np.round(r2_score(y_test,
    predictions),3)}')
42
43        return
44
```

Listing A.19: Data evaluation code.(only the relevant part of the code is presented)

```
1
2     rf = RandomForestRegressor(n_estimators = 70, random_state = 0,
    max_depth=30)
3
4     evaluate_model(rf, x_train, x_val, x_test, y_train, y_val, y_test
    )
5
```

Listing A.20: Random Forest Regressor.(only the relevant part of the code is presented)

```
rf = KNeighborsRegressor(algorithm='brute', leaf_size=30, metric='minkowski', n_neighbors=3, weights='distance')

evaluate_model(rf, x_train, x_val, x_test, y_train, y_val, y_test)
```

Listing A.21: K-Nearest Neighbors Regressor.(only the relevant part of the code is presented)

A.6 Artificial Neural Networks

```
scaler=MinMaxScaler(feature_range=(0,1))

df_values = df_int.values #numpy array

df_values_scaled = scaler.fit_transform(df_values)

dataset = pd.DataFrame(df_values_scaled)
```

Listing A.22: Feature Scaling.(only the relevant part of the code is presented)

```
X = dataset.iloc[:,:7].values #Features: 'Depth','Hookload kkgf', 'Downhole Weight on Bit kkgf', 'Average Downhole Torque kNm', 'Average Rotary Speed rpm', 'Average Standpipe Pressure kPa'.

y = dataset.iloc[:,7:8].values #Labels: 'ROP mph'

#Create train and test data set with an 60:40 split
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.4,random_state = 0,shuffle = True)

#Further divide training data set into train and validation data set with an 80:20 split
x_train, x_val, y_train, y_val = train_test_split(x_train,y_train,test_size=0.2,random_state = 0, shuffle = True)
```

Listing A.23: Data "random sampling".(only the relevant part of the code is presented)

```
X = dataset.iloc[:,:7].values #Features: 'Depth','Hookload kkgf', 'Downhole Weight on Bit kkgf', 'Average Downhole Torque kNm', 'Average Rotary Speed rpm', 'Average Standpipe Pressure kPa'.

y = dataset.iloc[:,7:8].values #Labels: 'ROP mph'
```

```
#Create train and test data set with an 60:40 split
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.4,random_state = 0,shuffle = False)

#Further divide training data set into train and validation data set with an 80:20 split
x_train, x_val, y_train, y_val = train_test_split(x_train,y_train,test_size=0.2,random_state = 0, shuffle = False)
```

Listing A.24: Data "sequential sampling".(only the relevant part of the code is presented)

```
model = Sequential()

model.add(Dense(30,input_dim=x_train.shape[1],activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(30,activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(1,activation='relu'))

model.compile(optimizer='adam',loss="mse",metrics=["mean_squared_error", rmse, r_square])

# callbacks
stop = EarlyStopping(monitor='val_loss', patience=5)

result = model.fit(x_train,y_train, validation_data=(x_val,y_val), epochs=100, batch_size=32, callbacks=[stop])

y_pred = model.predict(x_test)
```

Listing A.25: Artificial Neural Network Architecture.(only the relevant part of the code is presented)

```
print("Mean absolute error (MAE): %f" % sklearn.metrics.mean_absolute_error(y_test,y_pred))
print("R square (R^2): %f" % sklearn.metrics.r2_score(y_test, y_pred))
```

Listing A.26: Model results.(only the relevant part of the code is presented)

A.7 Long Short Term Memory

```
scaler=MinMaxScaler(feature_range=(0,1))

df_values = df_calc_downhole_test.values #data set used for model

df_values_scaled = scaler.fit_transform(df_values)

df_scaled = pd.DataFrame(df_values_scaled)

df_scaled.head()
```

Listing A.27: Feature Scaling.(only the relevant part of the code is presented)

```
# split into training and testing sets

values = reframed1.values

test_pct = 0.4*len(reframed1) #percentage of data assigned for testing

train = values[:round(len(reframed1)-test_pct), :]

test = values[round(len(reframed1)-test_pct):, :]
```

Listing A.28: Data selected for LSTM model.(only the relevant part of the code is presented)

