FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation:

Spring semester, 2022

MSc. Petroleum Engineering / Drilling and Well Engineering

Open

Author: Carol Alexandra Ladino García

.....
Carol Alexandra Ladino

Programme coordinator:
Øystein Arild

Faculty Supervisor(s):
UiS – Prof. Dan Sui
UiS – PhD. Jie Cao

Thesis title:
FIELD-SCALE GENERALITY OF THE MACHINE LEARNING MODELS

Credits (ECTS): 30 ECTS

Keywords:

Machine Learning, Rate of Penetration, Regression, Drilling.

Pages: 68

+ appendix: 59

Stavanger, 15th June 2022

Abstract

Drilling performance is directly related to fundamental aspects such as drilling variables that can affect the performance of the operation, the well stability, efficiency of drilling equipment, use of new technologies and operational parameters. Approximately 30% of the total time of construction of a well corresponds to the time rotating and sliding, in this order of ideas the optimization of the rate of penetration "ROP" has a direct impact on time and cost reduction. This reduction has as an added value: making viable economically the drilling campaigns and development of the fields. That is why one of the main objectives of the operating companies is to reduce the total time in which the true depth is reached, to reduce the costs of the operation but without affecting the main objective of the well drilling operations. To consider a good performance of the operations, many factors are involved being the rate of penetration one of the most important, without leaving behind the HSE performance, the stability of the well, integrity of the formation and final cost of the project.

On the other hand, the data driven machine learning models are significantly different in conception process from physics-based models. The physic-based models try to understand the problem and propose proper models resembling he problem under certain assumptions and constraints. They seek methodology to reasonably determine the results given input. On the contrary, the machine learning models consider little about the details of the problem but train a working model mapping directly from inputs (knowns) to outputs (unknowns) through a black box of neutral networks. After that, researchers try to unveil the black box to analyze what happens there and enlighten what knowledge learned from there as to improve the model interpretability.

Along the project, the relevant parameters for the machine learning predictive model were chosen considering the correlation and their dependency to ROP, the model was fed up, trained, and tested with the data set of one well and its accuracy was improved using hyperparameter tunning. After it, the algorithm was tested with five different data sets keeping constant the chosen parameters. Among them it was possible to determine that the Random Forest, Gradient Boosting and K Neighbors regressor were the ones with the highest coefficient of determination and the best performance, considering that any model in general can be improved reckoning also the importance of the learned lessons or field experience from petroleum engineering knowledge to enhance the quality of the inputs and the outputs of the model.

Acknowledgments

I would like to express my deepest gratitude to the University of Stavanger, especially to Dan Sui and Jie Cao for their invaluable support, patience, and feedback given since the beginning of this project. Additionally, I want to say thank you to each one of the professors and professionals who have been part of the development of my master's degree, since thanks to them, my professional knowledge has been strengthened.

I could not have undertaken this journey without the blessings of God, all the love of my mother, Gilma, to whom I owe each of the projects that I have completed throughout my life, to Chris for making me feel loved and at home.
I am also grateful to my family and friends, Fefita, Sergio, Yei and Pau for all this emotional and moral support during this adventure, because despite not being physically present, their love and time were key to not giving up. Their belief in me has kept my spirits and motivation high during this process.

I would be remiss not to mention that my master's degree would not have been possible without the motivation, example and experience gained at Equion Energia since I was lucky to have around me excellent and lovely professionals who inspired me to grow up.

Last but not least, thanks to all people I had the joy of meeting during this stage as a student, who taught me to be stronger, independent, and to value true friendship.

"Every journey, however long, starts with a single step"
Carol Ladino. 😊

List of figures

List of tables

List of equations

Nomenclature

ANN	Artificial Neural Network
API	Application Programming Interface
BHA	Bottom Hole Assembly
BOP	Blow Out Preventers
CSV	Comma-Separated Value
CV	Cross Validation
DWOB	Downhole Weight On Bit
DT	Decission Tree
DTQ	Downhole Torque
ECD	Equivalent Circulating Density
GB	Gradient Boosting
GR	Gamma Ray
HPHT	High Pressure High Temperature
i	Index
IADC	Internal Association of Drilling Contractors
ID	Inside Diameter
KNN	K-Nearest Neighbors
LWD	Logging While Drilling
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPD	Mean Absolute Percentage Deviation
MAPE	Mean Absolute Percentage Error
MD	Measure Depth
MedAE	Median Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
MSLE	Mean Squared Logarithmic Error
MWD	Measuring While Drilling
NN	Neural Network
OD	Outside Diameter
OLS	Ordinary Least Square
PDC	Polycrystalline diamond compact
R2	Coefficient of Determination
RF	Random Forest
RMSE	Root Mean Squared Error
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

Chapter 1.
Introduction

1.1. Background, Motivation, and Challenge

Drilling performance is directly related to fundamental aspects such as drilling variables that can affect the performance of the operation, the well stability, efficiency of drilling equipment, use of new technologies and operational parameters. Approximately 30% of the total time of construction of a well corresponds to the time rotating and sliding, in this order of ideas the optimization of the rate of penetration "ROP" has a direct impact on time and cost reduction. This reduction has as an added value: making viable economically the drilling campaigns and development of the fields.

According to the above, it can be said that one of the main objectives of the operating companies is to reduce the total time in which the TD is reached, to reduce the costs of the operation but without affecting the main objective of the well drilling operations. To consider a good performance of the operations, many factors are involved being the rate of penetration one of the most important, without leaving behind the HSE performance, the stability of the well, integrity of the formation and final cost of the project.

Many theoretical and practical studies have been developed previously giving a technical way of what happens operationally, being now, an investigation with so much future since different models with different variables can be developed in order to meet the final objective, to achieve an increasingly accurate prediction of ROP. Lately, the interest in data science and machine learning has been increasing and many studies, not only in the oil and gas industry, have focused their methodology on model development with the aim of finding adequate solutions to different problems.

The motivation of this study is to perform and implement a code to select the relevant parameters from a general data set and to generate a model able to predict ROP and compare these values to the real data obtained from the drilled well.

1.2. Objectives and Scopes

The main objective of the present study is to compare and identify the parameters that can affect the accuracy of ROP prediction modelling process. Taking into account that the general objective can address too many years of research, conditions and methods, the specific objectives of this project are considered below:

- Understand all the parameters and factors that can be involved or affect the ROP behavior during the drilling phase of a well.
- Identify the importance and relevancy of some of the parameters measured during drilling operations.
- Use a well data set to train and test the model and to predict ROP, choosing an appropriate data set that contains relevant and consistent information.
- Make a comparison among all methods considering their accuracy.
- Build the model and verify the prediction accuracy with never-before-seen data taken from other drilled wells, predicted ROP vs actual ROP.

The first objective is very important since it is relevant to understand everything that can affect the ROP of the wells and factors that can become relevant in their behavior, after being clear about the above, a decision can be made on its relevance and importance.
Currently, service companies have been developing high-tech tools that allow having as much data as possible, both in memory and in real time, which help to make decisions, but it is important to determine and classify which of the measured parameters are relevant to have a good prediction and optimization of ROP.

One of the wells, the well *N-NA_F-9_Ad*, was selected with the aim of training and testing the model so that it could later be implemented in the other wells to be studied. Finally, and as the last objective, it was defined which of the techniques evaluated are the ones that deliver a better prediction of the ROP having as baseline the actual information.

1.3. Tool Box and Structure

The base of this project is coding and building a machine learning model that helps to analyze ROP behavior and a way to predict. Jupyter Notebook [1] will be the application of choice as it is user-friendly, handles the selected programming language Python [2], and is helpful to analyze the results from the data.
Several packages were installed and used as a tool to develop the coding work related to the project, see Appendix A.

One of the main steps was to review, collect and select the input data. Collecting the data depends on the number of sensors run in the BHA and the quality and continuity of the information given by them since the accuracy and quality of the built model depends on that. Along the Chapter 4, a Machine Learning model was developed using random forest algorithm where the inputs were identified, data set was split into testing and training data set and a hyperparameter tuning was performed in order to improve the accuracy and to reduce the coefficient of determination. Finally, the Machine Learning model was implemented to verify its accuracy against never-before-seen data to make an analytical comparison of the results explained in Chapter 5.

Chapter 2.
Theoretical background

2.1. Drilling Operations

To understand the topics that are going to be developed throughout the project, performing a short briefing can be considered very important. Despite not being the first step when it comes to exploration and exploitation of oil basins, drilling a well is one of the most important steps during the productive life of a field.

2.1.1. Drilling Rig Components – Surface

A drilling rig, no matter if it is offshore or onshore operations, consists of five (5) main components:

2.1.1.1. Rotary System

The term rotary comes from the physical movement of the drill string and bit, which applies a rotary shear force to the rock at the bottom of the hole. The rotation can be applied on the surface to the entire string or by a downhole motor to a part of the assembly bottom (Bottom hole assembly, BHA). The drill string consists of a steel pipe that conducts the drilling fluid inside it to the drilling bit. This pipe or string is a mix of standard drill pipe, heavyweight drill pipe, drill collars, or pipes of different diameters and caliber.

It is also very important to have an efficient rotation system, that mainly includes a swivel, Kelly rotary drive, and rotary table. The working principle of the rotation system is the Kelly, which is connected to the drill pipe driven by the rotary table and then the whole drilling string can be rotated for drilling the well [3].

2.1.1.2. Circulation System

The drilling fluid is normally called drilling mud and it is the first barrier when talking about well control operations. The drilling fluid is stored in tanks or pools, and from there the mud can be pumped through the standpipe to the swivel where it enters into the Kelly or the top drive, then down the drill string to the bit. After it, the fluid is recirculated returning to the surface through the annulus. When fluid is returned to the surface, the mud is passed through various elements of the solid control system, such as screens, desanders, or centrifuges in order to remove all drilling cuttings before returning to being treated with certain chemicals and sending it back to mud tanks to complete de cycle.

2.1.1.3. Power System

The power system basically provides all the necessary power to carry out the drilling work. Normally this power is generated from local combustion generators.

2.1.1.4. Well Control and Monitoring System

Formations in the shallow section of a well are generally isolated by a casing which must be cemented in place. The annular space through which the mud returns to the surface is now the space between the inside of the casing and the outside of the drill string. To this liner preventer valves or BOPs (Blow Out Preventers) are connected, a series of valves and seals that can be used to shut off the annulus or the full head of the well in order to control high bottom-hole pressures when they occur. If there is a sudden pressure change in the well which pushes the formation fluid up to the surface, BOP will be closed the seal the well from a blowout [4].

2.1.1.5. Hoisting System

The rig has a system that has the responsibility of running in the hole (RIH) or pulling out of the hole (POOH) the drill string or casing, this system is called the hoisting system. It consists of the derrick, tackle and block system, and deadline anchor system. Tackles and blocks do the vertical movement of the pipe. The deadline anchor mainly helps the replacement of the drilling line when it was subjected to wearings.

2.2. Downhole Components

The configuration and setting of the BHA is a very important thing when talking about designing a drilling well program since the tools are going to be run can improve or affect the drilling performance, give information about the well, or simply have a successful operation. The importance of having downhole measurement tools is a reliable source of information to make decisions on the well site.

Measure While Drilling (MWD) systems allow the driller to gather and transmit information from the bottom of the hole back to the surface without interrupting normal drilling operations. This information can include some of the parameters that

are going to be analyzed through the development of the project, which are directional deviation data, data related to the petrophysical properties of the formations, and drilling data such WOB and torque. [5] The data is transmitted through the mud column in the drill string, to the surface, this transmission system is called mud pulse telemetry. All those tools are in constant improvement and can provide directional information, drilling parameters, and geological data. The latter tools are generally referred to as Logging While Drilling (LWD) and Wired Drill Pipe (WDP).

According to Lesso et al., *"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, drilling mechanics and drilling dynamics [6].

Some of the parameters or measurements that are going to be analyzed through the project include weight on bit (WOB), standpipe pressure (SPP), rate of penetration (ROP), average rotary speed or revolutions per minute (RPM), mudflow volume, mudflow density, the diameter of the hole, average hook load, measure depth, hope depth and gamma-ray (GR).

2.3. Drilling Performance

The operations involved in drilling a well can be illustrated by considering the sequence of events involved in drilling the well shown in ***Figure 1***.

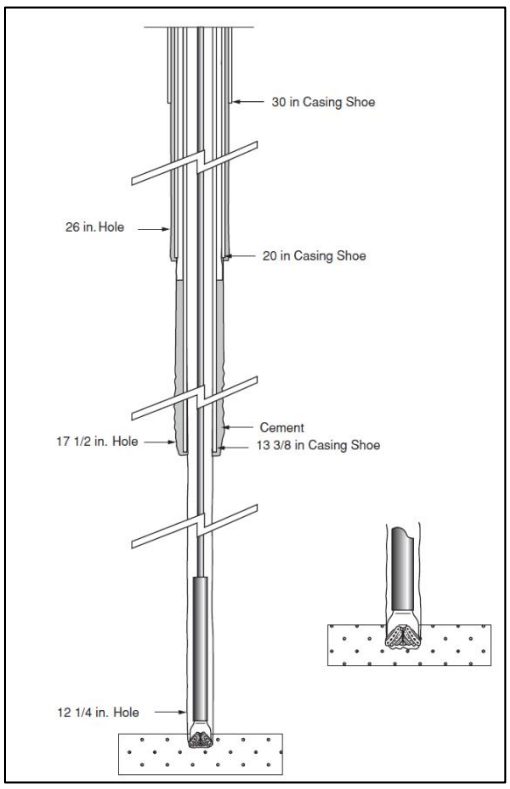

Figure 1. Typical hole and casing size *[5]*.

Drilling a well normally starts drilling a 36” hole using an 18 ½” bit and 36” hole opener. The hole is drilled with seawater, with the drilled cuttings settling onto the seabed because there is no riser or BOP installed at this stage. Then a 30” casing is run and cemented in the hole.
The 26” hole will generally be drilled with seawater. In most cases, this hole section is drilled without circulation back to the rig and in this case, the drilled cuttings are deposited on the seabed. The 26” hole is drilled by first drilling a small diameter (12 ¼”) pilot hole, logging the open formations, removing the diverter assembly, and then opening out to 26” diameter. The logging operation is performed to ensure there are no open hydrocarbon-bearing sand in the pilot hole section. Having drilled the 26" hole the diverter, riser, and hydraulic latch are recovered and laid down. The required length of 18 5/8" casing string is made up. Then the casing is run and cemented in the hole.
Before drilling the next section, BOP stack must be installed. The operation continues drilling de 17 ½” hole taking mud returns to the surface. When the casing setting depth is reached, the hole is circulated clean and the drilling assembly is recovered in preparation for running, setting, and cementing the 13 3/8” casing.
Then the 12 ¼” bit and BHA are made up and run to just above the cement inside of the 13 3/8” casing where a casing pressure test must be done to check the integrity of the casing shoe. The next section of the hole is drilled to the desired o required depth, cleaned out, and the 9 5/8” casing is run and cemented.
If more sections are required, the procedure drilling an 8 ½” hole and run and cement 7” casing [5].

2.3.1. Rate Of Penetration (ROP)

One of the main parameters registered and used to show the drilling performance is the rate of penetration, which is recorded as the well is being drilled and can be measured according to the drilling depth in a certain period, for example, meters per hour or feet per hour. This measurement is usually done by reading the chart on the geolograph or using a digital encoder that is attached to a part of the rig that moves in proportion to the movement of the drill string. Common attachment points are the drill line, drawworks drum, or crown sheaves.
Generally, ROP increases in fast drilling formations such as sandstone which can be called a positive drill break and, decreases in slow drilling formations such as shale also called a reverse break. ROP decreases in shale due to diagenesis and overburden stresses. Over pressured zones can give twice of ROP as expected which is indicative of a kick. If the main objective is optimizing the cost of the project, having a high ROP is normally an advantage [7].

2.3.2. Applications of ROP Measurements

ROP logs or track, serve as historical records of drilling performance and can help optimize drilling performance and formation evaluation [8]. Some of the applications are listed below.

2.3.2.1. Bit Selection

Rate of penetration logs will help to choose the best bit type with an appropriate mechanic and hydraulic parameters. Registered information from previous wells can help to correlate formations, ROP, and bit used in order to have a good selection of the bit for the new well to be drilled.

2.3.2.2. Mud Weight Adjustment

The drilling rate of shale is very sensitive to the pressure balance between the shale and wellbore. Shales drilled underbalanced are characterized by fast drilling, high gas readings, and large cuttings. Shales drilled overbalanced are characterized by slow drilling, low gas readings and, small cuttings. Recognizing this relation can aid in the selection of a mud weight to optimize drilling speed [8].

2.3.2.3. Correlation

The ROP log is the first source of data that can be used to correlate to nearby wells. This is done by comparing ROP to gamma ray or SP curves. This correlation can help determine structural and stratigraphic and stratigraphic position and is normally used to predict when the well will reach a zone of interest.

2.3.2.4. Lithology

Shales generally drill slower than sandstones or carbonates, and as a result, ROP logs tend to reflect indirectly the lithology of the formation [8].

2.3.2.5. Drilling Breaks

Another factor that is important to mention is called “the drilling breaks” which is basically a noticeable increase in the ROP. When it is a decrease, is called a negative drilling break.

The importance of the drilling break shape is that if ROP changes and there are no changes in the drilling parameters as WOB or RPM, so it can be said that a new formation is being drilled or there is a change in the lithology.

If there is a drilling break, there is a change in the porosity which can result in an increase of the formation pressure and having an influx, that is why, according to the ROP measure, it is recommendable to perform flow checks.

2.3.3. Factors affecting ROP

In order to develop an ROP model, looking at the factors affecting the ROP is very important. According to some research and experience, the relevant variables to take into account are listed below.

2.3.3.1. Bit Type

The penetration rate is highest when using bits with long teeth and large cone offset angle, but those bits are normally used and efficient in soft formations. The lowest cost per foot drilled usually is obtained by using the longest teeth that are consistent with bearing life at optimum bit operating conditions.
Fixed-cutter bits give a wedging-type rock destruction mechanism in which the bit penetration per revolution depends on the number of blades and the bottom-cutting angle. Diamond and PDC bits are designed for a given penetration per revolution by the selection of the size and number of diamond or PDC cutters. Developments in PDC bits have helped in achieving higher ROPs and longer bit life, but also involve a compromise between open, light-set bits for speed and heavy-set bits for durability. Hydraulic design improvements prevent bit balling, while mechanical-design enhancements increase the ROP [8].

2.3.3.2. Formation Characteristics

Many characteristics of the formation affect directly the ROP, one of the most important is the elastic limit and the ultimate strength of the formation. Mohr failure criterion is often used to characterize the strength of a formation. [9]
Another characteristic that can be mentioned is the mineral composition of the rock since it can have some effect on the penetration rate. Rocks that contain abrasive minerals can cause rapid dulling of the bit teeth but rocks that contain gummy clays can cause the bit to ball up and drill inefficiently.
Permeability of the formation, for example, allows the fluid to filtrate into the rock ahead of the bit and equalize the pressure differential acting on the chips formed beneath each tooth. The nature of fluids contained in pore spaces of the rock also affects the mechanism since more filtrate volume will be required to equalize the pressure in a rock containing a lighter fluid that a rock containing heavy fluid. [8]
Properties that affect ROP include mineralogy and hardness (if the rock is harder, the ROP is low), porosity (if the porosity of the rock is higher, the ROP is high), consolidation against cementation (if the rock is well consolidated, the ROP is low) mineral inclusions such as pyrite and chert, etc.
Additionally, but not least important, as much depth is the hole, the lithology is more compact, so the porosity decreases. This results in a decreased ROP and increased difficulty in drilling.

2.3.3.3. Drilling Fluid Properties

The properties of the drilling fluid that affect the ROP are density, rheological properties, filtration characteristics, solids content and, chemical composition.

ROP tends to decrease with increasing fluid density, viscosity, and solids content, and tends to increase with increasing filtration rate. The density, solids content, and filtration characteristics of the mud control, the pressure differential across the zone of crushed rock beneath the bit. The fluid viscosity controls the parasitic frictional losses in the drill string and, thus, the hydraulic energy available at the bit jets for cleaning. Increasing viscosity reduces ROP even when the bit is perfectly clean [8].
The chemical composition of the fluid also influences ROP by the hydration rate and bit balling tendency of some clays that are affected by the contact with some chemical components of the fluid.
An increase in drilling mud density causes an increase in the bottom hole pressure beneath the bit, due to the hydrostatic pressure of the column, and then an increase in the pressure differential between the borehole pressure and the formation fluid pressure, so if the differential pressure is positive, is correct to affirm the well is being drilled in overbalanced.
On the other hand, if the differential between the borehole pressure and the formation fluid pressure is negative, the well is being drilled in underbalanced condition, which is a very good way when talking about improve ROP.

2.3.3.4. Bit Operating Conditions

The operation conditions or drilling conditions also have a relevant effect on the improvement of ROP.

- **Revolutions per minute (RPM).** If the RPM is increased, then the ROP will increase. In soft formations ROP, is directly proportional to RPM and shows a linear increase. However, in hard formations, the ROP increase is not linear and will decrease with RPM increasing. The exception is, again, with the diamond bits or PDC when, even in hard formations, the ROP will increase linearly with the rotary speed.

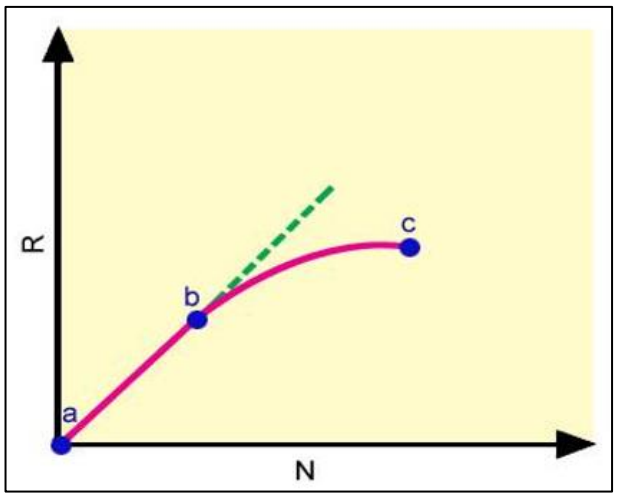

Figure 2. Relationship between ROP and RPM *[8].*

As it is shown in ***Figure 2*** where (R) is the ROP and (N) is the RPM, ROP increases linearly when the value of RPM increases. In the (b) point, the

foundering point is when the linearity is lost. This phenomenon is basically due to less efficient bottom hole cleaning, and it is also dependent on drilling fluids parameters.

- **Weight On Bit (WOB)** The WOB also affects the ROP. The relationship is linear since the WOB is duplicated, and the ROP will be duplicated too. The WOB is a parameter that must be controlled to avoid the bit tooth wear. See ***Figure 3***.

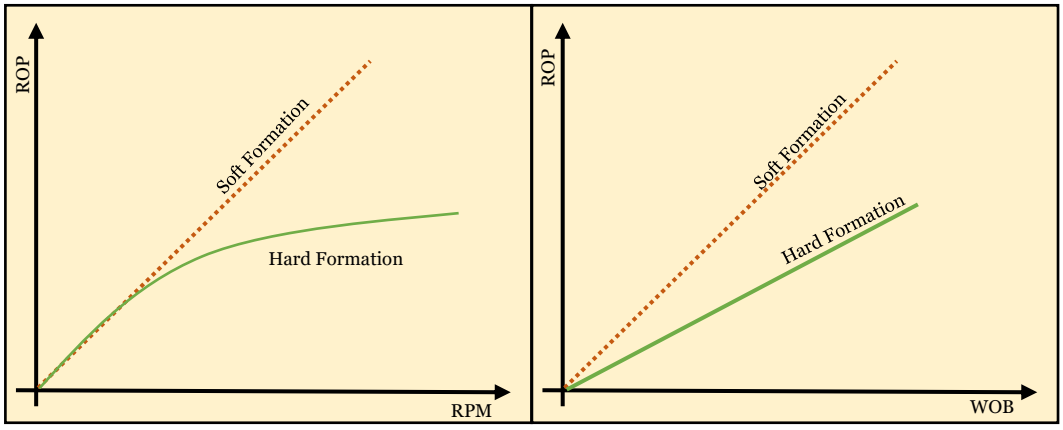

Figure 3. ROP vs RPM and ROP vs WOB *[8]*.

On the other hand, in the plot of ROP vs WOB (See ***Figure 4***) where tooth wear is not assumed and ROP is represented by "R" and WOB by "W", no significant ROP is obtained until the threshold formation stress is exceeded (See point a). ROP increases gradually and linearly with increasing values of WOB for low values of WOB (segment from a to b). A linear curve is observed at higher WOB (segment from b to c), where the segment has a much steeper slope which represents an increased drilling efficiency. Point "b" is the transition point where the rock failure mode changes from scraping or grinding to shearing. Beyond point "c", subsequent increases in WOB cause only slight improvements in ROP (segment from c to d). Sometimes a decrease in ROP is observed at extremely high values of WOB (segment from d to e).

The poor response of ROP at high WOB values is normally because of a less efficient hole cleaning related to a higher rate of cuttings generation, or because of complete penetration of a bit's cutting elements into the formation being drilled, without clearance for fluid bypass [8].

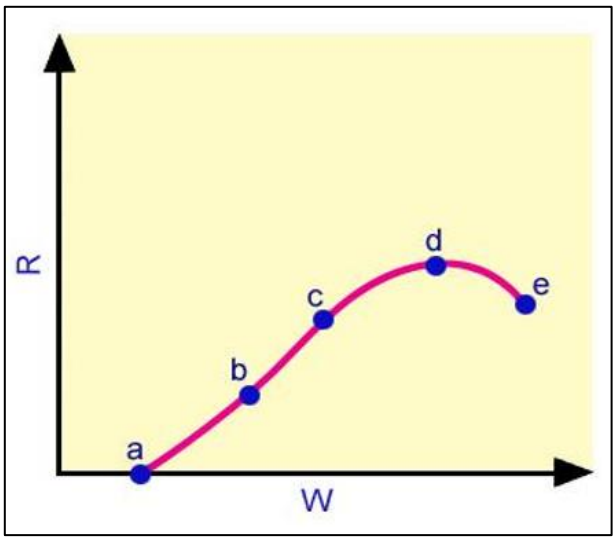

Figure 4. Relationship between ROP and WOB *[8].*.

2.3.3.5. Formation Pressure

A high formation pressure results in a higher differential of pressure which leads to a slower ROP but high values of formation pressure such a result of retention of formation fluids can lead to high porosity and an increase in ROP.

It is good to mention that ROP is a valuable indicator of lithology and a valuable aid for correct correlation.

2.3.3.6. Bit Hydraulics

Taking into account the relevance of cleaning the new drilling cuttings from the bottom of the hole, to maintain an optimal ROP, the cleaning of the hole must be effective since the cuttings can clog the bottom of the hole, in turn reducing the cutting surface of the hole. the bit, which would affect the ROP.

An important factor during drilling operations is the equivalent circulating density or ECD (An increase in mud density measured on the surface), due to frictional pressure losses in the annular differential pressure will increase. As well as an increase in the actual density of the drilling mud, these pressure losses will increase if the flow rate increases or if the flow regime is turbulent [4].

2.3.3.7. Bit Tooth Wear

Most bits are designed with certain footage or life cycle so as soon as they are run in the hole, the performance can be very good, but it will be decreasing because of tooth wear.

The tooth length of milled tooth rolling cutter bits is reduced continually by abrasion and chipping. The teeth are altered by hard facing to promote a self-sharpening type of tooth wear. However, while this tends to keep the tooth pointed, it does not compensate for the reduced tooth length. The teeth of tungsten carbide insert-type rolling cutter bits and PDC bits fail by breaking rather than by abrasion. Often, the entire tooth is lost when breakage occurs. Reductions in ROP due to bit wear usually

are not as severe for insert bits as for milled tooth bits unless many teeth are broken during the bit run [8].
The change in ROP is one of the main conditions to decide when the BHA needs to be pulled out of the hole because the bit must be changed for a new one.

2.3.3.8. Personal Efficiency

Manpower skill and experience of the driller are the keys to the success or failure of those operations and ROP is one of them. Overall, well costs as a result of any drilling problem can be extremely high; therefore, continuous training for personnel directly or indirectly involved is essential in order to achieve desired ROP [8].

2.4. ROP Models

The traditional models are empirical correlations developed based on regression analysis, and the data-driven models are developed based on machine learning techniques.

2.4.1. Traditional ROP Models

Traditional ROP models have been used for the prediction of ROP in drilling with some success. According to Hegde et al. [10], these traditional models have disadvantages such as the use of empirical coefficients, the requirement for auxiliary data such as bit properties, mud properties, bit design, among others, low accuracy in ROP predictions, and their conformity to one facies (since the empirical coefficients are highly dependent on lithology). These models remain unchanged; however, the empirical coefficients are continuously varied according to the calibration data.
It is important to investigate the equations of the physics-based models. This will lead to the understanding of the input parameters, and their importance in drilling, also reducing the overall cost of the drilling project by improving drilling operations.

2.4.1.1. Bingham's Model (1964)

The first major study was performed in the 1950s, where empirical relationships from WOB and RPM (R-W-N models) were developed [11]. The models are normally designed to work either for roller-cone bits or fixed-cutter bits. See ***Equation 1***.

$$ROP = K \left(\frac{W}{d_b}\right)^a * N$$

Equation 1. Bingham's Model

Where:
K = Constant of proportionality.
W = Weight on Bit [klbf].
d_b = Bit diameter.
a = Bit weight exponent.

N = Rotary speed of RPM.

For the previous equation, the results are highly dependent on the value of a_5, but the determination of such an exponent is not an easy task because 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 [12].

2.4.1.2. Bourgoyne and Young's Model (1974)

One of the ROP models which was widely accepted was presented by Bourgoyne [13]. This model estimates ROP as a function of eight parameters as it is shown in ***Equation 2***.

$$ROP = (f_1)\,(f_2)\,(f_3)\,\ldots\,(f_n)$$

Equation 2. Bourgoyne and Young's Model

Where:

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

$$f_2 = e^{2.303 * a_2 * (10000 - D)}$$

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

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

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

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

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

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

D = True vertical depth [ft].
K_s = Constant of proportionality.
N = Rotary speed of RPM.
g_p = Pore pressure gradient [lbm/gal].

ρ_c = Equivalent circulating density [psi].
F_j = Hydraulic impact force beneath the bit [lbf].
a_i = Chosen constants according to drilling conditions.
$\left(\frac{W}{d_b}\right)_t$ = Threshold bit weight per inch of bit diameter at which the drill begins to drill [1,000lbf/in].

The multiple constants assigned to "a" refers to a_1 is the formation strength parameter, a_2 is the normal compaction trend exponent, a_3 is the under compaction exponent, a_4 is the pressure differential exponent, a_5 is the bit weight exponent, $a6$ is the rotary speed exponent, $a7$ is the tooth wear exponent, and $a8$ is the hydraulic exponent. Coefficients $a1$ through $a8$ are determined with multiple regression techniques, using several data points to determine the eight unknowns that best fit a specific set of field data [14].
The Bourgoyne's Model was designed for roller-cone bits, as was mentioned before, but in recent years it has been applied for wells drilled with PDC bits [15].

2.4.1.3. Hareland's Model (1994)

Hareland proposed a bit-specific model which is specific to the drag bit. The model is defined by **Equation 3**.

$$ROP = 14.14 * N_c * RPM * \frac{A_v}{D_b}$$

Equation 3. *Hareland's Model*

Where:
A_v = Area of the rock compressed ahead of a cutter [in^2].
N_c = Number of cutters.
D_b = Bit diameter.
RPM= Revolutions per minute.

A_v is set based on the type of drag bit, in the case of a polycrystalline diamond cutter bit can be formalized as **Equation *4***.

$$A_v = cos\alpha sin\theta\left(\left(\frac{d_c}{2}\right)cos^{-1}\left(1-\frac{4WOB}{\cos(\theta\pi)N_c\sigma_c d_c^2}\right)-\left(\frac{2WOB}{cos(\theta\pi)N_c\sigma_c}-\frac{4WOB}{\cos(\theta\pi)N_c\sigma_c {d_c}^2}\right)^{0.5}\left(\frac{WOB}{\cos(\theta\pi)N_c\sigma_c}\right)\right)$$

Equation 4. Area of rock compressed ahead of a cutter.

Where:
d_c = Cutter diameter [in].
σ_c = Unconfined compressive strength [psi].
α = Cutter side rake angle [degrees].

θ = Cutter back rake angle [degrees].

2.4.1.4. Motahhari Model (2010)

Motahhari introduced a PDC bit specific model and incorporated it within a wear function and the effect of rock strength of ROP. This model takes into account perfect bit cleaning conditions and is defined by **Equation 5** [16].

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

Equation 5. Motahhari's Model

Where:
S = Confined rock strength.
W_f = Wear function.
RPM_t = Rotary speed or RPM.
D_b = Bit diameter.
WOB = Weight on Bit.
α = ROP model exponent.
γ = ROP model exponent.

The wear function is shown in **Equation 6**.

$$W_f = k_w f \left(\frac{WOB}{N_C} \right)^{\rho} * \frac{1}{S^{\tau} * A_w^{\rho+1}}$$

Equation 6. Wear function.

Where:
N_c = Number of cutters on the bit face.
$k_w f$ = Wear function constant.
ρ = Wear function exponent.
τ = Wear function exponent.
A_w= It 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 hole using IADC dull grading, so to find this factor it needs to be estimated by a constant degradation factor as a function of depth.

The application of this model is highly dependent on the value of W_f, which is difficult to implement as it has been shown to introduce various fitting parameters [17].

The confined rock strength used in the previous model requires laboratory testing at different confining pressures which are seldom undertaken. Field based correlations and rock failure envelopes can be used to determine the confined rock strength.

Unconfined compressive rock strength (UCS) data in this dataset come from calculations using field-based correlations on sonic log measurements [10].

2.4.2. Data Driven ROP Models

Data-driven modeling's successful application in different industries has caused an increasing interest in the subject from the Oil and Gas Industry and is regarded as the future of the segment due to its potential for optimizing drilling operations [18].
When talking about data driven models, is important to mention that different ML models can be applied according to what is going to be analyzed and for each well or case, its data should be analyzed independently.

2.4.2.1. Ensemble Methods

According to Géron et al. [19] *"a group of predictors is called a ensemble… and an Ensemble Learning Algorithm is called an Ensemble method"*. Ensemble methods are techniques that create multiple models and then combine them looking to get improved results. They usually give more accurate solutions than the solutions given by a single model.

Ensemble uses two types of methods, one of them is called bagging, that works creating different training subset from sample training data with replacement and the final output is based on majority voting, for example random forest. On the other hand, is boosting method, that combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy, for example Ada Boost.

The type of base learners used will define the classification of the ensemble model. The ensemble is called homogeneous when all learners belong to the same type, and it is called heterogeneous when there are different types of learners. One example of homogenous learners can be Random Forest, that is based on Decision Trees (DT) and it is used to get a single result from the different possibilities provided by each tree. RF is a well-known and powerful ML algorithm [6].

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 complex.

Chapter 3.
Machine Learning Models

3.1. Machine Learning Models

Supervised learning is a method where a computer algorithm is trained on input data that has been labeled for a particular output. The model is usually trained until it can detect the underlying patterns and relationships between the input data and the output labels, enabling it to yield accurate labeling when presented with never-before-seen data [20]. Supervised learning is normally defined by **Equation 7**, where the goal is to approximate the mapping function so well that when you have new input data you can predict the output variables for that data.

$$Y = f(X)$$

Equation 7. Supervised Learning.

Where:
X = Input variables.
Y = Output variables.

Supervised learning cases can be further grouped into regression and classification cases. Through this project, regression cases are going to be developed.

3.1.1. Random Forest Algorithm

Random Forest is a supervised ML algorithm that is normally used in classification and regression problems, also it is one of the best techniques with high performance which is widely used for its efficiency. It can handle binary, continuous, and categorical data.

RF is based on bagging or bootstrap aggregation as an ensemble technique. Bagging chooses a random sample from the data set. Hence each model is generated from the samples provided by the original data with replacement known as row sampling. This step of row sampling with replacement is called bootstrap. Since each model is trained independently which generates results. The final output is based on majority voting after combining the results of all models, this step which involves combining all the results and generating output based on majority voting is known as aggregation.

3.1.2. Gradient Boosting Regressor.

GB builds an additive model in a forward stage-wise fashion, it allows for the optimization of arbitrary differentiable loss functions. In each stage, a regression tree is fit on the negative gradient of the given loss function [21]. Regression trees are most teamed with boosting. See **Figure 5.**

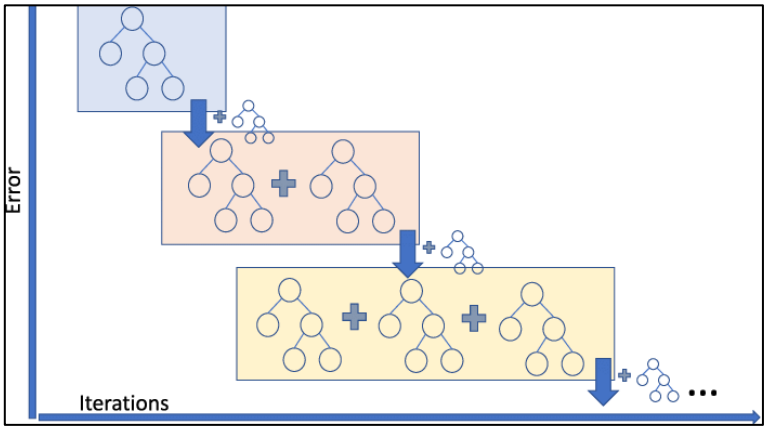

Figure 5. Schematical representation of gradient boosting regression regarding algorithm iterations *[22].*

Hyperparameter tuning is a way to set the parameters to improve the results of the model and it consists of setting the value of parameters that the algorithm cannot learn on its own. In order to find these parameters, this process is carried out where it is commonly sought to make the algorithm use various combinations of values until it finds the values that are best for the model [23].

Considering the above, there are several hyperparameters that we need to adjust, and they are as follows [21].

- **Number of Estimators.** Show the total number of trees in the ensemble or the number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting, so a large number usually results in better performance.
- **Maximum depth.** Refers to the number of leaves of each tree or the maximum depth of the individual regression estimators. Tuning this parameter is recommended to have a better performance since the best value depends on the interaction of the input variables.

- **Learning Rate.** Hyperparameter scales the contribution of each tree. There is a trade-off between the learning rate and the number of estimators.
- **Subsample.** The fraction of samples to be used for fitting the individual base learns. If it is smallest than 1.0 this results in Stochastic Gradient Boosting. Choosing a subsample number less than one (1) leads to a reduction of variance and an increase in bias.
- **Random State.** Controls the random seed given to each tree estimator at each boosting iteration also it controls the random permutation of the features at each split.

3.1.3. Random Forest Regressor

RF regressor is a meta estimator that fits several classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [21]. See ***Figure 6*** [24]**.**

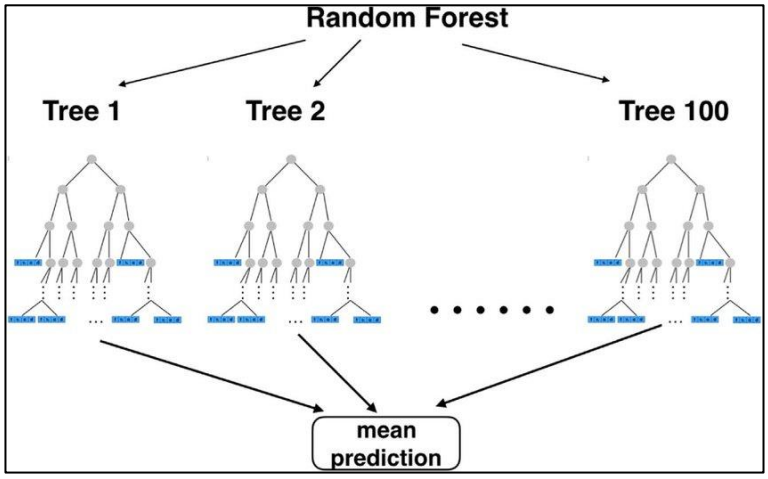

Figure 6. Random Forest Regressor *[24]*.

In the case of a random forest, hyperparameters include the number of decision trees in the forest and the number of features considered by each tree when splitting a node. Hyperparameter tuning relies more on experimental results than theory, and thus the best method to determine the optimal settings is to try many different combinations to evaluate the performance of each model. However, evaluating each model only on the training set can lead to one of the most fundamental problems in machine learning: overfitting [25].

If the model is optimized for the training data, it will score very well on the training set, but it will not be able to generalize to new data such as in a test set. According to Koehrsen et al. [25] *"When a model performs highly on the training set but poorly on the test set, this is known as overfitting, or essentially creating a model that knows the training set very well but cannot be applied to new problems. It's like a student who has memorized the simple problems in the textbook but has no idea how to apply concepts in the messy real world."*.

When an overfitting model is found (See **Figure 1**) it can look perfect on the training set, but it will be useless in a real application that is why a cross-validation should be done looking to improve the hyperparameter results.

There are some hyperparameters that we need to adjust, and they are as follows.

- **Number of Estimators.** So, like the one mentioned in GB regressor, the number of estimators refers to the number of trees in the forest.
- **Random State.** Controls both the randomness of the bootstrapping of the sample used when building trees.
- **Maximum Depth:** It is the maximum depth of the tree or the longest path between the root node and the leaf node.

3.1.4. Multi-Layer Perceptron Regressor

A perceptron is recognized as an algorithm, but it was initially intended as an image recognition machine. It gets its name from performing the human-like function of perception, seeing, and recognizing images.

The multilayer perceptron (MLP) has input and outputs layers, and one more hidden layer with many neurons stacked together. It falls under the category of feedforward algorithms because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like the perceptron.
Each layer is feeding the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer [26].

Backpropagation is the learning mechanism that allows the MLP to iteratively adjust the weights in the networks, with intending to minimize the cost function [27]. The function that combines inputs and weights in a neuron, for instance, the weighted sum, and the threshold function, for instance, ReLU, must be differentiable. See ***Figure 7*** [26]**.**

In each iteration, after the weighted sums are forwarded through all layers, the gradient of the MSE is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That is how the weights are propagated back to the starting point of the neural network.

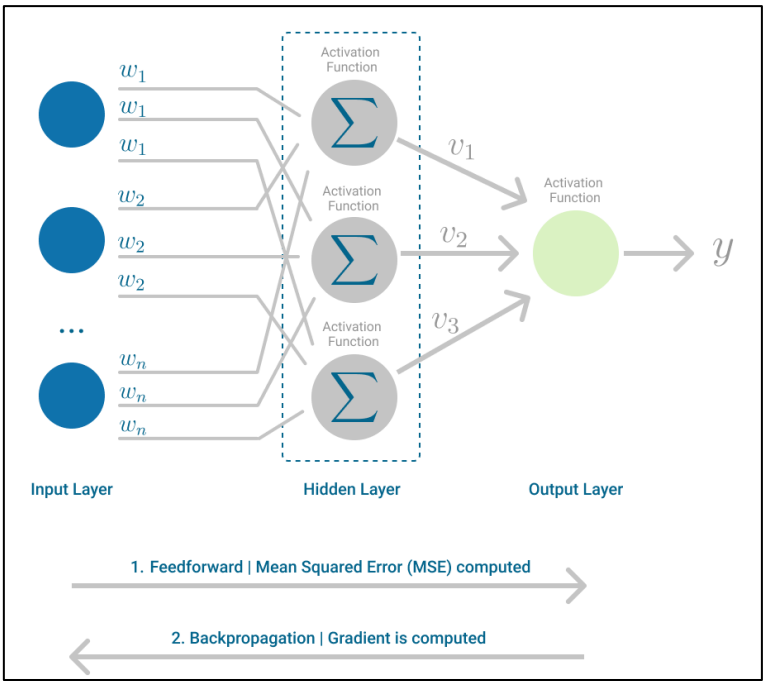

Figure 7. Multilayer Perceptron, highlighting the Feedforward and Backpropagation steps *[26]*.

There are several hyperparameters that needs to be adjusted, the first one to tune is the number of neurons in each hidden layer.

