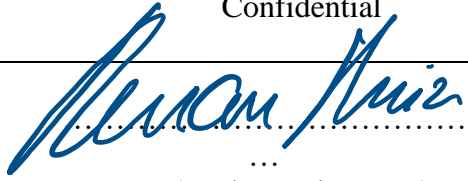




University of
Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

| | |
|--|--|
| Study program/Specialization: MSc. in Petroleum Engineering /Drilling and Wells | Spring semester, 2021 Confidential |
| Author: Renán Gonzalo Ruiz Beviglia |  (Author's signature) |
| Programme coordinator: Øystein Arild Supervisors: UiS - Prof. Dan Sui Equinor – Åsmund Hjulstad | |
| Title of master's thesis: REAL-TIME DATA DRIVEN ROP AND TORQUE MODELLING AND OPTIMIZATION USING MACHINE LEARNING | |
| Credits (ECTS): 30 | |
| Keywords: Machine Learning, Torque, Optimization, Modelling, Rate of Penetration, Torque, Drilling | Number of pages: 66 + supplemental material/other: 37 Stavanger, 27 th June 2020 |

Acknowledgements

To my family, my parents and my brother, for their unconditional support and trust, they were present even in the distance, encouraging and accompanying me in all my projects.

To Dan Sui, for guiding me constantly and patiently in my work side by side in this new challenge, for generously sharing her time and knowledge with me.

To Andrzej Tunkiel, for regularly sharing new ideas and his valuable experience and knowledge with me.

To Åsmund Hjulstad, when I asked him more than once, he always gave me his reflections, suggestions and new concerns to continue improving my work.

To my friends and colleagues who did not allow me to give up the goal, for sustaining my motivation with the right words at the right time.

Abstract

Obtaining the maximum rate of penetration (ROP) is one of many techniques to reduce cost and Non-Productive Time (NPT) in drilling wells. Many parameters affect ROP, including hole cleaning, tooth wear, etc. The study was developed in three parts. First, data was selected, pre-processed and cleaned. In the second part, four machine learning (ML) models (Random Forest (RF), K-Nearest Neighbors (KNN), Gradient Boosting (GB) and AdaBoost (AB)) were implemented to create a ROP model and a Torque model and the section with the best performance was selected. Finally, two optimization algorithms were tested in selected data. In this case, Particle Swarm Optimization (PSO) and Differential Evolution (DE) algorithms were chosen. Once the optimization was performed, a sensitivity analysis was held to check ML methods performance.

In this study, two different parameters (ROP and Torque) were modelled and analysed. Both models use Bit depth, Weight on Bit (WOB), rotary speed (RPM) and pump flow rate (Q) as inputs to make a regression and predict Torque and ROP. In the last part of the study, a new approach is implemented, and modelling is carrying on along with the optimization each 30 meters simulating well drilling with the different optimizers. Hydromechanical Specific Energy (HMSE) was calculated for each 30-meter section and compared with the optimal values in both models. Finally, a sensitivity analysis was performed to evaluate every model and optimizer performance since it was not possible to perform a field experiment. Four ML models were implemented (RF, GB, AB and KNN) among the two algorithms of stochastic optimization and the best combination included GB and DE algorithms after calculating performance metrics and code running time.

Table of Contents

| | |
|---|------|
| Acknowledgements | II |
| Abstract | III |
| Table of Contents | IV |
| List of Abbreviations..... | VII |
| List of Figures | VIII |
| 1. Introduction | 1 |
| 1.1. Objective and motivation..... | 2 |
| 1.2. Methodology | 5 |
| 2. Literature Review | 6 |
| 2.1. ROP Traditional Models | 7 |
| 2.1.1. Bingham Model..... | 7 |
| 2.1.2. Bourgoyne and Young Model..... | 7 |
| 2.1.3. Winters, Warren, and Onyia..... | 9 |
| 2.1.4. Motahhari | 9 |
| 2.2. Data driven Techniques | 10 |
| 2.2.1. Machine Learning Approaches | 10 |
| Ensemble Learning | 10 |
| Gradient Boosting Regressor | 11 |
| Random Forest Regressor | 12 |
| AdaBoost Regressor | 12 |
| K-Nearest Neighbors Regressor | 13 |
| 2.2.2. Review of Data driven ROP Models and Optimization..... | 13 |
| 2.2.3. Issues and Discussions | 14 |
| 2.3. Optimization | 16 |
| 2.3.1. Differential Evolution Algorithm..... | 16 |

| | | |
|--------|---|----|
| 2.3.2. | Particle Swarm Optimization | 18 |
| 2.4. | Drilling Engineering Models | 19 |
| 2.4.1. | Downhole RPM Calculation | 19 |
| 2.4.2. | Downhole Torque Model | 20 |
| 2.4.3. | Downhole WOB Model | 21 |
| 2.4.4. | Density Model..... | 24 |
| 2.4.5. | Mechanical Specific Energy | 24 |
| 2.4.6. | Hydromechanical Specific Energy..... | 25 |
| 3. | Database Analysis | 27 |
| 3.1. | Johan Sverdrup Dataset | 28 |
| | Johan Sverdrup Field..... | 28 |
| 3.2. | Data Import and Visualization..... | 28 |
| 3.3. | Data Cleaning | 30 |
| 3.3.1. | Handling Missing Values..... | 30 |
| 3.3.2. | Handling Faulty Measurements | 31 |
| 3.3.3. | Removing Outliers | 31 |
| | Interquartile Range (IQR)..... | 32 |
| 3.3.4. | Removing Noise..... | 33 |
| | Radius Neighbours Regressor..... | 34 |
| | Median Filter | 35 |
| 3.4. | Data Selection | 35 |
| 4. | Machine Learning Implementation | 37 |
| 4.1. | Data Split | 38 |
| 4.2. | Performance metrics | 39 |
| | Coefficient of Determination (R ²)..... | 40 |
| | Mean Absolute Error (MAE)..... | 40 |
| | Standard Deviation (std) | 40 |

| | | |
|------|---|----|
| 4.3. | Regressors Implementation..... | 40 |
| 5. | Modelling and Optimization Implementation | 43 |
| 5.1. | Optimization Algorithms and constrains | 45 |
| 6. | Results and Discussion..... | 47 |
| 6.1. | Modelling Results | 48 |
| 6.2. | Modelling and Optimization Results | 51 |
| 6.3. | Sensitivity Analysis | 54 |
| 6.4. | Discussion..... | 57 |
| 7. | Conclusions and Future Work..... | 58 |
| 8. | References | 61 |
| 9. | Appendix | 67 |
| | Results Modelling | 68 |
| | Results Optimization PSO..... | 77 |
| | Python Code..... | 79 |

List of Abbreviations

| | |
|-------------|---------------------------------|
| AB | AdaBoost |
| ANN | Artificial Neural Networks |
| DE | Differential Evolution |
| DRPM | Downhole Revolutions per minute |
| DTOR | Downhole Torque |
| DWOB | Downhole Weight on Bit |
| GB | Gradient Boosting |
| HMSE | Hydromechanical Specific Energy |
| IQR | Interquartile Range |
| KNN | K-Nearest Neighbors |
| MAE | Mean Absolute Error |
| ML | Machine Learning |
| MSE | Mechanic Specific Energy |
| MWD | Measure while Drilling |
| NPT | Non-Productive Time |
| PSO | Particle Swarm Optimizer |
| Q | Flow rate |
| RF | Random Forest |
| RNN | Recurrent Neural Network |
| ROP | Rate of Penetration |
| RPM | Revolutions per minute |
| TOR | Surface Torque |
| WOB | Weight on Bit |

List of Figures

| | |
|---|----|
| Figure 1: Historical figures for 2008-2019 and forecast for 2020-2025. Source: NPD [1]. | 4 |
| Figure 2: Function approximation with regression trees. Source: [17]. | 11 |
| Figure 3: Simple tree that model the fertility of the sample based on the different conditions. Source: [19]. | 12 |
| Figure 4: Random train/test split (left) vs. Sequential train/test split (right). Each cell represents a row in the test matrix in depth growing order, where the blue ones are the training sample and white ones makes the test sample. | 15 |
| Figure 5: Common Terms Used in Evolutionary Computation. Source [35] | 17 |
| Figure 6: DE primary mutation operator illustration. A copy of member A is mutated by the addition of the vector difference between B and C. Source [35]. | 18 |
| Figure 7: Baker Hughes 6 ¾ in. Ultra XL/LS motor data and specifications, power performance section. Source [38]. | 19 |
| Figure 8: Free body pipe of a drill pipe unit of ds length (left). Source: [39]. Force acting in a curved drill string. Source: Johansick [40] | 20 |
| Figure 9: Drillstring element under influence of torsional and axial forces. Source: [30] | 22 |
| Figure 10: Plot of the different variables vs. time for well 3. | 30 |
| Figure 11: Variables vs. depth of well number 6 after removing outliers manually and using IPR method in well 1. | 33 |
| Figure 12: Variables vs. depth of well number 6 after removing noise using median average filter in well 1 | 34 |
| Figure 13: Variables vs. depth of well number 6 after removing noise using median average filter and Radius Neighbors Regressor in well 1 section 26". | 35 |
| Figure 14: Sequential split (left) and continuous learning split (right). Blue cells represent the training set and white cells represent the test set. | 39 |
| Figure 15: Flow-chart of the continuous learning approach in ROP and Torque modelling. | 41 |
| Figure 16: Flow-chart of the sequential split approach in ROP and Torque modelling. | 41 |
| Figure 17: Modelling and Optimization flow-chart | 44 |
| Figure 18: ROP models' MAEs using continuous learning approach | 48 |