```
# split into input and outputs

samples_n = n_steps * n_features

x_train, y_train = train[:, :samples_n], train[:, -1]

x_test, y_test = test[:, :samples_n], test[:, -1]

print(x_train.shape, len(x_train), y_train.shape)
```

Listing A.29: Splitting the data.(only the relevant part of the code is presented)

```
# reshape input to be 3D [samples, timesteps, features]

x_train = x_train.reshape((x_train.shape[0], n_steps, n_features))
```

```
x_test = x_test.reshape((x_test.shape[0], n_steps, n_features))

print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

Listing A.30: Reshaping the data.(only the relevant part of the code is presented)

```
model = Sequential()

model.add(LSTM(30, input_shape=(x_train.shape[1], x_train.shape
[2])))

model.add(Dropout(0.2))

model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')

# callbacks

stop = EarlyStopping(monitor='val_loss', patience=5)

# fit network

history = model.fit(x_train, y_train, epochs=100, batch_size=32,
validation_data=(x_test, y_test), verbose=2, shuffle=False,
callbacks=[stop])
```

Listing A.31: LSTM model.(only the relevant part of the code is presented)

```
# prediction

ypred = model.predict(x_test)

x_test = x_test.reshape((x_test.shape[0], n_steps*n_features))

# forecast inverse_transform
ypred_scaled = np.concatenate((x_test[:, -6:], ypred), axis=1)

ypred_scaled = scaler.inverse_transform(ypred_scaled)

ypred_scaled = ypred_scaled[:,6]

ypred_scaled.shape

# test inverse_transform
```

```
y_test = y_test.reshape((len(y_test), 1))

y_scaled = np.concatenate((x_test[:, -6:], y_test), axis=1)

y_scaled = scaler.inverse_transform(y_scaled)

y_scaled = y_scaled[:,6]

y_scaled.shape

```

Listing A.32: Invert scaling information.(only the relevant part of the code is presented)

```

print("Mean absolute error (MAE): %.4f" % sklearn.metrics.
mean_absolute_error(y_scaled,ypred_scaled))

print("R square (R^2): %.4f" % sklearn.metrics.r2_score(y_scaled,
ypred_scaled))

```

Listing A.33: Model results.(only the relevant part of the code is presented)

```

f, ax = plt.subplots(figsize=(15, 7))

plt.plot(df_int['Depth'], df_int['ROP'],c='r',label='Measured ROP
')

plt.plot(inv_x[:, 0], ypred_scaled[:],c='b',label='Predicted ROP'
)

ax.text(1910, 15, f'$R^2$ = {np.round(r2_score(y_test,ypred),3)}'
)

ax.set_xlabel('Depth [m]')

ax.set_ylabel('ROP [m/h]')

ax.xaxis.set_ticks_position('top')

ax.xaxis.set_label_position('top')

plt.legend()

plt.grid()

plt.show()

```

Listing A.34: Plotting the model results.(only the relevant part of the code is presented)

```
def lstm(test):

    reshape = test.reshape((1, 3, 7)) #Reshape input array
    hat1 = model.predict(reshape) #Predict ROP

    return hat1

```

Listing A.35: Model evaluation function.(only the relevant part of the code is presented)

```
# predicting ROP for (t) using (t-1) "predicted" value and
knowing ROP (t-2) and (t-3)

Test1 = np.zeros((len(x_test),21))

hat1 = np.zeros((len(x_test),1))

for i in range(len(Test)):

    if i < 1:

        Test1[i] = np.concatenate((x_test[i, :-1], y_train[[-1]])
)


        hat1[i] = lstm(Test1[i])

    else:

        Test1[i] = np.concatenate((x_test[i, :-1], hat[i-1]))

        hat1[i] = lstm(Test1[i])

```

Listing A.36: LSTM performance test #1.(only the relevant part of the code is presented)

```
# predicting ROP for (t) using (t-1) and (t-2) "predicted" values
and knowing ROP (t-3)

Test2 = np.zeros((len(x_test),21))

Input2 = np.zeros((len(x_test),21))

hat2 = np.zeros((len(x_test),1))

for i in range(len(Test)):

    Test2 [i] = x_test[i, :]

    if i < 1:
```

```
            Input2[i] = np.concatenate((Test2[i, :-1], y_train[[-1]])
)

            hat2[i] = lstm(Input2[i])
        else:

            Input2[i] = np.concatenate((x_test[i, :7], Input2[i-1,
14:],x_test[i, 14:-1],hat2[i-1]))

            hat2[i] = lstm(Input2[i])