- **Hidden Layer Sizes.** Determining this parameter is possible to specify the number of layers and the number of nodes that are required to have in the Neural Network Classifier. Each element represents the number of nodes at the ith position, where i is the index of the tuple. Thus, the length of the tuple indicates the total number of hidden layers in the neural network [28].
- **Alpha:** L2 penalty (regularization term) parameter [21].
- **Activation.** Represents the activation function for the hidden layers. It can be identity (linear bottleneck), logistic (logistic sigmoid function), tanh (hyperbolic tan function) or relu (rectified linear unit function).
- **Learning Rate:** schedule for weight updates. It can be constant, invscaling or adaptive.
- **Solver:** This parameter specifies the algorithm for weight optimization over the nodes, it can be 'lbfgs' which is an optimizer in the family of quasi-Newton methods, 'sgd' which refers to stochastic gradient descent or, 'adam' which refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba.
  According Scikit learn et al. [21] *"The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better."*

3.1.5. AdaBoost Regressor

An AdaBoost Regressor [29] is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases. This algorithm is sensitive to outliers and is thus useful to check for outliers in the data set. See **Figure *8*** [30]**.**

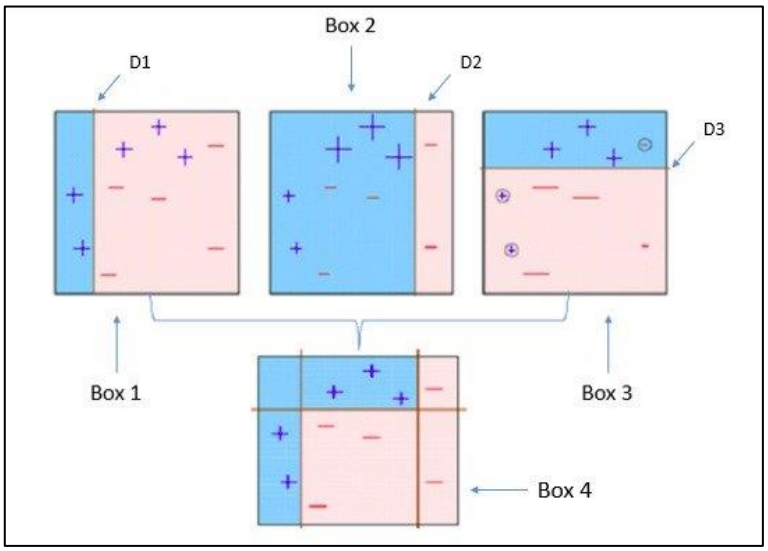

Figure 8. AdaBoost Regressor *[30].*

Some of the hyperparameters need to be adjusted, they are [21]:

- **Number of estimators:** It is the maximum number of estimators at which boosting is terminated. In the case of a perfect fit, the learning procedure is stopped early.
- **Learning Rate:** There is a trade between the learning rate and the number of estimators. Weight applied to each classifier at each boosting iteration, a higher learning rate increases the contribution of each classifier.
- **Random State:** Control the random seed given at each base estimator at each boosting iteration.

3.1.6. K-Neighbors Regressor

The k-Nearest Neighbors (kNN) algorithm is one of the simplest ML algorithms because building the model just consists of storing the training dataset. To predict for a new data point, the algorithm finds the closest data points in the training dataset, its "nearest neighbors" [31].
K-Neighbors Regressor is a variant of kNN where the target is predicted by local interpolation of the targets associated with the nearest neighbors in the training set [21].

Some of the hyperparameters need to be adjusted, they are [21]:

- **Number of neighbors:** It is the, as your name said, the number of neighbors to use by default for kneighbors queries. Using only a single neighbor, each point in the training set has an obvious influence on the predictions, and the predicted value goes through all the data points. This leads to a very unsteady prediction. Considering more neighbors leads to smoother predictions, but these do not fit the training data as well. In **Figure 9** , the blue points are the responses to the training data, while the red line is the prediction made by some model for all points on the line [31].

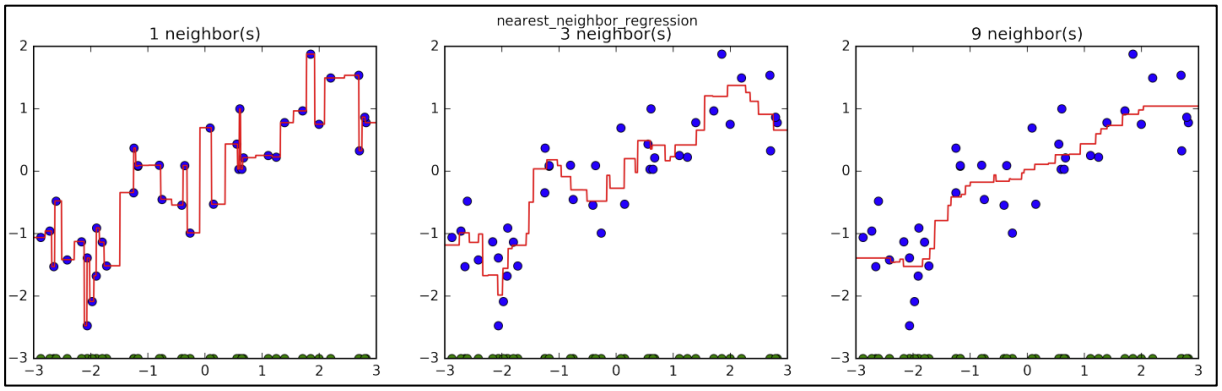

Figure 9. Behavior according to the number of neighbors *[31]*.

- **Weights:** It is the weight function used in prediction. The possible values are [21]:
  - Uniform, uniform weights. When all points in each neighborhood are weighted equally.
  - Distance, weight points by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  - Callable: a user-defined function that accepts an array of distances and returns an array of the same shape containing the weights.

3.1.7. Linear Regression

Linear models are a class of models that make a prediction, as its name said, using a linear function of the input features. One of the simplest and most classic linear methods for regression is called Ordinary Least Squares (OLS).

Linear regression finds the parameters “w” and “b” that minimize the mean squared error between predictions and the true regression targets on the training set. Linear regression has no parameters, which is a benefit, but it also has no way to control model complexity [31]. See **Figure *10***.

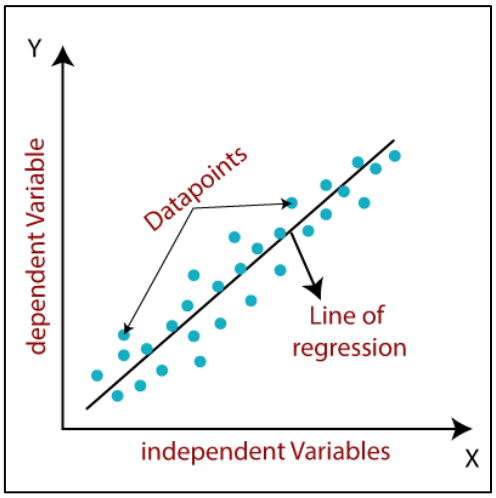

Figure 10. Linear Regression in Machine Learning *[31]*.

3.1.8. Advantages and Disadvantages

Some of the general advantages and disadvantages when talking about regression models are shown in ***Table 1***.

Regression Model	Advantages	Disadvantages
Linear Regression	• Works well irrespective of the dataset size. • Gives information about the relevance of features.	• The assumption of Linear Regression.
Decision Tree Regression	• Interpretability. • Works well on both, linear and non-linear problems. • No need to apply feature scaling.	• Poor results on small datasets. • Overfitting can easily occur.
Random Forest Regression	• Powerful. • Accurate. • Good performance on many problems including non-linear.	• No interpretability. • Overfitting can easily occur. • Number of trees has to be chosen.

Table 1. Advantages and disadvantages of regression models

3.2. Training and Testing Data Set

Machine learning algorithms work in two stages, they are called training data set and test data set.
The training set is a portion of the actual dataset that is fed into the ML model to discover and learn patterns, so it is the one in charge of training the model, it is typically larger than testing data because the main objective is to feed the model with as much data as possible in order to find and learn meaningful patterns. Once data from the dataset are fed to an ML algorithm, it learns patterns from the data and makes decisions.
According to Barkved et al. [32] "*Algorithms enable machines to solve problems based on past observations. Kind of like learning from example, just like humans. The only difference is that machines require a lot more examples in order to be able to see patterns and learn. As machine learning models are exposed to more relevant training data, the more they improve over time*".

Once the ML model is built and fed with the training data, the model needs to be tested with unseen data. This data is known as testing data, and it can be used to evaluate the performance and progress of the algorithms' training and adjust or optimize looking for improving results.
The data set for testing has to be different from the training set because the model already knows the training data, so testing data is helpful to see if the model is working accurately or if it requires more training data to perform to the desired specifications.
To conclude, the main difference between the training and testing set is that one trains a model and the other confirms it works correctly.

3.2.1. Splitting Data.

In Data science, it is common to split the data into 80% for training and 20% for testing. According to Barkved et al. [32] *"…In supervised learning, the outcomes are removed from the actual dataset when creating the testing dataset. They are then fed into the trained model. The outcomes predicted by the trained model are compared with the actual outcomes. Depending on how the model performs on the testing dataset, we can evaluate the performance of the model…"*.
Train/test is a method to measure the accuracy of the ML model, where the data set is split into two sets in order to train and test the model.

"As a reminder, the reason we split our data into training and test sets is that we are interested in measuring how well our model generalizes to new, unseen data. We are not interested in how well our model fits the training set, but rather, how well it can make predictions for data that was not observed during training." [31]

Scikit-learn contains a function that shuffles the dataset and splits it. The function extracts some percentage of the rows to be used as a training set and the remaining data are declared as the test set.

3.2.2. Underfitting vs. Overfitting

To understand the root cause of poor model accuracy is important to know the model fit. Through it, it is possible to determine whether a predictive model is underfitting or overfitting the training data by looking at the prediction error in the training and evaluation data [33].
The model is underfitting the training data when it performs poorly on the training data, it happens because the model is not able to capture the relationship between the input and the target values.

On the other hand, the model is overfitting the training data when the model performs well on the training data but does not perform well on the evaluation data, it happens because the model is memorizing the data it has seen and is not able to generalize to unseen examples. See **Figure *11***.

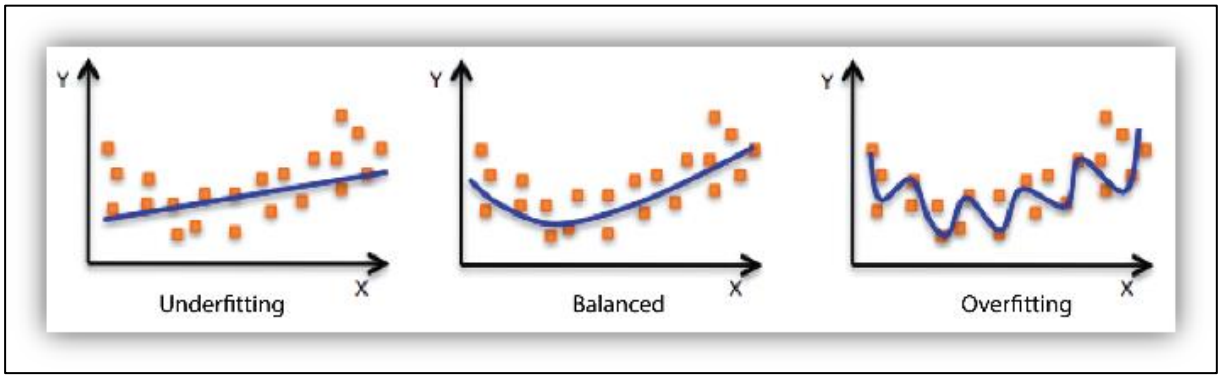

Figure 11. Model Fit: Underfitting vs. Overfitting *[33]*.

3.3. Model Evaluation and Improvement

Evaluation models and selecting parameters is one of the most important tips to qualify the accuracy of the model in supervised learning, that is why, after splitting, testing the data and, building the model, it is important to evaluate it.

3.3.1. Cross-Validation

The technique of Cross-validation (CV) consists of further splitting the training set into K numbers of subsets, called folds, then iteratively fitting the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data).
When talking about hyperparameter tuning, it has to be performed many iterations of the entire K-fold CV process, each time using different model settings, Then to compare

all of the models, select the best one, train it on the full training set and, then to evaluate on the testing set, everything can be done using Scikit-Learn [21].
Using Scikit-Learn's RandomizedSearchCv method, it is possible to define a grid of hyperparameter ranges and randomly sample from the grid, performing K-Fold CV with each combination value.
According to Guido and Muller [31], there are several benefits of using cross-validations instead of a single split into a training and test set. One of them is that when cross-validation is used, each example will be in the training set exactly once: each example is in one of the folds, and each fold is the test set once. Therefore, the model needs to generalize well to all the samples in the data set for all the cross-validation scores to be high.

3.3.2. Grid Search

Grid search is a method for adjusting the parameters in supervised models for the best generalization performance. Since finding the values of the important parameters of a model is a necessary task, it must be done for almost all models and datasets.
There is a huge risk to overfit the parameters so splitting the data into training and test data set in the first place has to be performed and after it, evaluate the model with an independent data set to the one used to create the model.

3.3.3. Grid Search with Cross-validation

For a better estimate of the generalization performance, instead of using a single split into a training and a validation set, using cross-validation to evaluate the performance of each parameter combination, is recommended [31].
According to Guido and Muller et al. [31] *"…the best parameter setting is selected. For each parameter setting, accuracy values are computed, one for each split in the cross-validation. Then the mean validation accuracy is computed for each parameter setting. The parameters with the highest mean validation accuracy are chosen…".*

3.3.4. Metrics and Scoring

There are three (3) different APIs for evaluating the quality of a model's prediction [21]:

- The estimators' score method: The estimators have a score method providing a default evaluation criterion for the problem they are designed to solve.
- Scoring parameter: Model evaluation tools using cross-validation rely on an internal scoring strategy.
- Metric functions: The sklearn.metrics module implements functions assessing prediction error for specific purposes.

3.3.5. Regression Metrics

Evaluating for regression can be done by analyzing over-predicting the target versus under-predicting the target. In some cases, using R^2 in the scoring method of all regressors is enough.

Below are some metrics for evaluating the performance of the regression model: [34]

3.3.5.1. Mean Absolute Error (MAE).

It measures the average magnitude of the errors in a set of forecasts, without considering their direction. It is not very sensitive to outliers compared to MSE. It measures accuracy for continuous variable data.
It gives a linear value, which averages the weighted individual differences equally. The lower the value, the better is the model's performance.
It is identified by **Equation 8.**

$$MAE = \frac{1}{n}\sum_{1}^{n}|y_i - \hat{y}_i|$$

Equation 8. Mean Absolute Error

Where:
n = Number of samples.
$\hat{y}_i$ = Predicted value of i-th sample.
y_i = The corresponding true value.

3.3.5.2. Mean Squared Error (MSE).

It is one of the most used in metrics, but least useful when a single bad prediction would ruin the entire predicting abilities of the model. It can be like MAE but differs from it in that it squares the difference before summing instead of just taking the absolute value. See **Equation 9.** Mean Squared Error [35].
MSE is most useful when the dataset contains outliers or unexpected values.

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

Equation 9. Mean Squared Error

Where:
n = Number of samples.
$\hat{y}_i$ = Predicted value of i-th sample.
y_i = The corresponding true value.

Since the difference is squared, the MSE generally is greater than the MAE and for this reason, MSE cannot be directly compared to the MAE, also the effect of the quadratic term in the MSE equation is most apparent when there are outliers in the data set. MSE is like a combination measurement of bias and variance of the prediction.

3.3.5.3. Root Mean Squared Error (RMSE).

In RMSE the errors are squared before they are averaged. This implies than RMSE assigns a higher weight to large errors, so RMSE is much more useful when large errors are present, and they affect the performance of the model. The common goal between MSE and RMSE is to measure how large the residuals are distributed. Their values lie in the range between zero and positive infinity and in this metric also, the lower is the value the better is the performance of the model.

Max Error. The max error function computes the maximum residual error. It is a metric that measure the worst case error between the predicted value and the true value, see **Equation *10***.

$$Max\ Error = max(|y_i - \hat{y}_i|)$$

Equation 10. Max Error

Where:
$\hat{y}_i$ = Predicted value of i-th sample.
y_i = The corresponding true value**.**

This metric shows the extent of error that the model had when it was fitted.

3.3.5.4. Mean Absolute Percentage Error (MAPE).

Also known as mean absolute percentage deviation (MAPD). Here, each prediction is scaled against the value it is supposed to estimate, MAPE is the percentage equivalent of MAE, see **Equation 11.** Mean Absolute Percentage Error.

$$MAPE = \frac{100}{n} \sum_{1}^{n} \frac{|y_i - \hat{y}_i|}{max\ (\epsilon, |y_i|)}$$

Equation 11. Mean Absolute Percentage Error

Where:
n = Number of samples.
$\hat{y}_i$ = Predicted value of i-th sample.
y_i = The corresponding true value.
ϵ = Arbitrary small yet strictly positive number to avoid undefined results when y is zero.

3.3.5.5. Median Absolute Error (MedAE).

It is robust to outliers. The loss is calculated by taking the median of all absolute differences between the target and the predictions [21]. See **Equation *12***.

$$MedAE = median(|y_i - \hat{y}_i|, \dots, |y_n - \hat{y}_n|)$$

Equation 12. Median Absolute Error

Where:
$\hat{y}_i$ = Predicted value of i-th sample.
y_i= The corresponding true value.

3.3.5.6. Coefficient of determination (R^2).

The value R² tells us how much variance in the outcome variable can be explained by the predictors.
It shows the proportion of variance that has been explained by the independent variables in the model. It also indicates of how well fits the model and it is a measure of how well-unseen samples are likely to be predicted by the model, through the proportion of explained variance. [21]
The best possible score is 1.0.
The coefficient of determination is defined as (see **Equation *13***).

$$R^2 = 1 - \frac{\sum_1^n (y_i - \hat{y}_i)^2}{\sum_1^n (y_i - \bar{y})^2}$$

Equation 13. Coefficient of determination.

Where:
$\hat{y}_i$ = The predicted value of i-th sample.
y_i = The corresponding true value.

$$\bar{y} = \frac{1}{n}\sum_1^n y_i$$

$$\sum_1^n (y_i - \hat{y}_i)^2 = \sum_1^n {\epsilon_i}^2$$

3.4. Selection of Parameters

Analyzing the traditional ROP models, it is noticeable that besides the Bingham ROP model (***Equation 1***) just requires few inputs. Other ROP traditional models like Burgoyne and Young (***Equation 2Equation 2***. *Bourgoyne and Young's Model*) and

Motahhari (**Equation *5***) are dependent of many inputs, and much of them are difficult to find so they need to be estimated which can end up in lack of accuracy predicting ROP.

When ROP is predicted using ML models, the information that can be used to feed up the model can be as much as possible, taking into account the availability of data, and how much ROP is affected by each of the parameters considered.

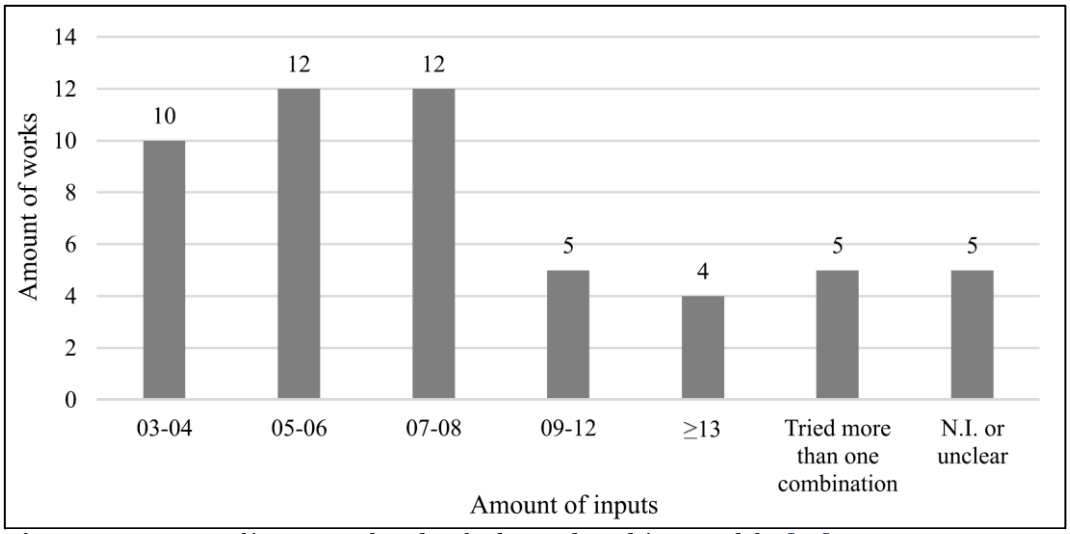

Figure 12. Amount of inputs employed to feed ROP data-driven models *[36].*

Depending on the availability the sensors or measures taken, configuration of the equipment or tools a large number of measurements can be stored. Barbosa shows in **Figure *12***. Amount of inputs employed to feed ROP data-driven models [36], from fifty three (53) different works analyzed, ten (10) reported the use of three (3) or four (4) inputs, twelve (12) works used five (5) or six (6) inputs, also twelve (12) works used seven (7) or weight (8) inputs Considering that five (5) works did not report the number of input data for their models, it can be said that almost 70% of the studies worked with less than nine inputs for their respective models [6].

This provides an interesting statistic to select the number of inputs, and to understand why even though many possible inputs are available most researchers prefer to select just a number of them.

Chapter 4.
Methodology

4.1. Methodology in general

In order to develop this research project, it will be carried out the following methodology in their respective order. Step by step of this chat is going to be mentioned and explained along this chapter.

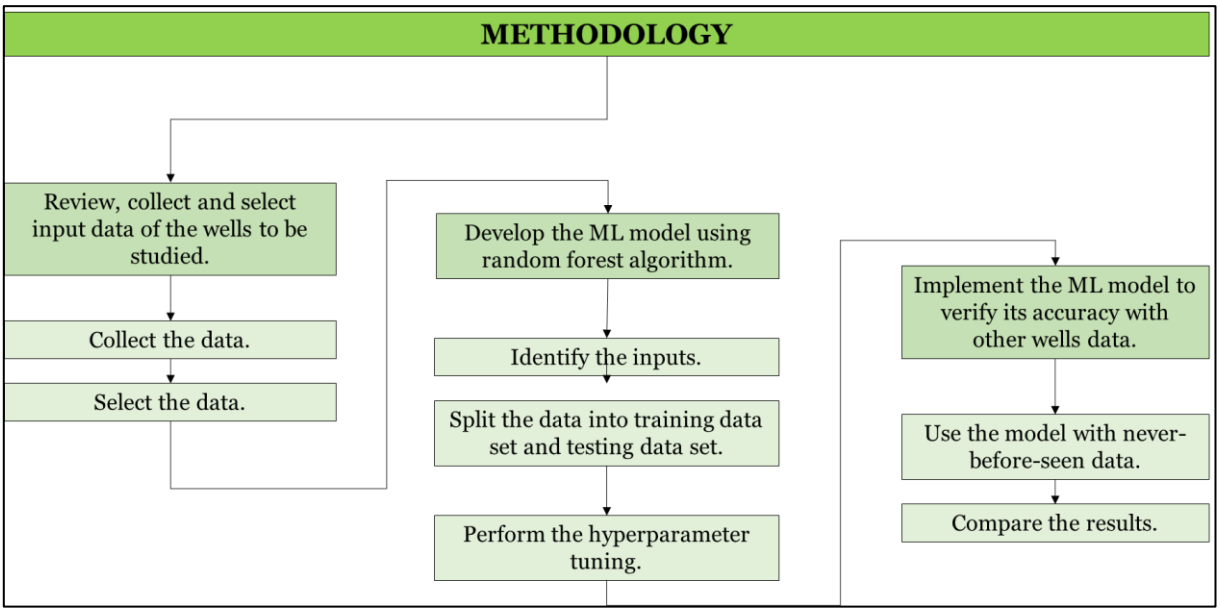

Figure 13. Methodology.

All the methodology in general was summarized in four steps, that are going to be describe below, which were:

- Select the training data set from the whole data set: Data set from Well 1 was used to train the model, keeping a ratio of 80/20 (80 training/20 testing).

- Develop the model with different ML models.
- Test the model on unseen single wells where the model was tested with never-before-seen data from Well 2, 3, 4 and 5.
- Evaluate the model taking into account the results given by the metrics.

4.2. Volve Data Set

In 2018 Equinor and the partners of the society working for Volve field, have decided to make public all the downhole and production data from the field [37]. All the data is located in a repository at the Equinor's web site and is completely available for access for researching purposes.

4.2.1. Volve Field

Volve field is located in Block 15/9 in the southern part of the Norwegian North Sea, and it is situated approximately 200 km west of Stavanger and 8 km from Sleipner Ost Field. Oil was discovered in 1993, but the development was approved in 2005, and the field and the oil is located in the middle Jurassic sandstone formations. Recoverable reserves are estimated at 78.6 million barrels of oil and 1.5 billion cubic meters of gas.

The field started production in 2008 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. The plateau production of Volve was around a daily production of 56,000 barrels of oil a day with a total recovery of 63 million barrels.

4.2.2. Wells Information

Six wells were taken to make the analysis and development of the current project. In the ***Table 2*** is possible to identify the wells and a nomenclature that is going to be used along the project

Well Date Set	Project nomenclature
USROP_A 0 N-NA_F-9_Ad	Well 1
USROP_A 1 N-S_F-7d	Well 2
USROP_A 2 N-SH_F-14d	Well 3
USROP_A 3 N-SH-F-15d	Well 4
USROP_A 4 N-SH_F-15Sd	Well 5
USROP_A 5 N-SH-F-5d	Well 6

Table 2. Complete wells data set

4.3. Data Analysis

In order to set up the ML model, some data was required. The data set used came from drilling information of six (6) wells, showed in ***Table 2***. This information is saved in a comma-separated value (csv) format archive, so the handling, selection, and processing data to execute the study was easier.

The data was pre-processed and normalized in previous research works to improve the accuracy of the model, so it can be used to feed the different ML algorithms.
As it was mentioned before, all the work that is going to be described is performed using Jupyter Notebook as a main application and Python as a programming language.

4.3.1. Importing and Visualizing the Data

The first step into the analysis is importing and visualizing the data. The raw data contains information from drilling operations organized according to the depth of the well and some parameters that are shown in **Table *3***.

Number	Variables	Units	Data Type
1	Measured Depth	[m]	float64
2	Weight on Bit	[kkgf]	float64
3	Average Standpipe Pressure	[kPa]	float64
4	Average Surface Torque	[kN.m]	float64
5	Rate of Penetration	[m/h]	float64
6	Average Rotary Speed	[rpm]	float64
7	Mud Flow In	[L/min]	float64
8	Mud Density In	[g/cm3]	float64
9	Diameter	[mm]	float64
10	Average Hookload	[kkgf]	float64
11	Hole Depth (TVD)	[m]	float64
12	USROP Gamma	[gAPI]	float64

Table 3. Features from data sets.

Using pandas, the main data frame was generated looking to visualize the data frame with all data set is going to be analyzed along the project. For example, in the **Table *4*** is shown the complete data with all its features. The process that is going to be described was done for each one of the wells in order to get the initial conditions of the data set.

	Measured Depth m	Weight on Bit kkgf	Average Standpipe Pressure kPa	Average Surface Torque kN.m	Rate of Penetration m/h	Average Rotary Speed rpm	Mud Flow In L/min	Mud Density In g/cm3	Diameter mm	Average Hookload kkgf	Hole Depth (TVD) m	USROP Gamma gAPI
0	491.033	5.842270	9440.922214	0.244047	42.864024	84.000	2784.321942	1.210000	311.15	93.780222	490.760309	150.88
1	491.185	6.241431	9499.941336	0.244047	42.315384	84.000	2784.321942	1.210000	311.15	93.979803	490.910880	150.88
2	491.222	6.241431	9499.941336	0.244047	42.315384	84.000	2784.321942	1.210000	311.15	93.979803	490.910880	150.88
3	491.338	6.368437	9313.024466	0.216931	42.894504	84.000	2784.321942	1.210000	311.15	94.079593	491.061756	146.26
4	491.341	6.368437	9313.024466	0.216931	42.894504	84.000	2784.321942	1.210000	311.15	94.079593	491.061756	146.26
...	...	...	...	...	...	...	...	...	...	...	...	...
13741	1205.755	12.455646	10986.726800	9.666982	40.614600	194.680	1927.456082	1.198264	215.90	88.119390	1013.078086	25.37
13742	1205.789	12.407112	10953.887070	9.597835	39.549934	194.571	1927.467438	1.198264	215.90	88.213283	1013.095459	25.37
13743	1205.810	12.407112	10953.887070	9.597835	39.549934	194.571	1927.467438	1.198264	215.90	88.213283	1013.106432	25.37
13744	1205.880	12.407112	10953.887070	9.597835	39.549934	194.571	1927.467438	1.198264	215.90	88.213283	1013.142703	25.37
13745	1205.999	12.407112	10953.887070	9.597835	39.549934	194.571	1927.467438	1.198264	215.90	88.213283	1013.142703	25.37

13746 rows × 12 columns

Table 4. Complete data set, Well 1 (13746 rows x 12 columns).

After having the features that are going to be taken into account, it was performed a check to verify if there are some missing values inside of the data set so in case of being found, perform a correct handling of them. Also, as it can see in **Figure *14* Figure *14*.** Probability density distributions., there are probability density distribution plots of the data of all wells to be studied. According to Fernández et al, [38] "*The drilling data is not always distributed and intuitive to understand since it is usually skewed. Nevertheless, data that follows certain distributions can be valuable, and identifying its probability distribution is critical for further analysis.*"

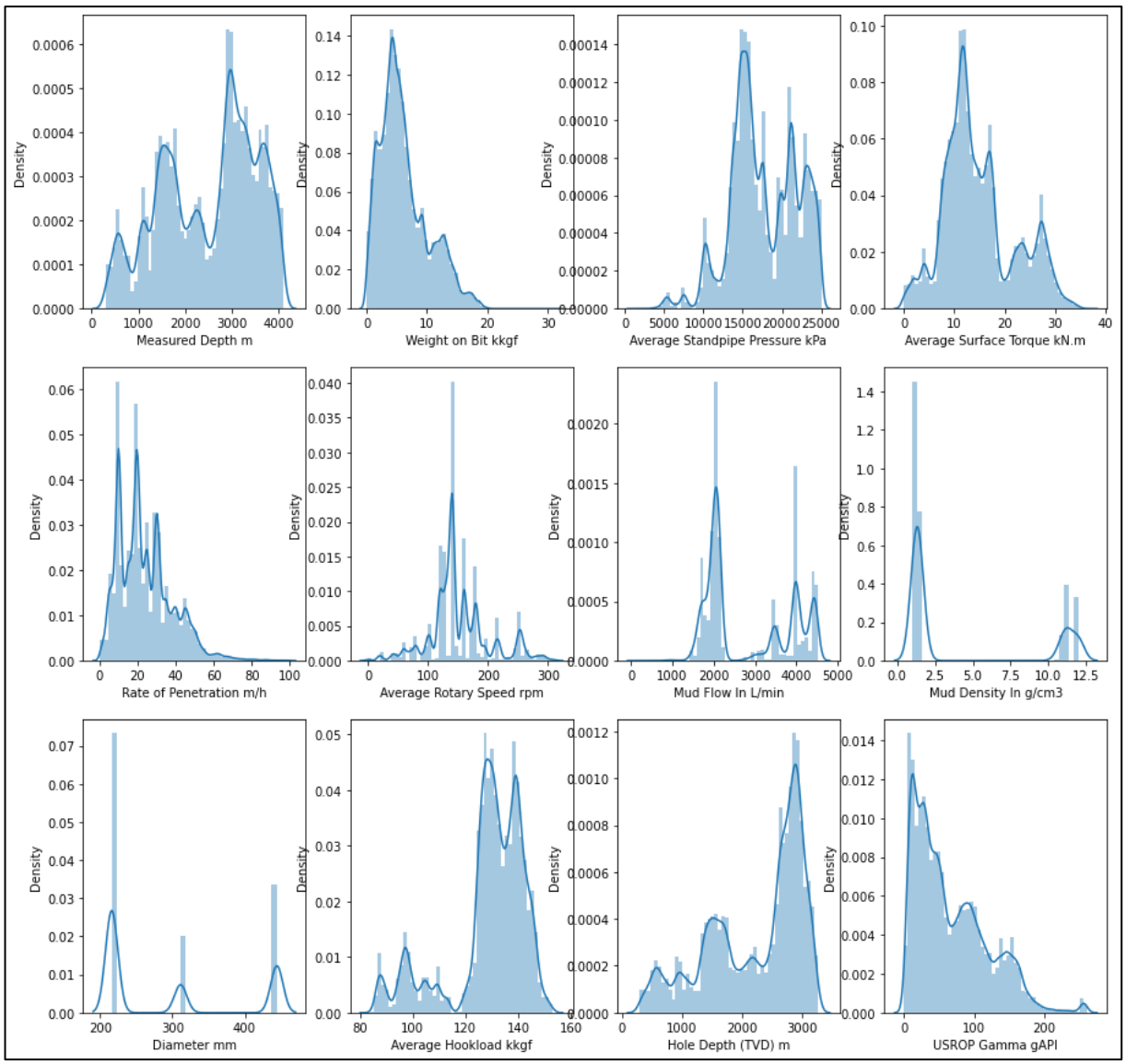

Figure 14. Probability density distributions.

On the other hand, another tool that is widely used for data visualization is the heat map, by means of which it was possible to visualize the correlation of the parameters with respect to the ROP for each of the wells. This is especially useful since it is necessary to analyze many variables and choose the ones that are related to the ROP. The heat map was performed in each one of the wells and according to the results a summary table was performed.

The **Figure *15*** presents the heat map, for the well that is used as an example, where the diagonal set of squares going from top left to bottom right are in the darkest blue, this represents the intersection between the same variables, and that is why its relationship value is the highest one. Numerically, this will represent a Pearson correlation coefficient of one, where the values can be from negative one (-1) to one (+1), being one the value related to the highest correlation between same parameters or variables. Negative values indicate that the variables tend to move in opposite directions being a clear example of no direct correlation between both of them.
The intensity of the color will decay depending on the correlation level between variables.
For the study case, the variables with darkest color found in the "Rate of Penetration" row, are the ones which are going to be selected as main features (See red square in **Figure *15***).

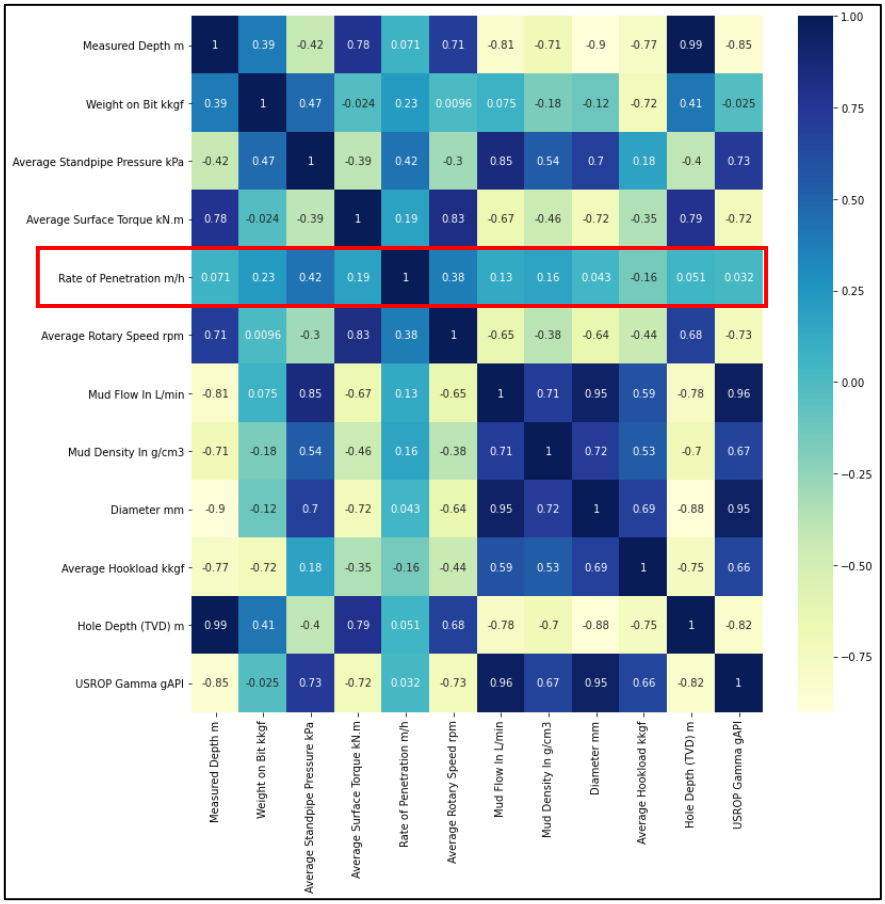

Figure 15. Heatmap, raw data of variables for Well 1

After doing the heat maps for each well, the data was collected and summarized in the **Table *5*** where all the variables are shown in the columns and the wells in the rows. According to the relationship value given by the heatmap, it was assigned a number for each parameter from 1 to 5, being "5" the one that has the highest value of correlation.

The row denotated as "Count" contains the number of times that the parameter was found as a parameter with high correlation to ROP, and the row "Sum" shows the total sum of the values assigned according to the correlation, with the aim of being able to weight and select the most relevant parameters according to the data of the wells that are available.

For example, for the heat map shown in **Figure *15*** (Well 1), the variable with the highest correlation is *Average Standpipe Pressure*, with a value of 0.42 in the heat map and five (5) in the **Table** 5. ROP dependent variables **according to heat** maps *into about different wells*. following the same criteria, six (6) variables were selected as the dependent variables, those which had more ponderation and correlation to ROP for each heat map, in the example case, they were *Average Standpipe Pressure, Average Rotary Speed, Weight on Bit, Average Surface Torque* and *Mud Density In* and leaving ROP as an independent variable. All heat maps for wells are shown in the Appendix B.2.

After choosing the variables for each well, and give a value, as a final result it was found that the selected six (6) variables to develop the ML model will be those which got the highest values.

ROP DEPENDENT VARIABLES								
WELL	**Well 1**	**Well 6**	**Well 5**	**Well 4**	**Well 3**	**Well 2**	**COUNT**	**SUM**
Measured Depth	-	-	-	5	-	3	2	**8**
Weight on Bit	3	-	-	2	1	-	3	**6**
Average Standpipe Pressure	5	3	1	-	5	-	4	**14**
Average Surface Torque	2	2	-	3	-	5	4	**12**
Average Rotary Speed	4	4	2	1	-	-	4	**11**
Mud Flow In	-	5	5	-	3	-	3	**13**
Mud Density In	1	-	-	-	4	2	3	7
Diameter	-	-	4	-	2	-	2	6
Average Hookload	-	-	-	-	-	-	0	0
Hole Depth (TVD)	-	-	-	4	-	4	2	8
USROP Gamma	-	-	3	-	-	1	2	4

Table 5. ROP dependent variables according to heat maps into about different wells.

According to experience, some of the variables taken into account in the initial data are relevant at the time of optimizing the ROP, such as *Average Hook load*, the results obtained by the analyzed data show that the highest levels of correlation are found in those that are shown in the **Table *6***. Since *Measure Depth* and *Hole Depth* refers to depth of the well, *Weight on Bit (WOB)* is going to be included into the variables to evaluate. It is important to mention than in **Table *6***, the variables are organized according to the one which got a higher ponderation being 5 the one with maximum value and one the variable that have less correlation.

5	1. Average Standpipe Pressure
4	2. Mud Flow In
3	3. Average Surface Torque
2	4. Average Rotary Speed
1	5. Measure Depth (MD) 6. Weight on Bit (WOB)

Table 6. Selected variables to develop ML model

The heat maps and well statistics for all wells can be found in the *Appendix B*.

4.3.2. Data Selection

After select the variables that are going to be taken into account to feed the model according to its correlation to ROP, it was performed a validation to verify the continuity of the information and its behavior for each of the wells. The ***Figure 16***, shows the measure depth in the "x" axis vs the selected variables in the "y" axis for the Well 1 (Other wells' plots can be found in the *Appendix B.)*

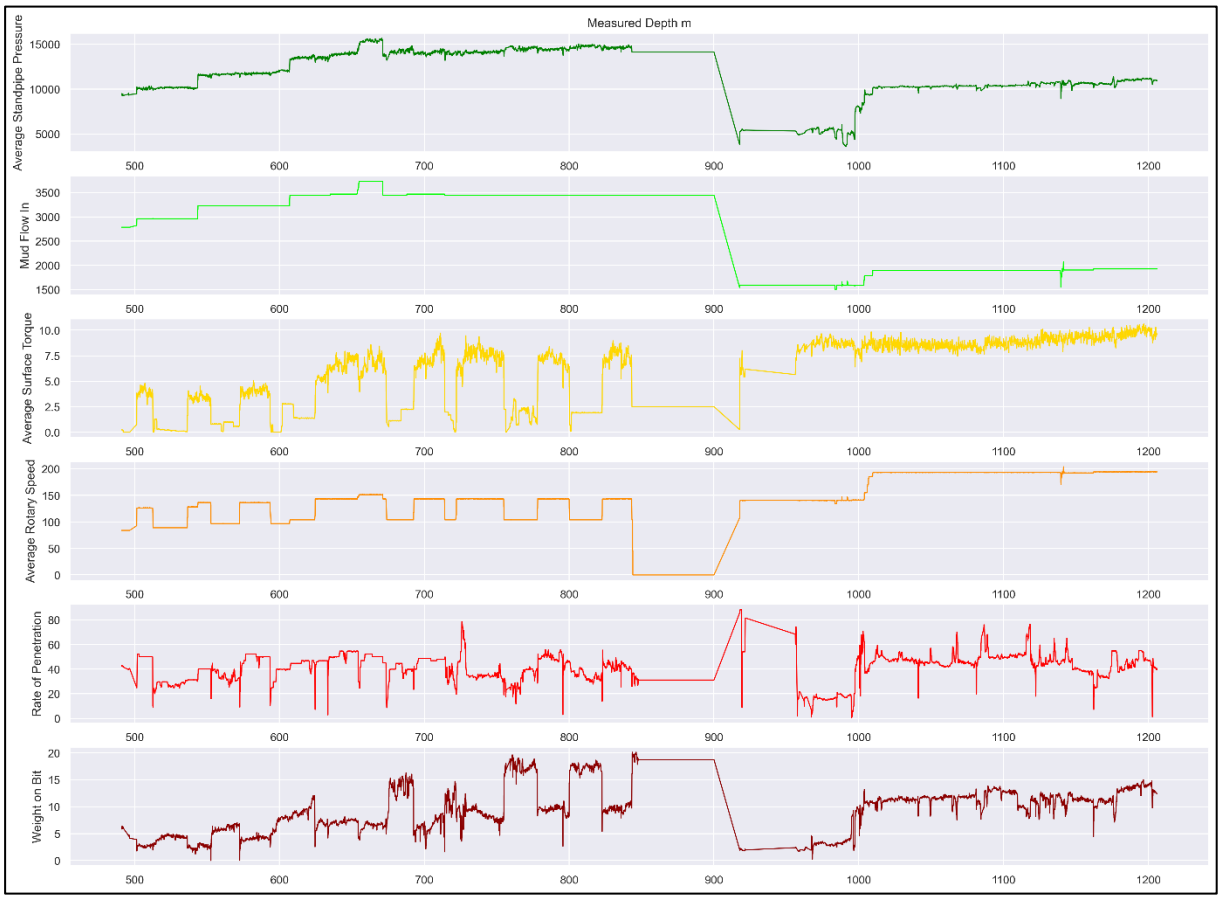

Figure 16. Measure depth vs. Selected variables (Well 1)

In the plots, it can be seen that despite the fact that the *Average Rotary Speed* is oscillating between 100 to 150 rpm, the ROP tends to have the same tendency since

they are directly proportional parameters as it was mentioned in Chapter 2. At the end of the well, between 1000 m to 1200 m there is a high value of *Average Rotary Speed,* but the ROP average is the same that in the shallower section of the well, so we can assume that they were drilling a hard formation probably with a tricone bit.
Taking a deep look into the WOB behavior vs the ROP, normally if the WOB is high, the ROP tends to increase unless there is an inefficient hole cleaning related to a higher rate of cuttings generation, or because of complete penetration of a bit's cutting elements into the formation being drilled, without clearance for fluid bypass [8], which can be a reason to explain the behavior of the plots at 680 m depth, where the WOB values are high, but the ROP kept constant or even with a small decrease of the rate with respect to the previous behavior and in turn, at the same depth, there is a peak in the average standpipe pressure which can explain an inefficient hole cleaning of the wellbore.