| | |
|---|----|
| Figure 19: Torque models' MAEs using continuous learning approach from one of the well sections | 49 |
| Figure 20: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well1 section 26". Predicted data (blue) is evaluated against test data (green)..... | 50 |
| Figure 21: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well1 section 26". Predicted data (blue) is evaluated against test data (green)..... | 50 |
| Figure 22: Code Runtime (seconds) for the two optimizers (Differential Evolution in blue and Particle Swarm in red) and using RF, GB, AB, and KN with the same parameters..... | 51 |
| Figure 23: ROP (blue), HMSE (red) and Torque (green) growth rate using GB Torque and ROP model and DE combination. Growth rate is negative because the three parameters decrease..... | 52 |
| Figure 24: ROP (blue), HMSE (red) and Torque (green) growth rate using AB ROP and Torque ROP model and DE combination..... | 53 |
| Figure 25: ROP (blue), HMSE (red), and Torque (green) average growth for all ML models for ROP optimization (maximize ROP) using DE algorithm..... | 53 |
| Figure 26: ROP (blue), HMSE (red), and Torque (green) average growth for all ML models for Torque optimization (minimize Torque) using DE algorithm..... | 53 |
| Figure 27: Sensitivity analysis case 1 for Torque models. The x-axis represents the model whose optimal parameters are implemented..... | 54 |
| Figure 28: Sensitivity analysis case 1 for ROP models. The x-axis represents the model whose optimal parameters are implemented..... | 55 |
| Figure 29: Sensitivity analysis case 2 for Torque models..... | 56 |
| Figure 30: Sensitivity analysis case 2 for ROP models..... | 56 |
| Figure 31: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 1 section 16". Predicted data (blue) is evaluated against test data (green)..... | 68 |
| Figure 32: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 1 section 16". Predicted data (blue) is evaluated against test data (green)..... | 69 |
| Figure 33: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 2 section 26". Predicted data (blue) is evaluated against test data (green)..... | 70 |
| Figure 34: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 2 section 26". Predicted data (blue) is evaluated against test data (green)..... | 70 |

| | |
|--|----|
| Figure 35: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 3 section 26". Predicted data (blue) is evaluated against test data (green)..... | 71 |
| Figure 36: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 3 section 26". Predicted data (blue) is evaluated against test data (green)..... | 72 |
| Figure 37: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 4 section 26". Predicted data (blue) is evaluated against test data (green)..... | 73 |
| Figure 38: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 4 section 26". Predicted data (blue) is evaluated against test data (green)..... | 73 |
| Figure 39: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 5 section 26". Predicted data (blue) is evaluated against test data (green)..... | 74 |
| Figure 40: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 5 section 26". Predicted data (blue) is evaluated against test data (green)..... | 75 |
| Figure 41: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 6 section 26". Predicted data (blue) is evaluated against test data (green)..... | 76 |
| Figure 42: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 6 section 26". Predicted data (blue) is evaluated against test data (green)..... | 76 |
| Figure 43: ROP (blue), HMSE (red) and Torque (green) growth rate using GB Torque model and PSO combination. Growth rate is negative because the three parameters decrease..... | 77 |
| Figure 44: ROP (blue), HMSE (red) and Torque (green) growth rate using GB ROP model and PSO combination. | 77 |
| Figure 45: ROP (blue), HMSE (red) and Torque (green) average growth rate for all ML models for ROP optimization (maximize ROP) using PSO algorithm. | 78 |
| Figure 46: ROP (blue), HMSE (red) and Torque (green) average growth rate for all ML models for ROP optimization (maximize ROP) using PSO algorithm. | 78 |

List of Tables

| | |
|--|----|
| Table 1: Initial features from a representative well | 28 |
| Table 2: Initial features from a representative well after merging and interpolating. | 31 |
| Table 3: input features for ROP and Torque models. | 38 |
| Table 4 and Table 5: Performance metrics for Machine Learning models using 60/40 split (Torque model on the left and ROP model on the right)..... | 49 |
| Table 6: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for Torque models from well 1 section 16”..... | 68 |
| Table 7: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for ROP models from well 1 section 16”..... | 68 |
| Table 8: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for Torque models from well 2 section 26”..... | 69 |
| Table 9: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for ROP models from well 2 section 26”..... | 69 |
| Table 10: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for Torque models from well 3 section 26”..... | 71 |
| Table 11: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for ROP models from well 3 section 26”..... | 71 |
| Table 12: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for Torque models from well 4 section 26”..... | 72 |
| Table 13: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for ROP models from well 4 section 26”..... | 72 |
| Table 14: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for Torque models from well 5 section 26”..... | 74 |
| Table 15: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for ROP models from well 5 section 26”..... | 74 |
| Table 16: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for Torque models from well 6 section 26”..... | 75 |
| Table 17: Mean Absolute Error (MAE), coefficient of determination (R ²) and standard deviation (std) for ROP models from well 6 section 26”..... | 75 |

1. Introduction

1.1. Objective and motivation

Drilling is a crucial and expensive process in oil and gas industry. Drilling wells can be classified in two categories depending the objective: exploration wells and development (or production) wells. While the goal of the first ones is to find hydrocarbons and gather data for development, the seconds goal is to produce. Pressure is typically uncertain during drilling exploration wells and therefore mud weight and casing design practices are conservative and the ROP is low, making impossible to optimize it.

According to NPD, overall costs during oil and gas operation can be divided in different categories: Investments, Operating Costs, Exploration costs, Disposal and cessation and other costs [1]. Then, development wells are included in investments and exploration wells in Exploration costs. Development wells made up 27% of the overall costs last year, which prices could round from NOK 200 million to NOK 700 million per unit in case of mobile rigs [2], which more production wells are drilled by. Moreover, the dominant cost elements are the oil services and rig rent, which account 30% and 45% of the drilling expenses respectively. Therefore, it is necessary to optimize drilling to save drilling costs and reduce Non-Productive Time (NPT).

According to Soares [3], drilling optimization is the “process of designing equipment and selecting operational parameters to minimize well drilling cost”. Thus, the rate of penetration (ROP), defined as the volume of rock removed expressed as depth per time unit, is a key metric to measure drilling performance. Although high ROP is considered a good metric to measure drilling efficiency and performance, drilling faster can affect cutting transport and lead to bore hole instability and poor hole cleaning [4]. ROP varies from the type is rock is being drilled and can give an idea of bit wear. Therefore, companies look for high ROP values while operating in recommend safety standards.

Generally, ROP optimization depends on dynamic and static drilling parameters. Dynamic parameters can be controllable (weight on bit [WOB], rotary speed [RPM], flow rate [Q]) or uncontrollable whether the driller can alter it manually during operations or not. Static parameters include formation properties such as compressive strength and formation pressure among other things.

In the industry exists two approaches to predict the ROP in a specific field: physics-based and data driven. Physics based models are formulas, mathematical functions

obtained during lab experiments. On the other hand, data driven approach use machine learning to create a model or formula that predicts ROP. Traditional models are deterministic and easy for optimization while they present low accuracy in ROP prediction, have empirical coefficients based on lithology that varied continuously and require static parameters as inputs that is not always available [5]. According to Hedge [5], data driven models gives accurate ROP prediction than physic based, since it does not contain any empirical constant, bit specifications and is not linked to a specific BHA. However, these models are purely dependent on data and therefore designed for a specific field. Currently, there is no model that predicts and optimize ROP accurately in all fields due to variation of the different drilling and geological parameters.

Hydromechanical Specific Energy (HMSE) is a new term introduced by Mohan [6] that “measures the energy to drill a unit volume of rock and remove it from underneath the bit”. It includes mechanical as well as hydraulic energy that will be explained with more detail later In this study, As MSE, it is a relevant parameter to describe drilling efficiency. Torque prediction can help to improve the HMSE as well as control vibration as it was performed in a recent study by Hedge [7]. Although for HMSE and MSE calculations Downhole Torque from MWD tools, it is possible to estimate TOB (Torque on Bit) with surface torque using a transformation matrix [8]. In his study Hedge showed that a reduce on Torque on Bit could lead to an ROP improvement and this test will be performed using surface Torque data.

The motivation of this work is to implement a code that can automatically improve drilling efficiency each 30 meters (one drilling stand) either by maximizing ROP or minimizing surface Torque. This code will provide the dynamic controllable parameters to achieve the desirable ROP (or Torque) and therefore save drilling costs. With Torque minimization the aim is to minimize the HMSE since these variables are directly related.

This thesis project is focused on drilling optimization and the goal is therefore maximize the Rate of Penetration and minimize Surface Torque. To achieve a good optimization, it is very important to have an accurate model to predict the metric is being optimized. This work is a first theoretical approach to optimize the parameters mentioned above in the same rock formation. Further considerations will need to do to perform an experimental test and will be mentioned in the Chapter 8.