```

Listing A.37: LSTM performance test #2.(only the relevant part of the code is presented)

```

    # predicting ROP for (t) using (t-1), (t-2) and (t-3) "predicted"
    values

    Test3 = np.zeros((len(x_test),21))

    Input3 = np.zeros((len(x_test),21))

    hat3 = np.zeros((len(x_test),1))

    for i in range(len(Test)):

        Test3 [i] = x_test[i, :]

        if i < 1:
            Input3[i] = np.concatenate((Test3[i, :-1], y_train[[-1]])
)

            hat3[i] = lstm(Input3[i])
        else:

            Input3[i] = np.concatenate((x_test[i, :7], Input3[i-1,
14:],x_test[i, 14:-1],hat3[i-1]))

            hat3[i] = lstm(Input3[i])

```

Listing A.38: LSTM performance test #3.(only the relevant part of the code is presented)

Appendix B

LSTM Performance Test Plots

B.1 LSTM Test #1

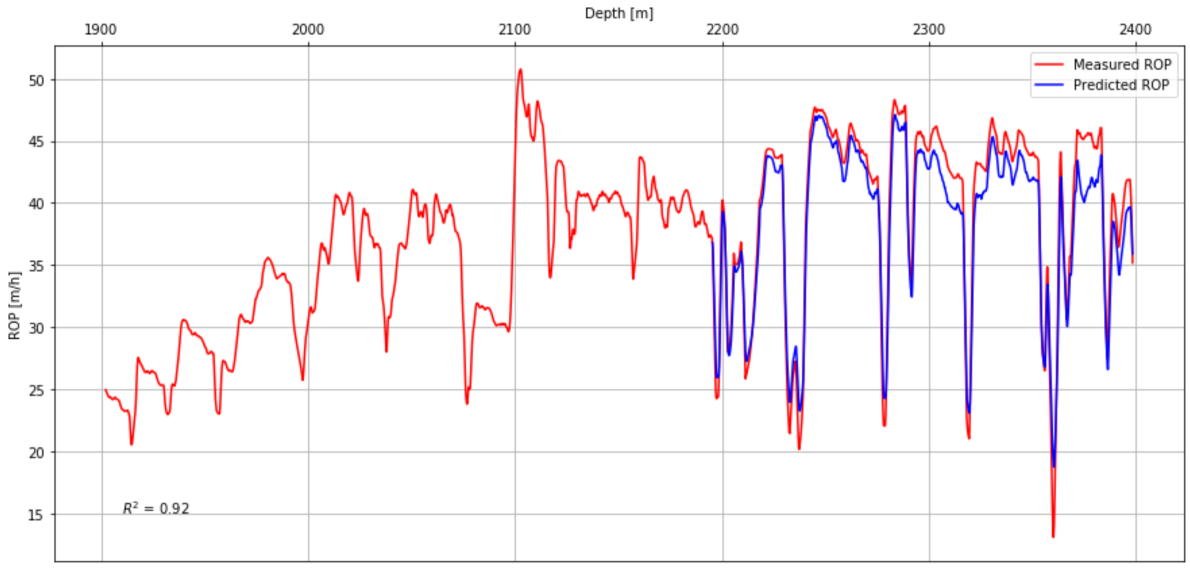

Figure B.1: LSTM model Test #1 results.