In the previous analysis it can be seen a summary of how the parameters are related to the ROP, and the way in which a good analysis of them in real time can lead to good decision making on the way.

4.4. ROP modeling

As it was mentioned before, Chapter 2, a data driven model is going to be implemented, using the data measured while drilling to predict ROP. The model building process incorporates feature engineering to ensure that the model is built in a robust fashion. Feature engineering is imperative to the success of the model since the input parameters used to build the model completely determine its outcome.

The model will be built as a function of feature vectors (pre-selected variables) to determine or predict the property "y", being ROP for this study work. The RF algorithm was used since the RF regression is a supervised learning algorithm that uses the ensemble learning method for regression, it is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.

4.4.1. Machine Learning Implementation

To perform the Machine Learning procedures of the project, Python [39] was used as the main tool, where different techniques were implemented using the selected features (See **Table *7***). Also, Scikit Learn [21], which is a module that contains RF algorithm, was used to implement the regressor models. The parameters or variables used as inputs to feed the model and compare its performance are shown in **Table *7***.

Number	Variables	Units	Data Type
1	Average Standpipe Pressure	[kPa]	float64
2	Mud Flow In	[L/min]	float64
3	Average Surface Torque	[kN.m]	float64
4	Average Rotary Speed	[rpm]	float64
5	Measured Depth	[m]	float64
6	Weight on Bit	[kkgf]	float64

Table 7. Selected features (variables).

As a first step, the relevant libraries were imported (See *Appendix A)* and additionally some model and tools from Scikit Learn. The **Figure *17***, show how the data was imported from the .CSV file to build a data frame getting a better visualization and dimension of the data that was used to feed up the model. The rows were organized, leaving the ROP measure at the end to avoid it being include into the prediction data.

```
In [48]:
df = pd.read_csv('USROP_A 0 N-NA_F-9_Ad.csv')
del df['Unnamed: 0']

df = df[['Average Standpipe Pressure kPa', 'Mud Flow In L/min','Average Surface Torque kN.m','Average Rotary Speed rpm','Mea

#print(df)
X = df.iloc[:,0:6]
y = df.iloc[:,11] #ROP
df = df.iloc[: , :12]

df
```

Out[48]:

	Average Standpipe Pressure kPa	Mud Flow In L/min	Average Surface Torque kN.m	Average Rotary Speed rpm	Measured Depth m	Weight on Bit kkgf	Hole Depth (TVD) m	Mud Density In g/cm3	Diameter mm	Average Hookload kkgf	USROP Gamma gAPI	Rate of Penetration m/h
0	9440.922214	2784.321942	0.244047	84.000	491.033	5.842270	490.760309	1.210000	311.15	93.780222	150.88	42.864024
1	9499.941336	2784.321942	0.244047	84.000	491.185	6.241431	490.910880	1.210000	311.15	93.979803	150.88	42.315384
2	9499.941336	2784.321942	0.244047	84.000	491.222	6.241431	490.910880	1.210000	311.15	93.979803	150.88	42.315384
3	9313.024466	2784.321942	0.216931	84.000	491.338	6.368437	491.061756	1.210000	311.15	94.079593	146.26	42.894504
4	9313.024466	2784.321942	0.216931	84.000	491.341	6.368437	491.061756	1.210000	311.15	94.079593	146.26	42.894504
...	...	...	...	...	...	...	...	...	...	...	...	...
13741	10986.726800	1927.456082	9.666982	194.680	1205.755	12.455646	1013.078086	1.198264	215.90	88.119390	25.37	40.614600
13742	10953.887070	1927.467438	9.597835	194.571	1205.789	12.407112	1013.095459	1.198264	215.90	88.213283	25.37	39.549934
13743	10953.887070	1927.467438	9.597835	194.571	1205.810	12.407112	1013.106432	1.198264	215.90	88.213283	25.37	39.549934
13744	10953.887070	1927.467438	9.597835	194.571	1205.880	12.407112	1013.142703	1.198264	215.90	88.213283	25.37	39.549934
13745	10953.887070	1927.467438	9.597835	194.571	1205.999	12.407112	1013.142703	1.198264	215.90	88.213283	25.37	39.549934

13746 rows × 12 columns

Figure 17. Initial data Well 1 (Screenshot from the main code)

After importing the data, it was filtered leaving only the selected features and having a data frame with 13746 rows and 6 columns (See **Figure *18***).

Out[12]:

	Average Standpipe Pressure kPa	Mud Flow In L/min	Average Surface Torque kN.m	Average Rotary Speed rpm	Measured Depth m	Weight on Bit kkgf
0	9440.922214	2784.321942	0.244047	84.000	491.033	5.842270
1	9499.941336	2784.321942	0.244047	84.000	491.185	6.241431
2	9499.941336	2784.321942	0.244047	84.000	491.222	6.241431
3	9313.024466	2784.321942	0.216931	84.000	491.338	6.368437
4	9313.024466	2784.321942	0.216931	84.000	491.341	6.368437
...	...	...	...	...	...	...
13741	10986.726800	1927.456082	9.666982	194.680	1205.755	12.455646
13742	10953.887070	1927.467438	9.597835	194.571	1205.789	12.407112
13743	10953.887070	1927.467438	9.597835	194.571	1205.810	12.407112
13744	10953.887070	1927.467438	9.597835	194.571	1205.880	12.407112
13745	10953.887070	1927.467438	9.597835	194.571	1205.999	12.407112

13746 rows × 6 columns

Figure 18. Selected variables and data frame (Screenshot from the main code).

4.4.2. Splitting Data

One of the most important things to start is splitting the data to train and test the model, as it was mentioned in the theoretical background. Working with machine learning models, the data should be divided into two or three parts to avoid overfitting and model bias, it could be training set (which is normally the largest one), testing set and in some cases the validation set.
For the data used to build the model, it was taking into account a ratio of splitting data of 80-20 percent, where 80 percent was used to train the model and 20 percent to test it.
Graphically, the split ratio could be represented in the ***Figure 19***, where the red points are the values taken as a training data and the black ones are the testing data points.

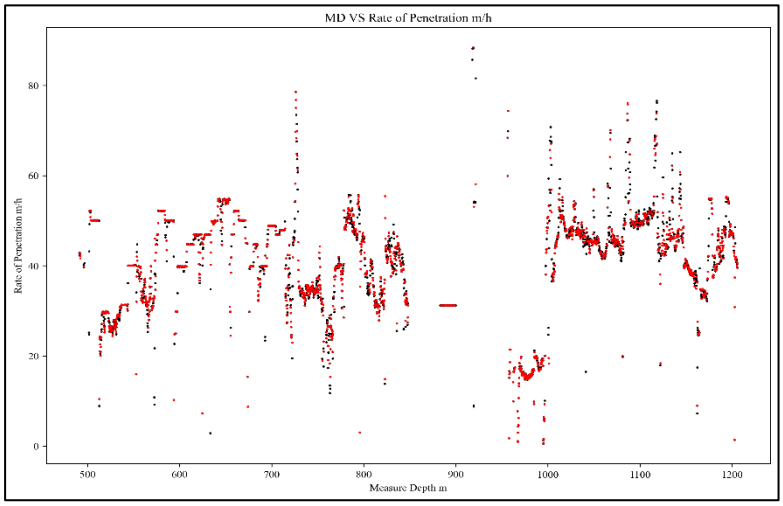

Figure 19. Example testing and training data.

Once the percentages of information to be assigned to the training and testing data were defined, a random sampling was considered to define the performance of the model.

The **Table *8*** summarize the number of values and how the data was split into training and data set. For the inputs, there is 10996 rows taken as a training data set times six (6) columns which refers to the selected features. On the other hand, there is the 20% of the total data which corresponds to the testing data set (2750 rows) times the six columns mentioned before.
As the ROP is the output, and, in this case the data that will be taken into account to compare how the accuracy of the model regarding to its prediction is, it has only one column which is in turn the value of the penetration rate.

Type	Amount (rows)	Amount (Columns)
Train data set (input)	10996	6
Test data set (input)	2750	6

Table 8. Testing and training data set.

Random Sampling basically consists of randomly select 80% of the data and assign them for the training set, and the 20% residual for the testing set. Random sampling is a good way to select how to split the data, since the behavior of the ROP tends to be "constant" according to the formations being drilled, type of bit or simply depth of the well, so if the main objective is predicting the ROP involving many parameters or environments, a random sampling is one of the best options to split the data.
On the other hand, one of the main disadvantages 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 some points [6], for this reason, the accuracy of the model was tested with never-before-seen data.

4.4.3. Grid Cross-validation

For each one of the regressors that are going to be used, the grid cross-validation was performed because grid search with CV is a commonly used method to adjust the parameters. The first step was specifying the parameters to search, then grid search CV will perform all the necessary model fits.
Fitting the grid search CV object, not only searches for the best parameters, it also automatically fits a new model on the whole training data set with the parameters that yielded the best cross-validation performance.
Parameters that were found, are called as "best parameter" and the best cross validation accuracy (the mean accuracy over the different splits for this parameter setting) is called "best score".

4.4.4. Gradient Boosting Regressor.

The first one of the regressors to be used was gradient boosting regressor (See code on Appendix A.3).

The parameters evaluated were:

- Number of estimators: 50, 200, 500 or 1000
- Learning Rate: 0.01 or 0.1
- Maximum depth of the three: 1, 2 or 4
- Subsample: 0.5, 0.75 or 1
- Random State: 1

Using the random sampling, the ***Figure 20***, shows the results of the GB regressor when randomly selected data, and it is possible to see the difference before and after the hyperparameter tuning is set, starting with a coefficient of determination of 0.8999

before and ending up in 0.9969, after setting the hyperparameter tuning, which means that this model has a high prediction accuracy.
The search uses the suggested parameters mentioned above, but also includes other hyperparameters of the model, the result of this search was implemented in the model.

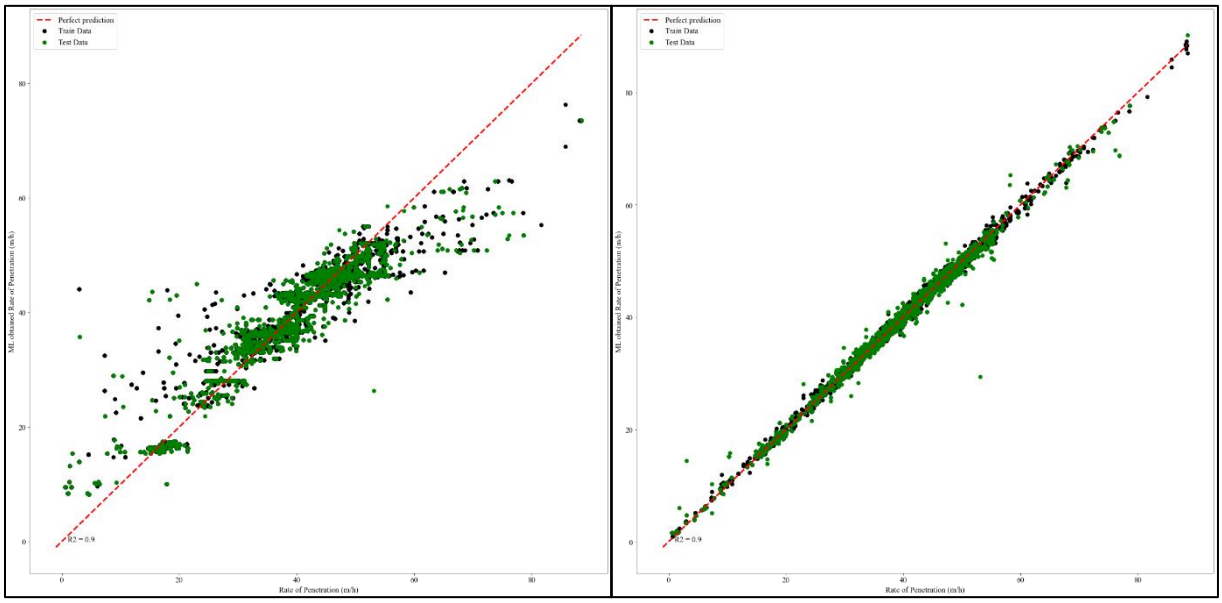

Figure 20. Gradient Boosting Regressor

As a result of the grid search CV, the best parameters were learning rate: 0.1, max depth: 4, number of estimators: 1000, random state: 1, and subsample: 0.5. In the ***Figure 21***, it can be observed the difference between before and after and how the model fits the points after adjusted the hyperparameters.

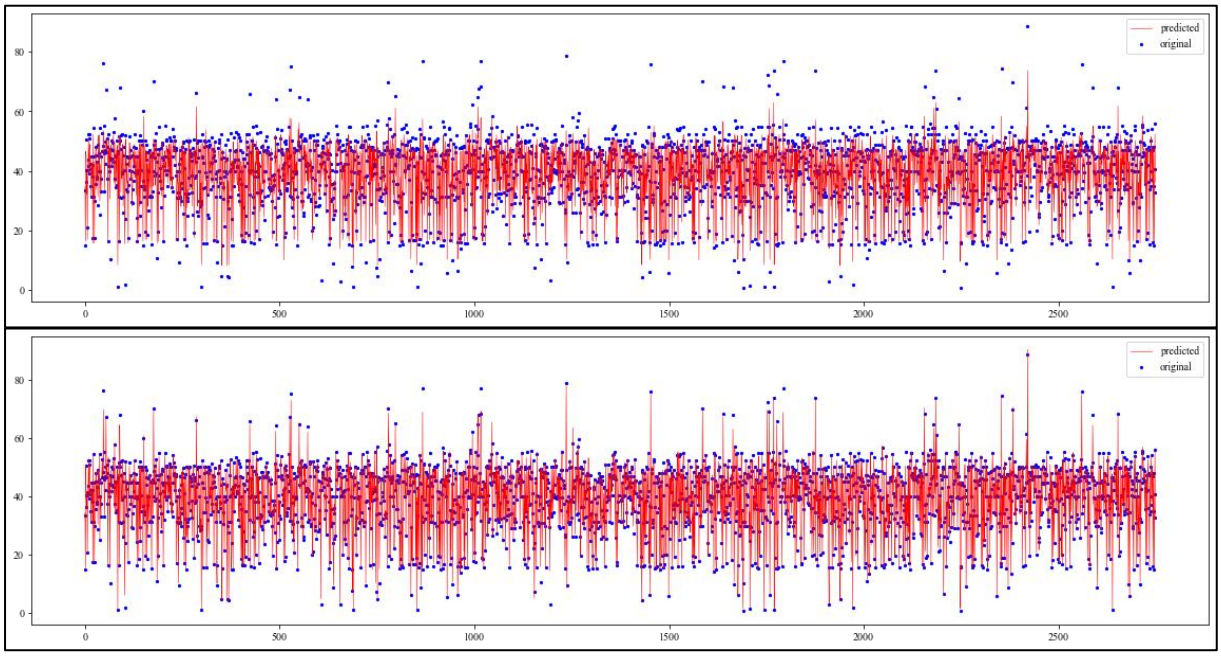

Figure 21. Predicted vs. Original Testing Data Set GB.

4.4.5. Random Forest Regressor

When the RF regressor was evaluated, the parameters evaluated were just two:

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

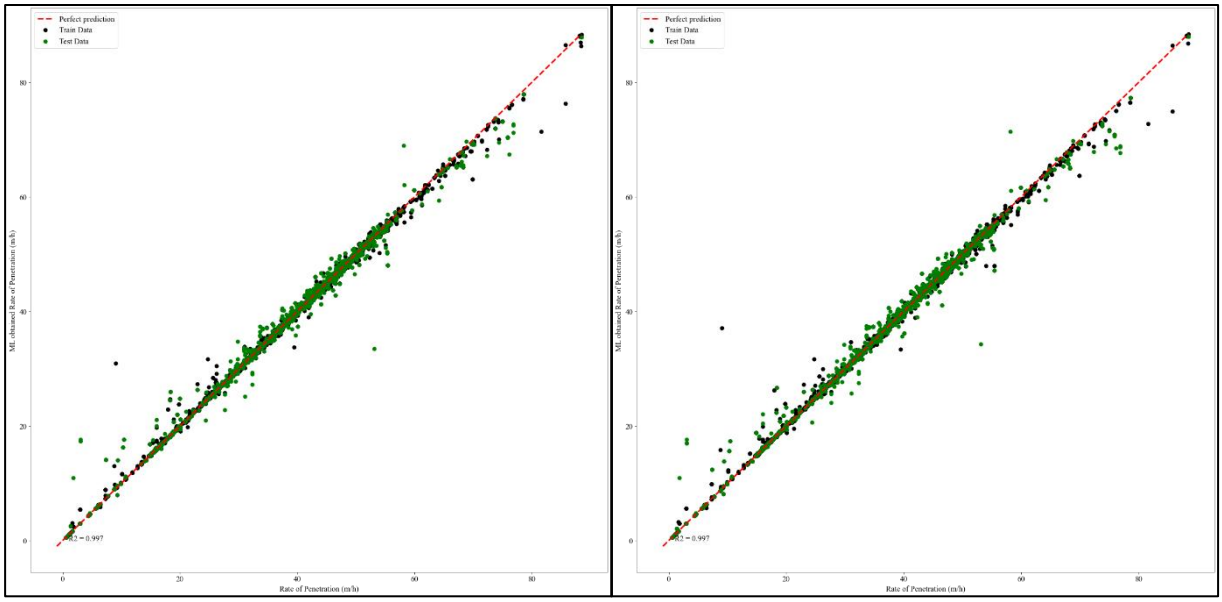

Figure 22. Random Forest Regressor

The ***Figure 22***, shows the result of the RF regressor when randomly selected data, and it is possible to see the difference before and after the hyperparameter tuning is set, starting with a coefficient of determination of 0.9973 before and ending up with a coefficient of determination of 0.997 which means that this model has a high prediction accuracy too.

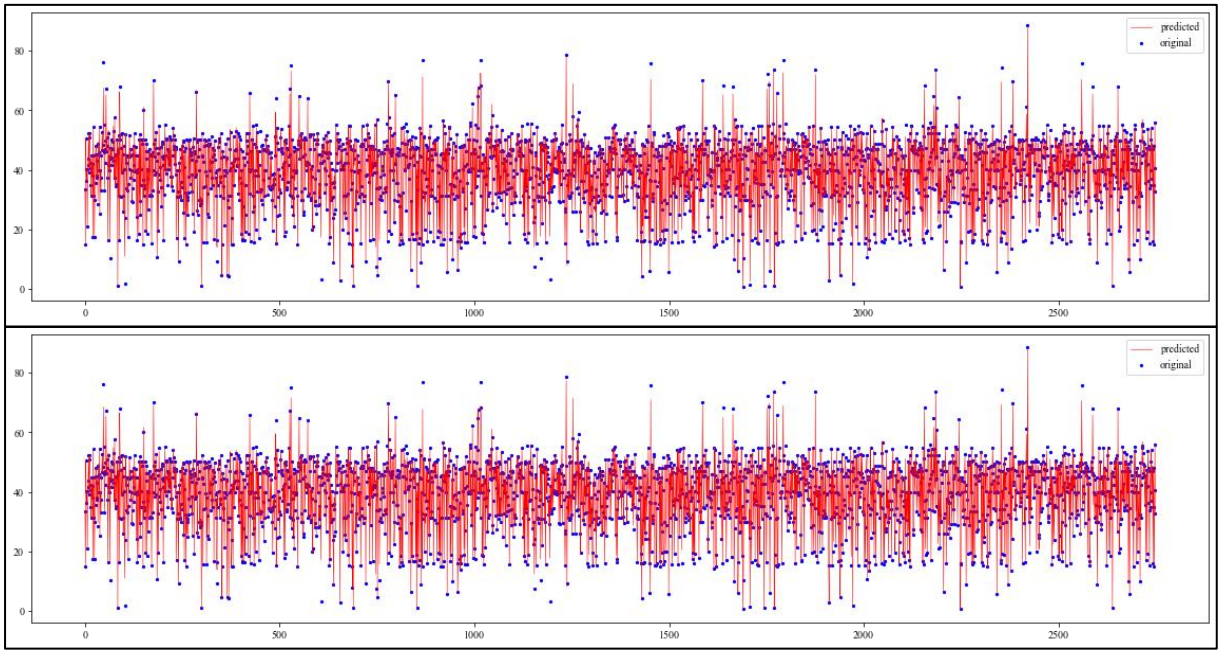

Figure 23. Predicted vs. Original Testing Data Set RF.

Having a result of the best parameters, a max depth: 30, number of estimators: 50 and a random state:0. In the ***Figure 23***, it can be observed the difference between before and after and how the model fits the points after adjusted the hyperparameters.

4.4.6. Multi-Layer Perceptron Regressor

Talking about the MLP regressor, the parameters evaluated were:

- Hidden Layer Sizes: (50,50,50), (50,50,100) or (100,1).
- Activation: Relu, tanh, or logistic
- Alpha: 0.0001 or 0.05
- Learning rate: constant or adaptive
- Solver: adam

The ***Figure 24***, shows the result of the MLP regressor when randomly selected data, the coefficient of determination obtained in this case is lower than the one from previous regressors, starting with a coefficient of determination of 0.56165 before and ending up with a coefficient of determination of 0.6032

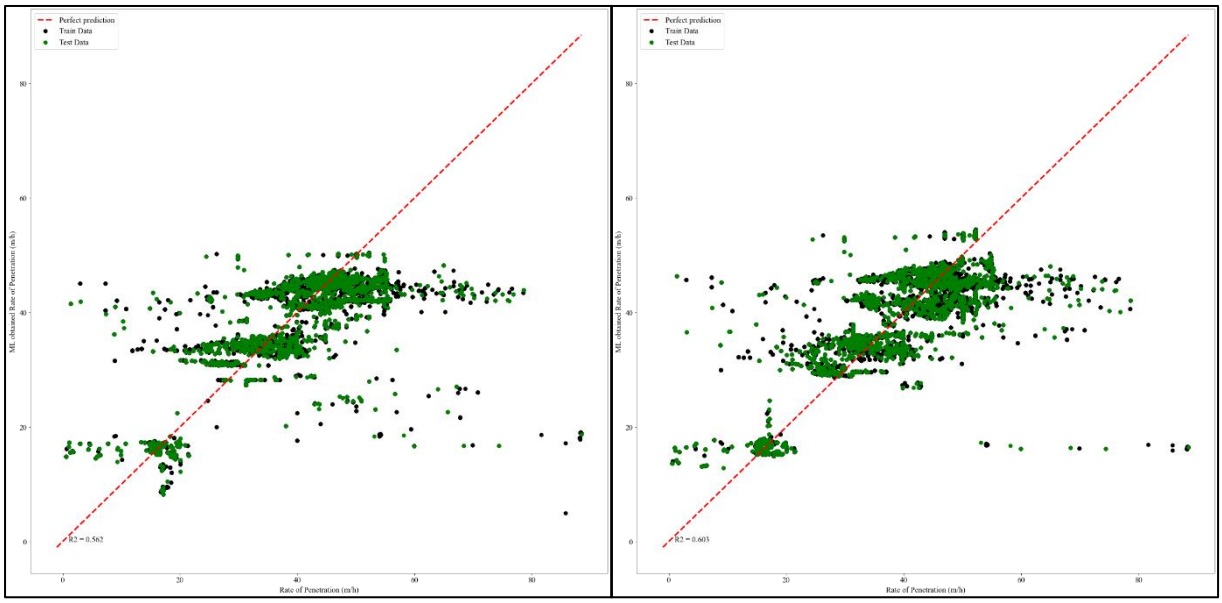

Figure 24. Multi-Layer Perceptron Regressor

Having a result of the best parameters, Hidden Layer Sizes: (50,50,50), activation: Relu, alpha: 0.05, learning rate: adaptive and solver: adam.

In the ***Figure 25***, it can be observed the difference between before and after and how the model fits the points after adjusted the hyperparameters.

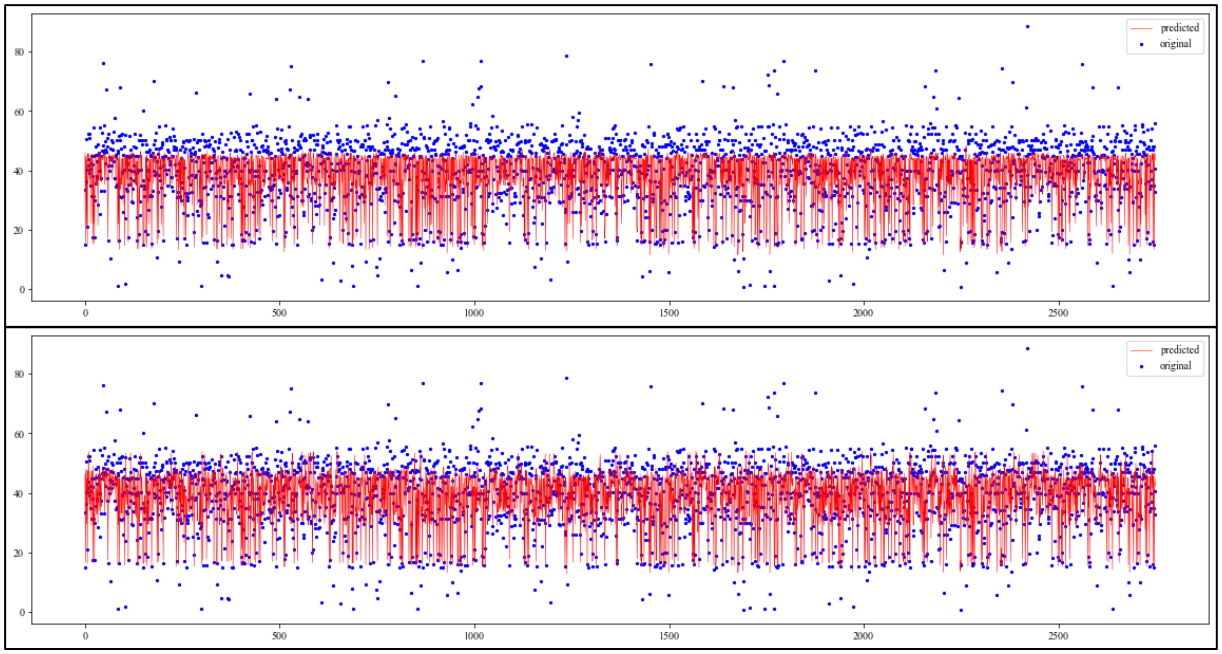

Figure 25. Predicted vs. Original Testing Data Set MLP.

4.4.7. AdaBoost Regressor

AdaBoost Regressor was evaluated under the follow parameters:

- Number of estimators: 500, 1000 or 2000
- Learning rate: 0.001, 0.01 and 0.1
- Random State: 1

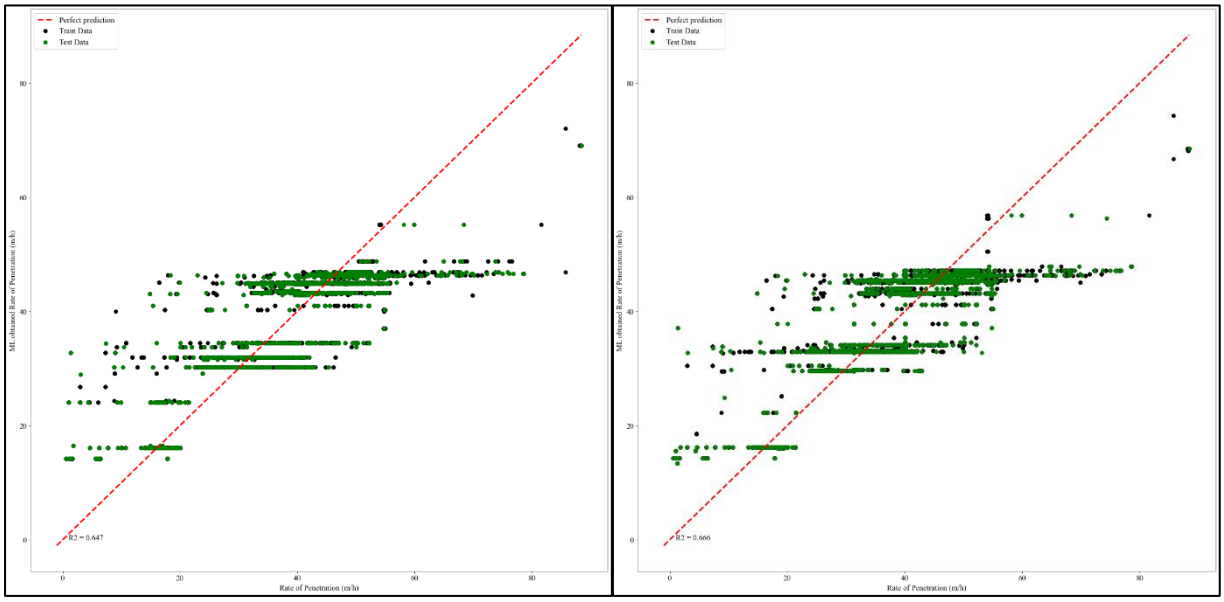

Figure 26. AdaBoost Regressor

The ***Figure 26***, shows the result of the AdaBoost regressor when randomly selected data, the coefficient of determination obtained in this case starts with a coefficient of determination of 0.647 before and ending up with a coefficient of determination of 0.666.

The result of the grid search CV gave a result of the best parameters, learning rate: 0.001, number of estimators: 2000 and a random state: 1.

In the ***Figure 27***, it can be observed the difference between before and after and how the model fits the points after adjusted the hyperparameters.

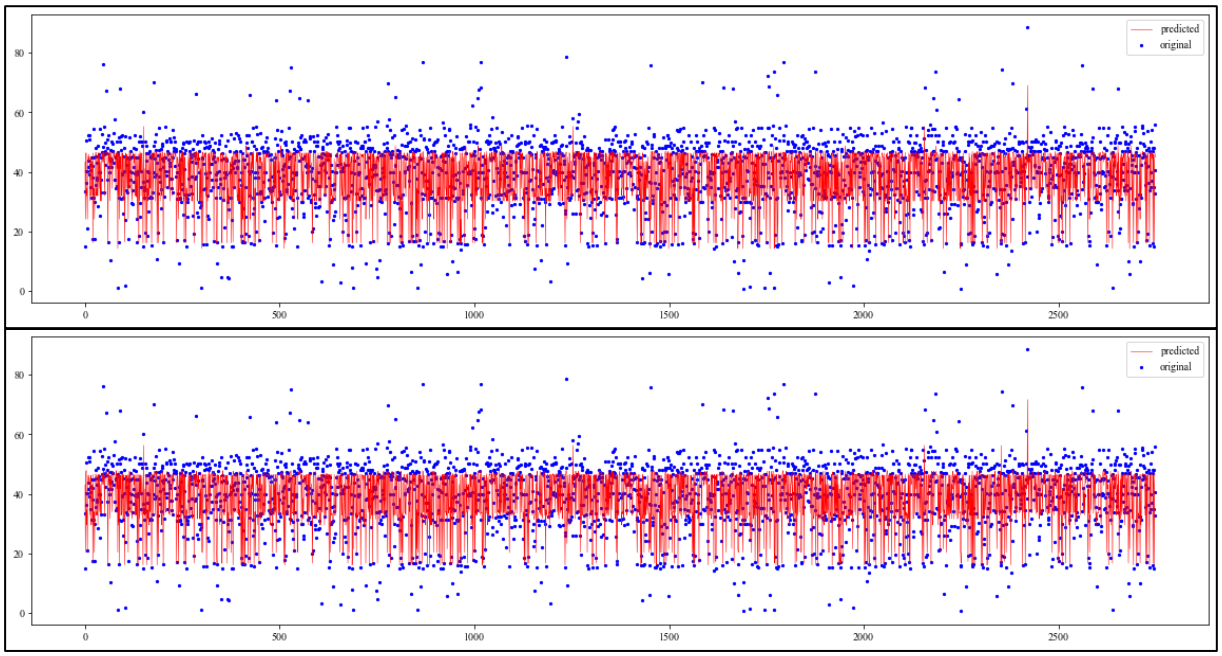

Figure 27. Predicted vs. Original Testing Data Set AdaBoost

4.4.8. K-Neighbors Regressor

Same as the other regressors evaluated before, the K-Neighbors regressor was evaluated under just the following parameters:

- Number of neighbors: 2, 3, 4, 5 or 6.
- Weights: uniform or distance.

Using the random sampling, the ***Figure 28***, shows the results of the KN regressor when randomly selected data, and it is possible to see the difference before and after the hyperparameter tuning is set, starting with a coefficient of determination of 0.939 before and ending up in 0.997, after setting the hyperparameter tuning, which means that this model has a high prediction accuracy.

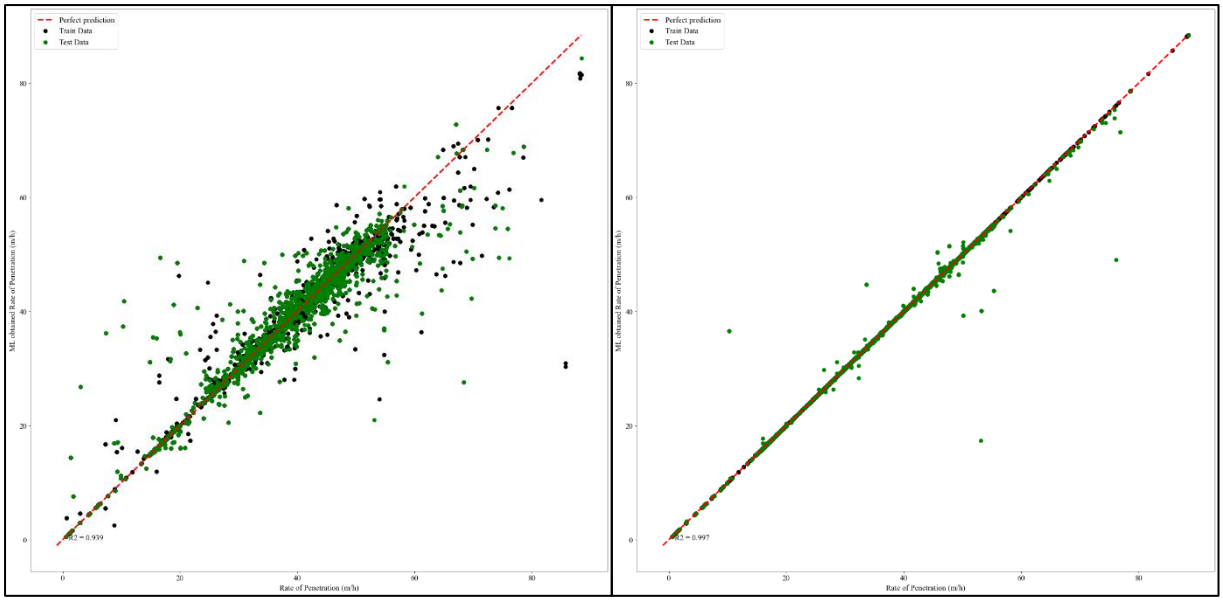

Figure 28. K-Neighbors Regressor.

The search uses the suggested parameters mentioned above, but also includes other hyperparameters of the model, the result of this search was implemented in the model.

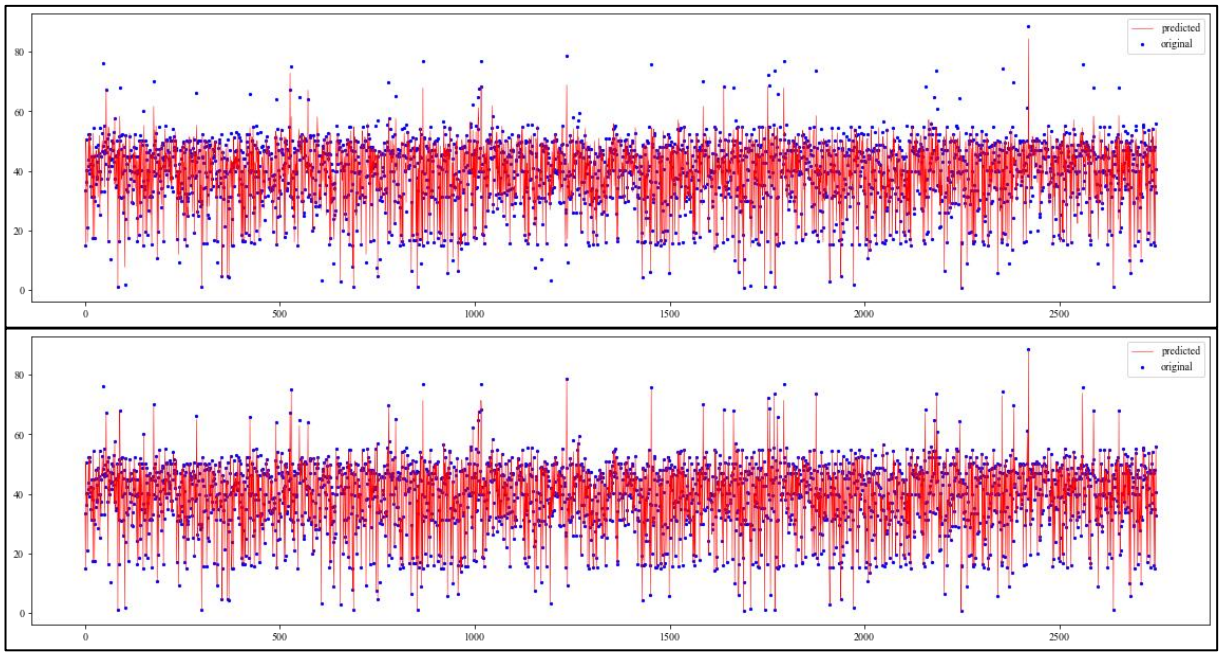

Figure 29. Predicted vs. Original Testing Data Set K-Neighbor

After the best parameters were found being a number of neighbors: 4 and weight: distance, the ***Figure 29*** shows the difference between before and after and how the model fits the points after adjusted the hyperparameters.

4.4.9. Linear Regression

The linear regression was the last one being evaluated. In the ***Figure 30***, it is possible to observe how is the correlation between the testing data and the prediction performed by the model.

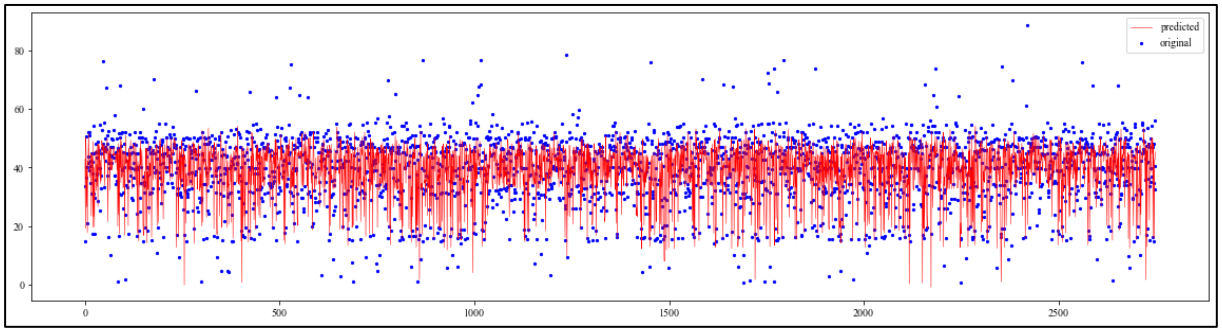

Figure 30. Predicted vs. Original Testing Data Set Linear Regression

In this case there were not hyperparameter tuning done, and the coefficient of determination has a value of 0.5774 which means that this regressor has a low prediction accuracy. See ***Figure 31***.

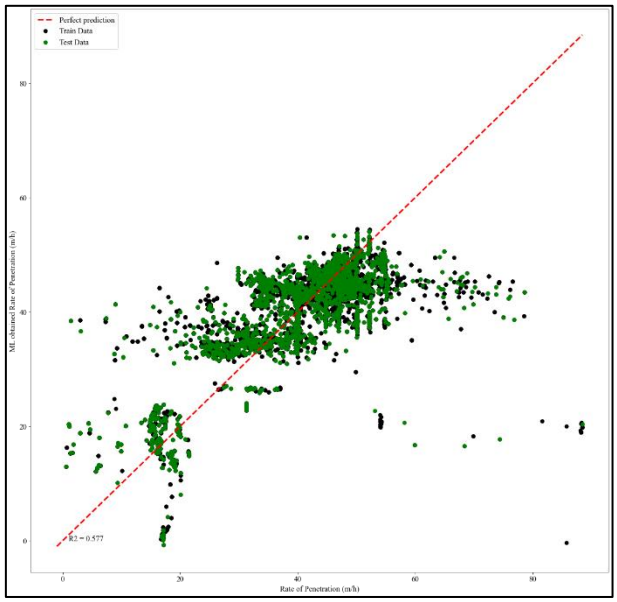

Figure 31. Linear Regression

4.5. Metrics

Some of the metrics mentioned in the Chapter 2, were evaluated for each one of the regressions performed in order to have a better understanding of the results and accuracy of the models to predict data. The results were summarized in the **Table 9**. Taking a look into the coefficients of determination (R2) the one with highest accuracy is the K-Neighbors regressor, having a coefficient of 0.996, since they memorized points instead of created correlations and the one with the lowest one is MLP regressor with a coefficient value of 0.603.

Regressor	Gradient Boosting	Random Forest	MLP	AdaBoost	K-Neighbors	Linear
Mean Absolute Error	0,42225	0,15859	5,253	5,077843	0,03804	5,60324
Mean Absolute Percentage Error	0,014704	0,00669	0,23884	0,228753	0,001269	0,26654
R2	0,99693	0,997	0,60321	0,666618	0.996897	0,57741
Mean Squared Error	0,43971	0,42938	56,8379	47,8187	0,4444921	60,5336
Root Mean Squared Error	0,66311	0,65527	7,53909	6,915106	0,666702	7,78033
Median Absolute Error	0,29761	0,01883	4,09340	3,711090	0,00000	4,340323

Table 9. Regression metrics of the trained model

Chapter 4.
Results and Discussion

5.1. Results

As testing the trained model with unseen data, to verify its accuracy, is one of the main objectives of the project, 5 of the 6 wells data set were used as a test data set. In that order of ideas Well 2, 3, 4, 5 and 6 were used to test the trained and adjusted model. The distribution of probability of each one of the wells is shown in Appendix B and along this chapter, the Well 2 is going to be used as an example to show the procedure step by step. By the end, comparative results and metrics are shown to verify the results and accuracy of the model.

5.1.1. Metrics Analysis

In the ***Figure 32***, is possible to identify which one of the models have a better accuracy according with the data used for training and testing it. In those cases, the three (3) best options to evaluate the model with never-before-seen data, it will be Random Forest, Gradient Boosting and K-Neighbors Regressor.

In the ***Table 1***, a summary of advantages and disadvantages was summarized, where it was said that the decision tree regression and random forest models work well being so accurate but sometimes can result in overfitting. The ***Figure 32*** confirms it, since the highest values of the coefficient of determination bellow to these types of regressions.