The goal of this study is to optimize the ROP and surface Torque in real time when varying surface controllable drilling parameters of a specific well section. Six wells from Johan Sverdrup field drilled last year were provided by Equinor and only one well section was chosen. The project can be summarized in the following tasks:

- Clean and process all dataset.
- Understand Machine Learning Models and how to Implement algorithms in the dataset
- Implement Machine Learning Algorithms in the dataset
- Evaluate Machine Learning Models Performance in different well sections
- Select well section with accurate ROP and Torque Prediction
- Implement new continuous learning modelling and optimization method
- Evaluate ROP and Torque optimization including HMSE calculations
- Sensitivity analysis of the different Machine Learning Models

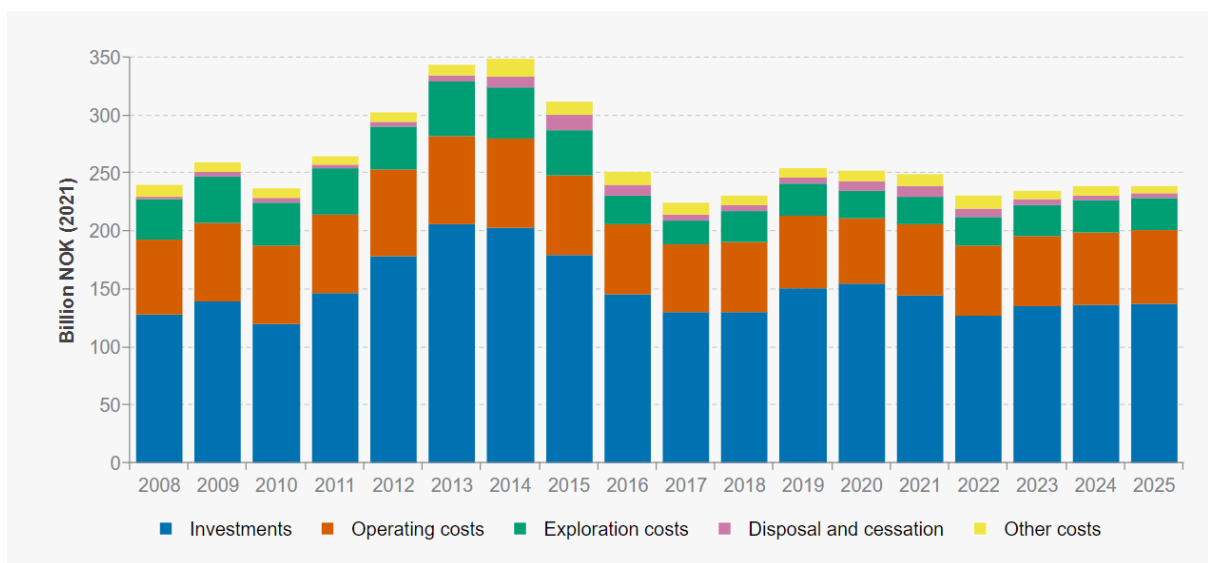


Figure 1: Historical figures for 2008-2019 and forecast for 2020-2025. Source: NPD [1].

1.2. Methodology

For the purpose of coding, Jupyter Notebook was used as it is a user-friendly application that handles and run Python code while describes data analysis. In this study, Python was used as a coding language due to its simplicity and versatility, making it very attractive for beginners. It is also free, open source and support huge libraries which can be imported by Python package manager (pip). All the libraries used in this study are explained in the Appendix.

It is very relevant to have an appropriate and cleaned data set, having enough observation variables to run a Machine Learning Model. This is in fact the first and the most time-consuming part of the study, which consist on data selection, fill data gaps with interpolation, remove outliers and noises and correct faulty measurements. Moreover, a machine learning algorithm was used to evenly distribute data points in depth to easily implement a Regression using ML and therefore remove more noise that was impossible using filter. Anyway, all this process will be explained deeply in next chapters.

Second part of this study consists in the implementation of different ML techniques (Random Forest, Gradient Boosting, AdaBoost and k-Nearest Neighbors) and assess which algorithm provides the best ROP and Torque prediction. In this project, two parameters (ROP and Torque) were modelled using Machine Learning technique. All the models were evaluated using different evaluation metrics which will be explained later in Chapter 4. The result will be eight predictive models which will finally be used in the last part of this study.

The last part of this study is modelling and optimization using the same ML models and two different stochastic optimization techniques: Differential Evolution (DE) and Particle Swarm Optimization (PSO). Since one of the goals of this project is to reduce drilling cost, it is very important to choose the best optimization algorithms and constrains in order to improve ROP, reduce Torque and HMSE and therefore save costs and mitigate drilling problems. Different physical boundaries were considered as constrains for the optimization algorithm based on the variables of the ML model previously chosen. Finally, a sensitivity analysis was executed for the different ROP and Torque models.

2. Literature Review

2.1. ROP Traditional Models

Many traditional models have been used for ROP modelling successfully. The goal of this section is to introduce the most popular ROP physics models in the industry. These models have been developed experimentally by regression methods and based on drilling knowledge and only the most widely used will be presented. These are outlined below:

2.1.1. Bingham Model

Bingham [9] is the oldest traditional ROP model. It is designed for any bit type and considers ROP as a function of WOB, RPM and bit diameter. Although it is a straightforward model, it is still a good starting point for ROP prediction:

$$ROP = k \left(\frac{WOB}{d_b} \right)^a RPM \quad (1)$$

where ROP is the rate of penetration (ft/hr), WOB is the weight on bit (klb), RPM is the rotary speed (revolutions/min), d_b is the bit diameter (in), and 'a' and 'k' are rock formation constants obtained by linear regression.

2.1.2. Bourgoyne and Young Model

Bourgoyne and Young ROP model [10] was developed in 1974 after multiple regression analysis of drilling data obtained in short intervals. In those years, there was one model for ROP optimization, one for jet bit hydraulics optimization and one for detecting abnormal pressure from field data. This model combines these three processes into one single model, including effects of formation strength, formation depth, formation compaction, differential pressure, bit diameter and weight, bit wear, rotary speed and bit hydraulics and it is expressed as a function of eight components:

$$ROP = f_1 * f_2 * f_3 * f_4 * f_5 * f_6 * f_7 * f_8 \quad (2)$$

$$f_1 = e^{a_1} \quad (3)$$

$$f_2 = e^{a_2(13000-TVD)} \quad (4)$$

$$f_3 = e^{a_3 TVD^{0.69} (P_{pore} - 10.5)} \quad (5)$$

$$f_4 = e^{a_4 TVD (P_{pore} - ECD)} \quad (6)$$

$$f_5 = \left(\frac{\left(\frac{W}{d_b} \right) - \left(\frac{W}{d_b} \right)_t}{4 - \left(\frac{W}{d_b} \right)_t} \right)^{a_5} \quad (7)$$

$$f_6 = \left(\frac{RPM}{100}\right)^{a_6} \quad (8)$$

$$f_7 = e^{-a_7 h} \quad (9)$$

$$f_8 = e^{a_8 \frac{\rho q}{350 \mu d_n}} \quad (10)$$

Where, f_i includes drilling parameters and a_i are the variable coefficients calculated using linear regression. Coefficients are described in the following way:

- f_1 represents the formation strength influence, where a_1 is the formation strength parameter.
- f_2 represents the formation depth influence, where TVD is true vertical depth (ft) and a_2 the exponent of normal compaction trend.
- f_3 represents pore pressure influence, where P_{pore} is the pore pressure gradient (ppg) and a_3 the undercompaction exponent.
- f_4 represents the differential pressure effect, where ECD is the equivalent circulating density (ppg) and a_4 the pressure differential exponent.
- f_5 represents the variation of WOB and bit diameter and changes for bit type, where $\frac{w}{d_b}$ is the applied WOB per inch (1000 lb/in), d_b the bit diameter, $\left(\frac{w}{d_b}\right)_t$ the threshold WOB per inch (1000 lb/in) at which the bit begins to drill and a_5 the bit weight exponent.
- f_6 represents the influence of the RPM. In this case the author normalizes to 1 a rotary speed of 100 rpm, but this number can change in terms of the average rotary speed of the dataset. a_6 is the rotary speed exponent.
- f_7 represents the drill bit wear, where h is the fractional tooth weight that has been worn away and a_7 the tooth wear exponent.
- f_8 represents the hydraulic effects, where d_n is the bit nozzle diameter (in), μ the apparent viscosity (cp) of the drilling fluid at 10000 sec^{-1} and a_8 the hydraulic exponent.