B.2 LSTM Test #2

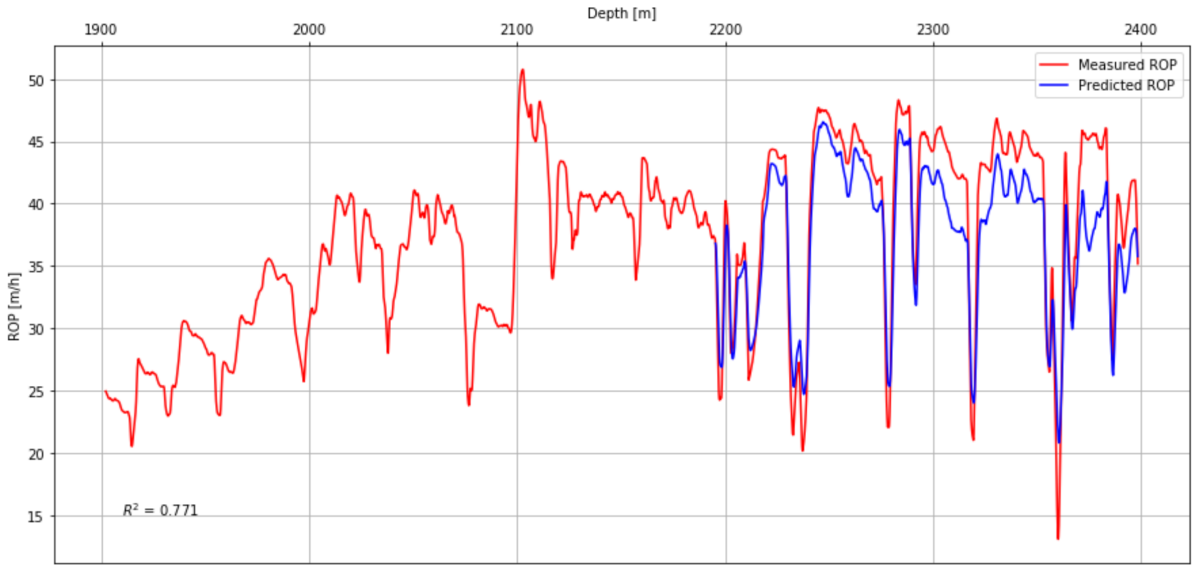

Figure B.2: LSTM model Test #2 results.

B.3 LSTM Test #3

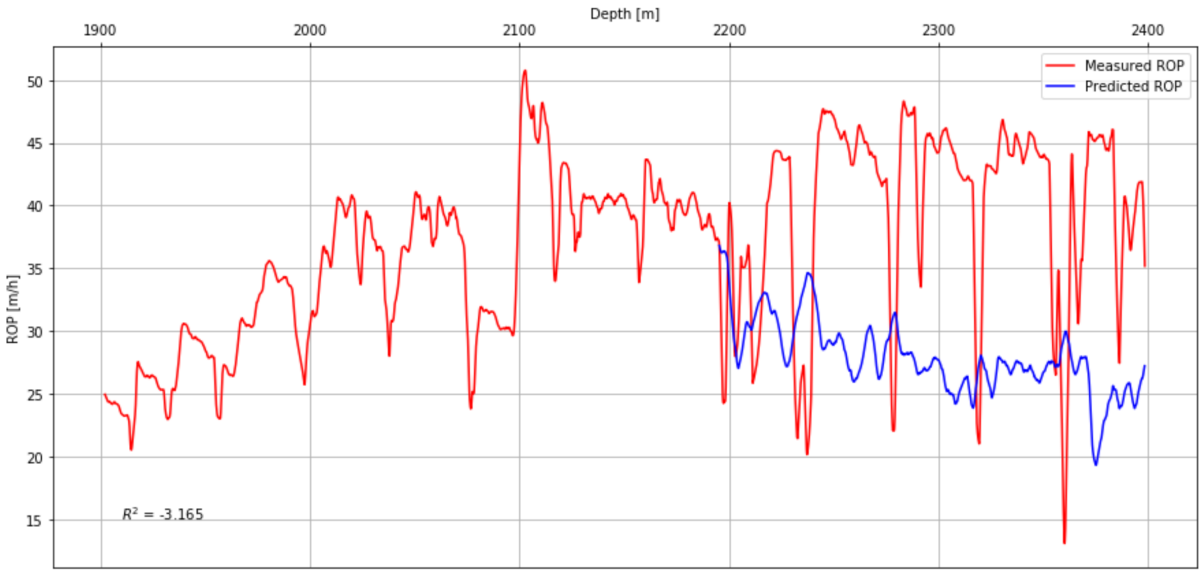

Figure B.3: LSTM model Test #3 results.