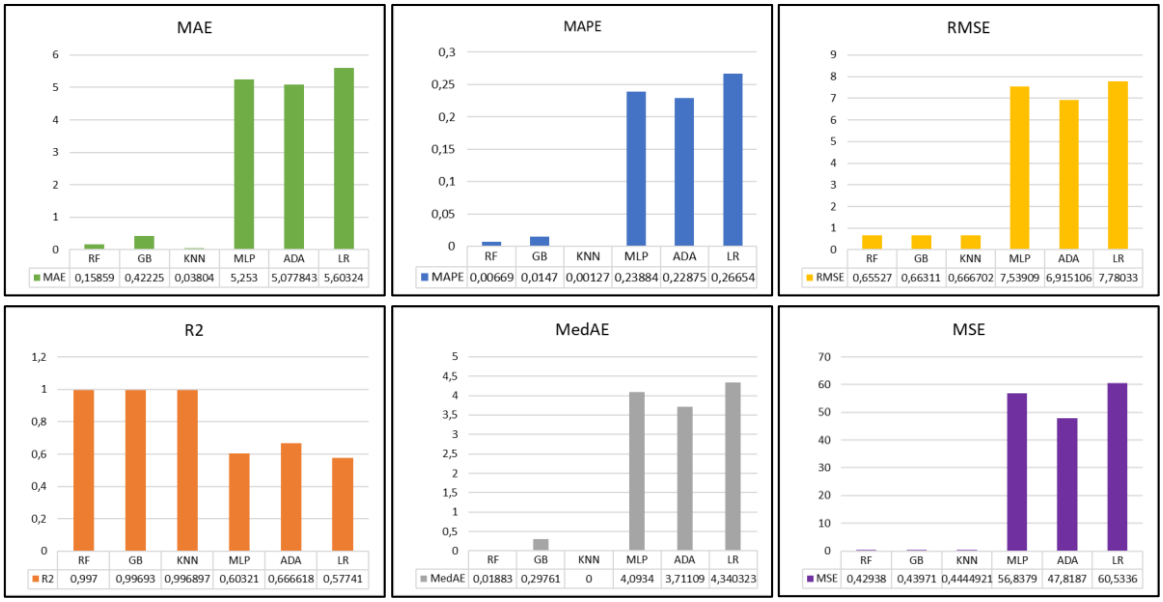

Figure 32. Metrics Analysis

5.1.2. Testing the model with other wells data set

In order to evaluate the accuracy of the prediction given by the model, it is important to test the defined model with never-before-seen data, 100% new data, in this case the data is provided by other wells drilled nearby the well that was used to train the model. After the model was trained with the data set from Well 1, and with a working ratio of 80% training data size vs 20% testing data size, the program was run for each one of the models according to each well data set, using the 100% of the data for testing the trained model, the results that are going to be shown below are just for one of the wells analyzed, to see the other wells' data plots, see Appendix B.4.

The well data set "*USROP_A 1 N-S_F-7d*" which is the Well 2, is going to be used as an example to show the procedure to evaluate the model. First of all, the data was imported keeping the same structure as the training data set. After importing the data, just a small test was run in order to check the difference between running a predetermined model, without hyperparameters defined, and running the model tested with the previous data set.

The ***Figure 33*** shows that comparison, having in the right side the one without the trained model, and in the right side, the one which was evaluated with the defined model.

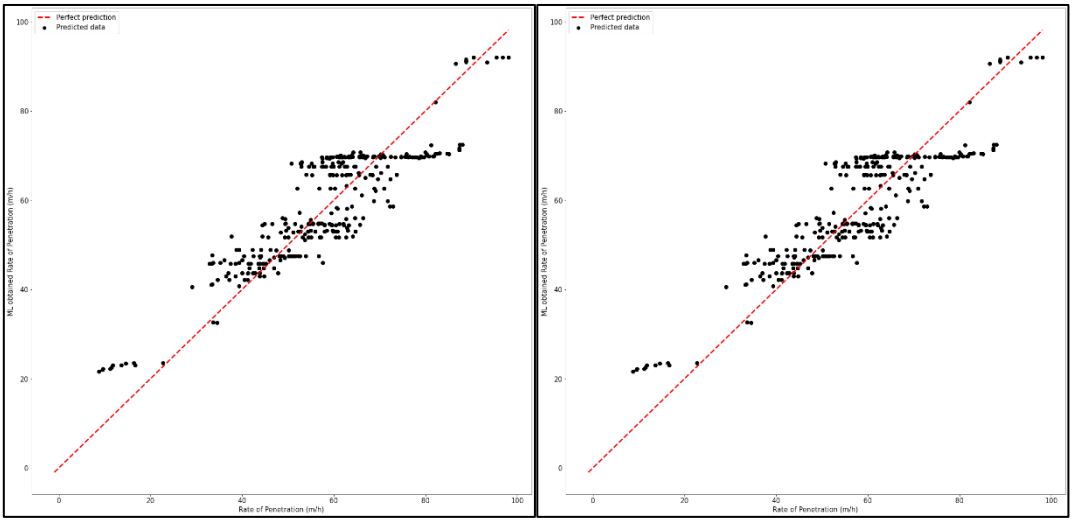

Figure 33. AdaBoost regressor model prediction Well 2

After it, the predicted data was plotted to find the accuracy to the model. In the ***Figure 34***, is possible to see the original values for ROP (field/operational data) in color green, the prediction performed without the model implementation in black color, and in purple, the predicted data provided by the model after performing the hyperparameter tunning (HT).

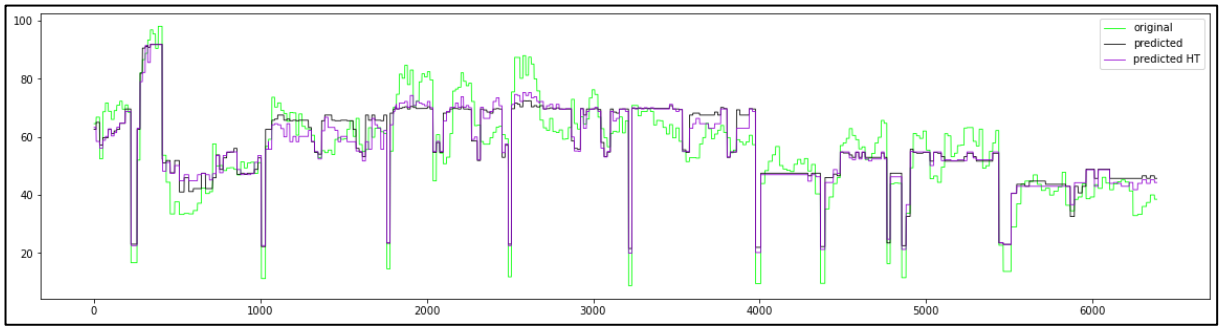

Figure 34. AdaBoost regressor predicted data set Well 2

For this case, the model gave a coefficient of determination of 0.8186. Perhaps the lines are not so close, it is possible to determine that the trend is constant which can result in a good but not perfect prediction.
After doing this procedure with each one of the wells, and each one of the regressions evaluated, the predictions were plotted to define graphically its behavior and accuracy. (See ***Figure 35***, ***Figure 36***, ***Figure 37***, ***Figure 38*** and ***Figure 39***). In the plots is possible to identify and compare the accuracy of the models, and how each one of the evaluated regressions, fits or does not fit with the predicted ROP.

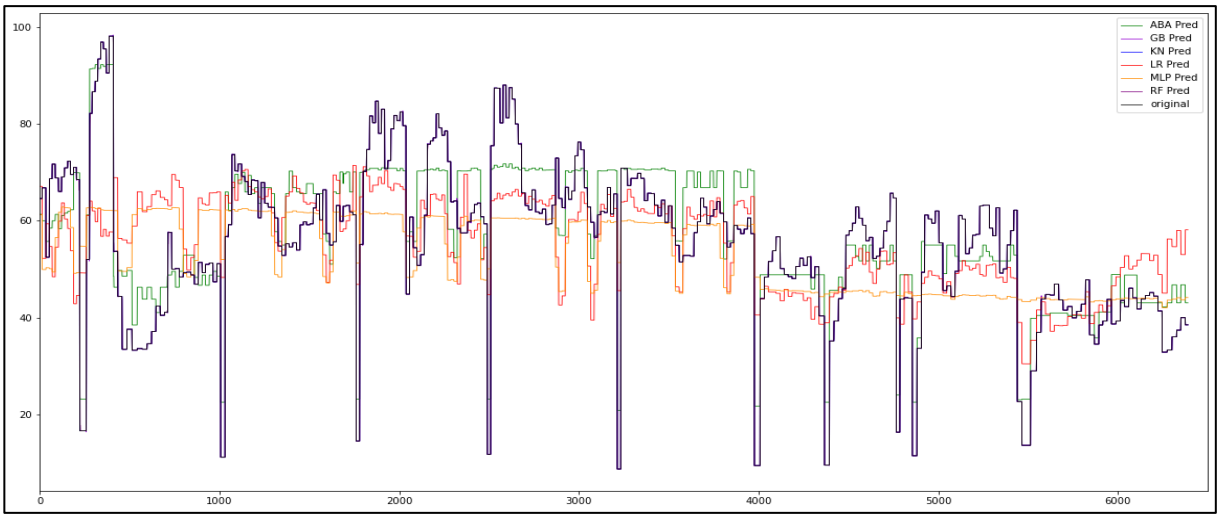

Figure 35. ROP predictions Well 2

Due to the depth of the Well 2, in the ***Figure 35*** is possible to see the difference between the accuracy of each one of the predictions given by the different models, for example the AdaBoost regressor, represented in green color, is keeping the trend of the original prediction (color black), but the values are not as accurate as the gradient boosting regressor prediction (pink color) that is even hidden by the perfect accuracy. It is important to mention that one of the most relevant things when talking about machine learning, is having as much data as possible to improve the accuracy and applicability of the model.

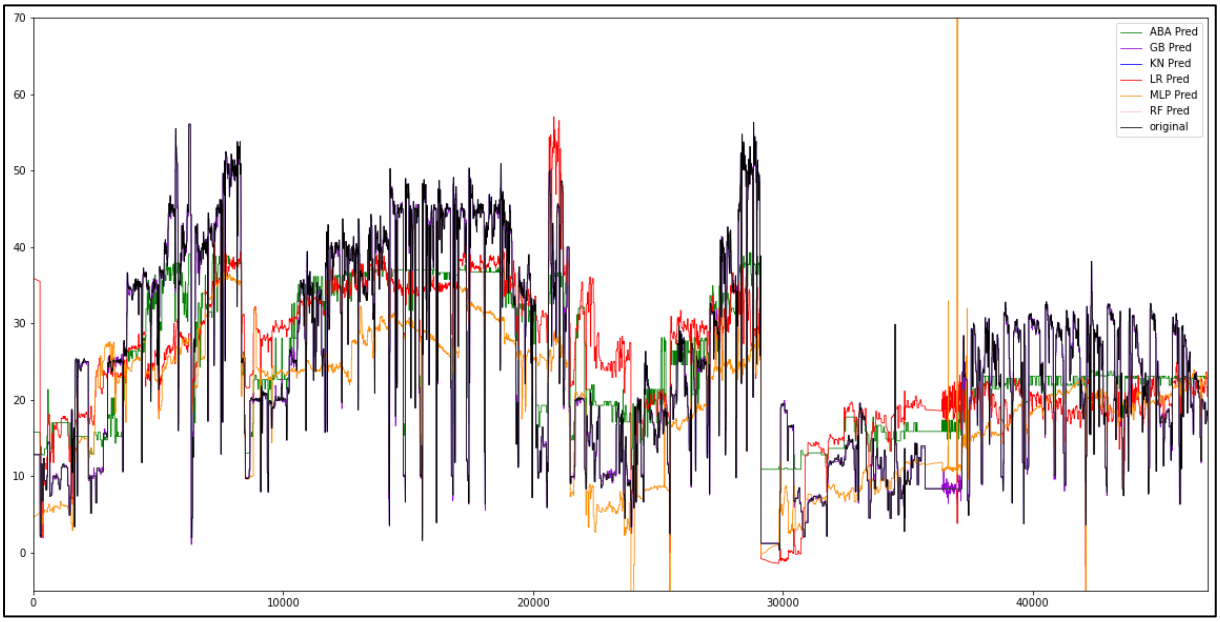

Figure 36. ROP predictions Well 3

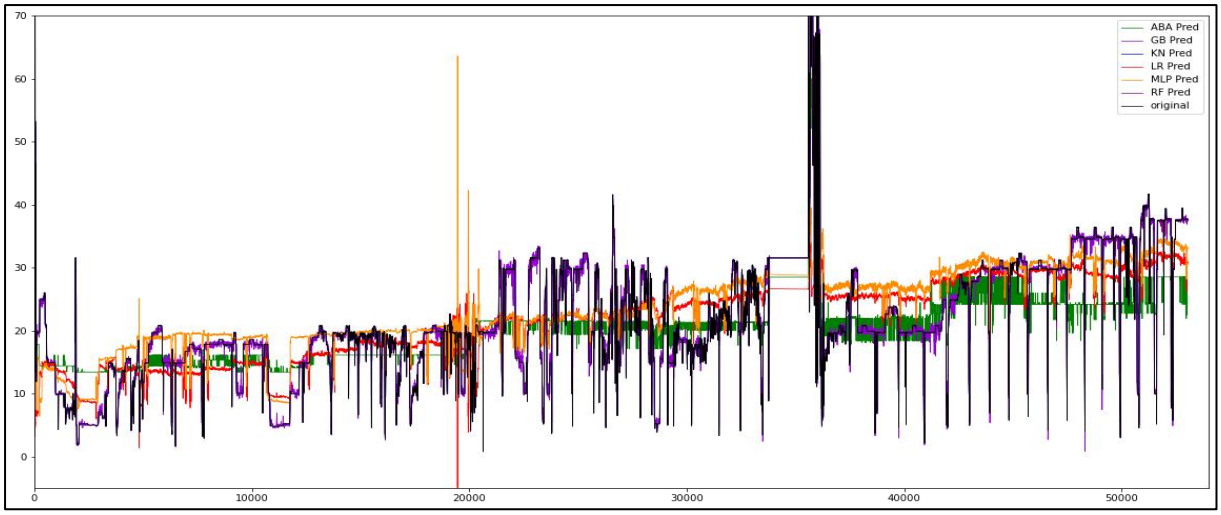

Figure 37. ROP predictions Well 4

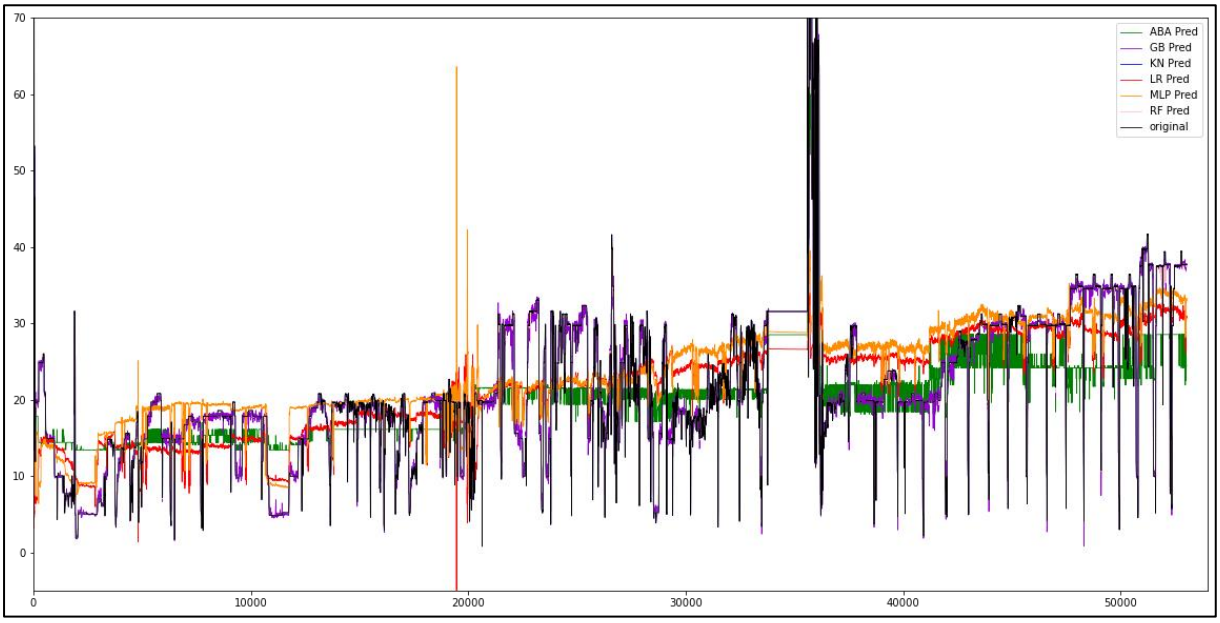

Figure 38. ROP predictions Well 5

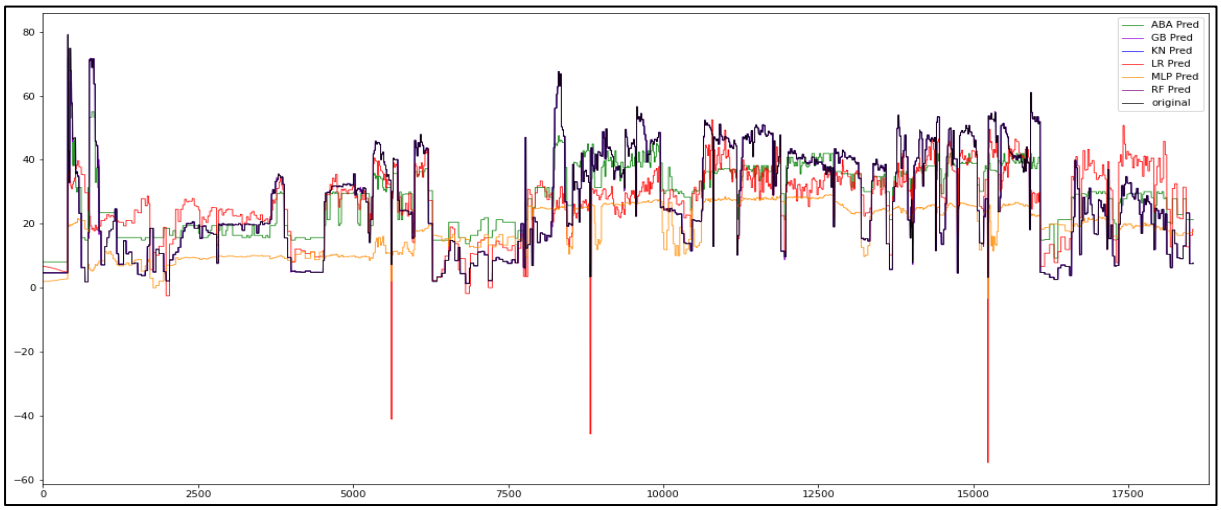

Figure 39. ROP predictions Well 6

Since the plots can be not very clear because of the amount of analyzed data, plots were generated considering only the selected models, ones which have a higher coefficient of determination. For those cases, in the following plots is shown the ROP predicted by GB regressor, K Neighbors and RF regressor.

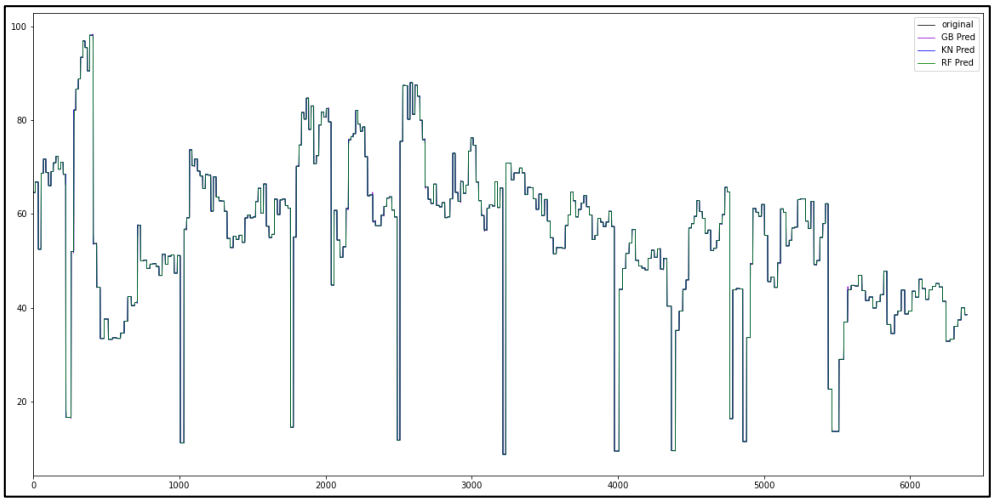

Figure 40. Plot predicted ROP selected models, Well 2

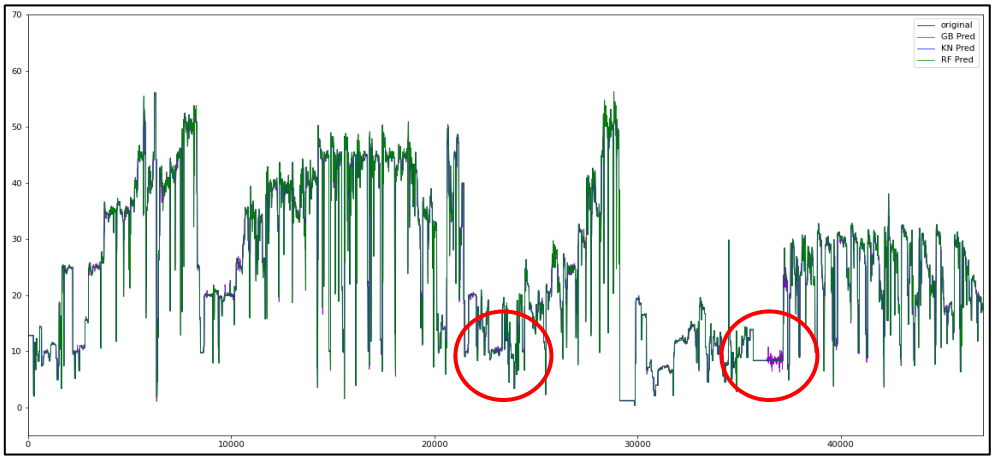

Figure 41. Plot predicted ROP selected models, Well 3

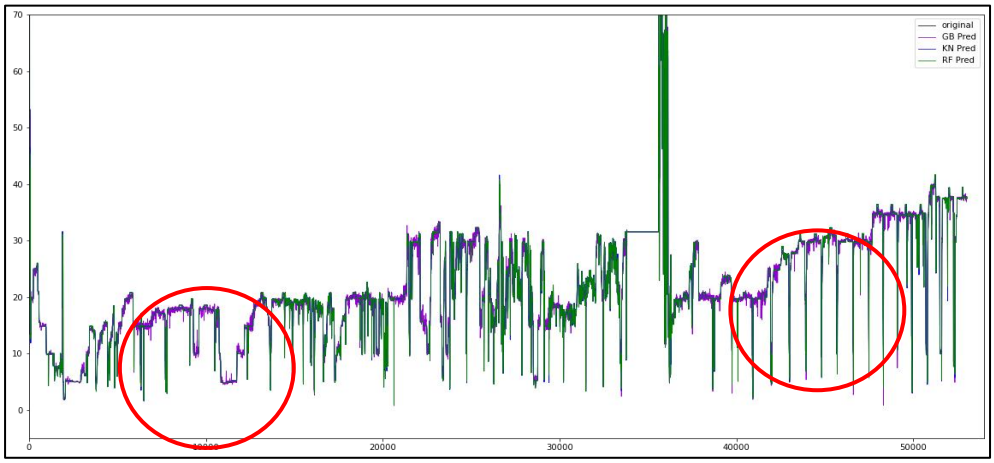

Figure 42. Plot predicted ROP selected models, Well 4

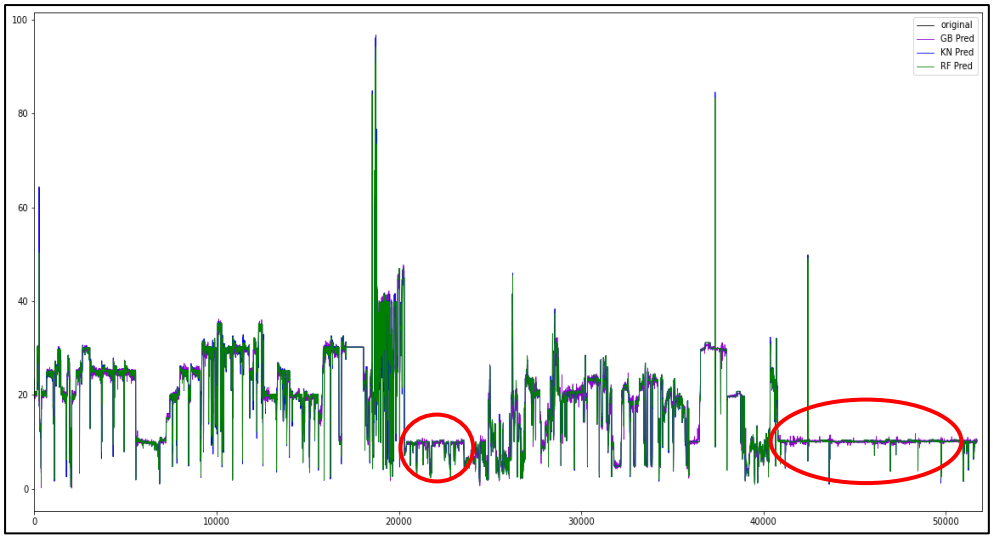

Figure 43. Plot predicted ROP selected models, Well 5

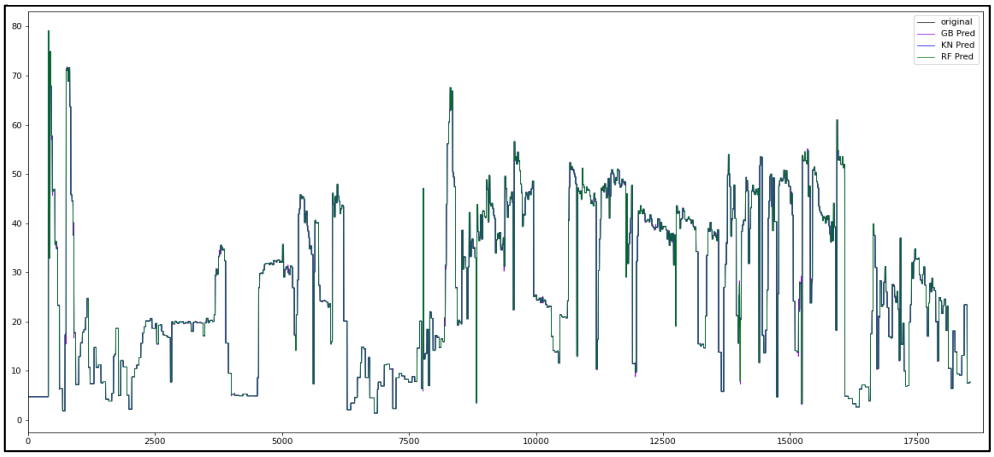

Figure 44. Plot predicted ROP selected models, Well 6

The ***Figure 40***, ***Figure 41***, ***Figure 42***, ***Figure 43*** and ***Figure 44***, show the ROP predicted in each one of the wells by using GB regressor, K Neighbors and RF regressor. Despite the accuracy of the model is very good, there are some points (red circle) where there is some noise or simple the prediction is not very accurate, but it is always close to the main prediction which means that these models can be used in different type of wells.

The main question after visualizing the plots is which one should be selected as the best model prediction ROP? Having a look back to the **Table 9** and the comparison made in the ***Figure 32***, GB regressor, K Neighbors and RF regressor are the ones which have the highest coefficient of determination since they memorized points instead of created correlations. As described by Gulli et al. [40] "*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*".

An important factor to take into account, is that since the model can have a very good accuracy, many external factors that were not consider during the study can affect the

behavior of the prediction if it is implemented in another well. A model can be more and more accurate if each single time it is adjusted until get 100% of reliability.

To conclude the final results and show why it was said that the GB regressor, K Neighbors and RF regressor are the ones which have the highest coefficient of determination or accuracy to the model tested on non-before-seen data, the metrics of each well were compilated in ***Table 10***, ***Table 11***, ***Table 12***, ***Table 13*** and ***Table 14***. If the reader takes a seen on the values, for example the coefficient of determination has the highest values for GB, K Neighbor and RF regressor in all tested data set of the six wells.
On the other hand, or another metric that can give good information regarding to the accuracy of the model is the MSE (Mean Squared Error), where the lowest values are the ones regarding to the GB, K Neighbor and RF regressor in all tested data set of the six wells.

WELL 2						
	AdaBoost	**GB**	**K Neighbor**	**Linear Regression**	**MLP**	**RF**
MAE	5,7483	0,01547	0	9,6954	10,1278	0,02052
MAPE	0,1345	0,000299	0	0,2638	0,3091	0,00495
R2	0,8186	0,9999	1	0,346	0,25136	0,9997
MSE	46,5892	0,001583	0	168,221	192,281	0,07462
RMSE	6,8256	0,03978	0	12,97	13,866	0,27317
MedAE	5,5276	0,00644	0	7,19423	7,53235	2,84E-14

Table 10. Metrics, testing model Well 2

WELL 3						
	AdaBoost	**GB**	**K Neighbor**	**Linear Regression**	**MLP**	**RF**
MAE	5,9347	0,3791	0	7,8425	10,254	0,0044
MAPE	0,5192	0,0194	0	0,5178	0,4594	0,0002
R2	0,729	0,9985	1	0,4979	0,1285	0,999
MSE	47,826	0,2706	0	88,607	153,809	0,005
RMSE	6,9156	0,5202	0	9,1413	12,402	0,0711
MedAE	6,0685	0,2848	0	7,2189	9,5187	1,42E-14

Table 11. Metrics, testing model Well 3

WELL 4						
	AdaBoost	**GB**	**K Neighbor**	**Linear Regression**	**MLP**	**RF**
MAE	4,8515	2,4895	0	5,0329	7,5903	0,0266
MAPE	0,3547	0,1758	0	0,3439	0,4122	0,00137
R2	0,5992	0,8339	1	0,4311	0,0703	0,9994
MSE	37,1435	15,3975	0	52,7275	86,172	0,0533
RMSE	6,0945	3,9239	0	7,2614	9,2829	0,2309
MedAE	3,6999	1,4958	0	4,3275	7,6099	2,40E-04

Table 12. Metrics, testing model Well 4

WELL 5						
	AdaBoost	**GB**	**K Neighbor**	**Linear Regression**	**MLP**	**RF**
MAE	6,3748	2,0163	0	4,2685	5,4668	0,0607
MAPE	0,6615	0,1874	0	0,3769	0,4199	0,00419
R2	0,3156	0,8603	1	0,5428	0,4299	0,99854
MSE	53,869	11,0027	0	35,9957	44,884	0,11493
RMSE	7,3396	3,31704	0	5,9997	6,6995	0,33901
MedAE	6,2857	1,0936	0	3,06172	4,3677	1,28E-03

Table 13. Metrics, testing model Well 5

WELL 6						
	AdaBoost	**GB**	**K Neighbor**	**Linear Regression**	**MLP**	**RF**
MAE	7,9429	2,8285	0	8,3846	12,4141	0,00854
MAPE	0,71926	0,16786	0	0,5647	0,8095	0,000294
R2	0,6587	0,9359	1	0,50808	0,0787	0,9999
MSE	86,486	16,2282	0	124,643	233,442	0,0194
RMSE	9,2998	4,0284	0	11,1644	15,279	0,1393
MedAE	7,9693	2,01076	0	6,6847	11,7	1,42E-14

Table 14. Metrics, testing model Well 6

5.2. Discussions

Despite the amount of data that has been available for many years, people in the oil industry do not know what to do with it, it has been stored and, in some cases, purged without knowing all the benefits that can be obtained from it. It can be said that much of the value and economic benefit that can be obtained from the correct management and implementation of this information can be too high.

A good use of data and its analysis will always be a good tool to strengthen planning and make predictions that help and contribute to decision making, especially in real time. A good start for this is to carry out a diagnostic analysis where the reason for having high or low rate of penetration values is explained and where the descriptive statistics take a value that is too strong, and where analytics shows that it is possible to stop assuming to begin with to read the data thanks to a much faster processing, and in the same way make decisions that will contribute to the predictions when the same conditions are present, or inputs in cases of modeling and machine learning.

There is no commercial tool or application that exactly predicts the ROP that is going to be obtained according to the different parameters, for which predictive analysis is a very good tool. After performing the predictive analysis, it is suggested to move on to the prescriptive analysis where it is sought to get to*, How to make it happen? How to optimize it?* and *What parameters do I need so that the ROP is the maximum?* In the study carried out throughout this project, base information has been taken which has been a key piece to develop the models, but despite this there are more factors or

parameters which, through experience, have shown that they can affect or vary the ROP noticeably. These factors were mentioned in Chapter 1.

If the objective is to optimize the ROP prediction to the maximum, one of the key factors to take into account without a doubt is the lithological properties of the column to be drilled, added to a correct selection and design of the bit. It is clear that, in most cases, especially when it comes to drilling exploratory wells, where the geological column is unknown, which leads to obtaining values different from those expected, the performance of the drill bit may not be as expected, or the footage may be less than planned. Normally a drill bit is chosen expecting a certain performance according to how its behavior has been in other wells and other formations, but there is no tool that exactly predicts the ROP that it will have according to the various parameters, for this, predictive analyzes are a very good tool.

At this point, another question is part of the scoop for future studies or projects, it is which parameters should be considered at the time of well planning and which should be taken into account during the operation if the objective is to feed a built model with the information obtained in real time considering as a relevant factor, the incorporation of rapid data acquisition tools such as wired drill pipe.

After running the models and analyzing the information shown in the graphs, it can be concluded that the database is biased and it is taken into account that it is a model for which it predicts better taking into account the parameters chosen according to their relationship with the ROP and how these affect it, therefore when the model was implemented in the new wells, and using the same type of parameters as input, the prediction had an optimal behavior, but when talking about real cases, where the decision making must be as fast as possible, you must follow and read the information given by the well.

Analytically predicting the ROP can contribute positively when it comes to drilling a well as fast as possible, but empirically, the ideal is to have as input to the predictions factors that can be controlled in real time, or with which the operation can be optimized. In the case of this study, the factors used as inputs were, for the most part, data measured throughout the operation that can hardly be programmed or controlled and that in turn depend on external factors, such as the average surface torque that depends on type of level of vibrations, geometry of the hole, hook load, elasticity, and hardness of the formation with respect to the cutting properties of the bits, among others. The selected variables according to the correlation and impact on the ROP, can be read during operations and keys to understand what is happening in the well, but considering an ideal case in which the objective is to automate the drilling of a well, the operational properties should be adjusted considering the values of the variables as input.
Since an important factor is the knowledge of expertise of people involved in drilling operations looking forward to improving and to enhance the quality of the inputs and

the outputs of the model, some opinions regarding to why modelling ROP is a challenging topic were taken considering the different point of view of professionals with experience in drilling jobs. Some of them are mentioned below.

"The analytically calculated ROP is made based on assumptions that are far from reality, in most of the cases. On the other hand, in places where geological uncertainty weighs heavily at the time of defining the prognosis of the lithological column, the real results definitely differ from the theoretical ones".

W. Carreño,
Company Man (Onshore drilling operations)
Colombia.

"In well drilling planning, offset wells are analyzed as a starting point for the expected ROP, in development wells where the area is known, it is expected to develop an ROP close to the plan, however differences in this value may occur by BHA design, BHA hanging or differences in lithology."

P. Perez,
Drilling Engineer, development wells (Onshore drilling planning)
Colombia.

"ROP can hardly be predicted with a high level of reliability. In my experience as a tool pusher, driller, and rig manager, one of my biggest goals is to drill the hole as fast as possible, while keeping in mind optimal penetration rates. I must not say that it is a matter of luck, since I consider that the operational parameters should be adjusted according to what the data shows us, and that is why among colleagues the phrase "The well speaks to you" is very well known. Data is a key factor in making decisions, but the expertise of the people in charge of analyzing it and knowing what to do with it is key".

G. Bonilla,
Rig Manager (Onshore drilling operations)
Colombia.

"When planning a well, the engineering team assumes ideal conditions while considering margins of error, as a rule of thumb. Regardless of the expected performance of both the drill bit and the downhole tools, external factors such as geological structures and faults result in changes in plans, operating conditions, and penetration rates far from those expected. Surprisingly, in some cases we have benefited when we encounter soft formations, but we have also incurred additional time for small layers of hard formations that must be drilled with bits designed for soft formations. As a directional engineer I must reach the desired depth, but the path does not always behave as per planned".

C. Vorkinn,
Directional Driller (Offshore drilling operations)
Norway.

Chapter 6.
Conclusions and Future Work

6.1 Conclusion

Another important point to consider is finding a way to optimize the parameters to the maximum so that they can be managed and controlled during drilling obtaining optimal ROP. In the same way take into account the specific mechanical energy and its impact on the efficiency of drilling.
Along the planning, execution and development of the project, many questions and point of views were considered to find a good approach to improve the performance given by the machine learning models and how the accuracy of the model can be affected by choosing some parameters. According to the objectives set, the conclusions of the work are:

- A complete data set from six wells was used to select the variables to develop the machine learning model.
- The most relevant parameters for the machine learning predictive model were chosen considering the correlation and their dependency to ROP, they were: Average standpipe pressure, mud flow in, average surface torque, average rotary speed, measured depth and weight on bit.
- The data set from one of the wells was used to train and test the models and to establish the accuracy of the model.
- Hyperparameter tunning was performed to improve the accuracy of the model as much as possible adjusting the metrics and coefficients of determination.
- The algorithm was tested with five different data sets, using the same parameters chosen for training and testing the model, observing that the regressions with the best performance were Random Forest, Gradient Boosting and K Neighbor Regressor with a coefficient of determination around 90%.

- The performance of the model and any model in general can be improved by considering learned lessons or field experience from petroleum engineering knowledge to enhance the quality of the inputs and the outputs of the model.

6.2 Future Work

It is known that a large number of investigations related to machine learning and data analysis always have something to improvise, either in terms of information, considerations or the use of new technologies capable of providing excellent quality data. Due to the scope of this project, several points remain open for future research, for example:

- Feeding up the model with as much information as possible, not only from the same field or cluster, but also with data from non-adjacent wells.
- As it was mentioned in the conclusions, use the information related to the stratigraphic column, field properties, geology, properties of the bit, among others can lead up into a very good performance of machine learning models and why not an optimal automatization of the drilling operations.
- Split the data obtained along the well, in order to predict the next sections to be drilled, considering that it can be applied just when the geological data used to train the model match with the new well to be drilled.

References

[1] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla y C. Willing, «Jupyter Notebooks – a publishing format for reproducible computational workflows,» de *20th International Conference on Electronic Publishing*., 2016.

[2] G. Vassum, L. Fred y Drake, Python 3, Reference Manual, Scotts Valley, CA: CreateSpace, 2009.

[3] D. Luo, «Drilling Equipment,» University of Alaska Fairbanks, April 2011. [En línea]. Available: http://ffden-2.phys.uaf.edu/212_spring2011.web.dir/Dan_Luo/web%20page/Drilling%20rig.html. [Último acceso: 12 May 2022].

[4] DataLog, Manual de Perforacion, procedimientos y Operaciones en el pozo, vol. 1.0, Calgary, Alberta, Canada: DataLog, 2002.

[5] Heriot Watt University, Drilling Engineering, Edinburgh: Heriot-Watt University, 2005.

[6] M. Encinas, Data Driven ROP Modeling Analysis and Feasibility Study, Stavanger, Rogaland: University of Stavanger, Faculty of Science and technology, 2020.

[7] Drilling Manual, «Drilling Manual,» 18 November 2007. [En línea]. Available: https://www.drillingmanual.com/major-factors-affecting-drilling-penetration-rates-rop/. [Último acceso: 14 May 2022].

[8] Petroleum Engineer's Notebook, «Petroleum Engineer's Notebook,» [En línea]. Available: https://merben.ning.com/articles/factors-affecting-rop. [Último acceso: 14 May 2022].

[9] A. Hackston y E. Rutter, «The Mohr–Coulomb criterion for intact rock strength and friction a re-evaluation and consideration of failure under polyaxial stresses,» *Solid Earth,* p. 16, 01 April 2016.

[10] C. Hegde, H. Daigle, H. Millwater y K. Gray, «Analysis of rate of penetration (ROP) prediction in drilling using physics-based and data-driven models,» *Journal of Petroleum Science and Engineering,* vol. 159, nº 0920-4105, pp. 295-306, 2017.

[11] C. Soares y K. Gray, «Real-time predictive capabilities of analytical and machine learning rate of penetration (ROP) models.,» *Journal of Petroleum Science and Engineering,* 2019.

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

[13] A. Bourgoyne y F. Young, «A multiple regression approach to optimal drilling and abnormal pressure detection.,» 1974.

[14] C. Hegde, C. Soares y K. Gray, «Rate of penetration (ROP) modeling using hybrid models: deterministic and machine learning,» *Unconventional Resources Technology Conference,* p. 19, 2018.

[15] E. Wiktorski, A. Kuznetcov y D. Sui, «Rop optimization and modeling in directional drilling process.,» 2017.

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

[17] C. Soares y K. Gray, «Real-time predictive capabilities of analytical and machine learning rate of,» *Journal of Petroleum Science and Engineering,* 2019.

[18] B. Mantha y R. Samuel, «Rop optimization using artificial intelligence techniques with statistical regression,» 2016.

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

[20] P. D, «SearchEnterpriseAI - Supervised Learning,» TechTarget, March 2021. [En línea]. Available: https://www.techtarget.com/searchenterpriseai/definition/supervised-learning. [Último acceso: 18 May 2022].

[21] Scikit learn org, "User Guide," Scikit-learn developers (BSD License), 2007 - 2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html. [Accessed 05 May 2022].

[22] I. Martinsen y K. Baturynska, «Research Gate,» Journal of Intelligent Manufacturing, 01 January 2021. [En línea]. Available: https://www.researchgate.net/figure/Schematical-representation-of-gradient-boosting-regression-in-regards-to-algorithm_fig3_340524896. [Último acceso: 05 May 2022].

[23] «Python, Data Science and programming,» Research techniques and education, 14 January 2019. [En línea]. Available: https://educationalresearchtechniques.com/2019/01/14/gradient-boosting-regression-in-python/. [Último acceso: 05 May 2022].

[24] G. Nedjati-Gilani, T. Schneider, M. Hall y N. Cawley, «Machine learning based compartment models with permeability for white matter microstructure imaging,» February 2017. [En línea]. Available: https://www.researchgate.net/publication/313489088_Machine_learning_based_compartment_models_with_permeability_for_white_matter_microstructure_imaging. [Último acceso: 06 May 2022].

[25] W. Koehrsen, «Towards Data Science,» 10 January 2018. [En línea]. Available: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74. [Último acceso: 06 May 2022].

[26] C. Bento, «Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis,» 21 September 2021. [En línea]. Available: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141. [Último acceso: 06 Mayo 2022].

[27] D. Rumelhart, G. Hinton y R. WIlliams, Learning representations by Back-propagating Errors, vol. 6088, Nature 323, 1986, pp. 533 - 536.

[28] M. Fuchs, «Python Netlify App,» 02 February 2021. [En línea]. Available: https://michael-fuchs-python.netlify.app/2021/02/03/nn-multi-layer-perceptron-classifier-mlpclassifier/. [Último acceso: 06 May 2022].

[29] Y. Freund y R. Schapire, «Experiments with a newboosting algorithm.,» de *Machine Learning: Proceedings of the Thirteenth International Conference*, New Jersey, 1996.

[30] D. Ramman y K. Vinita, «Forecasting the Grant Duration of a Patent usingPredictive Analytics,» *International Journal of Computer Applications,* vol. 178, 2019.

[31] A. Guido y C. Muller, Introduction to Machine Learning with Python, Sebastopol, CA.: O'Reilly Media, Inc, 2016.

[32] K. Barkved, «The Difference Between Training Data vs. Test Data in Machine Learning,» 11 February 2022. [En línea]. Available: https://www.obviously.ai/post/the-difference-between-training-data-vs-test-data-in-machine-learning. [Último acceso: 11 May 2022].

[33] Amazon Web Services, «AWS,» 02 August 2016. [En línea]. Available: https://docs.aws.amazon.com/machine-learning/latest/dg/machinelearning-dg.pdf#model-fit-underfitting-vs-overfitting. [Último acceso: 11 May 2022].

[34] P. Agrawal, «Analytics Vidhya,» 16 March 2021. [En línea]. Available: https://medium.com/analytics-vidhya/metrics-for-regression-model-84e3bc28fc7f. [Último acceso: 10 May 2022].

[35] M. Fuchs, «Metrics for Regression Analysis,» Python Netlify App, 30 June 2019. [En línea]. Available: https://michael-fuchs-python.netlify.app/2019/06/30/metrics-for-regression-analysis/. [Último acceso: 10 May 2022].

[36] L. Barbosa, A. Nascimiento, M. Mathias y J. Andrade de Carvalho, «Machine learning methods applied to drilling rate of penetration prediction and optimization - a review,» *Journal of Petroleum Science and Engineering,* p. 183, 2019.

[37] E. News, «Disclose all volve data.,» Equinor News, 2018.

[38] M. Fernández, J. Cao y D. Sui, «Evaluation and interpretation on data driven ROP models from engineering perspectives.,» de *Conference on Ocean, Offshore and Arctic Engineering*, Hamburg, 2022.

[39] Drake, G. V. Rossum y L. Fred, Python 3 Reference Manual., Scotts Valley, CA: CreateSpace, 2009.