2.1.3. Winters, Warren, and Onyia

Winters Warren and Onyia [11] developed a new model for roller cone bit, taking into account bit design, operating conditions, and rock mechanics. It includes rock ductility and cone offset as new and important features for ROP modelling:

$$\frac{1}{ROP} = \frac{\sigma D^2}{N WOB} \left(\frac{a \sigma D \epsilon}{WOB} + \frac{\Phi}{WOB} \right) + \frac{b}{N D} + \frac{c \rho \mu \epsilon}{I_m} \quad (11)$$

Where σ is the rock compressive strength (psi), D the bit diameter (in), N the rotary speed (rpm), WOB the weight on bit (lb), Φ the cone offset (in), ϵ the rock ductility, ρ the mud density (ppg), μ the mud viscosity (cp), I_m the modified jet impact force and a, b and c the bit design constants. The modified jet impact force is calculated as follows:

$$I_m = (1 - A_v^{-0.122}) F_j \quad (12)$$

Where A_v is the ratio of jet velocity to return velocity, F_j the jet impact force, and A_v is calculated as follows assuming three jets:

$$A_v = \frac{v_n}{v_f} = \frac{0.15 D^2}{3 d_n^2} \quad (13)$$

Where V_n is the nozzle velocity and V_f the return fluid velocity.

2.1.4. Motahhari

Motahhari [12] developed in 2010 a new method for ROP prediction for polycrystalline diamond compacts (PDC) bits and positive displacement motors. According to Barros and Motahhari [12] [13], this model is very useful for horizontal and directional drilling operations with motors. The model equation is the following:

$$ROP = W_f \frac{G WOB^\alpha N_t^\gamma}{d_b CCS} \quad (14)$$

Where d_b is the bit diameter (in), CCS the confined rock strength (psi), N_t the total rotary speed (rpm), WOB the weigh on bit (lb), G the coefficient determined by the bit geometry, cutter size and design, W_f the bit wear function and α and γ the ROP model exponents. Bit wear function is calculated as follows:

$$W_f = k_{wf} \left(\frac{WOB}{n_c} \right)^\rho \frac{1}{CCS^\alpha A_w^{\rho-1}} \quad (15)$$

Where k_{wf} is the wear function constant, ρ and τ the wear function exponents, n_c the number of cutters and A_w is the wear flat area underneath of a single cutter, which

according to Motahhari [12]“is a function of wear depth on a cutter face and PDC layer thickness”.

2.2. Data driven Techniques

It is commonly said that ML is a subfield of artificial intelligence. Machine Learning (ML) means create mathematical models to understand data [14]. The program “learns” from the data when “giving to the models tunable parameters that can be adaptable to observe the data” [14]. Basically, it starts separating your dataset into training data and test data. The training data is used to generate the mathematical model while the test data is compared with the predicted data once the model fits with training data. Python has a powerful tool called Scikit learn that contains multiple machine learning models to implement in the code.

There are two categories of Machine Learning, Supervised Learning and Unsupervised Learning, but the difference is not relevant for this study. It is important to mention that this study works with Machine Learning Regressors, one subdivision of Supervised Learning methods, since parameters are continuous quantities.

2.2.1. Machine Learning Approaches

Machine Learning has a lot of advantages if used properly for ROP or Surface Torque prediction. For instance, all the shortcomings from the traditional models can be solved since they do not contain empirical constants and they are independent of the bit type and bottom hole assembly (BHA). In a study developed by Hegde [5], three traditional models were compared with data driven models showing better performance, improving the R^2 value from 0.12 to 0.84. Predictions of machine learning algorithms are purely based on input data and parameters; therefore, data quality is essential to implement these models. Four machine algorithms were implemented and each one has its advantages and disadvantages. First, a few concepts will be explained:

Ensemble Learning

According to Zhou [15], “Ensemble learning is a machine learning paradigm where multiple learners are trained to solve the same problem”. Usually, ordinary machine learning approaches learn one hypothesis from training data, while these methods learn from a set of hypotheses and combine them. An ensemble is constructed from base

learners, created from training data by a base learning algorithm (a decision tree, a neural network, etc). After this process, these learners are combined, and the most popular combination is selected.

Gradient Boosting Regressor

Gradient Boosting is part of Boosting algorithms family and is based on decision trees where the objective is to minimise the loss function of the model. It is a generic algorithm to find approximate solutions to the additive modelling problem and thus, it is more flexible than AdaBoost. Friedman firstly introduced it in 2001. As any boosting method, Gradient Boosting (GB) adds new models to the ensemble sequentially [16], and at each iteration a new base learner (in this case, a decision tree) is trained regarding the error of the whole ensemble.

GB consists of three elements: a loss function, a weak learner and an additive model. It uses decision trees as weak learners due to its ability to handle mixed data types and model complex functions [16]. Figure 2 illustrates the process of approximation of Regression trees.

The advantages include high accuracy, high flexibility, no need for data pre-processing, and good manage of missing data. However, Gradient Boosting algorithms are very sensitive to small data changes, so it is mandatory to use cross-validation.

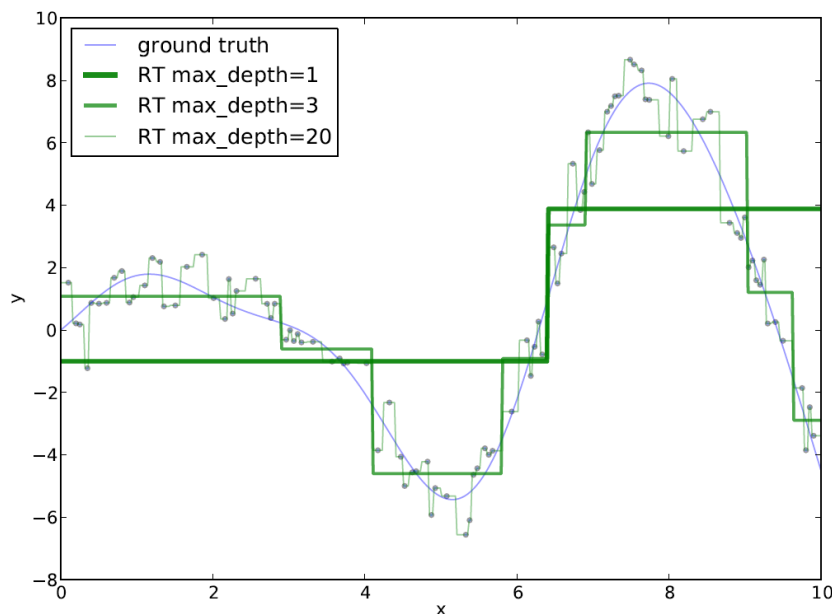


Figure 2: Function approximation with regression trees. Source: [17]

Random Forest Regressor

Random forests are ensemble learners that use a combination of decision trees that “grow in accordance to a random parameter”, according to Biau [18]. It was proposed by Leo Breiman in the 2000’s where each tree is built randomly, therefore the name “random forest”. Decision trees are, as VanderPlas says [14], “intuitive ways to classify objects”: each tree is subdivided in nodes and these nodes, based on a cut-off value of one of the features, splits the data into two groups [14]. Random Forests are one of the most accurate ensemble learning techniques. Random Forests can be used in Classification and Regression problems depends if we have categorical or continuous variables.

Advantages include excellent prediction, no need for data preparation, fast training, good handling of missing data and finally, it works with large datasets. On the other hand, it has its limitations with regressions and there is a risk of overfitting the model. Fig. 3 is an example of a tree taken from Yoo’s study [19]. This is an individual tree with two nodes that models Fertility based on Education, Agriculture and Examination of the specimen.

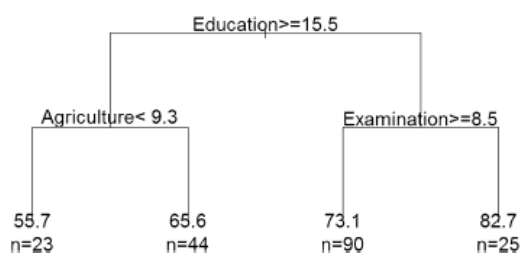


Figure 3: Simple tree that model the fertility of the sample based on the different conditions. Source: [19]

AdaBoost Regressor

AdaBoost (Adaptive Boosting) is a machine learning algorithm formulated by Yoav Freund and Robert Schapire in 1997. According to Zhou [20]: “boosting refers to a family of algorithms that are able to convert weak learners to strong learners”. As a boosting algorithm, it works with Decision Stumps (trees in a Random Forest, but not “fully grown”) using a forest of that decision stumps. Each decision stump has one node and two leaves. AdaBoost Algorithm has many advantages: is fast, simple, easy to program and flexible to combine with other machine learning algorithms. On the other hand, it is also very sensitive to noises and outliers.

K-Nearest Neighbors Regressor

According to Zhou [20]: KNN assumes that similar objects in input space are similar in the output space; thus similar data are near each other. This author also considers as a lazy learning approach due to not having an explicit training process. It has many advantages as its simplicity and versatility. However, it is susceptible to small changes in data and cannot handle missing values. Moreover, it does not work with datasets with many features.