[40] A. Gulli y P. Sujit, Deep Learning with Keras., Packt Publishing, 2017.

[41] Y. M. El-Nadi y M. Osman, «Establishing the Penetration Rate Formula For Drilling Oil and Gas Wells and Its Solution,» *Journal of Egyptian Society of Engineers,* vol. 3, nº 3, 1976.

[42] A. Olatunbosun y A. Kester, «Comparative Analysis of Drilling Rate of Penetration Models,» *American Journal of Engineering Research (AJER),* vol. 9, nº 4, pp. 125-132, 8 April 2020.

Appendices

Appendix A
Python Code

A.1 Installed Packages

Package	Version	Package	Version
absl-py	1.0.0	*numpydoc*	1.1.0
alabaster	0.7.12	*oauthlib*	3.2.0
anaconda-client	1.7.2	*olefile*	0.46
anaconda-navigator	2.1.4	*openpyxl*	3.0.7
anaconda-project	0.9.1	*opt-einsum*	3.3.0
anyio	2.2.0	*packaging*	20.9
appdirs	1.4.4	*pandas*	1.2.4
argh	0.26.2	*pandocfilters*	1.4.3
argon2-cffi	20.1.0	*paramiko*	2.7.2
asn1crypto	1.4.0	*parso*	0.7.0
astroid	2.5	*partd*	1.2.0
astropy	4.2.1	*path*	15.1.2
astunparse	1.6.3	*pathlib2*	2.3.5
async-generator	1.10	*pathspec*	0.7.0
atomicwrites	1.4.0	*pathtools*	0.1.2
attrs	20.3.0	*patsy*	0.5.1
autopep8	1.5.6	*pep8*	1.7.1
Babel	2.9.0	*pexpect*	4.8.0
backcall	0.2.0	*pickleshare*	0.7.5

backports.functools-lru-cache	1.6.4	*Pillow*	8.2.0
backports.shutil-get-terminal-size	1.0.0	*pip*	21.0.1
backports.tempfile	1.0	*pkginfo*	1.7.0
backports.weakref	1.0.post1	*plotly*	5.3.1
bcrypt	3.2.0	*plotly-express*	0.4.0
beautifulsoup4	4.9.3	*pluggy*	0.13.1
bitarray	1.9.2	*ply*	3.11
bkcharts	0.2	*prometheus-client*	0.10.1
black	19.10b0	*prompt-toolkit*	3.0.17
bleach	3.3.0	*protobuf*	3.20.1
bokeh	2.3.2	*psutil*	5.8.0
boto	2.49.0	*ptyprocess*	0.7.0
Bottleneck	1.3.2	*py*	1.10.0
Brotli	1.0.9	*pyasn1*	0.4.8
brotlipy	0.7.0	*pyasn1-modules*	0.2.8
cachetools	5.0.0	*pycodestyle*	2.6.0
certifi	2021.10.8	*pycosat*	0.6.3
cffi	1.14.5	*pycparser*	2.20
chardet	4.0.0	*pycurl*	7.43.0.6
click	7.1.2	*pydocstyle*	6.0.0
cloudpickle	1.6.0	*pydot*	1.4.2
clyent	1.2.2	*pydotplus*	2.0.2
colorama	0.4.4	*pyerfa*	1.7.3
comtypes	1.1.9	*pyflakes*	2.2.0
conda	4.12.0	*Pygments*	2.8.1
conda-build	3.21.4	*pygtrie*	2.4.2
conda-content-trust	0+unknown	*PyJWT*	2.1.0
conda-package-handling	1.7.3	*pylint*	2.7.4
conda-repo-cli	1.0.4	*pyls-black*	0.4.6
conda-token	0.3.0	*pyls-spyder*	0.3.2
conda-verify	3.4.2	*PyNaCl*	1.4.0
contextlib2	0.6.0.post1	*pyodbc*	4.0.0-unsupported
cryptography	3.4.7	*pyOpenSSL*	20.0.1
cycler	0.10.0	*pyparsing*	2.4.7
Cython	0.29.23	*pyreadline*	2.1
cytoolz	0.11.0	*pyrsistent*	0.17.3
dash	0.21.1	*PySocks*	1.7.1
dash-core-components	0.23.0	*pytest*	6.2.3
dash-html-components	0.11.0	*python-jsonrpc-server*	0.4.0
dash-renderer	0.13.0	*python-language-server*	0.36.2

dash-table	5.0.0	*python-dateutil*	2.8.1
dask	2021.4.0	*PyWavelets*	1.1.1
decorator	5.0.6	*pywin32*	227
defusedxml	0.7.1	*pywin32-ctypes*	0.2.0
diff-match-patch	20200713	*pywinpty*	0.5.7
distributed	2021.4.0	*PyYAML*	5.4.1
docutils	0.17	*pyzmq*	20.0.0
entrypoints	0.3	*QDarkStyle*	2.8.1
et-xmlfile	1.0.1	*QtAwesome*	1.0.2
fastcache	1.1.0	*qtconsole*	5.0.3
filelock	3.0.12	*QtPy*	1.9.0
flake8	3.9.0	*regex*	2021.4.4
Flask	1.1.2	*requests*	2.25.1
Flask-Compress	1.10.1	*requests-oauthlib*	1.3.1
flatbuffers	2.0	*require-python-3*	1
fsspec	0.9.0	*rope*	0.18.0
future	0.18.2	*rsa*	4.8
gast	0.5.3	*Rtree*	0.9.7
gevent	21.1.2	*ruamel-yaml-conda*	0.15.100
gitdb	4.0.7	*scikit-image*	0.18.1
glob2	0.7	*scikit-learn*	1.0.2
gmpy2	2.1.2	*scipy*	1.6.2
google-auth	2.6.6	*seaborn*	0.11.1
google-auth-oauthlib	0.4.6	*Send2Trash*	1.5.0
google-pasta	0.2.0	*setuptools*	52.0.0.post 20210125
graphviz	0.20	*simplegeneric*	0.8.1
greenlet	1.0.0	*singledispatch*	0.0.0
grpcio	1.44.0	*sip*	4.19.13
h5py	2.10.0	*six*	1.15.0
idna	2.10	*smmap*	4.0.0
imagecodecs	2021.3.31	*sniffio*	1.2.0
imageio	2.9.0	*snowballstemmer*	2.1.0
imagesize	1.2.0	*sortedcollections*	2.1.0
importlib-metadata	4.11.3	*sortedcontainers*	2.3.0
iniconfig	1.1.1	*soupsieve*	2.2.1
intervaltree	3.1.0	*Sphinx*	4.0.1
ipykernel	5.3.4	*sphinxcontrib-applehelp*	1.0.2
ipython	7.22.0	*sphinxcontrib-devhelp*	1.0.2
ipython-genutils	0.2.0	*sphinxcontrib-htmlhelp*	1.0.3
ipywidgets	7.6.3	*sphinxcontrib-jsmath*	1.0.1
isort	5.8.0	*sphinxcontrib-qthelp*	1.0.3

itsdangerous	1.1.0	*sphinxcontrib-serializinghtml*	1.1.4
jdcal	1.4.1	*sphinxcontrib-websupport*	1.2.4
jedi	0.17.2	*spyder*	4.2.5
Jinja2	2.11.3	*spyder-kernels*	1.10.2
joblib	1.1.0	*SQLAlchemy*	1.4.7
json5	0.9.5	*statsmodels*	0.12.2
jsonschema	3.2.0	*sympy*	1.8
jupyter	1.0.0	*tables*	3.6.1
jupyter-client	6.1.12	*tabulate*	0.8.9
jupyter-console	6.4.0	*tblib*	1.7.0
jupyter-core	4.7.1	*tenacity*	8.0.1
jupyter-packaging	0.7.12	*tensorboard*	2.8.0
jupyter-server	1.4.1	*tensorboard-data-server*	0.6.1
jupyterlab	3.0.14	*tensorboard-plugin-wit*	1.8.1
jupyterlab-pygments	0.1.2	*tensorflow*	2.8.0
jupyterlab-server	2.4.0	*tensorflow-io-gcs-filesystem*	0.25.0
jupyterlab-widgets	1.0.0	*termcolor*	1.1.0
keras	2.8.0	*terminado*	0.9.4
Keras-Preprocessing	1.1.2	*testpath*	0.4.4
keyring	22.3.0	*textdistance*	4.2.1
kiwisolver	1.3.1	*tf-estimator-nightly*	2.8.0.dev2021122109
lazy-object-proxy	1.6.0	*threadpoolctl*	2.1.0
libarchive-c	2.9	*three-merge*	0.1.1
libclang	14.0.1	*tifffile*	2021.4.8
llvmlite	0.36.0	*toml*	0.10.2
locket	0.2.1	*toolz*	0.11.1
lxml	4.6.3	*tornado*	6.1
Markdown	3.3.6	*tqdm*	4.59.0
MarkupSafe	1.1.1	*traitlets*	5.0.5
matplotlib	3.3.4	*typed-ast*	1.4.2
mccabe	0.6.1	*typing*	3.7.4.3
menuinst	1.4.16	*typing-extensions*	3.7.4.3
mistune	0.8.4	*ujson*	4.0.2
mkl-fft	1.3.0	*unicodecsv*	0.14.1
mkl-random	1.2.1	*urllib3*	1.26.4
mkl-service	2.3.0	*watchdog*	1.0.2
mock	4.0.3	*wcwidth*	0.2.5
more-itertools	8.7.0	*webcolors*	1.11.1
mpmath	1.2.1	*webencodings*	0.5.1

msgpack	1.0.2	*Werkzeug*	1.0.1
multipledispatch	0.6.0	*wheel*	0.36.2
mypy-extensions	0.4.3	*widgetsnbextension*	3.5.1
navigator-updater	0.2.1	*win-inet-pton*	1.1.0
nbclassic	0.2.6	*win-unicode-console*	0.5
nbclient	0.5.3	*wincertstore*	0.2
nbconvert	6.0.7	*wrapt*	1.12.1
nbformat	5.1.3	*xlrd*	2.0.1
nest-asyncio	1.5.1	*XlsxWriter*	1.3.8
networkx	2.5	*xlwings*	0.23.0
nltk	3.6.1	*xlwt*	1.3.0
nose	1.3.7	*xmltodict*	0.12.0
notebook	6.3.0	*yapf*	0.31.0
numba	0.53.1	*zict*	2.0.0
numexpr	2.7.3	*zipp*	3.4.1
numpy	1.20.1	*zope.event*	4.5.0
numpy-financial	1.0.0	*zope.interface*	5.3.0

A.2 General Well Statistics Python Code

Importing libraries

In [ ]:
```
import pandas as pd
from sklearn import datasets
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
from glob import glob
import warnings
import numpy as np
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression

warnings.filterwarnings("ignore")
well_list = glob(r'USROP_A 0 N-NA_F-9_Ad.csv', recursive=False)

df = pd.read_csv('USROP_A 0 N-NA_F-9_Ad.csv')
df = pd.read_csv(well_list[0])
df.drop('Unnamed: 0',axis=1, inplace=True)
df.dropna(axis=0, how='any', inplace=True)

```

Loading the raw data

In [ ]:
```
df.head()
```

In [ ]:
```
df
```

In [ ]:
```
# Descriptive statistics are very useful for initial exploration of the variables

df.describe(include='all')
```

Dealing with missing values

In [ ]:
```
# data.isnull() # shows a df with the information whether a data point is null
# Since True = the data point is missing, while False = the data point is not missing, we can sum them
# This will give us the total number of missing values feature-wise

df.isnull().sum()
```

In [ ]:
```
# correlation matrix
sns.pairplot(df)
```

In [ ]:
```
plt.figure(figsize=(12,12))
corr = df.corr()
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,cmap="YlGnBu",annot=True)
```

In [ ]:
```
plt.figure(figsize=(12,12))
```

Correlated variables (ROP)

In [ ]:
```
f, (ax1, ax2, ax3) = plt.subplots(1, 3, sharey=False, figsize =(18,5)) #sharey -> share 'ROP' as y

ax1.scatter(df['Average Standpipe Pressure kPa'],df['Rate of Penetration m/h'])
ax1.set_title('Average Standpipe Pressure and ROP')
ax2.scatter(df['Average Rotary Speed rpm'],df['Rate of Penetration m/h'])
ax2.set_title('Average Rotary Speed and ROP')
ax3.scatter(df['Weight on Bit kkgf'],df['Rate of Penetration m/h'])
ax3.set_title('Weight on Bit and ROP')

```

In [ ]:
```
f, (ax1, ax2) = plt.subplots(1, 2, sharey=False, figsize =(18,5)) #sharey -> share 'ROP' as y

ax1.scatter(df['Weight on Bit kkgf'],df['Rate of Penetration m/h'], color="green", marker=".", s=5 )
ax1.set_title('WOB VS Rate of Penetration m/h')
ax2.scatter(df['Average Rotary Speed rpm'],df['Rate of Penetration m/h'], color="red", marker=".", s=5 )
ax2.set_title('Average Rotary Speed VS Rate of Penetration m/h')


```

In [ ]:
```
f, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots(6, 1, sharey=False, figsize =(20,15)) #sharey -> share 'depth' as y

ax1.plot(df['Measured Depth m'],df['Average Standpipe Pressure kPa'], color='green', linewidth=1, markersize=12)
ax1.set_title('Measured Depth m')
ax1.set_ylabel('Average Standpipe Pressure')

ax2.plot(df['Measured Depth m'],df['Mud Flow In L/min'], color='lime', linewidth=1, markersize=12)
ax2.set_ylabel('Mud Flow In')

ax3.plot(df['Measured Depth m'],df['Average Surface Torque kN.m'], color='gold', linewidth=1, markersize=12)
ax3.set_ylabel('Average Surface Torque')

ax4.plot(df['Measured Depth m'],df['Average Rotary Speed rpm'], color='darkorange', linewidth=1, markersize=12)
ax4.set_ylabel('Average Rotary Speed')

ax5.plot(df['Measured Depth m'],df['Rate of Penetration m/h'], color='red', linewidth=1, markersize=12)
ax5.set_ylabel('Rate of Penetration')

ax6.plot(df['Measured Depth m'],df['Weight on Bit kkgf'], color='darkred', linewidth=1, markersize=12)
ax6.set_ylabel('Weight on Bit')

plt.savefig('plot.png', dpi=300, bbox_inches='tight')

```

Continuity of Data

In [ ]:
```
plt.plot(df['Hole Depth (TVD) m'],df['Rate of Penetration m/h'], color='blue', linewidth=0.5, markersize=12)

# x-axis label
plt.xlabel('Hole Depth (TVD) m')
# frequency label
plt.ylabel('Rate of Penetration m/h')
# plot title
plt.title('TVD m VS Rate of Penetration m/h')
# showing legend
#ax.legend()

plt.rcParams["figure.figsize"] = (8,2)
plt.xlim(450, 1000)
plt.ylim(0, 100)

# function to show the plot
#ax.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
plt.plot(df['Measured Depth m'],df['Rate of Penetration m/h'], color='blue', linewidth=0.5, markersize=12)

# x-axis label
plt.xlabel('Measured Depth m')
# frequency label
plt.ylabel('Rate of Penetration m/h')
# plot title
plt.title('Measured Depth m VS Rate of Penetration m/h')
# showing legend
#ax.legend()
plt.rcParams["figure.figsize"] = (8,2)

# function to show the plot
#ax.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()

```

In [ ]:
```
plt.plot(df['Hole Depth (TVD) m'],df['Average Rotary Speed rpm'], color='red', linewidth=0.5, markersize=12)

# x-axis label
plt.xlabel('Hole Depth (TVD) m')
# frequency label
plt.ylabel('Average Rotary Speed rpm')
# plot title
plt.title('TVD m VS Average Rotary Speed rpm')
# showing legend
#plt.legend()

plt.rcParams["figure.figsize"] = (8,2)

# function to show the plot
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
plt.plot(df['Measured Depth m'],df['Average Rotary Speed rpm'], color='red', linewidth=0.5, markersize=12)

# x-axis label
plt.xlabel('Measured Depth m')
# frequency label
plt.ylabel('Average Rotary Speed rpm')
# plot title
plt.title('Measured Depth m VS Average Rotary Speed rpm')
# showing legend
#plt.legend()

plt.rcParams["figure.figsize"] = (8,2)
# function to show the plot
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
plt.plot(df['Hole Depth (TVD) m'],df['Weight on Bit kkgf'], color='purple', linewidth=0.5, markersize=12)

# x-axis label
plt.xlabel('Hole Depth (TVD) m')
# frequency label
plt.ylabel('Weight on Bit kkgf')
# plot title
plt.title('TVD m VS Weight on Bit kkgf')
# showing legend
#plt.legend()

plt.rcParams["figure.figsize"] = (8,2)
# function to show the plot
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()

```

In [ ]:
```
plt.plot(df['Measured Depth m'],df['Weight on Bit kkgf'], color='purple', linewidth=0.5, markersize=12)

# x-axis label
plt.xlabel('Measured Depth m')
# frequency label
plt.ylabel('Weight on Bit kkgf')
# plot title
plt.title('Measured Depth m VS Weight on Bit kkgf')
# showing legend
#plt.legend()

plt.rcParams["figure.figsize"] = (8,2)
# function to show the plot
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()

```

In [ ]:
```
plt.plot(df['Hole Depth (TVD) m'],df['Average Standpipe Pressure kPa'], color='green', linewidth=0.5, markersize=12)

# x-axis label
plt.xlabel('Hole Depth (TVD) m')
# frequency label
plt.ylabel('Average Standpipe Pressure kPa')
# plot title
plt.title('TVD m VS Average Standpipe Pressure')
# showing legend
#plt.legend()

plt.rcParams["figure.figsize"] = (8,2)
# function to show the plot
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()
```

In [ ]:
```
plt.plot(df['Measured Depth m'],df['Average Standpipe Pressure kPa'], color='green', linewidth=0.5, markersize=12)

# x-axis label
plt.xlabel('Measured Depth m')
# frequency label
plt.ylabel('Average Standpipe Pressure kPa')
# plot title
plt.title('Measured Depth m VS Average Standpipe Pressure')
# showing legend
#plt.legend()

plt.rcParams["figure.figsize"] = (8,2)
# function to show the plot
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
plt.show()
```

Probability Distribution

In [ ]:
```
plt.figure(figsize=(15,15))
for i in range(len(df.columns)):
    plt.subplot(3,4,i+1)
    plt.title(df.columns.values[i])
    sns.distplot(df[df.columns.values[i]])
plt.show()
```

In [ ]:
```
1
```

A.3 Training and Testing the model

- **Gradient Boosting Regressor**

Import the relevant libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot  as plt
import warnings
from numpy.random import seed
import tensorflow as tf
import math
import pickle
from tensorflow import keras
from scipy.stats import pearsonr

warnings.filterwarnings('ignore')
seed(0)
tf.random.set_seed(0)
```

SKlearn model libraries

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import pydot

from sklearn.neural_network import MLPRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from tensorflow import keras

import seaborn as sns
import io
```

Importing Data

In [ ]:
```
df = pd.read_csv('USROP_A 0 N-NA_F-9_Ad.csv')
del df['Unnamed: 0']

df = df[['Average Standpipe Pressure kPa', 'Mud Flow In L/min','Average Surface Torque kN.m','Average Rotary Speed rpm','Measured Depth

#print(df)
X = df.iloc[:,0:6]
y = df.iloc[:,11] #ROP
df = df.iloc[: , :12]

df
```

In [ ]:
```
X
```

Training the model

In [ ]:
```
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=0.2, random_state=42) #test:20% train:80%
plt.rcParams['font.sans-serif'] = ['Times New Roman']

print(train_x.shape, test_x.shape, train_y.shape, test_y.shape)
```

Gradient Boosting Regressor

In [ ]:
```
#gbr = GradientBoostingRegressor(n_estimators=1000, learning_rate=0.1, max_depth=4, random_state=1, subsample=0.5)

# with default parameters
gbr = GradientBoostingRegressor()
gbr.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypred = gbr.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

```

```
#Predicting training data set
train_predictions = gbr.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = gbr.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)



#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:

```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:

```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

Hyperparameter Tuning

In [ ]:

```
gbr=GradientBoostingRegressor()
search_grid={'n_estimators':[50,200,500,1000],'learning_rate':[0.01,.1],'max_depth':[1,2,4],
             'subsample':[.5,.75,1],'random_state':[1]}
search=GridSearchCV(estimator=gbr,param_grid=search_grid,scoring='neg_mean_squared_error',n_jobs=1)
result= search.fit(train_x,train_y)

print("Best parameters Gradient Boosting Regressor:",result.best_params_)
print("Best score GBR:",result.best_score_)
```

In [ ]:

```
#Defined hyperparameter tuning
gbr = GradientBoostingRegressor(n_estimators=1000, learning_rate=0.1, max_depth=4, random_state=1, subsample=0.5)
gbr.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypredHT = gbr.predict(X)

#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)


#Predicting training data set
train_predictions = gbr.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = gbr.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypredHT])], [-1, np.max([y, ypredHT])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
df1['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

- **K Neighbors Regressor**

In [ ]:
```
# with default parameters
KN = KNeighborsRegressor()
KN.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypred = KN.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

#Predicting training data set
train_predictions = KN.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = KN.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
             linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics

print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

Hyperparameter Tuning

In [ ]:
```
KN = KNeighborsRegressor()
seed = 13
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)

param_list2 = [{'n_neighbors': [2,3,4,5,6], 'weights': ['uniform','distance']}]
# Search for best hyperparameters
grid = GridSearchCV(estimator=KN, param_grid=param_list2, cv=kfold, scoring='r2')
grid.fit(train_x,train_y)

print(grid.best_score_)
print(grid.best_estimator_)
print(grid.best_params_)
```

In [ ]:
```
#Defined hyperparameter tuning
KN = KNeighborsRegressor(n_neighbors=4, weights='distance')
KN.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypredHT = KN.predict(X)

#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)

#Predicting training data set
train_predictions = KN.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = KN.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)

plt.show()
```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
df1['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

- **Linear Regression**

In [ ]:
```
# with default parameters
LR = LinearRegression()
LR.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypred = LR.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

#Predicting training data set
train_predictions = LR.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = LR.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)
```

```
#Regression Metrics

print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)
```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

- **Multi-Layer Perceptron Regressor**

In [ ]:
```
# with default parameters
MLP = MLPRegressor()
MLP.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypred = MLP.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

#Predicting training data set
train_predictions = MLP.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = MLP.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics

print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)
```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

Hyperparameter Tuning

In [ ]:
```
param_grid = {'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,1)],'activation': ['relu','tanh','logistic'],
              'alpha': [0.0001, 0.05],'learning_rate': ['constant','adaptive'],'solver': ['adam']}
MLP = MLPRegressor()
grid_searchMLP= GridSearchCV(MLP,param_grid,cv=5, scoring='neg_mean_squared_error', verbose=0, n_jobs=-1)

grid_result = grid_searchMLP.fit(train_x,train_y)
best_params = grid_result.best_params_

print("Best parameters MLP Regressor:",best_params)
```

In [ ]:
```
#Defined hyperparameter tuning
MLP = MLPRegressor(hidden_layer_sizes=(50, 50, 50), activation='relu',
                   solver='adam', learning_rate='adaptive', alpha=(0.05))
MLP.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypredHT = MLP.predict(X)

#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)

#Predicting training data set
train_predictions = MLP.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = MLP.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
df1['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

- **Random Forest Regressor**

In [ ]:
```
# with default parameters
RF = RandomForestRegressor(random_state = 0)
RF.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypred = RF.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

#Predicting training data set
train_predictions = RF.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = RF.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)
```

```
#Regression Metrics

print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

Hyperparameter Tuning

In [ ]:
```
param_list1 = {"n_estimators": [10,30,50,70], "max_depth": [10,20,30]}
RF = RandomForestRegressor(random_state = 0)
grid_search1 = GridSearchCV(RF, param_grid=param_list1 , cv=10, scoring='r2' , n_jobs=4).fit(train_x,train_y)

print("Best parameters RandomForest:", grid_search1.best_estimator_)
print("Best score RFR:", grid_search1.best_score_)
```

In [ ]:
```
#Defined hyperparameter tuning
RF = RandomForestRegressor(n_estimators = 50, random_state = 0, max_depth=30)
RF.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypredHT = RF.predict(X)

#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)

#Predicting training data set
train_predictions = RF.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = RF.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
df1['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

A.4 Implementation of the model

- **AdaBoost Regressor**

Import the relevant libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot  as plt
import warnings
from numpy.random import seed
import tensorflow as tf
import math
import pickle
from tensorflow import keras

warnings.filterwarnings('ignore')
seed(0)
tf.random.set_seed(0)
```

SKlearn model libraries

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import pydot

from sklearn.neural_network import MLPRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from tensorflow import keras

import seaborn as sns
import io
```

Importing Data

In [ ]:
```
df = pd.read_csv('USROP_A 2 N-SH_F-14d.csv')
del df['Unnamed: 0']

df = df[['Average Standpipe Pressure kPa', 'Mud Flow In L/min','Average Surface Torque kN.m','Average Rotary Speed rpm','Measured Depth

#print(df)
X = df.iloc[:,0:6]
y = df.iloc[:,11] #ROP
df = df.iloc[: , :12]

df
```

In [ ]:
```
X
```

AdaBoost Regressor

In [ ]:
```
# with default parameters
AB = AdaBoostRegressor()
#AB.fit(train_x, train_y)
AB.fit(X,y)

#Predicting data set
ypred = AB.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

plt.scatter(y, ypred, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
             linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)
```

```
#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)
```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

AdaBoost Regressor

In [ ]:
```
#Defined hyperparameter tuning
rf = AdaBoostRegressor(learning_rate=0.001, n_estimators=2000, random_state=1)
#AB.fit(train_x, train_y)
AB.fit(X,y)

#Predicting data set
ypredHT = AB.predict(X)

#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)

plt.scatter(y, ypred, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)
```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")
plt.plot(x_ax, ypredHT, lw=0.8, color="darkviolet", label="predicted HT")


plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
df1_14d=pd.DataFrame()

df1_14d['ROP'] = df['Rate of Penetration m/h']
df1_14d['Perfect Prediction'] = ypred
df1_14d['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1_14d
```

- **Gradient Boosting Regressor**

In [ ]:
```
#gbr = GradientBoostingRegressor(n_estimators=1000, learning_rate=0.1, max_depth=4, random_state=1, subsample=0.5)

# with default parameters
gbr = GradientBoostingRegressor()
#gbr.fit(train_x, train_y)
gbr.fit(X,y)

#Predicting data set
ypred = gbr.predict(X)
plt.scatter(y, ypred, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
             linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics

errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')

MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')

R2 = r2_score(y, ypred)
print('R2=',R2)

MSE=mean_squared_error(y, ypred)
print('MSE:',MSE)

RMSE=math.sqrt(MSE)
print('RMSE:',RMSE)

MedAE=median_absolute_error(y, ypred)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

Defined Model

In [ ]:
```
#Defined hyperparameter tuning
gbr = GradientBoostingRegressor(n_estimators=1000, learning_rate=0.1, max_depth=4, random_state=1, subsample=0.5)
#gbr.fit(train_x, train_y)
gbr.fit(X,y)

#Predicting data set
ypredHT = gbr.predict(X)
plt.scatter(y, ypredHT, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypredHT])], [-1, np.max([y, ypredHT])], c="red", linewidth=3,
             linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")
plt.plot(x_ax, ypredHT, lw=0.8, color="darkviolet", label="predicted HT")


plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
df2_14d=pd.DataFrame()

df2_14d['ROP'] = df['Rate of Penetration m/h']
df2_14d['Perfect Prediction'] = ypred
df2_14d['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df2_14d
```

- **K Neighbors Regressor**

In [ ]:
```
# with default parameters
KN = KNeighborsRegressor()
#KN.fit(train_x, train_y)
KN.fit(X,y)

#Predicting data set
ypred = KN.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

plt.scatter(y, ypred, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

Defined Model

In [ ]:
```
#Defined hyperparameter tuning
KN = KNeighborsRegressor(n_neighbors=4, weights='distance')
#KN.fit(train_x, train_y)
KN.fit(X,y)

#Predicting data set
ypredHT = KN.predict(X)
```

```
#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)

plt.scatter(y, ypredHT, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypredHT])], [-1, np.max([y, ypredHT])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)
```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")
plt.plot(x_ax, ypredHT, lw=0.8, color="darkviolet", label="predicted HT")


plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
df3_14d=pd.DataFrame()

df3_14d['ROP'] = df['Rate of Penetration m/h']
df3_14d['Perfect Prediction'] = ypred
df3_14d['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df3_14d
```

- **Linear Regression**

In [ ]:
```
# with default parameters
LR = LinearRegression()
#LR.fit(train_x, train_y)
LR.fit(X,y)

#Predicting data set
ypred = LR.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

plt.scatter(y, ypred, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)
```

In [ ]:
```
df5_14d=pd.DataFrame()

df5_14d['ROP'] = df['Rate of Penetration m/h']
df5_14d['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df5_14d
```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

- **Multi-Layer Perceptron Regressor**

In [ ]:
```
# with default parameters
MLP = MLPRegressor()
#MLP.fit(train_x, train_y)
MLP.fit(X,y)

#Predicting data set
ypred = MLP.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

plt.scatter(y, ypred, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
#Defined hyperparameter tuning
MLP = MLPRegressor(hidden_layer_sizes=(50, 50, 50), activation='relu',
                   solver='adam', learning_rate='adaptive', alpha=(0.05))
#MLP.fit(train_x, train_y)
MLP.fit(X,y)

#Predicting data set
ypredHT = MLP.predict(X)

#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)

plt.scatter(y, ypred, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
            linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")
plt.plot(x_ax, ypredHT, lw=0.8, color="darkviolet", label="predicted HT")


plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
df4_14d=pd.DataFrame()

df4_14d['ROP'] = df['Rate of Penetration m/h']
df4_14d['Perfect Prediction'] = ypred
df4_14d['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df4_14d
```

- **Random Forest Regressor**

In [ ]:
```
# with default parameters
RF = RandomForestRegressor(random_state = 0)
#RF.fit(train_x, train_y)
RF.fit(X,y)

#Predicting data set
ypred = RF.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

plt.scatter(y, ypred, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
             linestyle="--", label="Perfect prediction")

plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
#Defined hyperparameter tuning
RF = RandomForestRegressor(n_estimators = 50, random_state = 0, max_depth=30)
#RF.fit(train_x, train_y)
RF.fit(X,y)

#Predicting data set
ypredHT = RF.predict(X)

#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)

plt.scatter(y, ypred, s=50, c="black", label="Predicted data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
             linestyle="--", label="Perfect prediction")
```

```
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
x_ax = range(len(y))
#plt.scatter(x_ax, y, s=5, color="blue", label="original")
plt.plot(x_ax, y, lw=0.8, color="lime", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="black", label="predicted")
plt.plot(x_ax, ypredHT, lw=0.8, color="darkviolet", label="predicted HT")


plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

In [ ]:
```
df6_14d=pd.DataFrame()

df6_14d['ROP'] = df['Rate of Penetration m/h']
df6_14d['Perfect Prediction'] = ypred
df6_14d['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df6_14d
```

- **AdaBoost Regressor**

In [ ]:
```
# with default parameters
AB = AdaBoostRegressor()
AB.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypred = AB.predict(X)

#Regression Metrics
errors = np.abs(ypred - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypred)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypred)
MSE=mean_squared_error(y, ypred)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypred)

#Predicting training data set
train_predictions = AB.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = AB.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
             linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics

print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)
plt.show()
```

Hyperparameter Tuning

In [ ]:
```
search_grid={'n_estimators':[500,1000,2000],'learning_rate':[.001,0.01,.1],'random_state':[1]}
AB = AdaBoostRegressor()
search=GridSearchCV(estimator=AB,param_grid=search_grid,scoring='neg_mean_squared_error',n_jobs=1,cv=3)
search.fit(train_x,train_y)

print("Best parameters AdaBoost:", search.best_params_)
print("Best score ABR:", search.best_score_)
```

In [ ]:
```
#Defined hyperparameter tuning
rf = AdaBoostRegressor(learning_rate=0.001, n_estimators=2000, random_state=1)
AB.fit(train_x, train_y)
#gbr.fit(X,y)

#Predicting data set
ypredHT = AB.predict(X)

#Regression Metrics
errors = np.abs(ypredHT - y)  # Print out the mean absolute error (mae)
MAPE=mean_absolute_percentage_error(y, ypredHT)
accuracy = 100 - np.mean(MAPE)
R2 = r2_score(y, ypredHT)
MSE=mean_squared_error(y, ypredHT)
RMSE=math.sqrt(MSE)
MedAE=median_absolute_error(y, ypredHT)

#Predicting training data set
train_predictions = AB.predict(train_x)
plt.scatter(train_y, train_predictions, s=50, c="black", label="Train Data", alpha=1)

#Predicting testing data set
test_predictions = AB.predict(test_x)
plt.scatter(test_y, test_predictions, s=50, c="green", label="Test Data", alpha=1)

plt.plot([-1, np.max([y, ypred])], [-1, np.max([y, ypred])], c="red", linewidth=3,
         linestyle="--", label="Perfect prediction")
text = "R2 = " + str(np.round(R2,3))

plt.annotate(text, (1,0),fontsize=15)
plt.xlabel("Rate of Penetration (m/h)",fontsize=15)
plt.ylabel("ML obtained Rate of Penetration (m/h)",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(loc = 'best', fontsize=15)

plt.show()
plt.rcParams["figure.figsize"] = (20,20)

#Regression Metrics
print('Mean Absolute Error:', round(np.mean(errors), 10), 'units')
print('MAPE:',MAPE, 'Accuracy:',round(accuracy, 2), '%.')
print('R2=',R2)
print('MSE:',MSE)
print('RMSE:',RMSE)
print('Median Absolute Error:',MedAE)

```

In [ ]:
```
x_ax = range(len(test_y))
plt.scatter(x_ax, test_y, s=5, color="blue", label="original")
plt.plot(x_ax, test_predictions, lw=0.5, color="red", label="predicted")

plt.legend()
plt.rcParams["figure.figsize"] = (20,5)

plt.show()
```

In [ ]:
```
df1=pd.DataFrame()

df1['ROP'] = df['Rate of Penetration m/h']
df1['Perfect Prediction'] = ypred
df1['Perfect Prediction HT'] = ypredHT
pd.set_option("display.max_rows", None, "display.max_columns", None)

df1
```

A.5 Predictions

Import the relevant libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot  as plt
import warnings
from numpy.random import seed
import tensorflow as tf
import math
import pickle
from tensorflow import keras

warnings.filterwarnings('ignore')
seed(0)
tf.random.set_seed(0)
```

SKlearn model libraries

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import pydot

from sklearn.neural_network import MLPRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from tensorflow import keras

import seaborn as sns
import io
```

Predictions USROP_A 1 N-S_F-7d well

In [ ]:
```
from test_AB_7d import df1_7d
from test_GB_7d import df2_7d
from test_KN_7d import df3_7d
from test_LR_7d import df4_7d
from test_MLP_7d import df5_7d
from test_RF_7d import df6_7d
```

In [ ]:
```
df=pd.DataFrame()

df['ROP'] = df1_7d['ROP']
df['ABA Pred'] = df1_7d['Perfect Prediction HT']
df['GB Pred'] = df2_7d['Perfect Prediction HT']
df['KN Pred'] = df3_7d['Perfect Prediction HT']
df['LR Pred'] = df4_7d['Perfect Prediction']
df['MLP Pred'] = df5_7d['Perfect Prediction HT']
df['RF Pred'] = df6_7d['Perfect Prediction HT']

```

Plot predicted ROP USROP_A 1 N-S_F-7d well

In [ ]:
```
x_ax = range(len(df['ROP']))

plt.plot(x_ax, df['ROP'], lw=0.8, color="black", label="original")
#plt.plot(x_ax, df['ABA Pred'], lw=0.8, color="green", label="ABA Pred")
plt.plot(x_ax, df['GB Pred'], lw=0.8, color="darkviolet", label="GB Pred")
plt.plot(x_ax, df['KN Pred'], lw=0.8, color="blue", label="KN Pred")
#plt.plot(x_ax, df['LR Pred'], lw=0.8, color="red", label="LR Pred")
#plt.plot(x_ax, df['MLP Pred'], lw=0.8, color="darkorange", label="MLP Pred")
plt.plot(x_ax, df['RF Pred'], lw=0.8, color="green", label="RF Pred")


plt.legend()
plt.xlim([0, 6500])
plt.rcParams["figure.figsize"] = (20,10)
plt.show()
```

Predictions USROP_A 2 N-SH_F-14d well

In [ ]:
```
from test_AB_14d import df1_14d
from test_GB_14d import df2_14d
from test_KN_14d import df3_14d
from test_MLP_14d import df4_14d
from test_LR_14d import df5_14d
from test_RF_14d import df6_14d
```

In [ ]:
```
df=pd.DataFrame()

df['ROP'] = df1_14d['ROP']
df['ABA Pred'] = df1_14d['Perfect Prediction HT']
df['GB Pred'] = df2_14d['Perfect Prediction HT']
df['KN Pred'] = df3_14d['Perfect Prediction HT']
df['MLP Pred'] = df4_14d['Perfect Prediction HT']
df['LR Pred'] = df5_14d['Perfect Prediction']
df['RF Pred'] = df6_14d['Perfect Prediction HT']

```

Plot predicted ROP USROP_A 2 N-SH_F-14d well

In [ ]:
```
x_ax = range(len(df['ROP']))

plt.plot(x_ax, df['ROP'], lw=0.8, color="black", label="original")
#plt.plot(x_ax, df['ABA Pred'], lw=0.8, color="green", label="ABA Pred")
plt.plot(x_ax, df['GB Pred'], lw=0.8, color="darkviolet", label="GB Pred")
plt.plot(x_ax, df['KN Pred'], lw=0.8, color="blue", label="KN Pred")
#plt.plot(x_ax, df['LR Pred'], lw=0.8, color="red", label="LR Pred")
#plt.plot(x_ax, df['MLP Pred'], lw=0.8, color="darkorange", label="MLP Pred")
plt.plot(x_ax, df['RF Pred'], lw=0.8, color="green", label="RF Pred")


plt.legend()
plt.xlim([0, 47000])
plt.ylim([-5, 70])
plt.rcParams["figure.figsize"] = (20,10)
plt.show()
```

Predictions USROP_A 3 N-SH-F-15d well

In [ ]:
```
from test_AB_15d import df1_15d
from test_GB_15d import df2_15d
from test_KN_15d import df3_15d
from test_LR_15d import df4_15d
from test_MLP_15d import df5_15d
from test_RF_15d import df6_15d
```

In [ ]:
```
df=pd.DataFrame()

df['ROP'] = df1_15d['ROP']
df['ABA Pred'] = df1_15d['Perfect Prediction HT']
df['GB Pred'] = df2_15d['Perfect Prediction HT']
df['KN Pred'] = df3_15d['Perfect Prediction HT']
df['LR Pred'] = df4_15d['Perfect Prediction']
df['MLP Pred'] = df5_15d['Perfect Prediction HT']
df['RF Pred'] = df6_15d['Perfect Prediction HT']
```

Plot predicted ROP USROP_A 3 N-SH-F-15d well

In [ ]:
```
x_ax = range(len(df['ROP']))

plt.plot(x_ax, df['ROP'], lw=0.8, color="black", label="original")
#plt.plot(x_ax, df['ABA Pred'], lw=0.8, color="green", label="ABA Pred")
plt.plot(x_ax, df['GB Pred'], lw=0.8, color="darkviolet", label="GB Pred")
plt.plot(x_ax, df['KN Pred'], lw=0.8, color="blue", label="KN Pred")
#plt.plot(x_ax, df['LR Pred'], lw=0.8, color="red", label="LR Pred")
#plt.plot(x_ax, df['MLP Pred'], lw=0.8, color="darkorange", label="MLP Pred")
plt.plot(x_ax, df['RF Pred'], lw=0.8, color="green", label="RF Pred")


plt.legend()
plt.xlim([0, 54000])
plt.ylim([-5, 70])
plt.rcParams["figure.figsize"] = (20,10)
plt.show()
```

Predictions USROP_A 4 N-SH_F-15Sd well

In [ ]:
```
from test_AB_15Sd import df1_15Sd
from test_GB_15Sd import df2_15Sd
from test_KN_15Sd import df3_15Sd
from test_LR_15Sd import df4_15Sd
from test_MLP_15Sd import df5_15Sd
from test_RF_15Sd import df6_15Sd
```

In [ ]:
```
df=pd.DataFrame()

df['ROP'] = df1_15Sd['ROP']
df['ABA Pred'] = df1_15Sd['Perfect Prediction HT']
df['GB Pred'] = df2_15Sd['Perfect Prediction HT']
df['KN Pred'] = df3_15Sd['Perfect Prediction HT']
df['LR Pred'] = df4_15Sd['Perfect Prediction']
df['MLP Pred'] = df5_15Sd['Perfect Prediction HT']
df['RF Pred'] = df6_15Sd['Perfect Prediction HT']
```

Plot predicted USROP_A 4 N-SH_F-15Sd well

In [ ]:
```
x_ax = range(len(df['ROP']))

plt.plot(x_ax, df['ROP'], lw=0.8, color="black", label="original")
#plt.plot(x_ax, df['ABA Pred'], lw=0.8, color="green", label="ABA Pred")
plt.plot(x_ax, df['GB Pred'], lw=0.8, color="darkviolet", label="GB Pred")
plt.plot(x_ax, df['KN Pred'], lw=0.8, color="blue", label="KN Pred")
#plt.plot(x_ax, df['LR Pred'], lw=0.8, color="red", label="LR Pred")
#plt.plot(x_ax, df['MLP Pred'], lw=0.8, color="darkorange", label="MLP Pred")
plt.plot(x_ax, df['RF Pred'], lw=0.8, color="green", label="RF Pred")

plt.legend()
plt.xlim([0, 52000])
plt.rcParams["figure.figsize"] = (20,10)
plt.show()
```

Predictions USROP_A 5 N-SH-F-5d well

In [ ]:
```
from test_AB_5d import df1_5d
from test_GB_5d import df2_5d
from test_KN_5d import df3_5d
from test_LR_5d import df4_5d
from test_MLP_5d import df5_5d
from test_RF_5d import df6_5d
```

In [ ]:
```
df=pd.DataFrame()

df['ROP'] = df1_5d['ROP']
df['ABA Pred'] = df1_5d['Perfect Prediction HT']
df['GB Pred'] = df2_5d['Perfect Prediction HT']
df['KN Pred'] = df3_5d['Perfect Prediction HT']
df['LR Pred'] = df4_5d['Perfect Prediction']
df['MLP Pred'] = df5_5d['Perfect Prediction HT']
df['RF Pred'] = df6_5d['Perfect Prediction HT']
```

Plot predicted USROP_A 5 N-SH-F-5d well

In [ ]:
```
x_ax = range(len(df['ROP']))

plt.plot(x_ax, df['ROP'], lw=0.8, color="black", label="original")
#plt.plot(x_ax, df['ABA Pred'], lw=0.8, color="green", label="ABA Pred")
plt.plot(x_ax, df['GB Pred'], lw=0.8, color="darkviolet", label="GB Pred")
plt.plot(x_ax, df['KN Pred'], lw=0.8, color="blue", label="KN Pred")
#plt.plot(x_ax, df['LR Pred'], lw=0.8, color="red", label="LR Pred")
#plt.plot(x_ax, df['MLP Pred'], lw=0.8, color="darkorange", label="MLP Pred")
plt.plot(x_ax, df['RF Pred'], lw=0.8, color="green", label="RF Pred")

plt.legend()
plt.xlim([0, 18800])
plt.rcParams["figure.figsize"] = (20,10)
plt.show()
```

In [ ]:
```

```

Appendix B
Well Statistics

B.1 General Statistics

1. Well 1, USROP_A 0 N-NA_F-9_Ad3

	Measured Depth m	Weight on Bit kkgf	Average Standpipe Pressure kPa	Average Surface Torque kN,m	Rate of Penetration m/h	Average Rotary Speed rpm	Mud Flow In L/min	Mud Density In g/cm3	Diameter mm	Average Hookload kkgf	Hole Depth (TVD) m	USROP Gamma gAPI
count	13 746	13 746	13 746	13 746	13 746	13 746	13 746	13 746	13 746	13 746	13 746	13 746
mean	844,155	9,289	11 562,065	5,937	39,101	143,320	2 714,106	1,207	269,962	92,707	781,324	103,791
std	216,075	4,450	2 779,706	3,282	11,969	41,557	777,596	0,010	47,190	4,393	157,351	64,629
min	491,033	0,005	3 592,720	0,014	0,549	-	1 506,518	1,190	215,900	84,727	490,760	11,270
25 %	653,009	5,711	10 266,862	2,510	31,797	104,000	1 895,507	1,198	215,900	87,970	644,362	26,310
50 %	808,030	9,472	11 498,973	7,389	40,717	143,190	3 226,269	1,200	311,150	94,030	776,790	144,720
75 %	1 043,332	11,861	14 132,901	8,677	47,512	193,000	3 447,223	1,210	311,150	96,815	929,222	158,570
max	1 205,999	20,102	15 664,406	10,616	88,441	204,170	3 734,574	1,230	311,150	104,304	1 013,143	204,761

Table B. 1. Well Statistics, USROP_A 0 N-NA_F-9_Ad

2. Well 2, USROP_A 1 N-S_F-7d

	Measured Depth m	Weight on Bit kkgf	Average Standpipe Pressure kPa	Average Surface Torque kN,m	Rate of Penetration m/h	Average Rotary Speed rpm	Mud Flow In L/min	Mud Density In g/cm3	Diameter mm	Average Hookload kkgf	Hole Depth (TVD) m	USROP Gamma gAPI
count	6389	6389	6389	6389	6389	6389	6389	6389	6389	6389	6389	6389
mean	472,31	4,65	12311,97	3,83	55,27	176,79	3915,37	1,03	444,50	98,78	472,27	83,04
std	98,16	1,59	2293,93	1,17	16,03	39,59	477,31	0,00	0,00	2,32	98,11	18,70
min	301,23	0,01	8949,00	0,31	8,77	103,00	3433,90	1,03	444,50	91,99	301,22	2,42
25 %	385,12	3,59	9997,00	3,45	44,79	143,00	3434,86	1,03	444,50	97,06	385,11	79,75
50 %	474,00	4,60	11723,00	3,94	57,34	192,00	3732,15	1,03	444,50	98,58	474,02	86,99
75 %	563,21	5,74	14636,00	4,51	63,95	212,00	4426,36	1,03	444,50	100,37	563,18	94,24
max	633,54	10,02	17754,00	7,22	98,11	212,00	4431,42	1,03	444,50	104,33	633,32	120,83

Table B. 2. Well Statistics, USROP_A 1 N-S_F-7d

3. Well 3, USROP_A 2 N-SH_F-14d

	Measured Depth m	Weight on Bit kkgf	Average Standpipe Pressure kPa	Average Surface Torque kN,m	Rate of Penetration m/h	Average Rotary Speed rpm	Mud Flow In L/min	Mud Density In g/cm3	Diameter mm	Average Hookload kkgf	Hole Depth (TVD) m	USROP Gamma gAPI
count	47645	47645	47645	47645	47645	47645	47645	47645	47645	47645	47645	47645
mean	2279,192	5,667	17083,348	11,147	24,571	137,107	3179,510	1,295	340,994	132,833	2199,297	65,181
std	733,447	2,638	3294,108	2,665	13,285	35,336	1038,328	0,070	107,498	13,508	638,556	36,397
min	987,948	0,469	4509,000	1,610	0,330	0,000	432,050	1,020	215,900	87,216	987,460	0,000
25 %	1617,223	4,171	15233,000	9,380	12,460	119,790	1984,990	1,270	215,900	128,953	1616,733	37,223
50 %	2277,889	5,364	17405,000	11,240	23,320	159,190	3523,390	1,310	444,500	138,284	2276,386	60,543
75 %	2931,021	6,547	18242,999	12,760	34,860	160,320	4155,930	1,350	444,500	140,435	2827,336	94,433
max	3466,033	15,092	24907,001	26,550	56,310	181,550	4538,450	1,380	444,500	152,927	2993,804	260,899

Table B. 3. Well Statistics, USROP_A 2 N-SH_F-14d

4. Well 4, USROP_A 3 N-SH-F-15d

	Measured Depth m	Weight on Bit kkgf	Average Standpipe Pressure kPa	Average Surface Torque kN,m	Rate of Penetration m/h	Average Rotary Speed rpm	Mud Flow In L/min	Mud Density In g/cm3	Diameter mm	Average Hookload kkgf	Hole Depth (TVD) m	USROP Gamma gAPI
count	53041	53041	53041	53041	53041	53041	53041	53041	53041	53041	53041	53041
mean	2640,153	6,183	17777,027	19,147	21,577	130,629	2933,601	11,453	303,343	129,058	2333,879	74,797
std	883,692	4,657	4138,872	7,894	9,628	19,348	1090,260	0,443	111,101	4,541	633,089	58,569
min	1306,525	0,005	4363,623	1,098	0,786	0,000	1083,309	10,682	215,900	114,795	1283,159	0,000
25 %	1668,576	1,891	14705,635	11,443	15,597	129,500	2077,184	11,183	215,900	126,136	1618,122	27,690
50 %	2845,860	5,080	15767,427	19,213	19,687	139,736	2121,383	11,266	215,900	128,303	2619,993	51,765
75 %	3302,752	9,430	22788,690	26,708	29,688	139,736	4408,794	11,934	444,500	130,839	2860,925	135,690
max	4065,346	19,858	24993,309	36,489	99,206	140,351	4453,121	12,017	444,500	149,743	3189,315	256,164

Table B. 4. Well Statistics, USROP_A 3 N-SH-F-15

5. Well 5, USROP_A 4 N-SH_F-15Sd

	Measured Depth m	Weight on Bit kkgf	Average Standpipe Pressure kPa	Average Surface Torque kN,m	Rate of Penetration m/h	Average Rotary Speed rpm	Mud Flow In L/min	Mud Density In g/cm3	Diameter mm	Average Hookload kkgf	Hole Depth (TVD) m	USROP Gamma gAPI
count	51708	51708	51708	51708	51708	51708	51708	51708	51708	51708	51708	51708
mean	2928,763	5,782	19160,422	14,434	17,326	174,271	2572,733	1,369	247,425	134,654	2484,778	57,162
std	815,094	4,231	3485,707	2,685	8,874	59,162	1037,703	0,066	44,822	6,947	519,979	49,610