2.2.2. Review of Data driven ROP Models and Optimization

As is mentioned by Alali [4], there is no reliable model that accurately predicts ROP. Many studies have been conducted to understand ROP behaviour and predict it based on data availability. For instance, Hedge [5] compared the performance of different ML algorithms with traditional models, showing an improvement on the R^2 from 0.12 to 0.84. Furthermore, Ahmed [21] developed a novel ROP model using artificial neural networks (ANN) with different input parameters and different data distributions and ANN structures with excellent precision (R^2 of 0.996). Brenjkar [22] developed three ROP neural network models from four drilled wells in southwestern Iran and compared them with Bingham and B&Y models. As a result, he obtained a high-performance model with R^2 and an average absolute percent relative error (AAPRE) of 0.948 and 5.531 respectively. Al-Abduljabbar [23] also created a new empirical correlation based on an optimized ANN model to predict ROP in horizontal carbonate reservoirs, with remarkable results in unseen data such R^2 and average absolute percentage error (AAPE) values of 0.946 and 5.29% respectively. Similar work was performed by Manta [24], where he designed a new model based on statistical regression and ANNs, using data from horizontal wells from the North Sea. Furthermore, Noshi [25] and Singh [26] implemented five and eight different ML models for ROP prediction respectively and all algorithms were compared. In addition, Tunkiel [27] and Soares [3] introduced a novel continuous learning approach, where each Machine Learning model is updated every specific number of meters simulating drilling conditions. Finally, last year Encinas developed a data driven ROP model to identify the influence of drilling parameters based on data from Volve field. He introduced a novel downhole WOB correction based on surface data. However, many of these studies are not available to replicate due to data

availability. Therefore, Tunkiel [28] published a dataset for ROP benchmarking, which is analysed in the following subsection.

On the other hand, other studies were conducted to optimize ROP. According to Gan [29] there are three different types of optimizations: robust based, moving horizon based and metaheuristic based. Wiktorski [30] understood the necessity of including the influence of wellbore trajectory, inclination, and azimuth in Burgoyne and Youngs model. Consequently, she developed an empirical model adding the dog leg severity (DLS) factor. Sui [31] designed a moving horizon method to predict ROP using a linear discrete-time model. However, as will be mentioned in section 2.3., for Machine Learning model optimization a metaheuristic approach is needed. Hedge continued his studies [5] [32] and implemented and evaluated different optimizers [33] in the best ML model from the previous research. Then, he implemented a drilling optimization model using the best ML model and best optimizer [7]. Alternatively, Alali [4] used another approach to optimize ROP while using data driven models. Using data from 2500 wells from an offshore field, where 18 wells with the best performance were chosen, he managed to create a two-phase model optimization to modify drilling controllable variables in real time. During field trial showed an improvement of 25% to 45% of ROP, saving 15% costs per foot.

2.2.3. Issues and Discussions

Many investigations have been conducted using data driven models to predict ROP, most of them comparing with traditional models. In one of his studies [5], Hedge mentioned different disadvantages of the traditional models, which includes ROP low accuracy predictions, variation of empirical coefficients (since these are obtained from linear regression for each lithology facie) and in many cases the requirement of auxiliary data (bit design and properties, mud properties, etc). In his investigation, he compared the performance and accuracy of three different machine learning models with three ROP traditional models for different formations in a field and showed a big improvement in case of data driven models.

As Tunkiel pointed in his study [28], both approaches require good data quality (in the case of traditional models to find accurate constants and for data driven approach to train the model). Therefore, a data driven model is only applicable for similar lithology,

drilling procedures and equipment and neighbour wells can be used as a reference starting point.

Other aspects that are concerning regarding previous investigation in data driven ROP models. For instance, Tunkiel [28] mentioned incorrect data split and the possibility to reproduce the experiments. As far as the author knows, there are three different ways to split the data for ML training: random sampling, sequential sampling, and continuous learning. The first one breaks training and test data randomly, which does not represent well drilling conditions, and one can reach R^2 levels of 0.95. This can be easily explained with fig. 4 that shows random and sequential sampling, where the depth grows from top to bottom. Assuming that test cells are the sample points that are “drilled”, it is clear that using sequential sampling is more advisable to simulate drilling process since it is illogic to predict a parameter from a depth you have already drilled and also impossible to “jump” through different depths without drilling. In this study, random sampling split is not considered due to the reasons mentioned above.

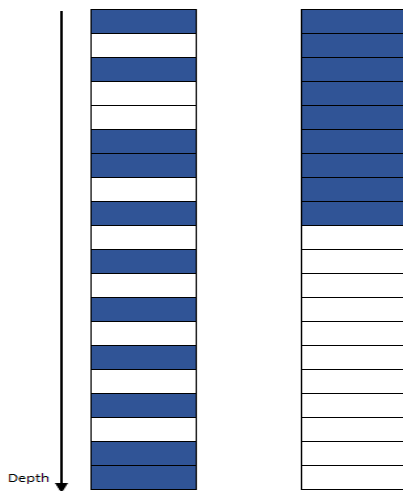


Figure 4: Random train/test split (left) vs. Sequential train/test split (right). Each cell represents a row in the test matrix in depth growing order, where the blue ones are the training sample and white ones makes the test sample.

Finally, data and source code are not available in most of these studies as Tunkiel [28] mentioned for experiment reproduction. For this study, data from Johan Sverdrup is confidential and will not be shared, but source code is available in the appendix and can be adapted to use in another field data.

2.3. Optimization

Optimization consists of finding the maximum or minimum value in a real function, therefore find the optimal value from the space solution. The complexity of the equation or problem will define the optimization algorithm to be used.

The interest in optimization has grown in the last few years. In fact, since the mid-sixties, many algorithms that focus on global optimization of “black-box” problems (the problems where exact analytical methods do not work) have been proposed. In his book, Boyd [34] defined a mathematical optimization for convex functions as a problem where the goal is to minimize the objective function within some constraints (boundaries). These are, however, not the functions used on this thesis. ML models are not mathematical equations, and therefore the need of metaheuristic to optimize this problem.

Metaheuristic, as Luke [35] defined, is a “subfield of stochastic optimization”, which he describes as “a class of algorithms and techniques which employ some degree of randomness to find optimal solutions to hard problems”. These algorithms work very well when having problems with little information and the space is too large.

In this study, two stochastic algorithms will be implemented: Differential Evolution and Particle Swarm. These algorithms are two different population methods, one of the subfields of stochastic optimization. Both have been used in the industry and much research have been conducted to evaluate its performance [33] [36], where the study of Hedge is the reference since it has been applied in the same field.

2.3.1. Differential Evolution Algorithm

Differential Evolution (DE) algorithm was introduced by Storn and Price in 1995. As an algorithm from the evolutionary family, it is based on Darwin's Natural Selection Theory, where the fittest individual survives. Fig. 5 describes the different terms used for this algorithm family, extracted from Luke’s book [35].

The process, according to Luke, is the following:

1. First, it starts with the initialization of a combination of random solutions (Creates a population).
2. Then, iterates three different procedures:
 - 2.1. It evaluates the fitness of all individuals.
 - 2.2. It uses these fitness values to breed a new children’s population.

2.3. Finally, it joins parents with children in the new generation, and cycle begins again.

The cycle stops until a specific number of generations is reached or until it fulfils a particular criterion. There are two new innovations (twists) introduced in the DE algorithm. First, everyone of the population creates a child and this child competes against parents to be included in the population. Second, the child's mutation size is determined based on the variance of the population, and therefore it is not fixed as in other algorithms in this family.

| | |
|----------------------------------|---|
| individual | a candidate solution |
| child and parent | a <i>child</i> is the Tweaked copy of a candidate solution (its <i>parent</i>) |
| population | set of candidate solutions |
| fitness | quality |
| fitness landscape | quality function |
| fitness assessment or evaluation | computing the fitness of an individual |
| selection | picking individuals based on their fitness |
| mutation | plain Tweaking. This is often thought as "asexual" breeding. |
| recombination or crossover | a special Tweak which takes two parents, swaps sections of them, and (usually) produces two children. This is often thought as "sexual" breeding. |
| breeding | producing one or more children from a population of parents through an iterated process of selection and Tweaking (typically mutation or recombination) |
| genotype or genome | an individual's data structure, as used during breeding |
| chromosome | a genotype in the form of a fixed-length vector |
| gene | a particular slot position in a chromosome |
| allele | a particular setting of a gene |
| phenotype | how the individual operates during fitness assessment |
| generation | one cycle of fitness assessment, breeding, and population re-assembly; or the population produced each such cycle |

Figure 5: Common Terms Used in Evolutionary Computation. Source [35]

DE employs vector operations on its different types of mutation. The easiest one to describe is the primary mutation operator. Here, for each member of the population, a new child is generated (offspring) by vector addition and subtraction from three different random individuals from the population. From fig. 6 from Luke's book it is easier to see the general idea: to mutate from one random individual (a) by adding a vector, which is the difference between vectors b and c. Then, the child is crossover with i. Finally, after all parents created the new children (though different mutations, not necessarily the one mentioned), all children compete with their parents to survive (they replace parents in the population if they have higher fitness value).

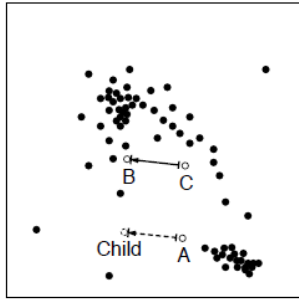


Figure 6: DE primary mutation operator illustration. A copy of member A is mutated by the addition of the vector difference between B and C. Source [35]

2.3.2. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population technique similar to DE. It is not based on evolution theory but inspired in the flocking behaviour of birds. It was created in 1995 by James Kennedy and Russell Eberhart [36]. The main difference with other population methods is that PSO does not resample or replace populations (there are no generations). It maintains a population whose individuals are tweaked based on new discoveries about the environment, like bird's behaviour. As Luke says [35], it is basically a form of mutation.

In this context, DE terms are not used, and individuals' population transforms into a swarm of particles. Then, this direct mutation that was mentioned above does not replace the particle genes, it only moves the particles around the space. According to Luke [35], a particle consists of two parts, the particle's location in space and the particle's velocity at each timestep. Moreover, each particle has a small memory that stores its own position (x^*), the best place that any of its informants (x^+) has discovered and the fittest location discovered by anyone ($x^!$). For each timestep, after evaluating the fitness score (location and how closed it is to the optimal point) of all particles, each particle could be added a velocity vector pointing to x^* , x^+ or $x^!$. This is performed randomly and based on the different scores given to the parameters to run the code, which are summarized below (for more information, the source is Luke's Book [35]):

- α : proportion of the original velocity retained
- β : ratio of the personal best (location) to be retained. The larger is β , the higher tendency the particle has to move to its own personal bests.
- γ : proportion of the informants' best location to be retained.

- δ : proportion of global best to be retained. The larger is δ , the higher tendency the particle has to move to global best.
- ϵ : jump size of particle. Large ϵ makes the system move quickly to best regions.

2.4. Drilling Engineering Models


2.4.1. Downhole RPM Calculation

It is a common practice in the NCS to drill the surface casing section using mud motor. Since all data was measured from surface in this study, it was necessary to calculate the downhole bit revolution to create an accurate model.

The calculation is very simple: downhole RPM (or total bit revolutions DRPM) is calculated as the summation of drilling mud motor revolutions (RPM_M), which is the RPM of the mud motor's rotor, and surface RPM (RPM_S):

$$DRPM = RPM_M + RPM_S \quad (16)$$

Mud motor's revolutions are calculated using speed to flow ratio [37]. These specifications are available in the drilling motor's Handbook. Below there is a screenshot of one of the Baker Hughes motors specifications [38]:



The image shows a technical specification table for a Baker Hughes 6 3/4 in. Ultra XL/LS mud motor. The table includes flow rate, speed, and various operational limits for different elastomer types (Standard or High Temperature, DuraMax).

| | US Units | SI Units |
|--|---------------|-----------------|
| Flow Rate | 265-660 gpm | 1000-2500 l/min |
| Speed | 40-105 rpm | |
| Speed to Flow Ratio | 0.16 rev/gal | 0.04 rev/l |
| No Load Pressure Drop | 350 psi | 24 bar |
| Standard or High Temperature elastomer | | |
| Operational Limits (Maximum Operational for MMTR) | | |
| Differential Pressure | 435 psi | 30 bar |
| Torque | 5,300 ft-lbs | 7,200 Nm |
| Power Output | 105 hp | 79 kW |
| Maximum Operational | | |
| Differential Pressure | 695 psi | 48 bar |
| Torque | 8,500 ft-lbs | 11,500 Nm |
| DuraMax elastomer | | |
| Operational Limits (Maximum Operational for MMTR) | | |
| Differential Pressure | 655 psi | 45 bar |
| Torque | 7,950 ft-lbs | 11,000 Nm |
| Power Output | 160 hp | 120 kW |
| Maximum Operational | | |
| Differential Pressure | 870 psi | 60 bar |
| Torque | 10,500 ft-lbs | 14,500 Nm |

Figure 7: Baker Hughes 6 3/4 in. Ultra XL/LS motor data and specifications, power performance section. Source [38].

Then, RPM_M is calculated for a specific flowrate as follows:

$$RPM_M = \text{speed to flow ratio} * Q \quad (17)$$

2.4.2. Downhole Torque Model

Torque (or torsion) occurs when a twisting moment is applied to the pipe [39]. High Torque and high drag usually happen together and have different causes, including differential sticking, sliding wellbore friction, tight hole conditions, keyseats, increase of cutting volume, and sloughing hole [40]. As Johansick says, torque and drag calculations are more relevant for directional wells and even critical in the case of highly deviated wells. Therefore, for planning these wells and ensure successful drilling operations it is vital to choose a good torque and drag criteria. First, it will be mentioned the theory for inclined wells and then for deviated wells from Johansick.

Figure 8 shows a free body diagram of a mass drill pipe section of length ds in an inclined well.

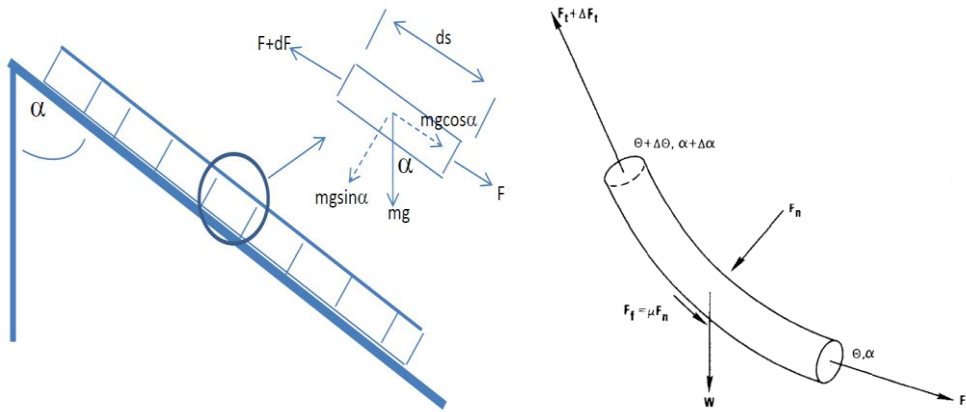


Figure 8: Free body pipe of a drill pipe unit of ds length (left). Source: [39]. Force acting in a curved drill string. Source: Johansick [40]

Applying Newton second law the drag force at the bottom is obtained:

$$F_T = F_B + \beta w \Delta s (\cos \alpha \pm \mu_d \sin \alpha) \quad (18)$$

Where F_B and F_T are the forces at the bottom and top, w the unit weight of the drill string, Δs the length of the drill string, μ_d the friction coefficient, β the buoyancy factor and α the inclination (+ for hoisting and – for lowering pipe). Also, according to Belayneh [39], torque is calculated as the normal moment multiplied by friction factor and pipe tool joint radius as follows:

$$T = r \beta w \Delta s \mu_d \sin \alpha \quad (19)$$

Where r is the pipe tool joint radius. For curved sections, the normal force is calculated from Johansick [40] equation:

$$F_n = \sqrt{(F_t \Delta\alpha \sin\alpha)^2 + (F_t \Delta\theta + \beta w \sin\theta)^2} \quad (20)$$

Where θ is the azimuth angle and α the inclination. This leads to tension increment, which is:

$$\Delta F_t = \beta w \cos\theta + \mu_d F_n \quad (21)$$

And torque increment:

$$\Delta M = \mu_d F_n r \quad (22)$$

Bit torque can be measured by MWD tools or in the laboratory. Usually, torque is measured in the surface and sometimes it is not reliable. In addition, force values F_T and F_B from equation 17 are not always available. Then, Pessier [41] developed a model as function of WOB [lbf], bit diameter (d_b [in]), and coefficient of sliding friction (μ_P):

$$T = \frac{\mu_P d_b WOB}{36} \quad (23)$$

This equation was later modified by Belayneh [42] considering rotation effect using the modified sliding friction factor, which is a function of axial velocity (V_a), rotary speed (RPM), and bit radius (r_b):

$$T = \frac{\mu_P \cos(\tan^{-1}(\frac{V_a}{r_b \text{ RPM}})) d_b WOB}{36} \quad (24)$$

2.4.3. Downhole WOB Model

As it was mentioned at the beginning of this study, rate of penetration is dependant of weight on bit. Thus, it is vital to have a good measure of downhole WOB. The introduction of measurement while drilling techniques has improved drilling performance in many wells. In fact, the availability of downhole sensors to accurately measure different drilling variables such as downhole WOB and Torque on bit help in decision making and decrease non-productive time (NPT). However, these tools are sometimes costly for some wells and consequently, this specific data is not handy.

Hareland [43] developed in 2014 a DWOB theoretical calculation based on a torque and drag model. The same model was also calibrated using field data and compared with the Autodriller system from Baker Hughes with similar results. Encinas [44] also

implemented this correction successfully in his ROP machine learning model last year. For the Hareland model, three effects are considered to adjust hook load:

1. Sheave effect:

$$HL_{a1} = \frac{HL_f (1 - \eta^n)}{n (1 - \eta)} \quad (25)$$

Where HL_f is the field hook load, η is the sheave efficiency and n the number of lines.

2. Static Hook effect:

$$HL_{a2} = \frac{HL_s (1 - \eta^n)}{n (1 - \eta)} \quad (26)$$

Where HL_s is the field hook weight, η is the sheave efficiency and n the number of lines.