min	1400,550	0,002	1432,662	0,008	0,399	0,000	185,421	1,300	215,900	84,048	1367,019	0,000
25 %	2183,961	3,080	15718,164	12,121	10,022	120,000	1705,919	1,320	215,900	129,147	2055,263	15,640
50 %	3056,812	4,345	20246,145	14,630	17,090	179,236	2016,073	1,320	215,900	135,112	2652,020	42,161
75 %	3664,481	7,440	21835,421	16,822	24,954	217,770	3987,856	1,450	311,150	140,435	2900,238	90,350
max	4090,001	31,411	24998,459	24,228	96,660	290,560	4173,644	1,480	311,150	152,213	3171,809	255,462

Table B. 5. Well Statistics, USROP_A 4 N-SH_F-15Sd

6. Well 6, USROP_A 5 N-SH-F-5d

	Measured Depth m	Weight on Bit kkgf	Average Standpipe Pressure kPa	Average Surface Torque kN,m	Rate of Penetration m/h	Average Rotary Speed rpm	Mud Flow In L/min	Mud Density In g/cm3	Diameter mm	Average Hookload kkgf	Hole Depth (TVD) m	USROP Gamma gAPI
count	18548	18548	18548	18548	18548	18548	18548	18548	18548	18548	18548	18548
mean	3333,935	6,700	20923,118	23,852	26,481	177,861	2040,051	1,409	2.159e+02	137,693	2930,595	42,091
std	296,633	3,190	2483,364	3,951	15,918	75,932	187,352	0,066	6.443e-11	5,600	223,785	41,121
min	2828,239	0,010	9782,000	5,570	1,360	24,160	1056,920	1,280	2.159e+02	114,055	2530,262	0,000
25 %	3007,302	4,456	20841,000	21,780	12,620	129,940	2066,940	1,350	2.159e+02	132,767	2679,852	12,320
50 %	3363,657	6,312	21162,000	23,930	24,100	179,860	2089,570	1,450	2.159e+02	136,938	2974,288	29,490
75 %	3592,875	8,657	22205,000	27,130	40,670	262,530	2090,060	1,450	2.159e+02	143,117	3129,706	54,600
max	3792,200	14,847	24997,000	30,880	79,140	311,230	2774,260	1,470	2.159e+02	148,379	3248,390	260,060

Table B. 6. Well Statistics, USROP_A 5 N-SH-F-5d

B.2 Heat Maps.

1. Well 1, USROP_A 0 N-NA_F-9_Ad

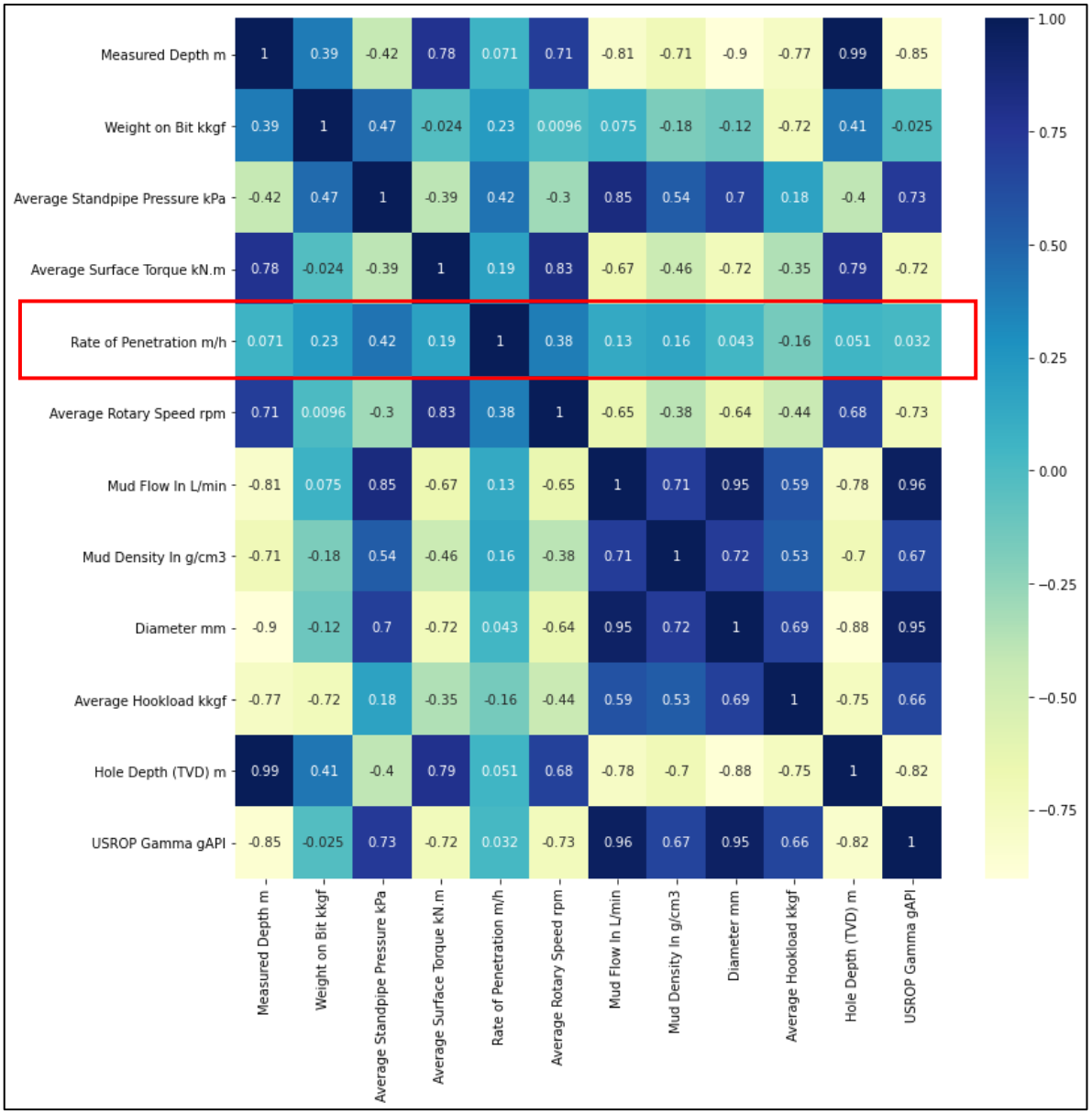

Figure B. 1. Heat Map USROP_A 0 N-NA_F-9_Ad

2. Well 2, USROP_A 1 N-S_F-7d

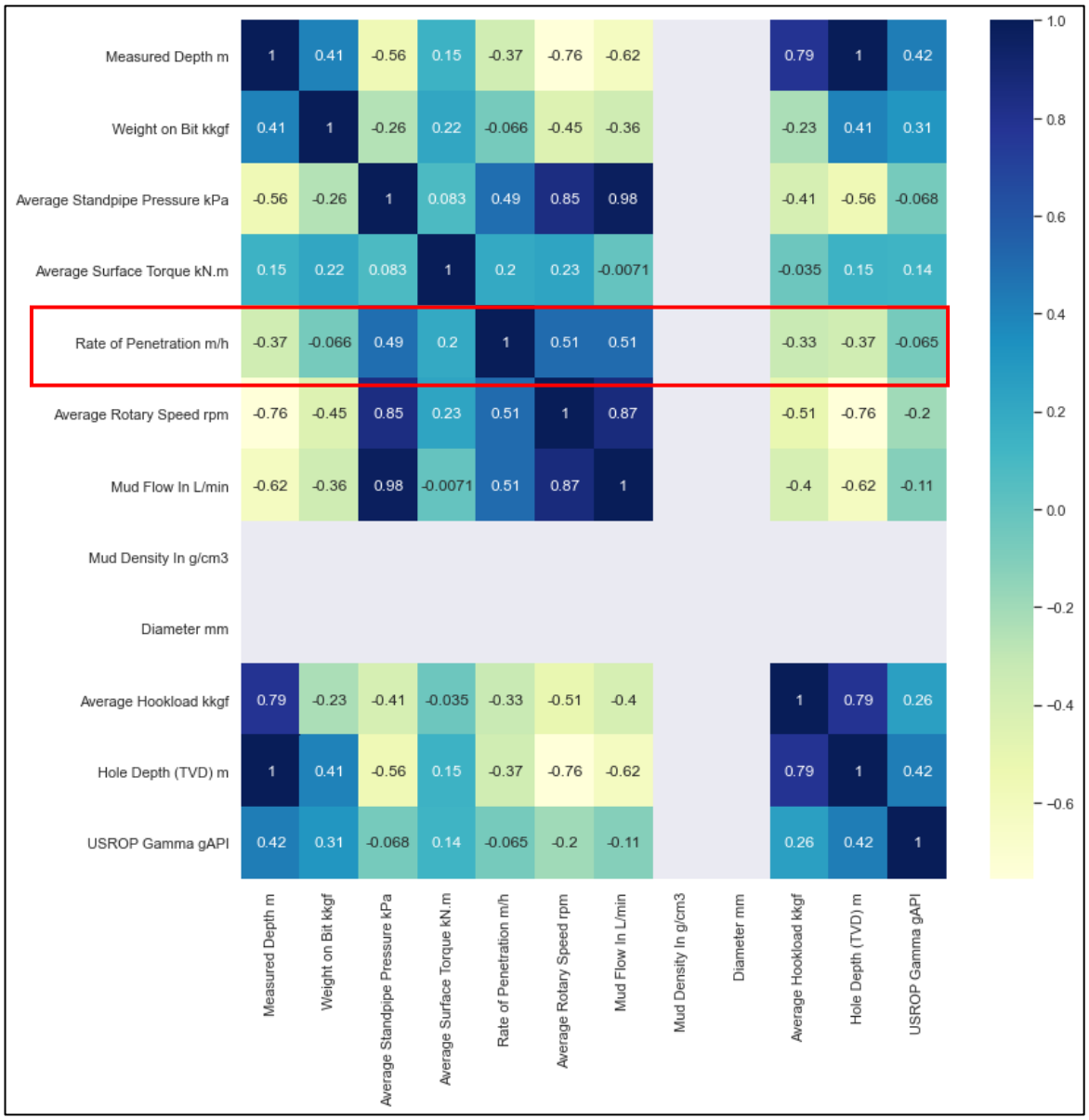

Figure B. 2. Heat Map USROP_A 1 N-S_F-7d

3. Well 3, USROP_A 2 N-SH_F-14d

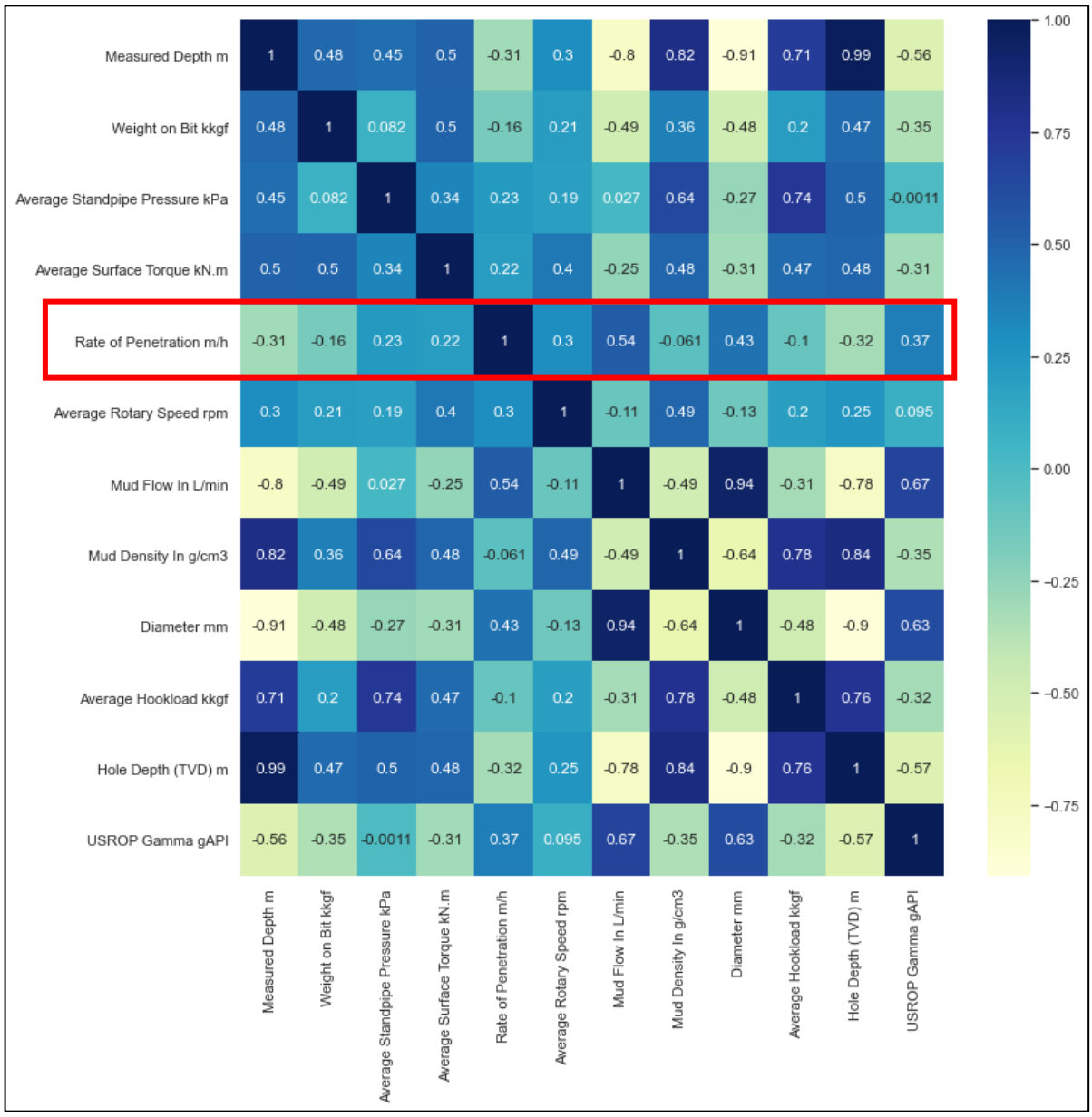

Figure B. 3. Heat Map USROP_A 2 N-SH_F-14d

4. Well 4, USROP_A 3 N-SH-F-15d

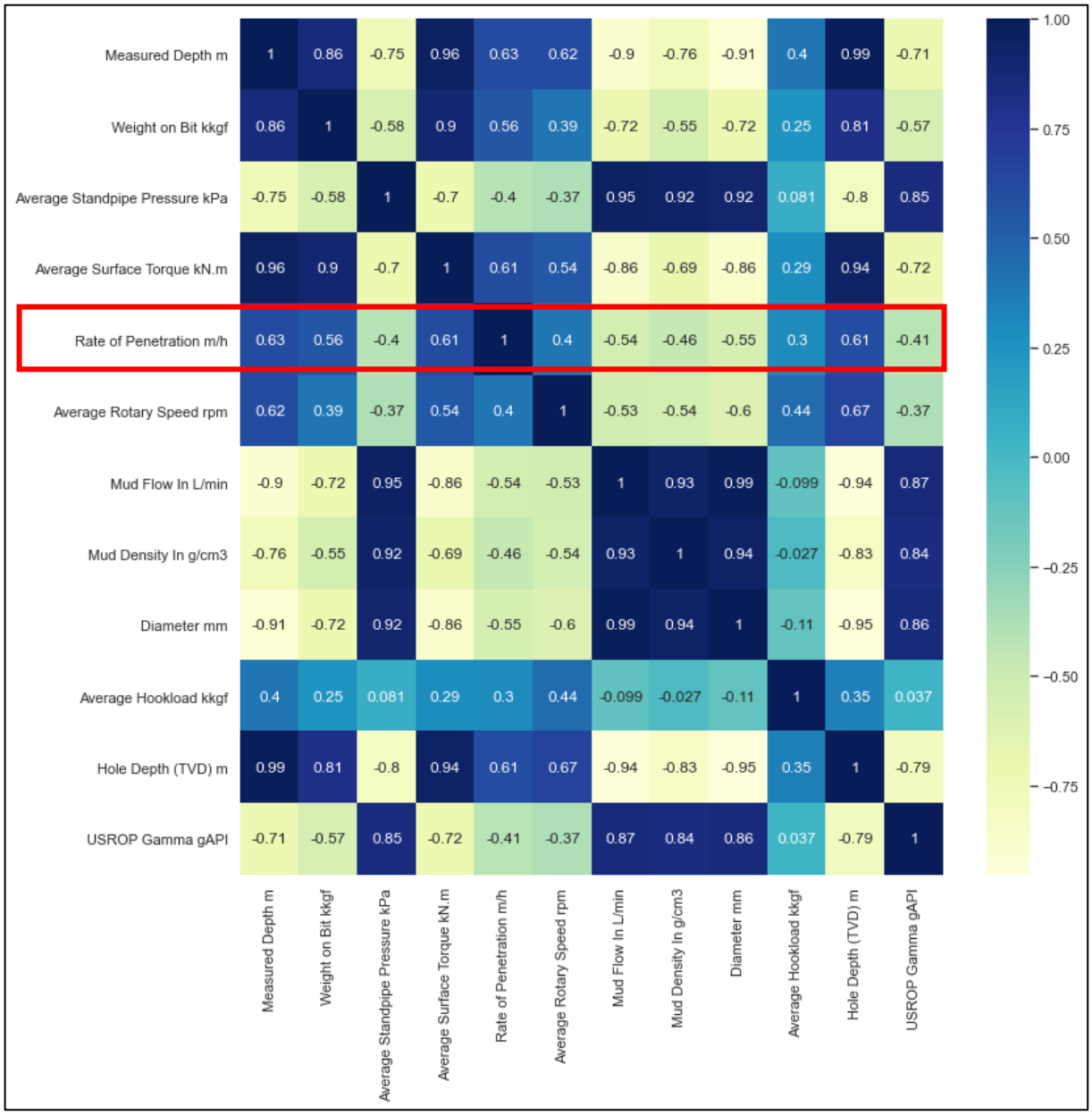

Figure B. 4. Heat Map USROP_A 3 N-SH-F-15d

5. Well 5, USROP_A 4 N-SH_F-15Sd

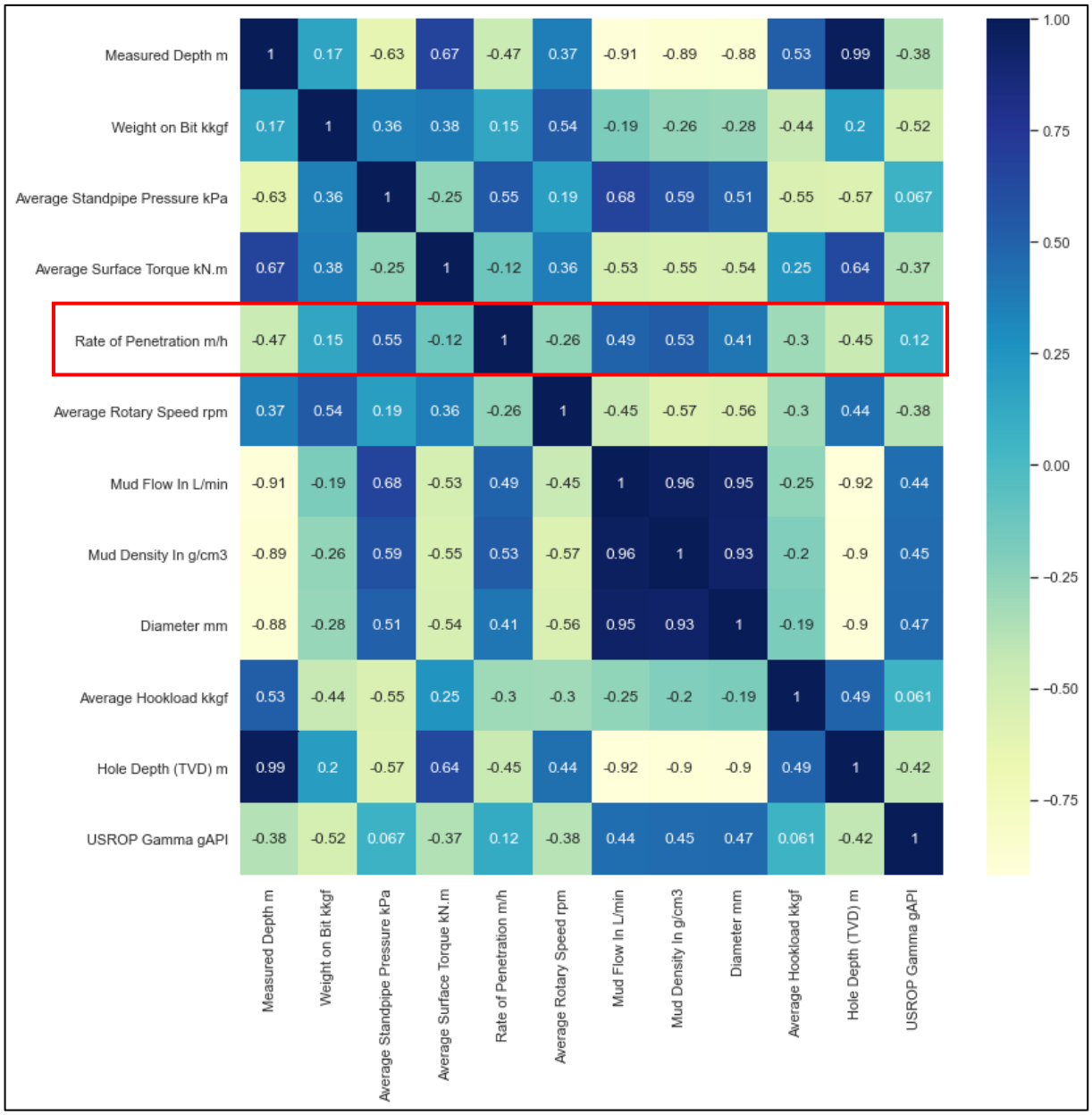

Figure B. 5. Heat Map USROP_A 4 N-SH_F-15Sd

6. Well 6, USROP_A 5 N-SH-F-5d

Figure B. 6. Heat Map USROP_A 5 N-SH-F-5d

B.3 Probability Distribution

1. Well 1, USROP_A 0 N-NA_F-9_Ad

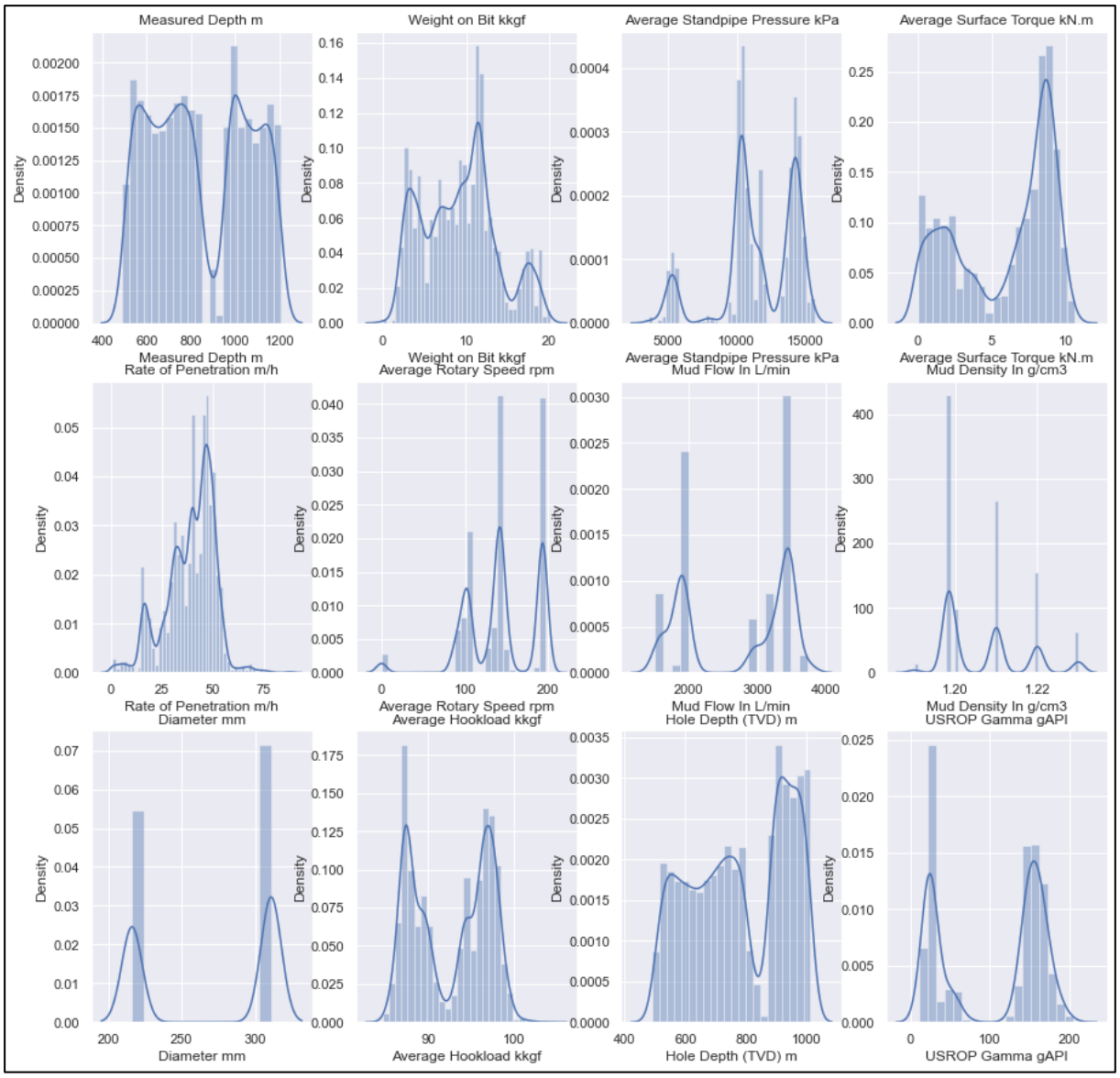

Figure B. 7. Probability Distribution Well 1

2. Well 2, USROP_A 1 N-S_F-7d

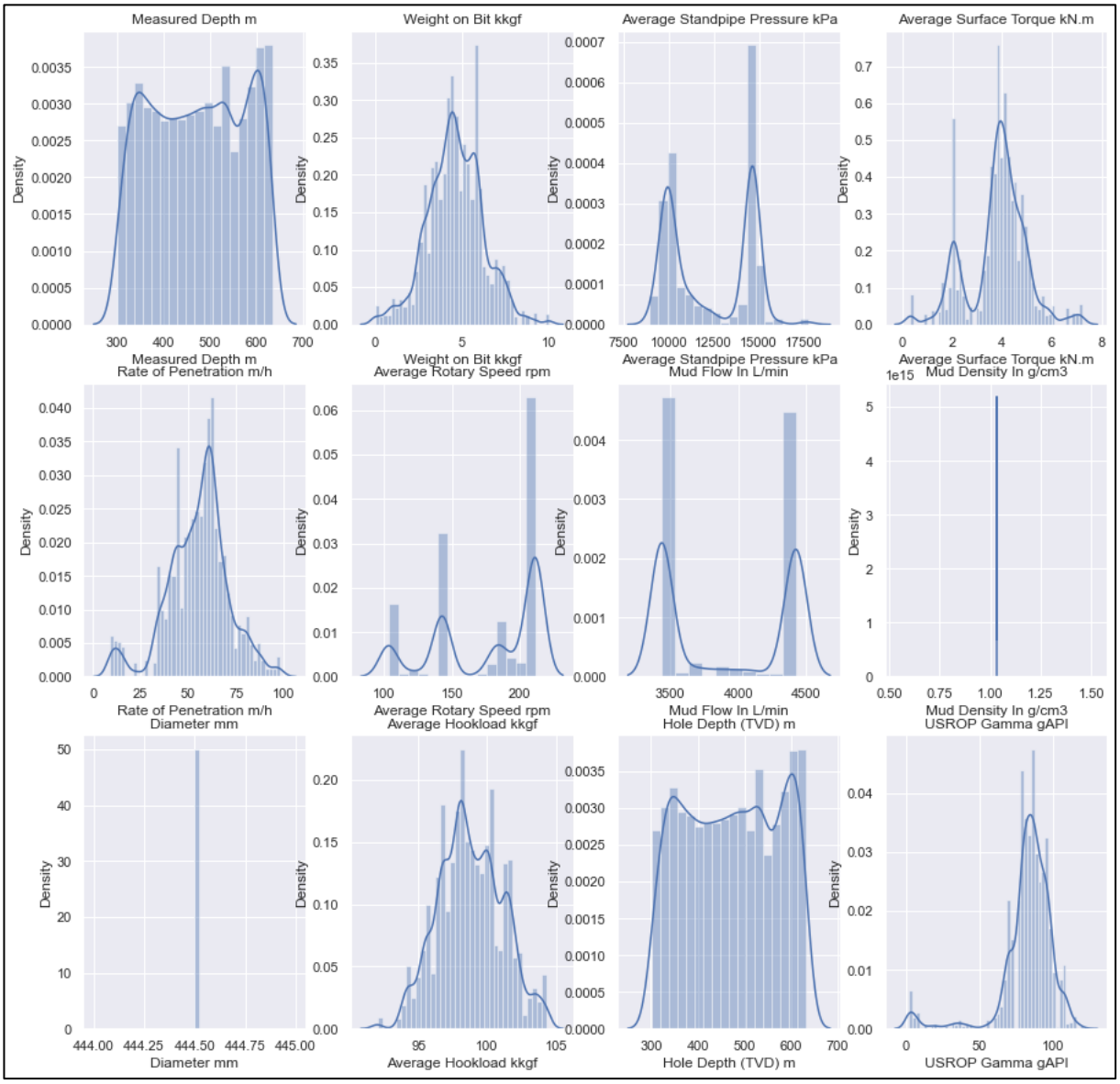

Figure B. 8. Probability Distribution Well 2

3. Well 3, USROP_A 2 N-SH_F-14d

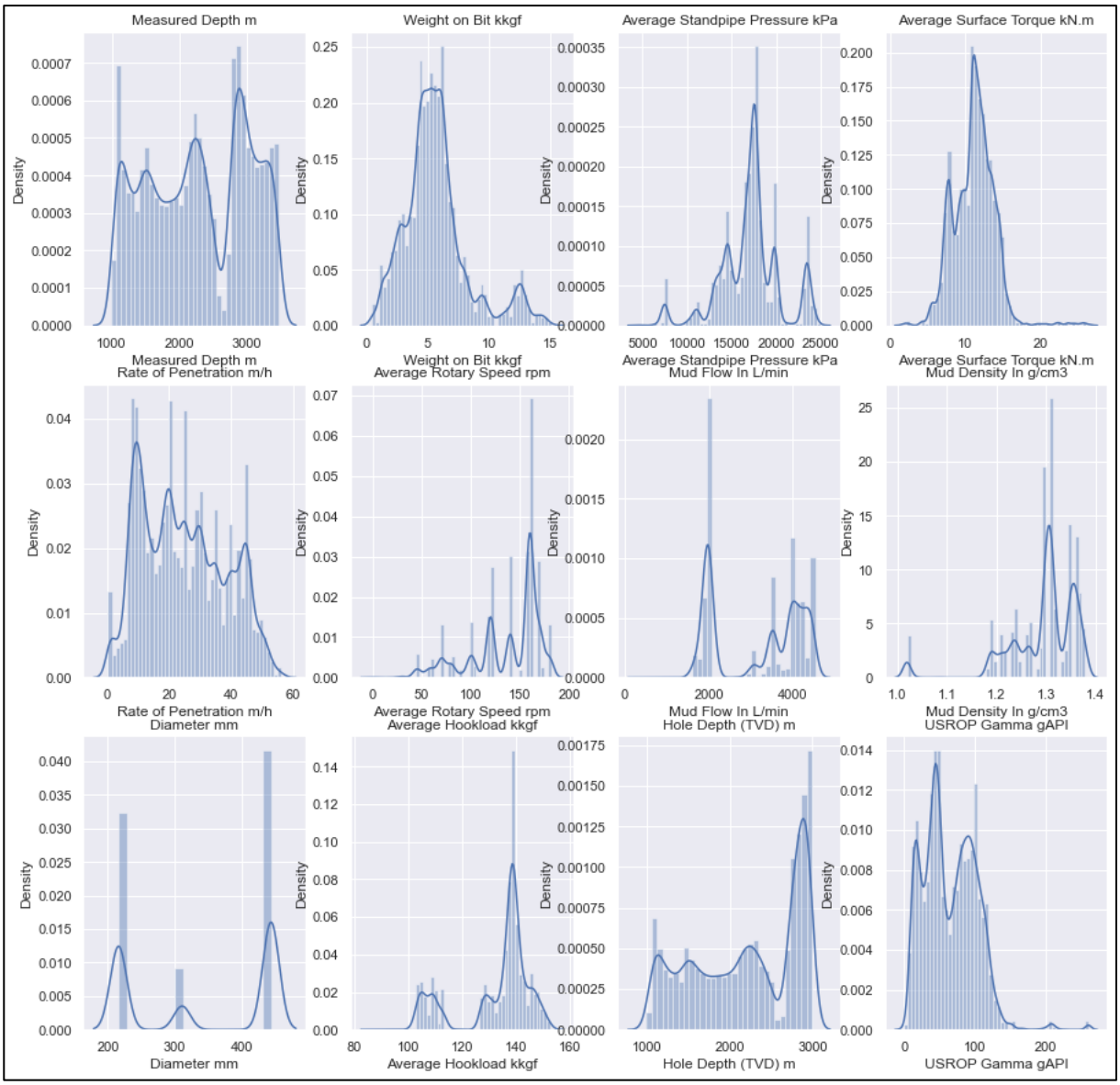

Figure B. 9. Probability Distribution Well 3

4. Well 4, USROP_A 3 N-SH-F-15d

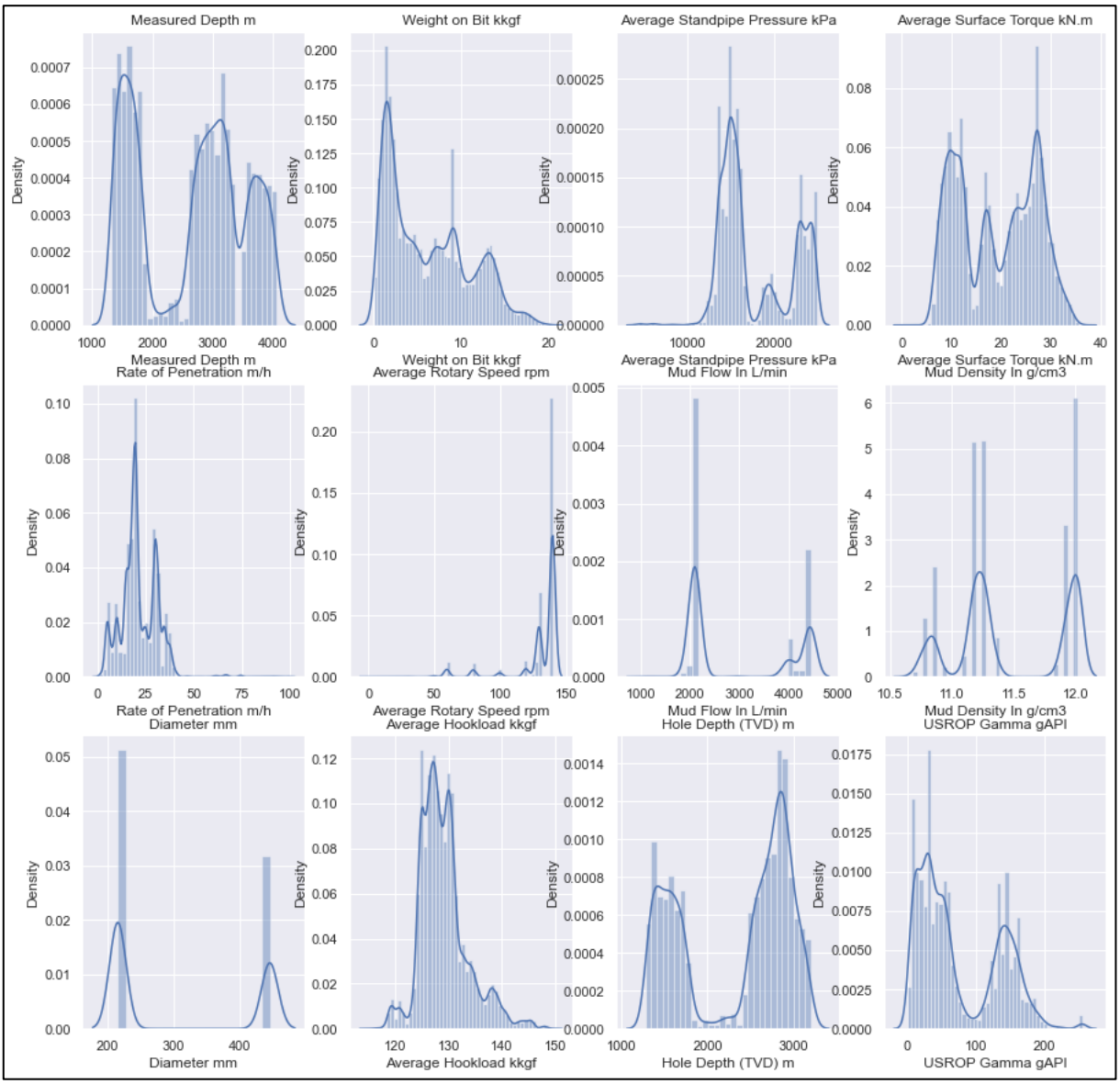

Figure B. 10. Probability Distribution Well 4

5. Well 5, USROP_A 4 N-SH_F-15Sd

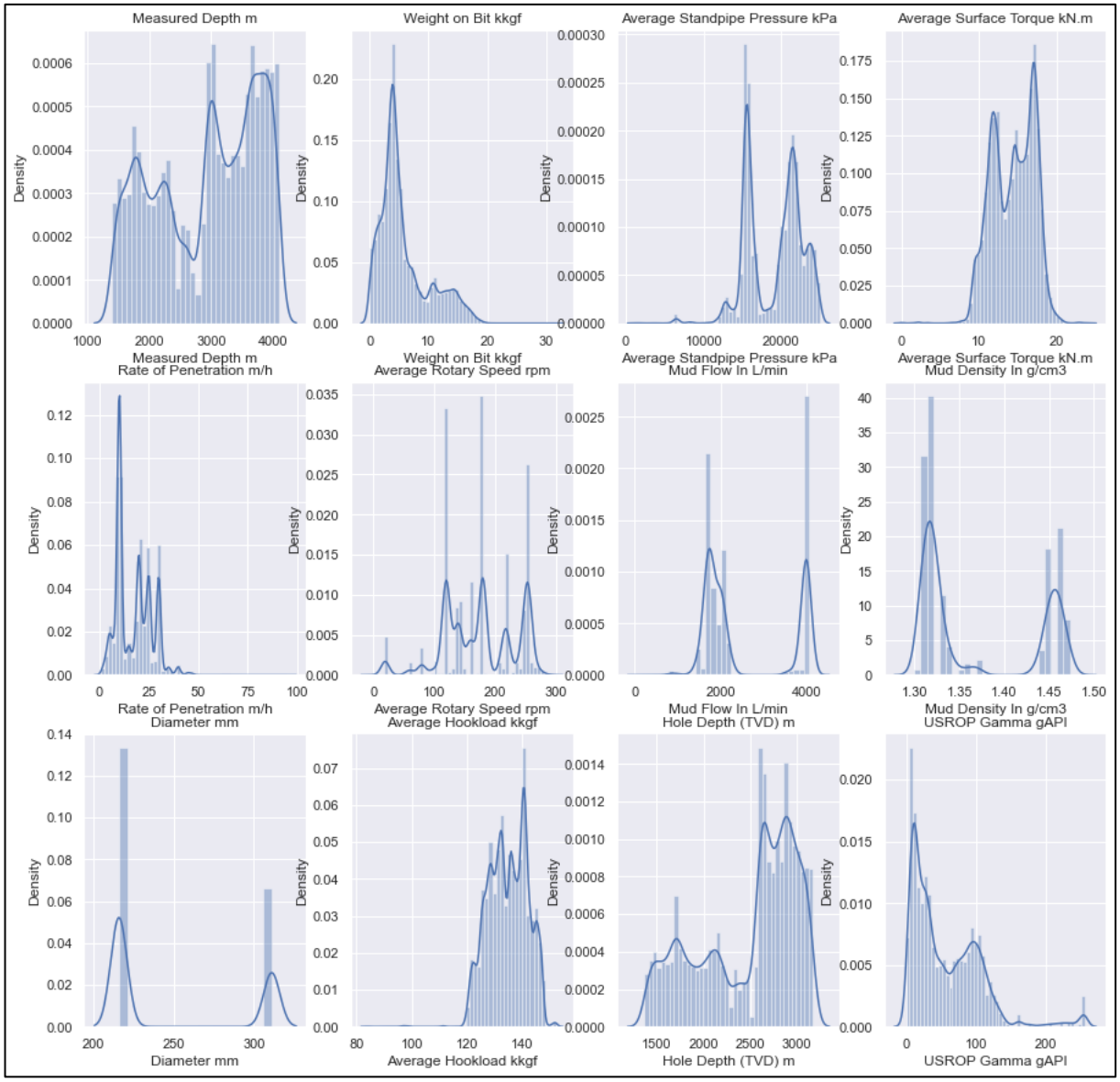

Figure B. 11. Probability Distribution Well 5

6. Well 6, USROP_A 5 N-SH-F-5d

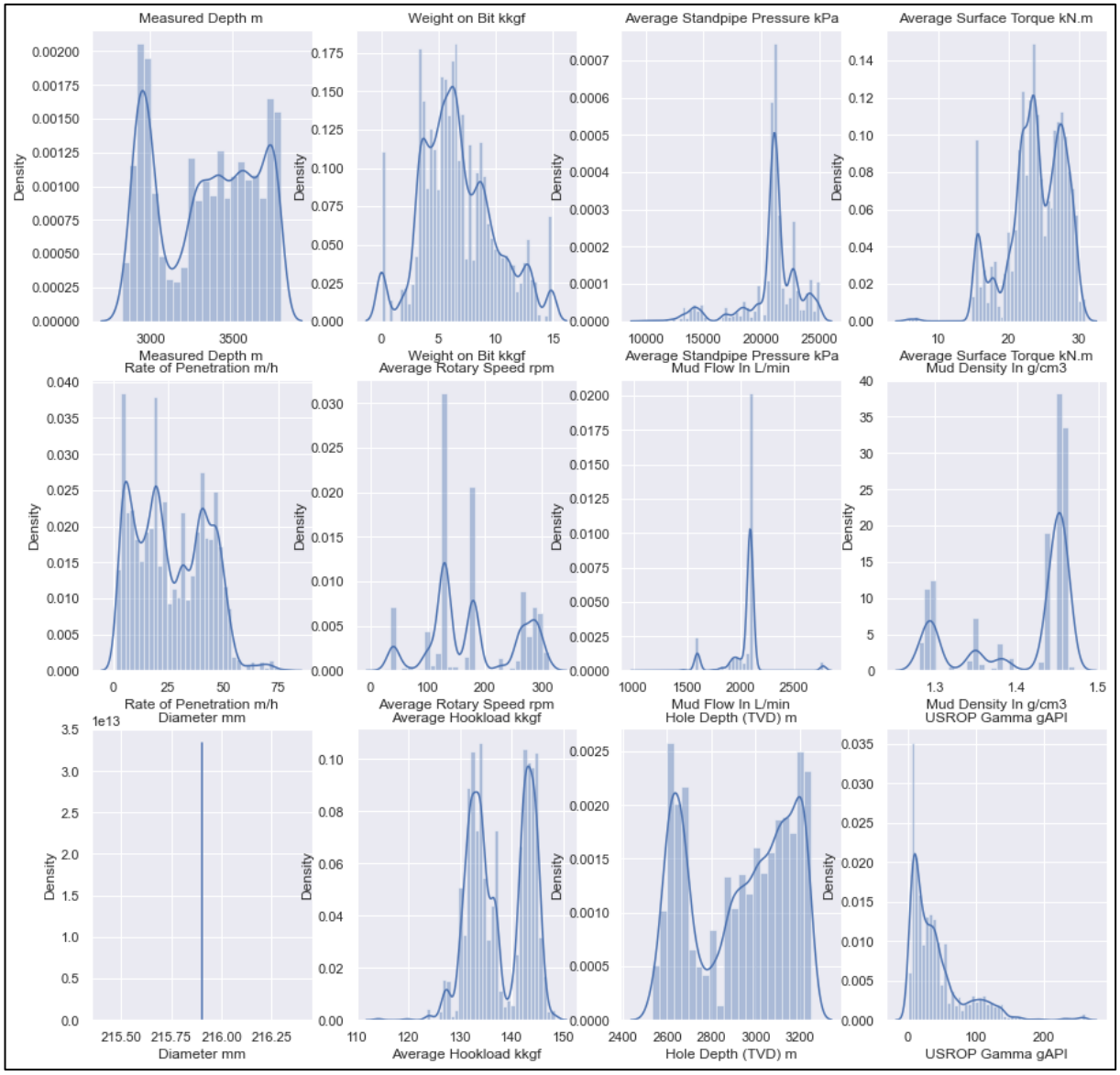

Figure B. 12. Probability Distribution Well 6

B.4 Plot Measure Depth (MD) vs Selected Variables.

1. Well 1, USROP_A o N-NA_F-9_Ad

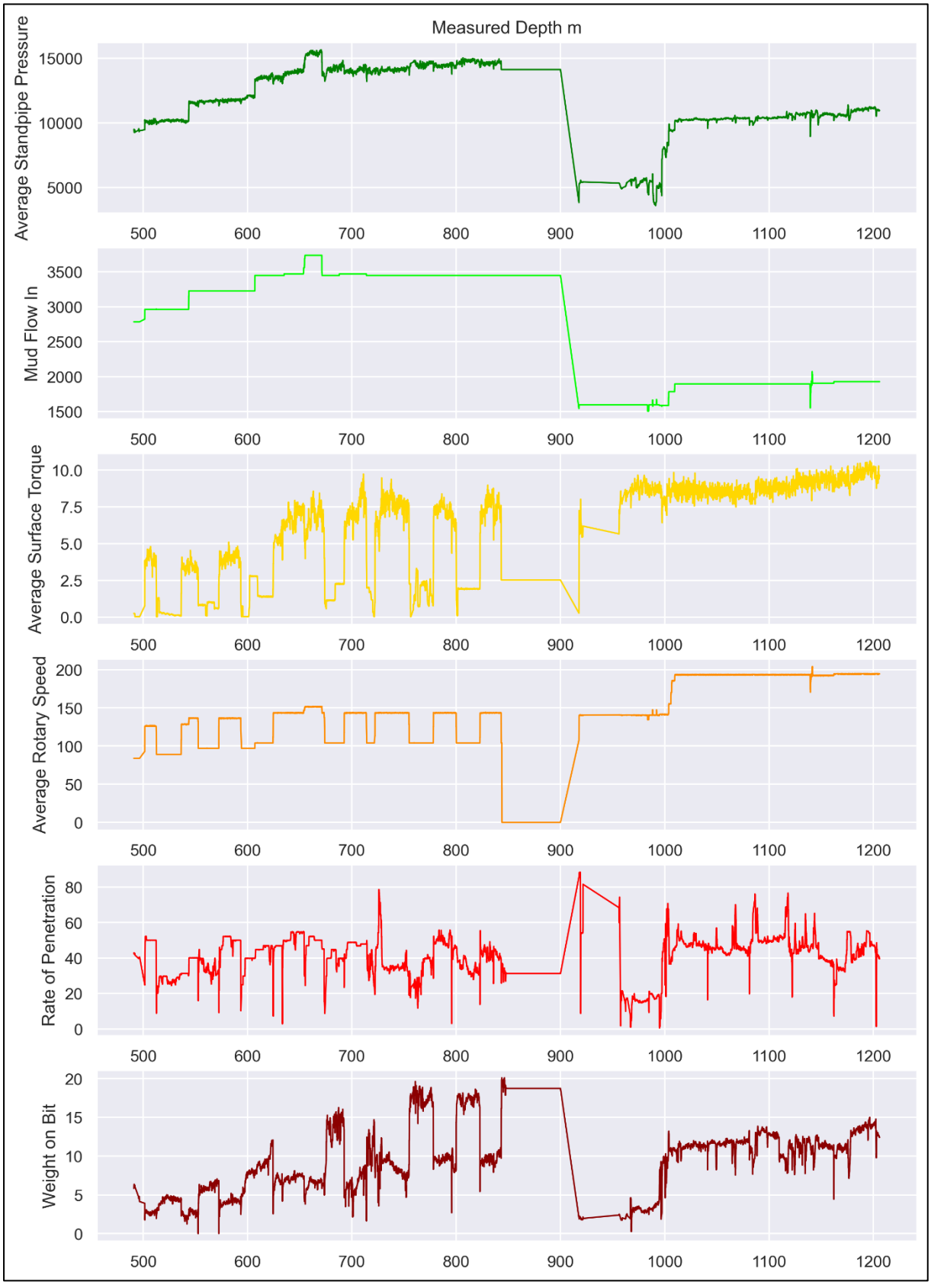

Figure B. 13. Measure depth vs. Selected variables Well 1

2. Well 2, USROP_A 1 N-S_F-7d

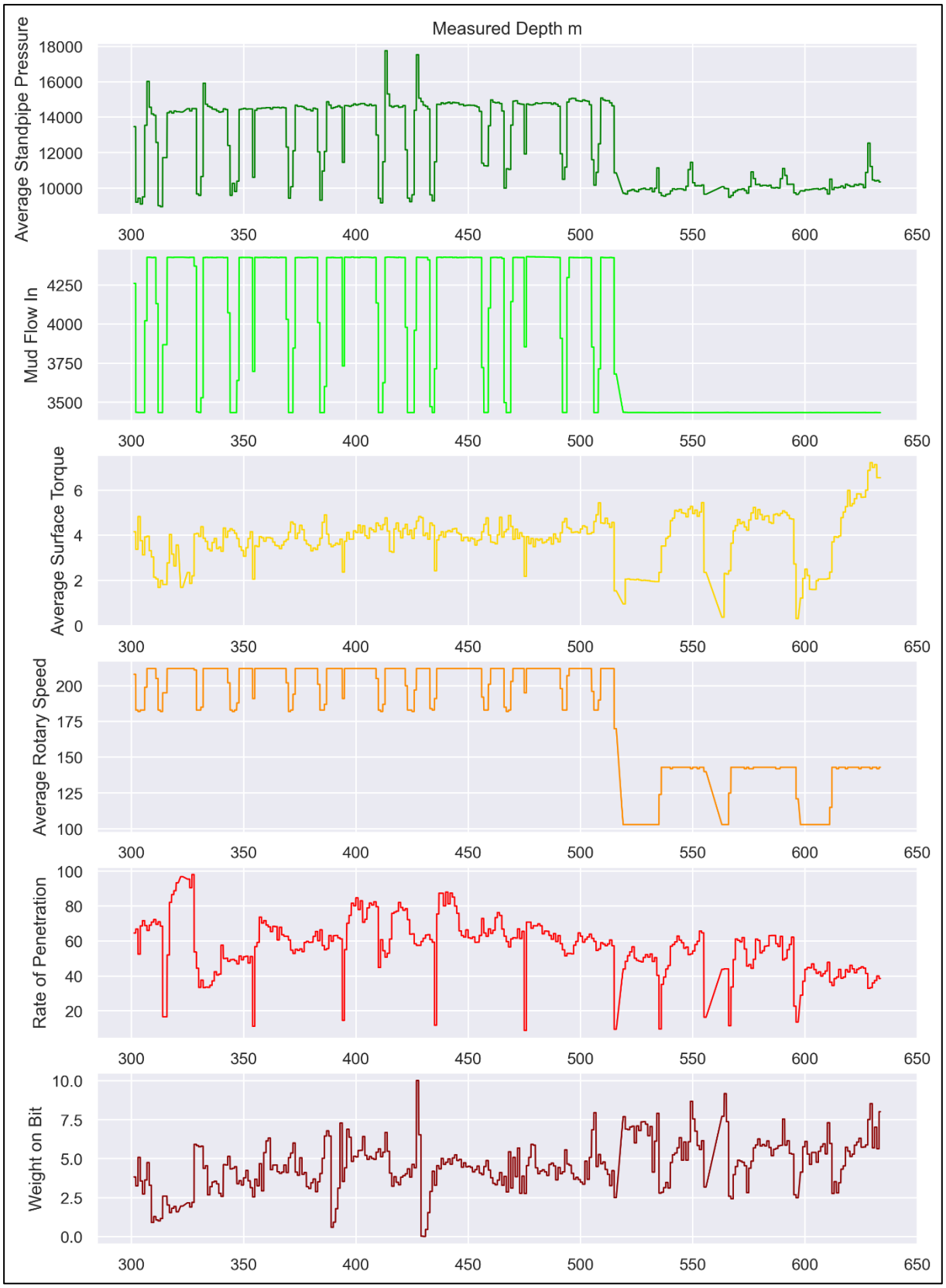

Figure B. 14. Measure depth vs. Selected variables Well 2

3. Well 3, USROP_A 2 N-SH_F-14d

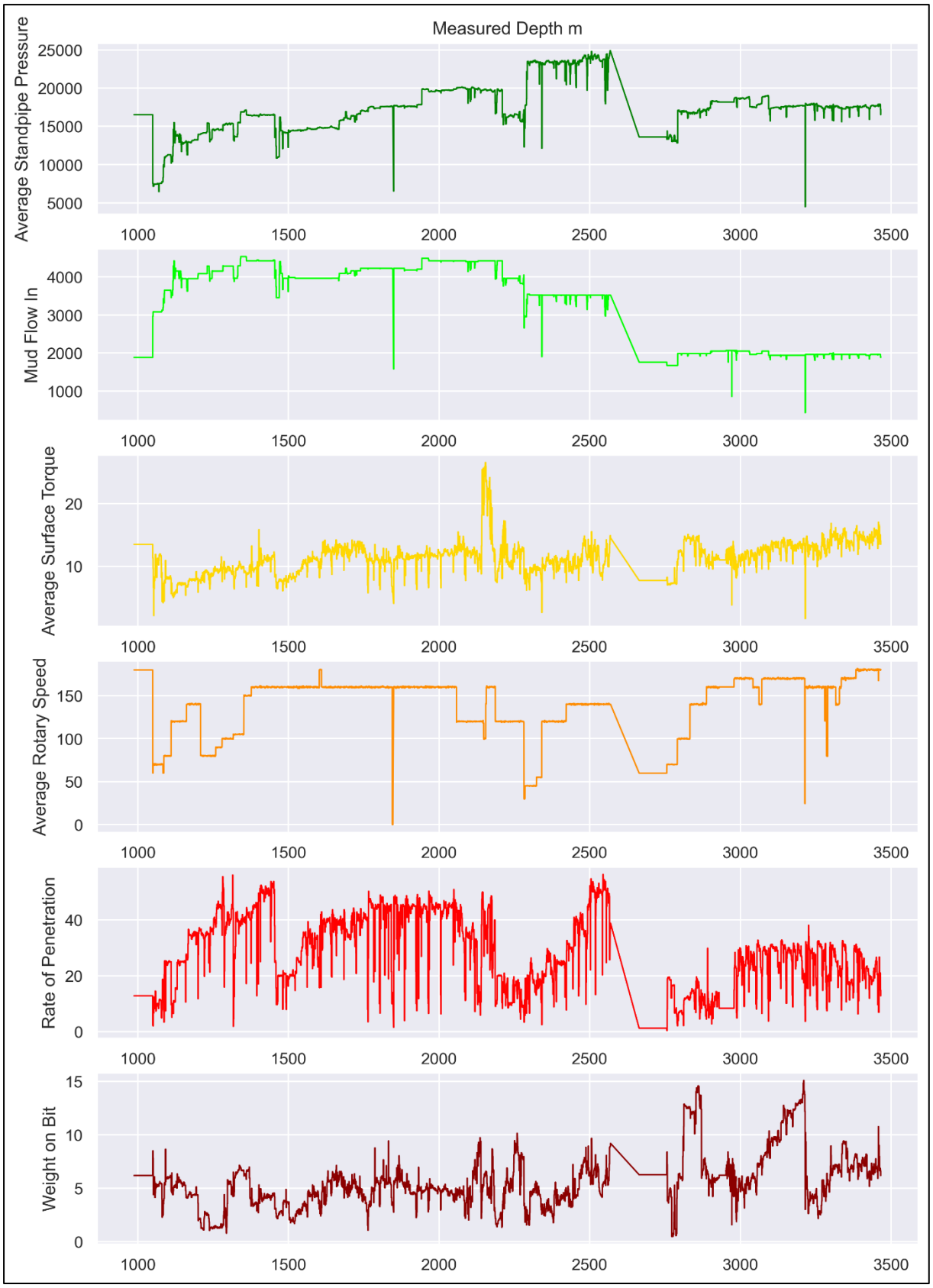

Figure B. 15. Measure depth vs. Selected variables Well 3

4. Well 4, USROP_A 3 N-SH-F-15d

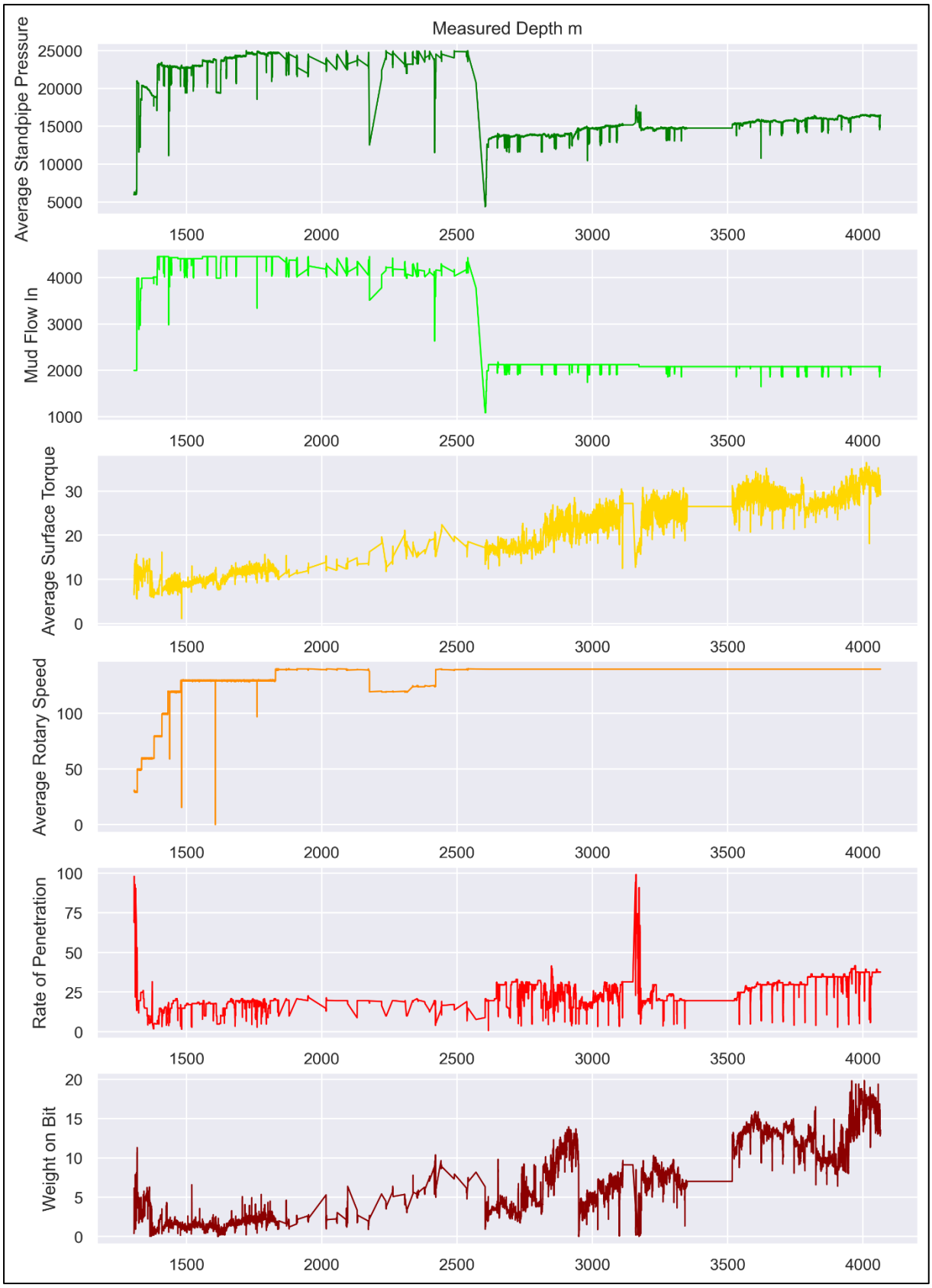

Figure B. 16. Measure depth vs. Selected variables Well 4

5. Well 5, USROP_A 4 N-SH_F-15Sd

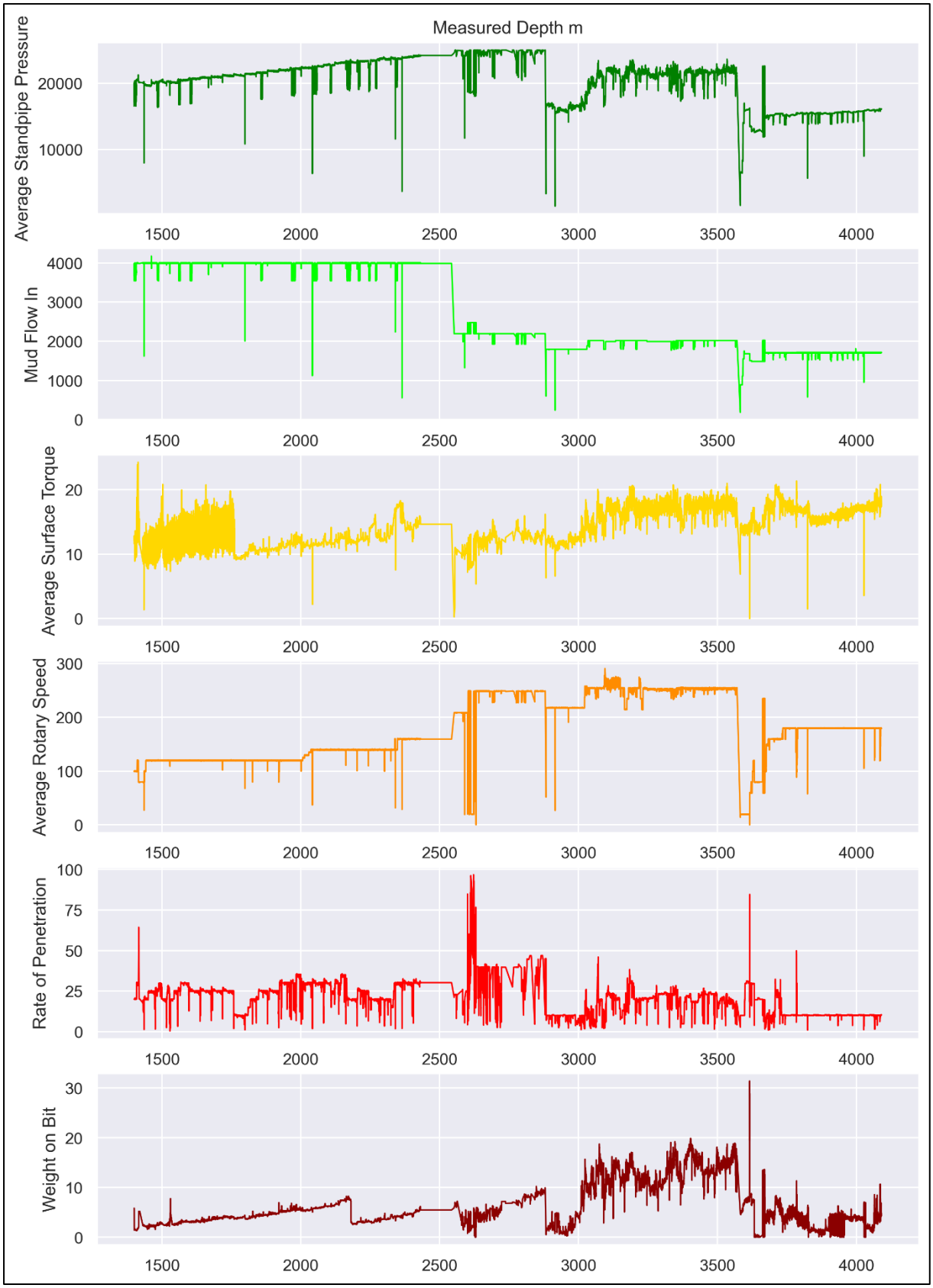

Figure B. 17. Measure depth vs. Selected variables Well 5

6. Well 6, USROP_A 5 N-SH-F-5d

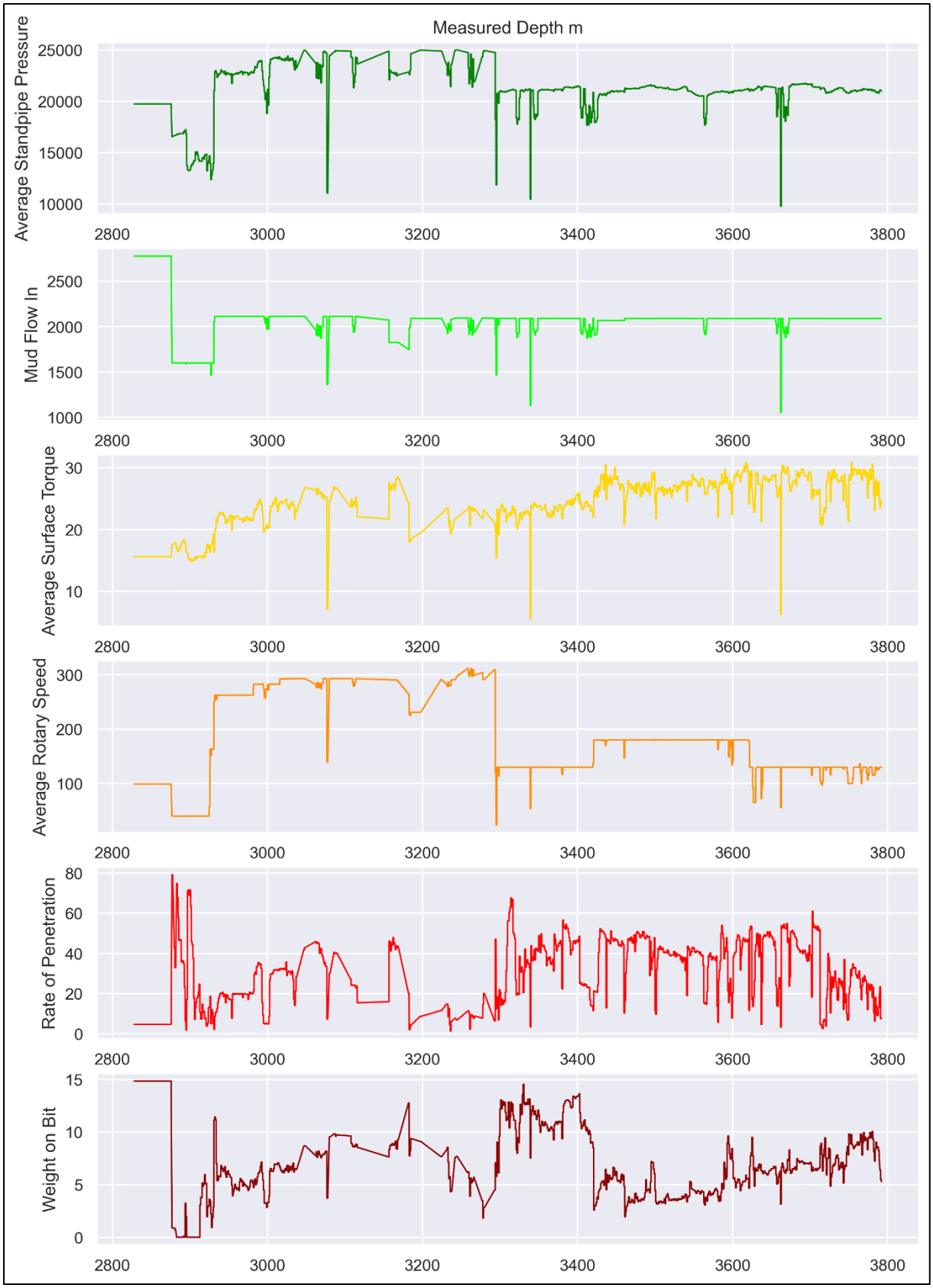

Figure B. 18. Measure depth vs. Selected variables Well 6

Appendix C
Machine Learning Results

C.1 Implementation of the Model.

1. Well 2, USROP_A 1 N-S_F-7d

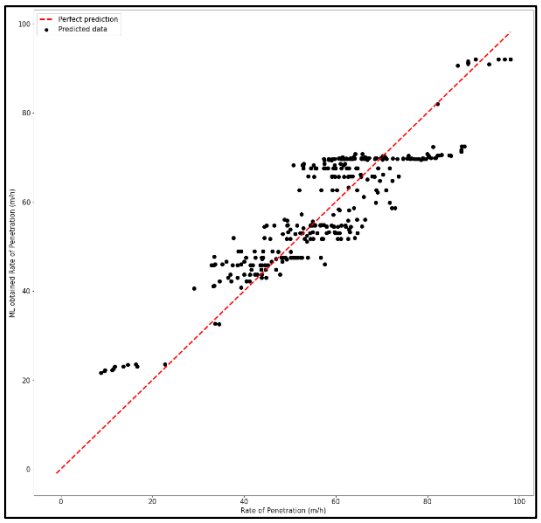

Figure C. 1 AdaBoost Reg, Model_Well 2

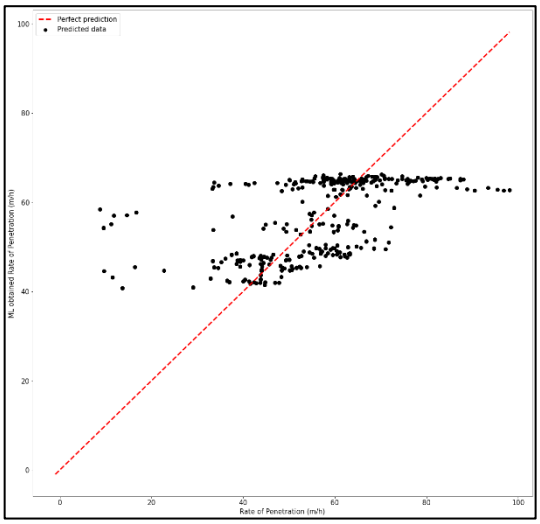

Figure C. 2 MLP Reg, Model_Well 2

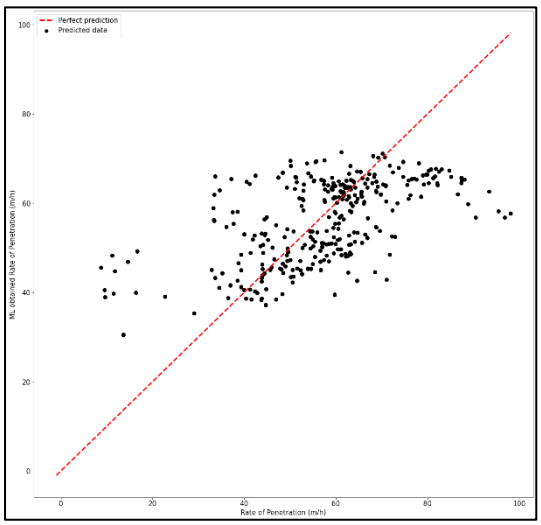

Figure C. 3 Linear Regression Model_Well 2

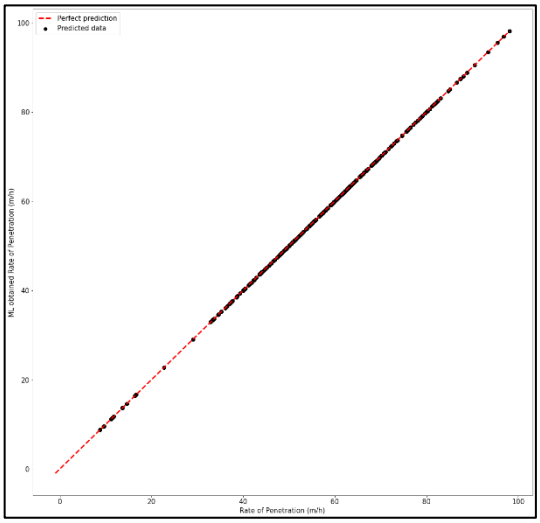

Figure C. 4 K- Neighbors Reg, Model_Well 2

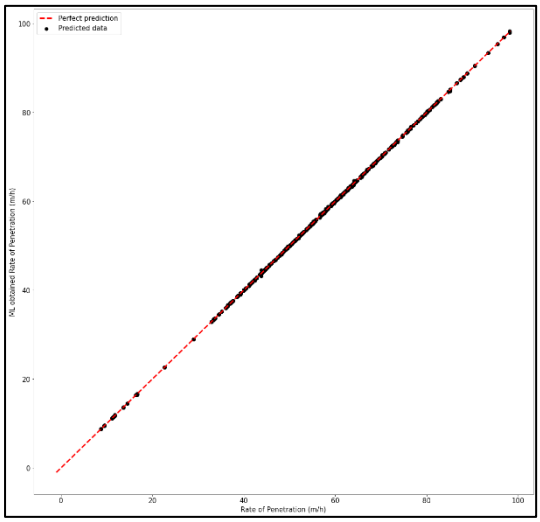

Figure C. 5 Gradient Boosting Regressor Model_Well 2

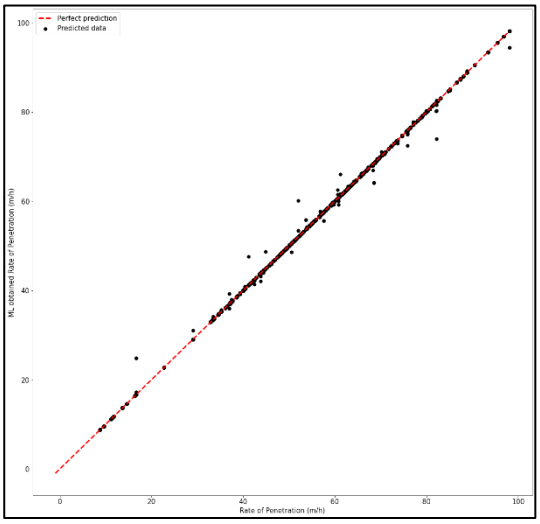

Figure C. 6 Random Forest Regressor Model_Well 2

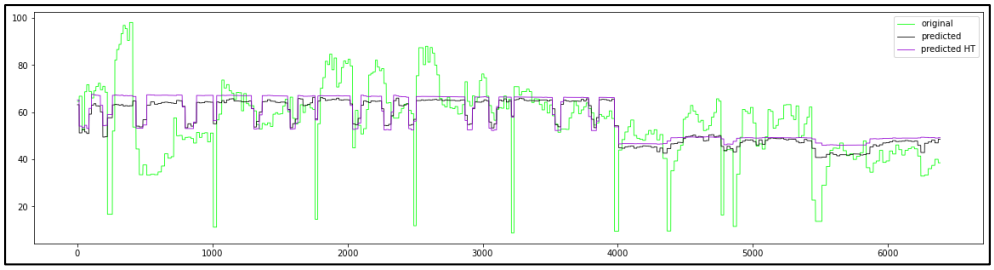

Figure C. 7 Multi-Layer Perceptron Regressor data set prediction_Well 2

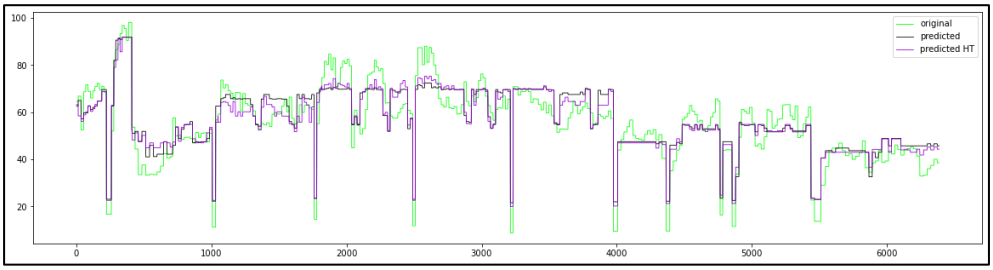

Figure C. 8 AdaBoost Regressor data set prediction_Well 2

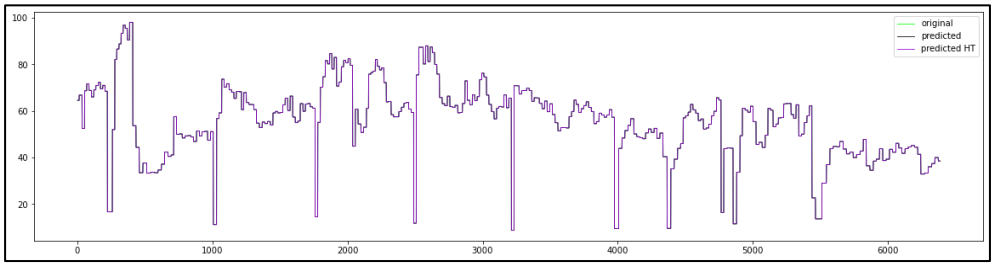

Figure C. 9 K-Neighbors Regressor data set prediction_Well 2

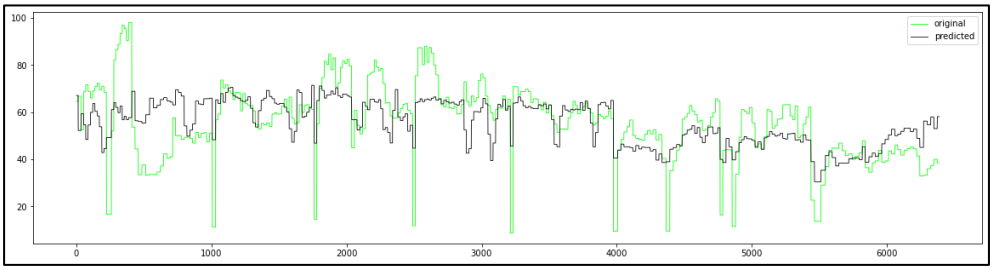

Figure C. 10 Linear Regression data set prediction_Well 2

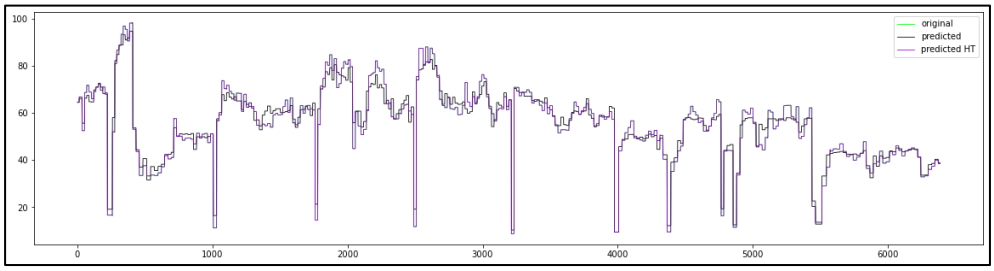

Figure C. 11 Gradient Boosting Regressor data set prediction_Well 2

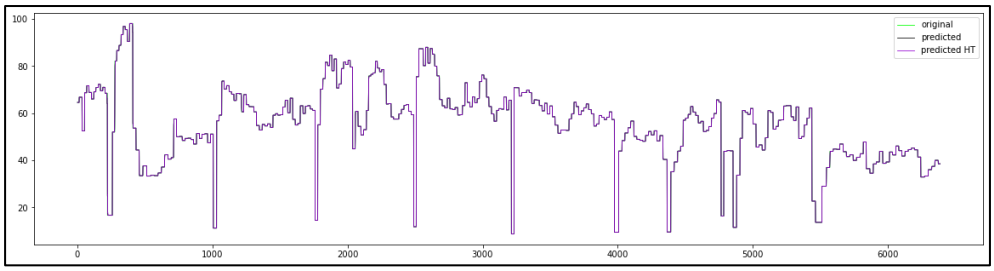

Figure C. 12 Random Forest Regressor data set prediction_Well 2

2. Well 3, USROP_A 2 N-SH_F-14d

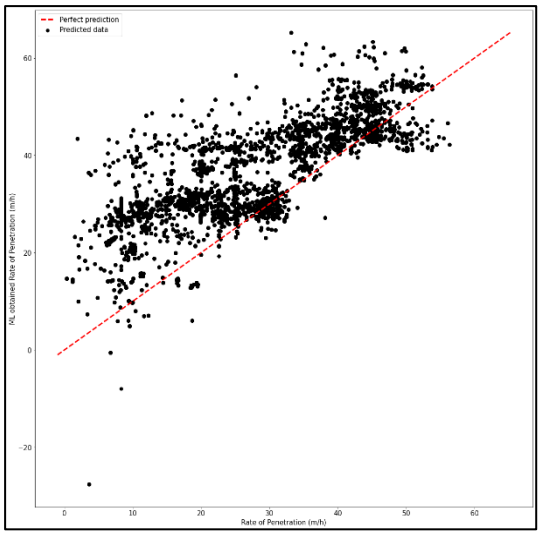

Figure C. 14 MLP Regressor Model_Well 3

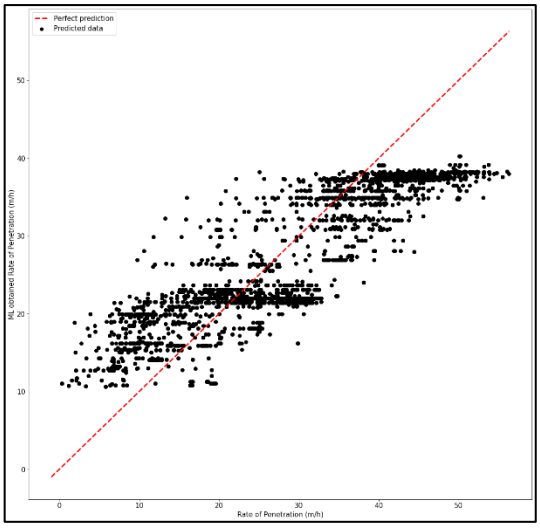

Figure C. 13 AdaBoost Regressor Model_Well 3

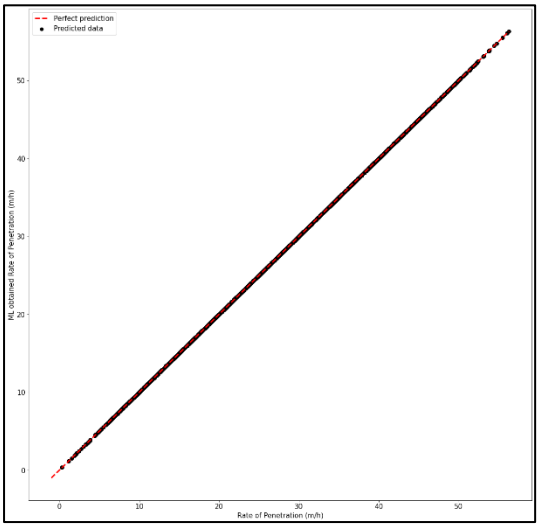

Figure C. 15 K- Neighbors Reg Model_Well 3

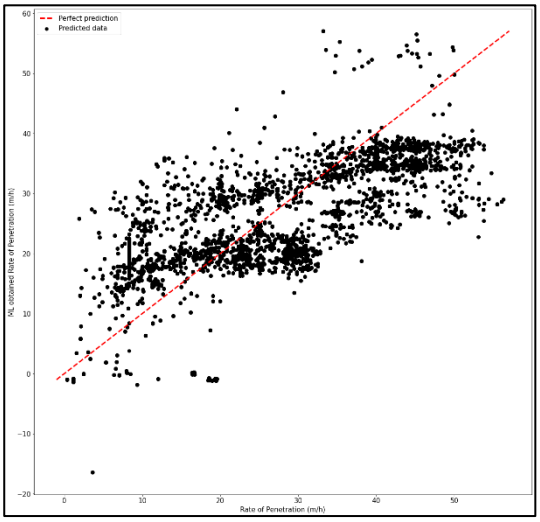

Figure C. 16 Linear Regression Model_Well 3

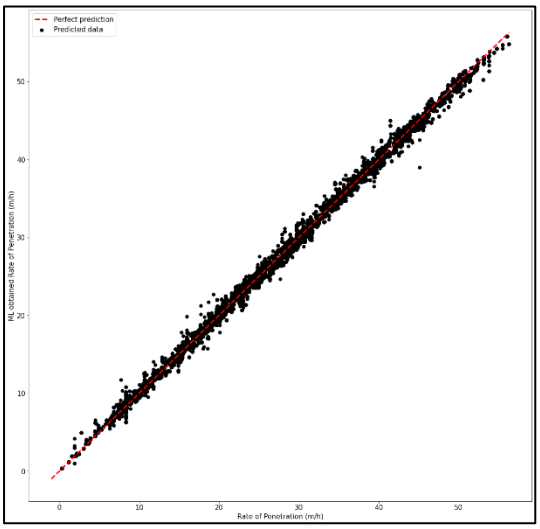

Figure C. 17 Gradient Boosting Regressor Model_Well 3

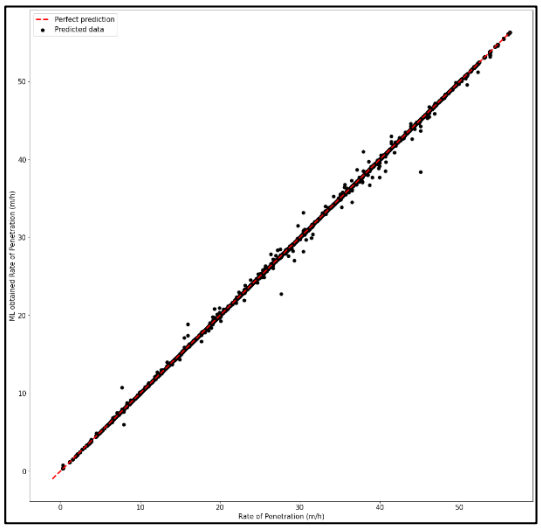

Figure C. 18 Random Forest Regressor Model_Well 3

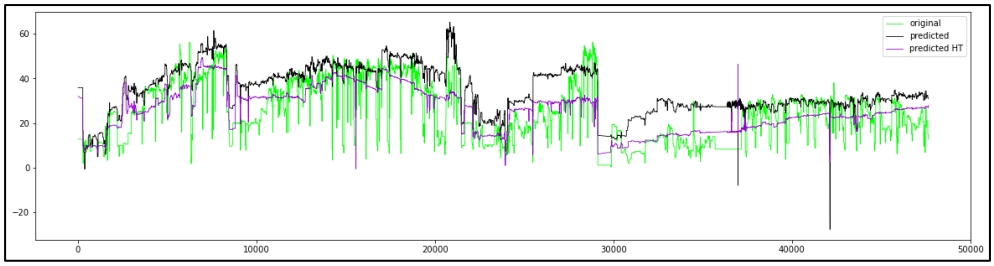

Figure C. 19 Multi-Layer Perceptron Regressor data set prediction_Well 3

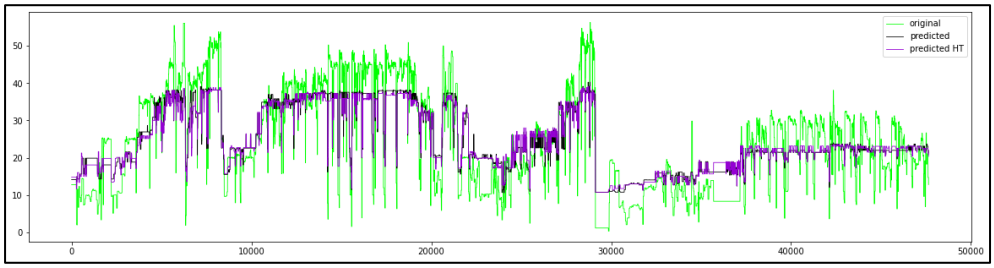

Figure C. 20 AdaBoost Regressor data set prediction_Well 3

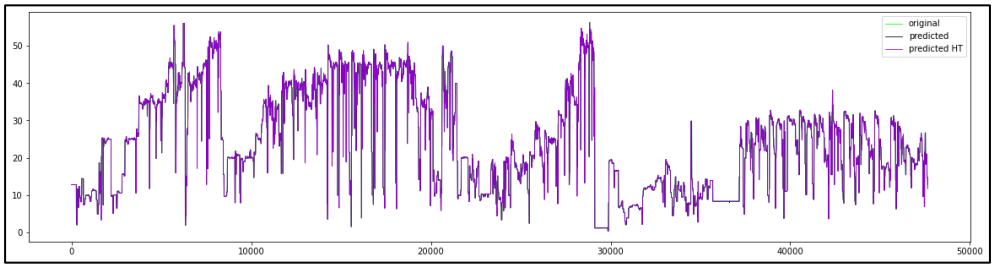

Figure C. 21 K-Neighbors Regressor data set prediction_Well 3

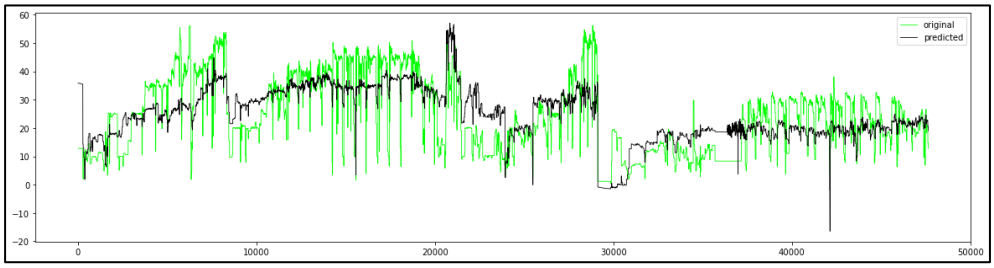

Figure C. 22 Linear Regression data set prediction_Well 3

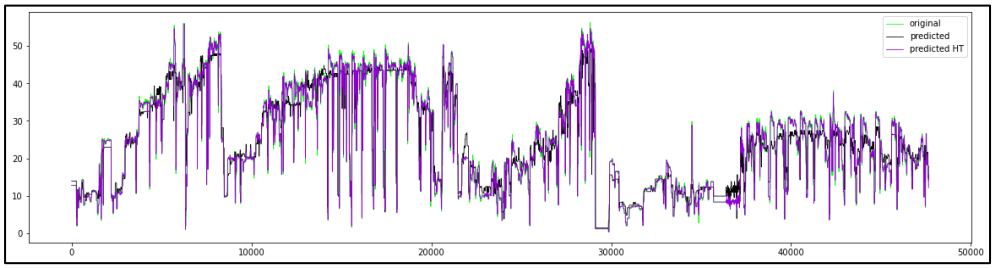

Figure C. 23 Gradient Boosting Regressor data set prediction_Well 3

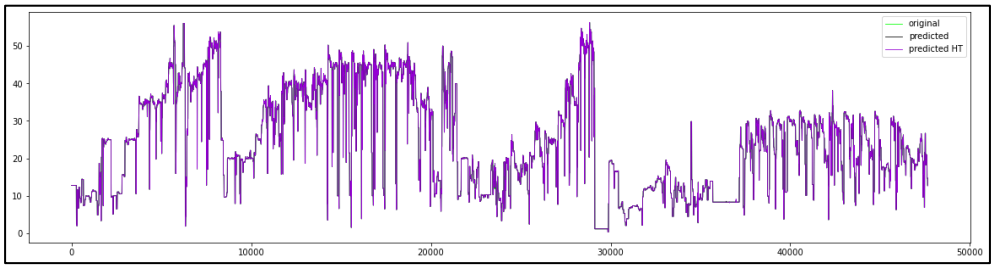

Figure C. 24 Random Forest Regressor data set prediction_Well 3

3. Well 4, USROP_A 3 N-SH-F-15d

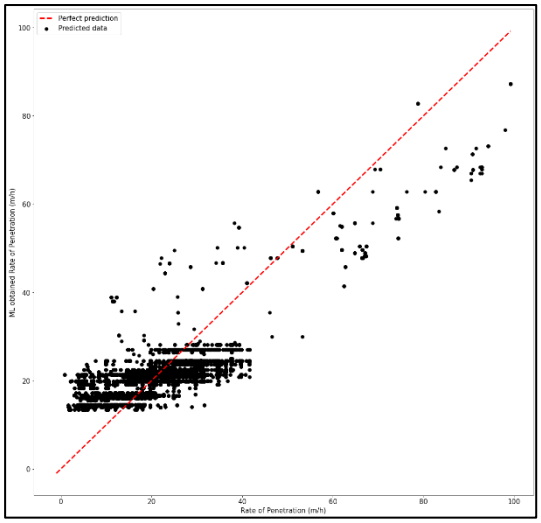

Figure C. 25 AdaBoost Reg Model_Well 4

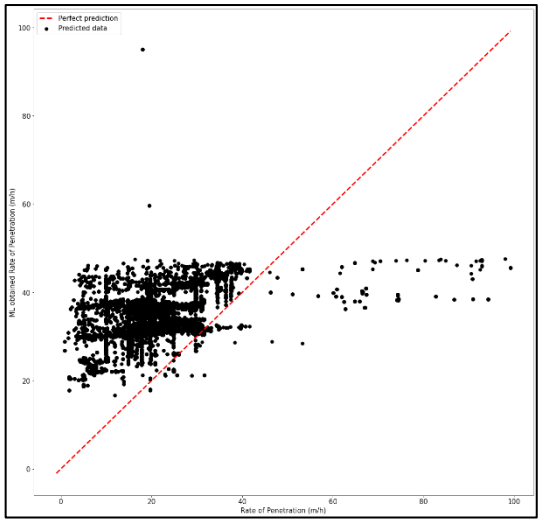

Figure C. 26 MLP Regressor Model_Well 4

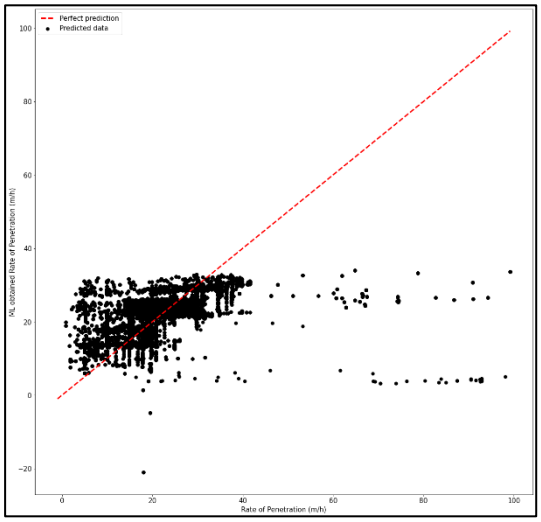

Figure C. 27 Linear Regression Model_15d

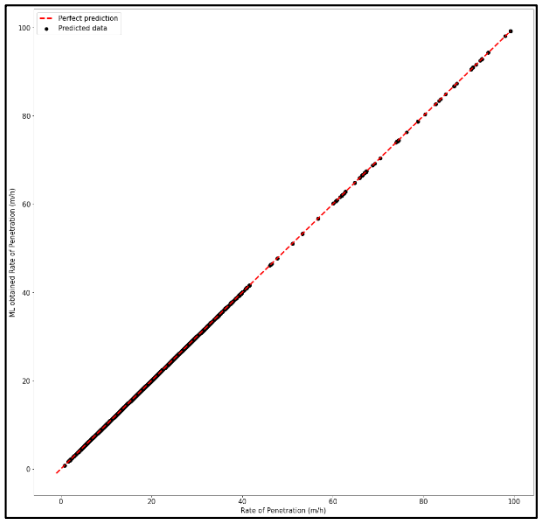

Figure C. 28 K-Neighbors Reg. Model_Well 4

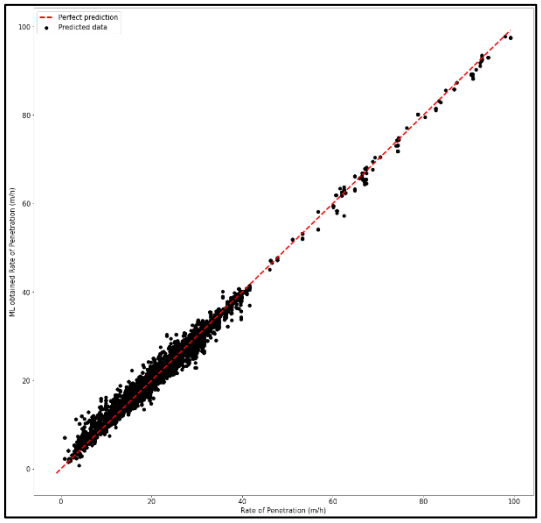

Figure C. 29 Gradient Boosting Regressor Model_Well 4

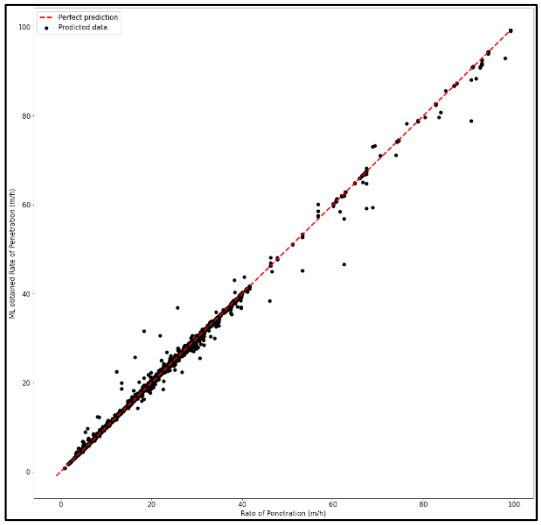

Figure C. 30 Random Forest Regressor Model_Well 4

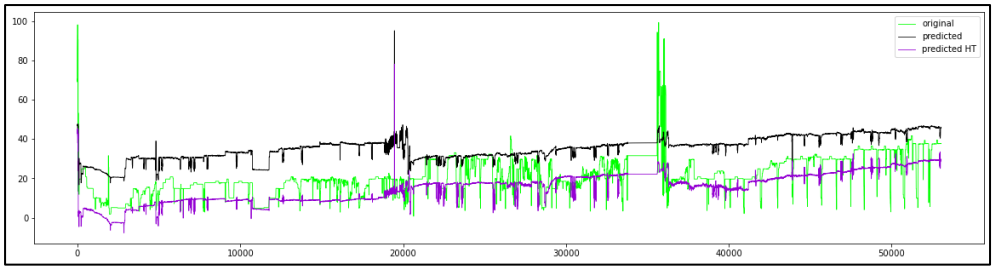

Figure C. 31 Multi-Layer Perceptron Regressor data set prediction_Well 4

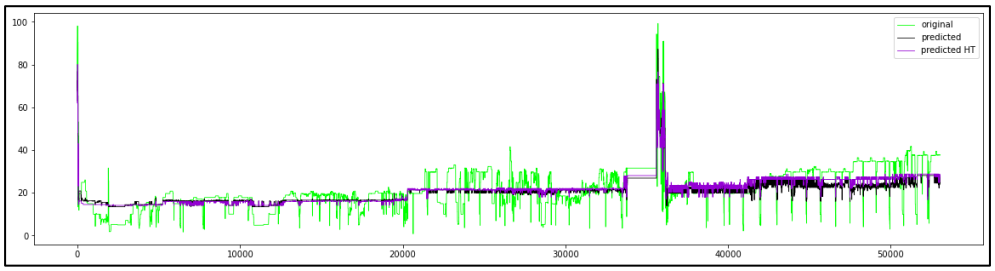

Figure C. 32 AdaBoost Regressor data set prediction_Well 4

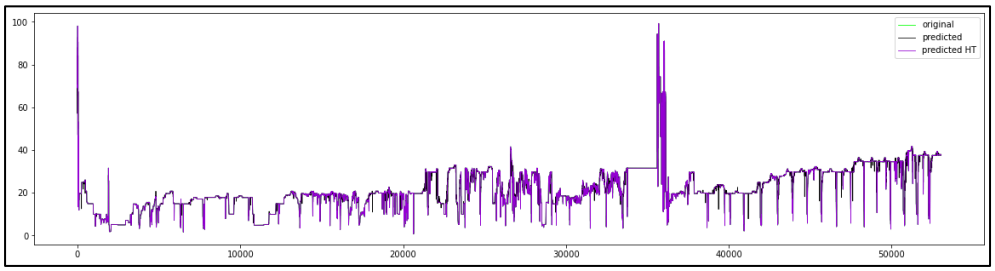

Figure C. 33 K-Neighbors Regressor data set prediction_Well 4

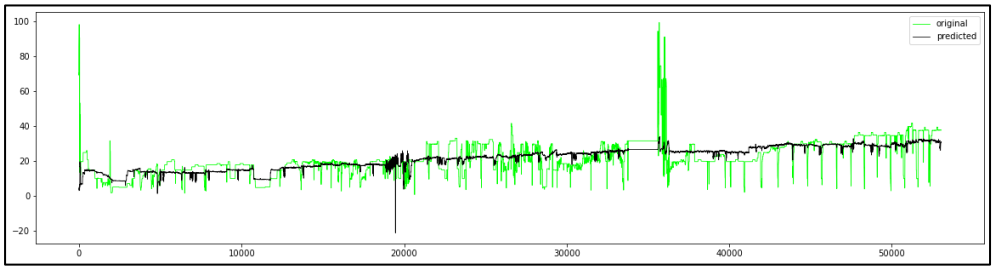

Figure C. 34 Linear Regression data set prediction_Well 4

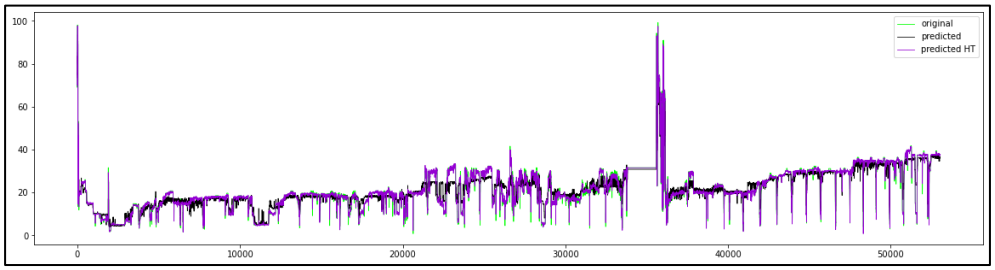

Figure C. 35 Gradient Boosting Regressor data set prediction_Well 4

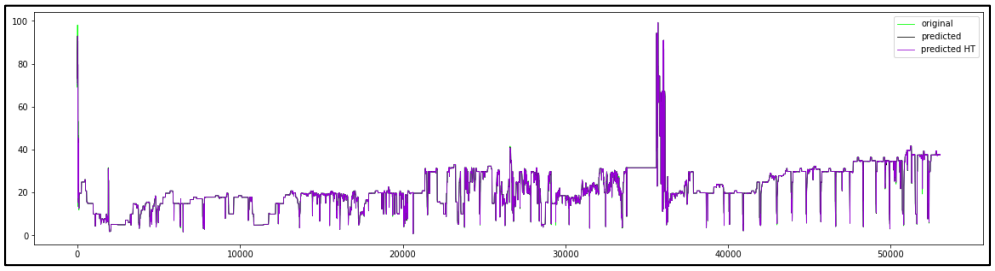

Figure C. 36 Random Forest Regressor data set prediction_Well 4

4. Well 5, USROP_A 4 N-SH_F-15Sd

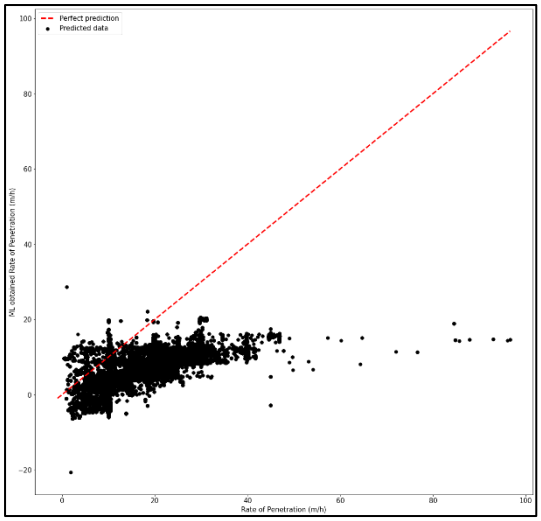

Figure C. 37 MLP Regressor Model_Well 5

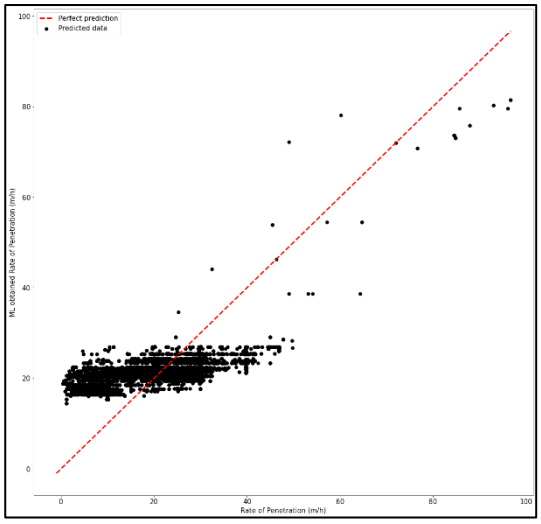

Figure C. 38 AdaBoost Regressor Model_Well 5

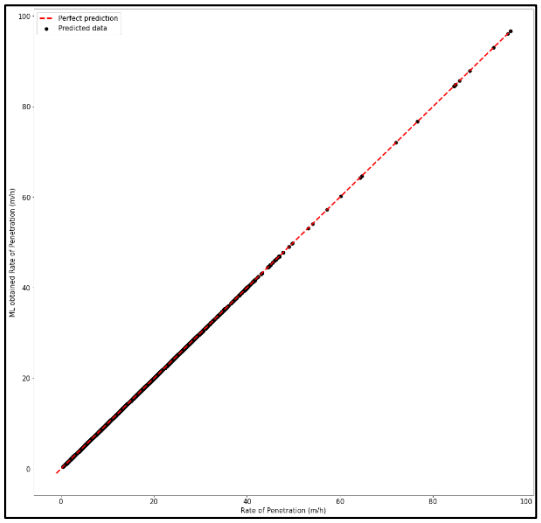

Figure C. 39 K-Neighbors Reg. Model_Well 5

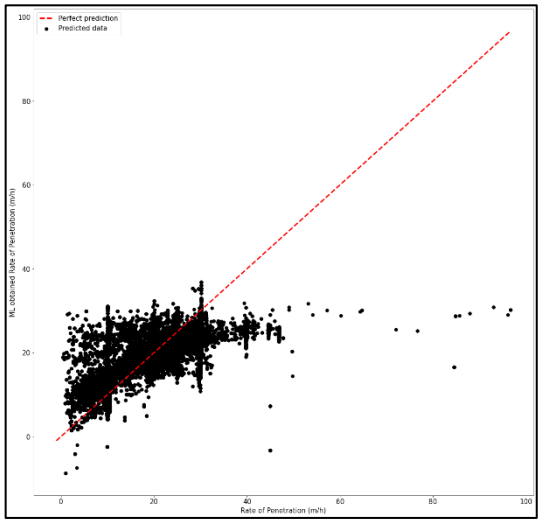

Figure C. 40 Linear Regression Model_Well 5

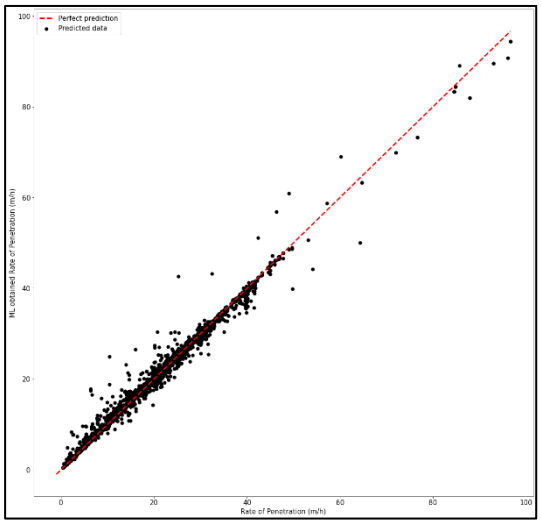

Figure C. 41 Random Forest Regressor Model_Well 5

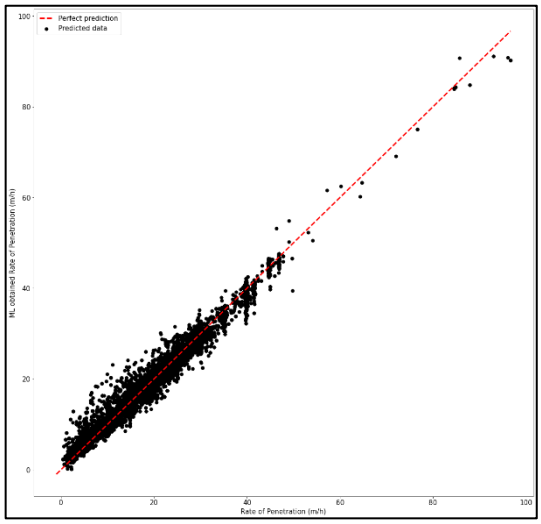

Figure C. 42 Gradient Boosting Regressor Model_Well 5

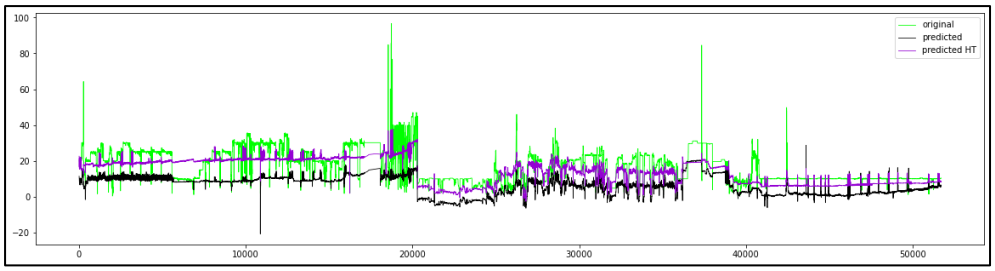

Figure C. 43 Multi-Layer Perceptron Regressor data set prediction_Well 5

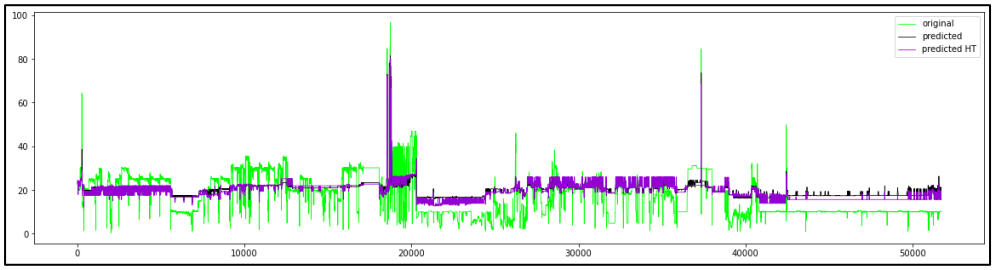

Figure C. 44 AdaBoost Regressor data set prediction_Well 5

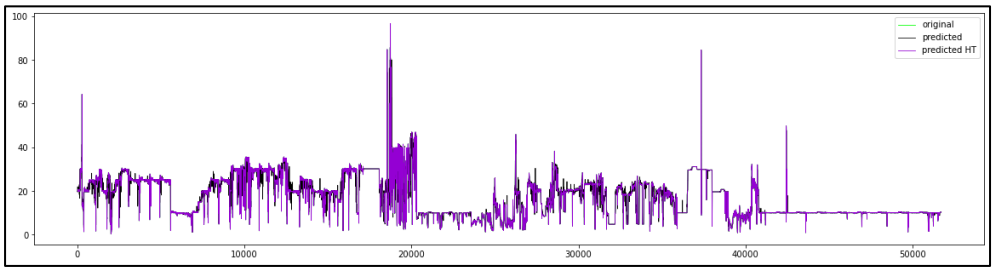

Figure C. 45 K-Neighbors Regressor data set prediction_Well 5

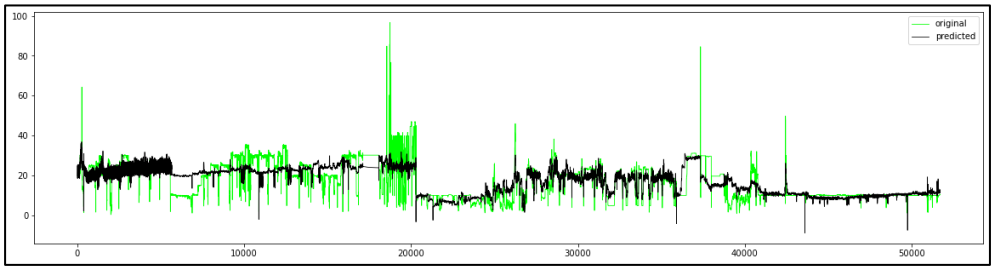

Figure C. 46 Linear Regression data set prediction_Well 5

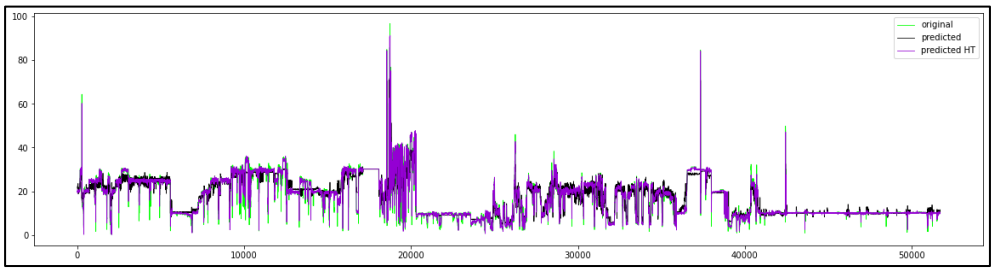

Figure C. 47 Gradient Boosting Regressor data set prediction_Well 5

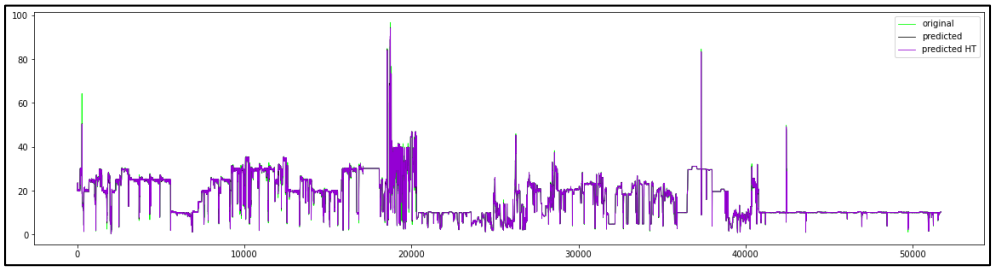

Figure C. 48 Random Forest Regressor data set prediction_Well 5

5. Well 6, USROP_A 5 N-SH-F-5d

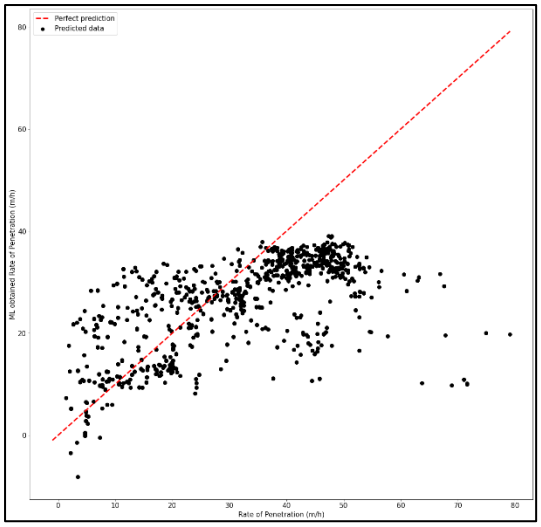

Figure C. 49 MLP Regressor Model_Well 6

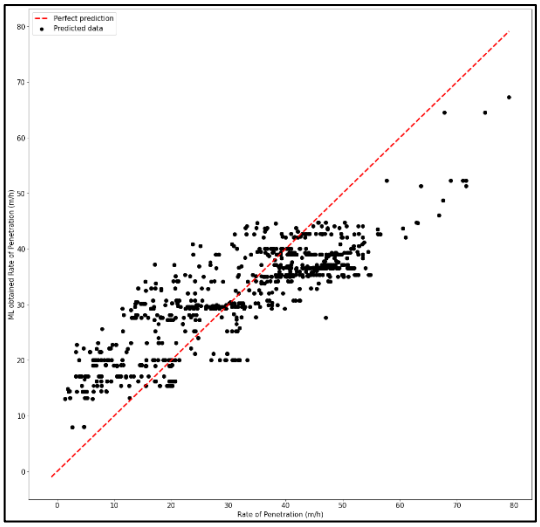

Figure C. 50 AdaBoost Regressor Model_Well *6*

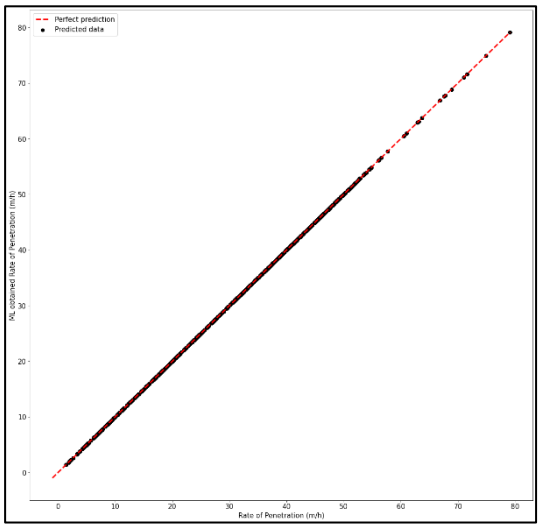

Figure C. 51 K-Neighbors Reg. Model_Well 6

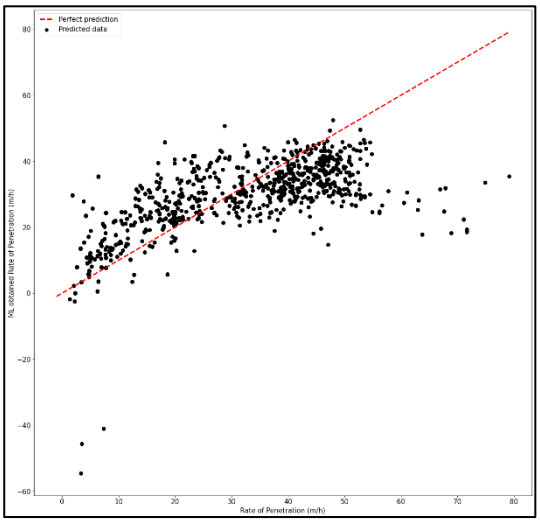

Figure C. 52 Linear Regression Model_Well *6*

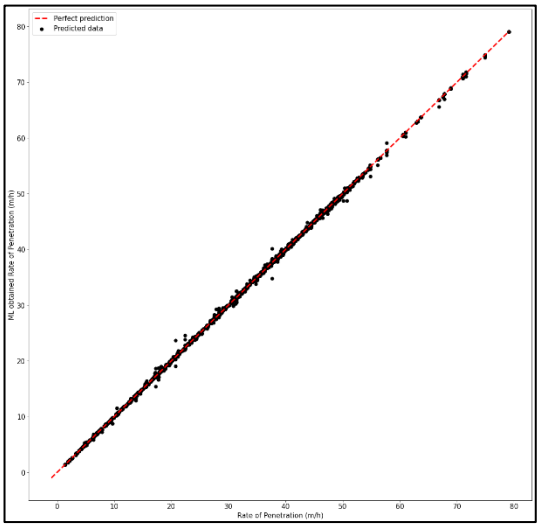

Figure C. 53 Gradient Boosting Regressor Model_Well 6

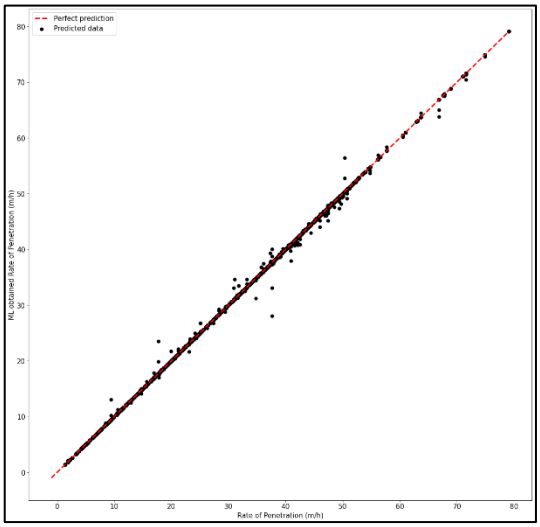

Figure C. 54 Random Forest Regressor Model_Well 6

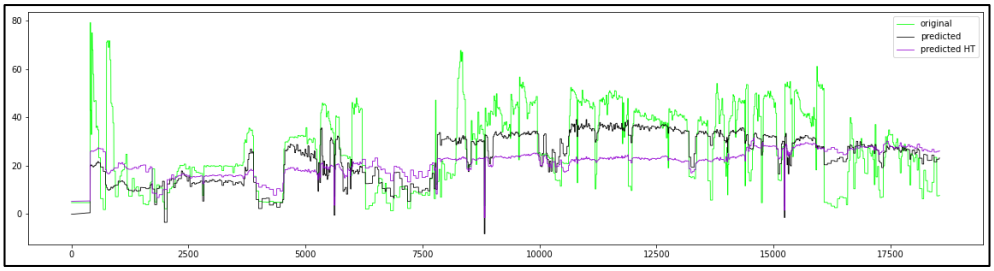

Figure C. 55 Multi-Layer Perceptron Regressor data set prediction_Well 6

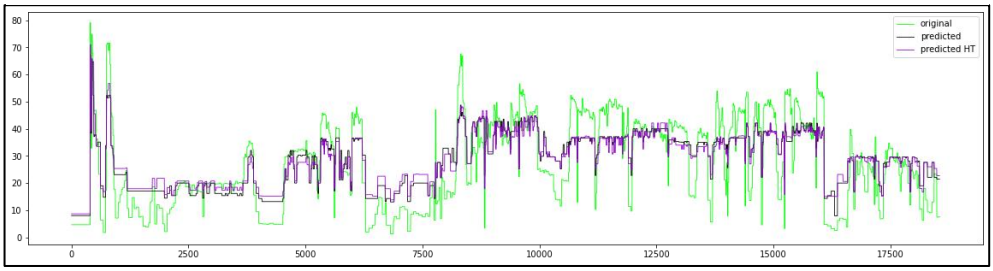

Figure C. 56 AdaBoost Regressor data set prediction_Well 6

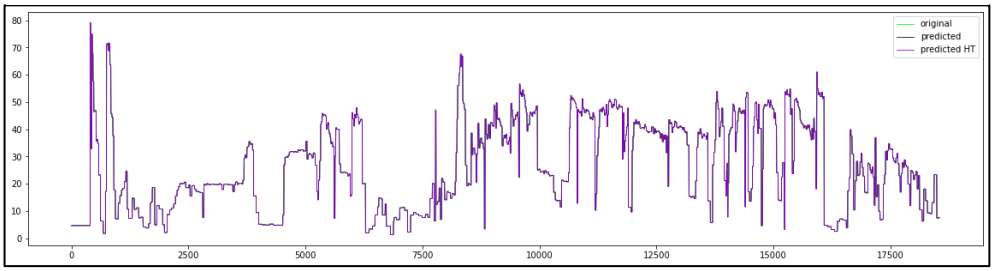

Figure C. 57 K-Neighbors Regressor data set prediction_Well 6

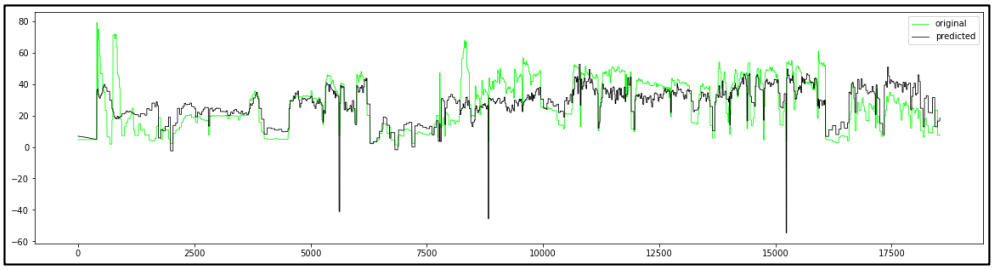

Figure C. 58 Linear Regression data set prediction_Well 6

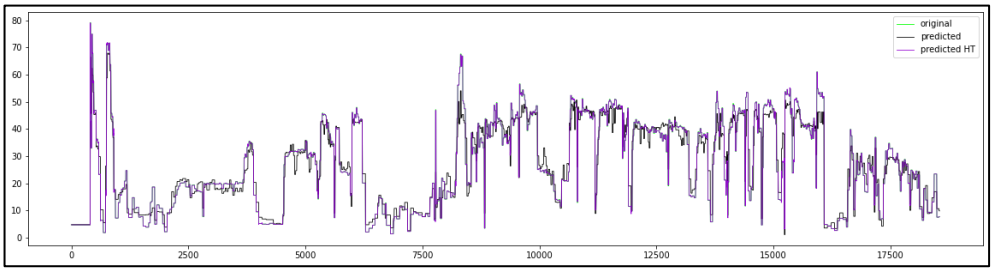

Figure C. 59 Gradient Boosting Regressor data set prediction_Well 6

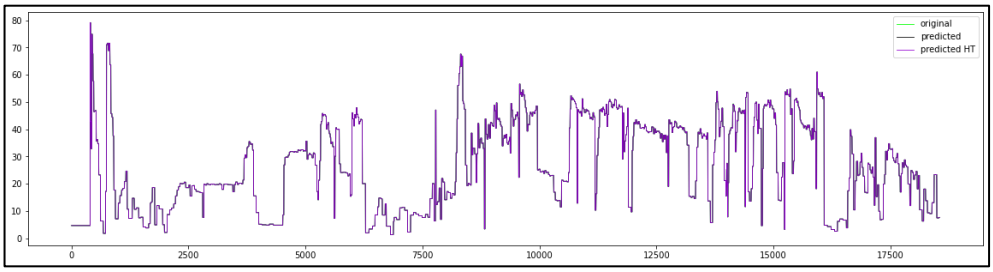

Figure C. 60 Random Forest Regressor data set prediction_Well 6