3. Standpipe pressure effect:

$$HL_{a3} = 5.095 \times 10^5 SPP OD^2 \quad (27)$$

Where SPP is the stand-pipe pressure and OD the outside diameter of the pipe. Finally, the Hook load correction is:

$$HL_{mod} = HL_{a1} - HL_{a2} - HL_{a3} \quad (28)$$

Chen [45] also modelled downhole WOB using Aadnøy drag model [46] for curved sections, since WOB is not modified for straight sections. The equation is as follows:

$$WOB_b = WOB e^{-\mu \Delta \alpha} \quad (29)$$

Where WOB is one weight on bit at a given depth, μ the friction factor and $\Delta \alpha$ inclination difference between two adjacent sections.

Wiktorski and Sui [47] proposed a finite element WOB model in the aim of modelling torsional and axial drill string vibrations. Figure 9 represents axial and torsional displacements acting in a drill string element.

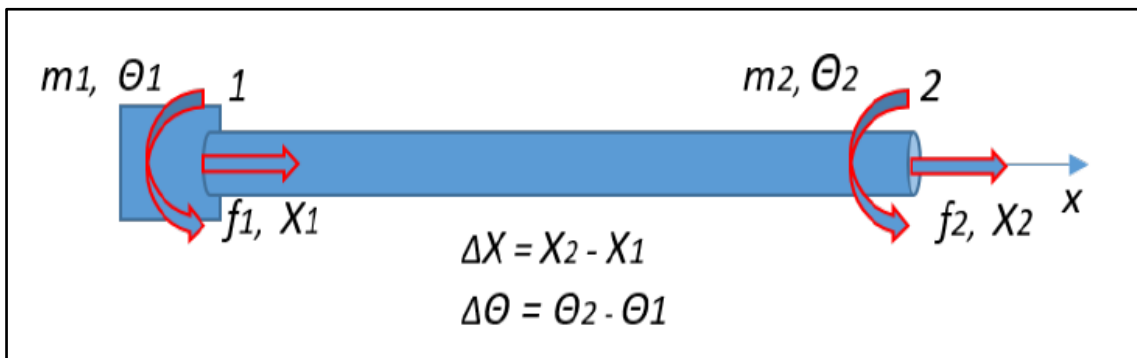


Figure 9: Drillstring element under influence of torsional and axial forces. Source: [30]

This problem is solved using a second-order differential equation as is shown below:

$$M\ddot{U}(t) + C\dot{U}(t) + KU(t) = F_{ext}(t) \quad (30)$$

Where M is the mass matrix, C is the damping matrix, K is the stiffness, and F_{ext} is the total external force applied to the drill string. U is the displacement, \dot{U} is the velocity, and \ddot{U} the acceleration (the three variables could be torsional or axial depending if we calculate torsional or axial displacement). In this system, damping is neglected and therefore C is equal to zero. Stiffness and mass matrix are represented in the following way:

$$K = \begin{bmatrix} k_i & -k_i \\ -k_i & k_i \end{bmatrix} \quad (31)$$

$$M = \begin{bmatrix} m_i & 0 \\ 0 & m_i \end{bmatrix} \quad (32)$$

Where k_i and m_i are the stiffness and mass elements respectively from rotation or translation movements. These are calculated as follows:

$$m_a = \rho_s A l / 2 \quad (33)$$

$$m_t = \rho_s I l \quad (34)$$

$$k_a = EA / l \quad (35)$$

$$k_t = G I / l \quad (36)$$

Where m_a is the axial mass element, m_t the mass torsional element, k_a the stiffness axial element, k_t the stiffness torsional element, ρ_s the drillstring density, A the cross sectional area of the drill string, l the drillstring length, E is the young's modulus and G the shear modulus. I is the inertia moment, which is calculated using the inner (d_i) and outer (d_o) diameter of the pipe as follows:

$$I = \frac{\pi (d_o^4 - d_i^4)}{32} \quad (37)$$

Then, based on different assumptions (for more information, read the paper), Wiktorski [47] formulates the effective downhole WOB model:

$$WOB = WOB_{set} \frac{\dot{x}_{total}}{\dot{\theta}_{total}} \delta \quad (38)$$

Where WOB_{set} is the downhole WOB during normal drilling, δ is a correction factor, and \dot{x}_{total} and $\dot{\theta}_{total}$ are the total axial and torsional displacements. The last two parameters mentioned are calculated by the sum of the relative velocity (axial or torsional), obtained from eq. 29 and the surface velocity that is an input parameter.

2.4.4. Density Model

During drilling, drilling fluid temperature and pressure increase as depth increases. Mud density is affected by the variation of these parameters; for instance, higher pressure induces a raise in mud density while higher temperature causes the opposite. It is very important not to neglect these effects on mud density, especially on high pressure high temperature (HPHT) wells [48], when wrong density estimation can result in wildly inaccurate pressure prediction. Therefore, the density model is a function of pressure (P) and temperature (T), and it can be linearized as follows:

$$\rho_m = \rho_{ref} + \frac{\rho_{ref}}{\beta}(P - P_{ref}) + \rho_{ref} \alpha (T - T_{ref}) \quad (39)$$

Where ρ_m is the mud density, ρ_{ref} , T_{ref} , P_{ref} the density, temperature and pressure used as reference, β the isothermal bulk modulus of the liquid, and α the cubical expansion coefficient of the liquid.

Usually, changes due to temperature and pressure for liquids are small and can be neglected.

2.4.5. Mechanical Specific Energy

In 1965, Teale [49] defined Mechanical Specific Energy (MSE) as “the energy required to excavate a unit volume of rock”. MSE has a variety of uses, including selection and optimization of drilling parameters and bit design. The old formula considers the axial and torsional energy involved when bit is removing a rock unit. MSE is calculated as follows:

$$MSE = \frac{4 WOB}{\pi d_{bit}^2} + \frac{480 RPM T}{d_{bit}^2 ROP} \quad (40)$$

Where d_{bit} is bit diameter, WOB is the weight on bit, RPM the rotary speed, ROP the rate of penetration, and T the Torque. Teale also introduced the concept of maximum mechanical efficiency, which is obtained when the MSE is closed to the uniaxial compressive strength of the rock (σ).

$$MSE = MSE_{min} \approx \sigma \quad (41)$$

$$ME_{ff} = \frac{MSE_{min}}{MSE} 100 \quad (42)$$

Where ME_{ff} is the mechanical efficiency. Then, Dupriest [50] introduces a correction factor in the newly modified MSE (MSE_{mod}):

$$MSE_{mod} = ME_{ff} \left(\frac{4 WOB}{\pi d_{bit}^2} + \frac{480 RPM T}{d_{bit}^2 ROP} \right) \quad (43)$$

2.4.6. Hydromechanical Specific Energy

In 2014, Mohan [6] modified Teale's equation by adding a hydraulic term, which represents "the hydraulic force exerted by the impact of drilling fluid on the formation". This impact force has a reaction force (pump off force) in accordance with Newton's third law and thus, the effective WOB decreases. The impact force is calculated as follows:

$$F_j = 0.000516 \rho_m Q V_n \quad (44)$$

Where ρ_m is the mud density, Q is the flow rate, and V_n is the nozzle exit velocity. This study also says that only 25-40% of this hydraulic energy reaches formation and thus defines the ratio (A_v) of nozzle jet velocity (V_n) to the return fluid velocity (V_f) considering that the available area of the total bit region is 15%.

$$A_v = \frac{V_n}{V_f} = \frac{0.15 d_{bit}^2}{n d_n^2} \quad (45)$$

Where d_n is the nozzle average diameter and n the number of nozzles.

This publication also remarked how the energy available at the formation is affected by the distance to the nozzle. This energy is "inversely proportional to the square of the distance from the nozzle to formation". Mohan presents a dimensionless variable (M) defined by Rabia that includes nozzle to formation distance (s), length of potential core (L), and the angle of axially symmetric jet (θ):

$$M = \frac{d_n + 2 L \tan(\theta/2)}{d_n + s \tan(\theta/2)} \quad (46)$$

With these variables, factor for reduction of energy is defined as follows:

$$\eta = \frac{1 - A_v^{-k}}{M^2} \quad (47)$$

Where k is different for each bit, and in this study is equal to 0.122 [6].

Finally, HMSE is calculated using effective WOB (WOB_e) as it was mentioned before and adding the hydraulic jet energy of the fluid to the formation (E_h) to the torsional (E_t) and axial energy (E_a).

$$HMSE = 4 \frac{E_a + E_t + E_h}{\pi d_{bit}^2 ROP} \quad (48)$$

$$HMSE = 4 \frac{WOB_e ROP + 120 \pi RPM T + \eta \Delta P_b Q}{\pi d_{bit}^2 ROP} \quad (49)$$

Hydraulic energy is calculated as the multiplication of the factor for energy reduction (η), the pressure loss across the bit (ΔP_b), and the flow rate (Q). According to this study [6], effective WOB is calculated as follows:

$$WOB_e = WOB - \eta F_j \quad (50)$$

Bit pressure loss equation was taken from Aadnøy's book [51]:

$$\Delta P_b = 8 \frac{\rho_m Q^2}{\pi^2 d_n^4 0.95^2} \quad (51)$$

Where ρ_m is the mud density.

3.Database Analysis

3.1. Johan Sverdrup Dataset

In this work, a confidential dataset of six wells from Equinor’s Johan Sverdrup field was used. For confidentiality, wells will be named as follow: well 1, well 2, well 3, well 4, well 5, and well 6. This field will be described briefly below.

Johan Sverdrup Field

Johan Sverdrup field started production in October 2019 with approximately 435,000 barrels per day. It is the third largest field in the NCS and is powered with hydraulic energy from shore. According to Equinor [52], it is expected to produce 535,000 barrels per day in mid-2021 and 720,000 barrels per day at plateau with a high recovery factor of 70% and low CO₂ emissions.

3.2. Data Import and Visualization

The first step in database analysis is data import. It is vital to select the valid data for the model to save time in data processing. The database import was performed in each well, and each one has its own csv file and json file. For each well were selected 16 features and since it is a time-based database, each physical parameter (any feature different than time) has its own sensor and therefore its own time feature. For a better understanding, a table is presented with the unit and data type of each parameter:

| Feature | Units | Data Type |
|---------------------|-------------------|-----------|
| time_DB | time units | object |
| Depth Bit | meters | float |
| time_DH | time units | object |
| Depth Hole | meters | float |
| time_RPM | time units | object |
| Rotary Speed | rev/s | float |
| time_TOR | time units | object |
| Torque | N.m | float |
| time_WOB | time units | object |
| Weight on Bit | N | float |
| time_ROP | time units | object |
| Rate of Penetration | m/s | float |
| time_FR | time units | object |
| Flow rate | m ³ /s | float |
| time_SPP | time units | object |
| Stand Pipe Pressure | Pa | float |

Table 1: Initial features from a representative well

Now, each parameter will be described. It is important to note that all the measurements were performed in the surface and during a time period since all information is time-based:

- time_DB is the time measurement of the bit depth. It indicates the exact moment (day, hour, minute and second) at which the bit position (depth) was measured.
- Depth Bit is the bit position. In this case the Measured Depth (MD).
- time_DH is the time measurement of the hole depth (DH). It indicates the exact moment (day, hour, minute and second) at which the borehole depth was measured.
- Depth Hole is the borehole position (depth), always higher than the bit depth.
- time_RPM is the time measurement of the surface rotary speed (RPM). It indicates the exact moment (day, hour, minute and second) at which the RPM was measured.
- Rotary Speed are the top drive revolutions measured from the surface.
- time_TOR is the time measurement of the surface torque. It indicates the exact moment (day, hour, minute and second) at which the torque was measured.
- Torque is the measurement of the surface torque.
- time_WOB is the time measurement of the surface weight on bit (WOB). It indicates the exact moment (day, hour, minute and second) at which the WOB was measured.
- Weight on Bit is the measurement of the surface weight on bit (WOB)
- time_ROP is the time measurement of the rate of penetration (ROP). It indicates the exact moment (day, hour, minute and second) at which the ROP was measured.
- Rate of Penetration is calculated over a time interval (usually 60 seconds), it measures the bit depth changes over time.
- time_FR is the time measurement of the flow rate (Q). It indicates the exact moment (day, hour, minute and second) at which the flow rate was measured.
- Flow Rate is measured from the mud pumps
- time_SPP is the time measurement of the surface stand-pipe pressure (SPP). It indicates the exact moment (day, hour, minute and second) at which the SPP was measured.
- Stand-Pipe Pressure is detected from a gauge located in the stan pipe.

3.3. Data Cleaning

After selecting the necessary information, it is important to improve the quality of the data to implement Machine Learning models and perform Optimization. The cleaning is done in four different steps: handling missing values, handling faulty measurements, removing outliers and removing noise. Each of these steps will be explained in the following subsections.

3.3.1. Handling Missing Values

Since it is a time-based data, all parameters have its corresponding time column, which means the time where sensor measure certain parameter. All the data measurements were performed in the same timeframe (on each well), so for example the starting row and ending row in time_DB and time_DH columns are the same (could be one or two seconds of difference, but is neglectable). When watching data structure, it was clear that some values were missing in almost all columns and that these were located at the end of the data frame. This showed that the frequency of the measurements on the different features varies and therefore, some variables have higher number of measurements than others. Thus, a dataframe was created for each variable (8 different dataframes were created) and missing values were deleted. Fig. 10 illustrates data of well 3 after this process.

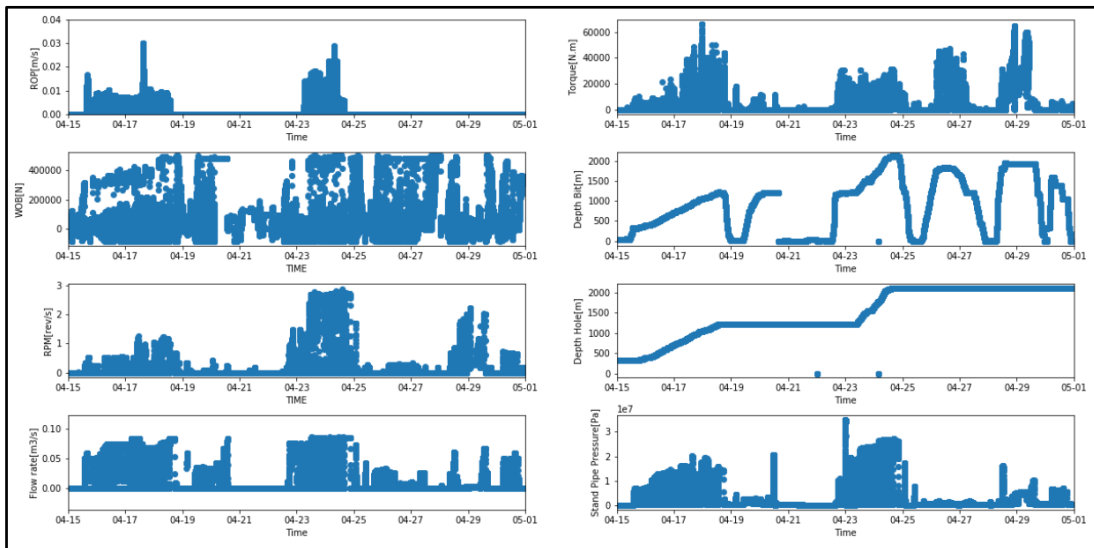


Figure 10: Plot of the different variables vs. time for well 3.

More pre-processing work was necessary prior to start removing outliers. Since it is very difficult to work with different time columns, it was necessary to merge all the

eight dataframes based on the time column with a higher number of measurements to avoid deleting relevant data. In all the wells time_DH was used as reference for merging except for the well number 2, where time_DB was used as reference.

Consequently, some missing values appeared in the different variables and it was necessary to interpolate data to fill that information. Linear interpolation was used since the frequency of measurements is low (usually between 2 or 3 seconds). After removing all duplicates and missing values, the data structure changed and was ready to remove faulty measurements. Table 2 shows the new structure of the data.

| Feature | Units | Data Type |
|---------------------|------------|-----------|
| time | time units | datetime |
| Depth Bit | meters | float |
| Depth Hole | meters | float |
| Rotary Speed | rev/min | float |
| Torque | kN.m | float |
| Weight on Bit | N | float |
| Rate of Penetration | m/h | float |
| Flowrate | l/min | float |
| Stand-Pipe Pressure | kPa | float |

Table 2: Initial features from a representative well after merging and interpolating.

3.3.2. Handling Faulty Measurements

Before removing outliers and noise it is necessary to delete some unnecessary data. Due to the frequency in sensors measurement (usually between 2 and 3 seconds) so many faulty measurements were recorded.

The first step was to remove all non-drilling data and select only data where ROP was higher than 0. It is normal to have negative ROP values when tripping out (lift all the drill string due to bit change or another problem) since depth decreases.

Secondly, negative WOB values were also deleted since this is not a realistic behaviour of drill string.

3.3.3. Removing Outliers

According to Sui [48], “an outlier is an observation point that is distant from other observations”. It also mentions that it can be due to an experimental error or variability in the measurement. The purpose of removing outliers in this study, was to narrow data in a realistic range [4]. Therefore, some outliers were removed manually after data observation and identifying some anomalies outside the range. In this case some

parameters were out of range, and flow rate lower than 2000 lt/min and Standpipe pressure lower than 5000 KPa were deleted. Also, in some wells ROP was out of the range and it was necessary to remove that noise that was not removed by the techniques explained below. After these steps, a more sophisticated technique is used to remove outliers, which is called the Interquartile Range (IQR).

Interquartile Range (IQR)

According to Hadi, “the interquartile range is the central 50% or the area between the 75th and the 25th percentile of a distribution” [53]. This is a solid method to remove points located far away from the range [44] and when the data is not normally distributed. However, relevant data could be incorrectly deleted, and it should be used carefully. This method uses the following formulas:

$$IQR = P_{75} - P_{25} \quad (52)$$

Afterwards, the IQR technique uses the following upper and lower cut-offs:

$$Lower\ Cutt - off = P_{25} - 1.5 IQR \quad (53)$$

$$Upper\ Cutt - off = P_{75} + 1.5 IQR \quad (54)$$

Consequently, a point that is not inside the range will be considered an outlier and removed. Results for applying the IQR method and removing outliers manually are presented below as an illustration in fig.11.

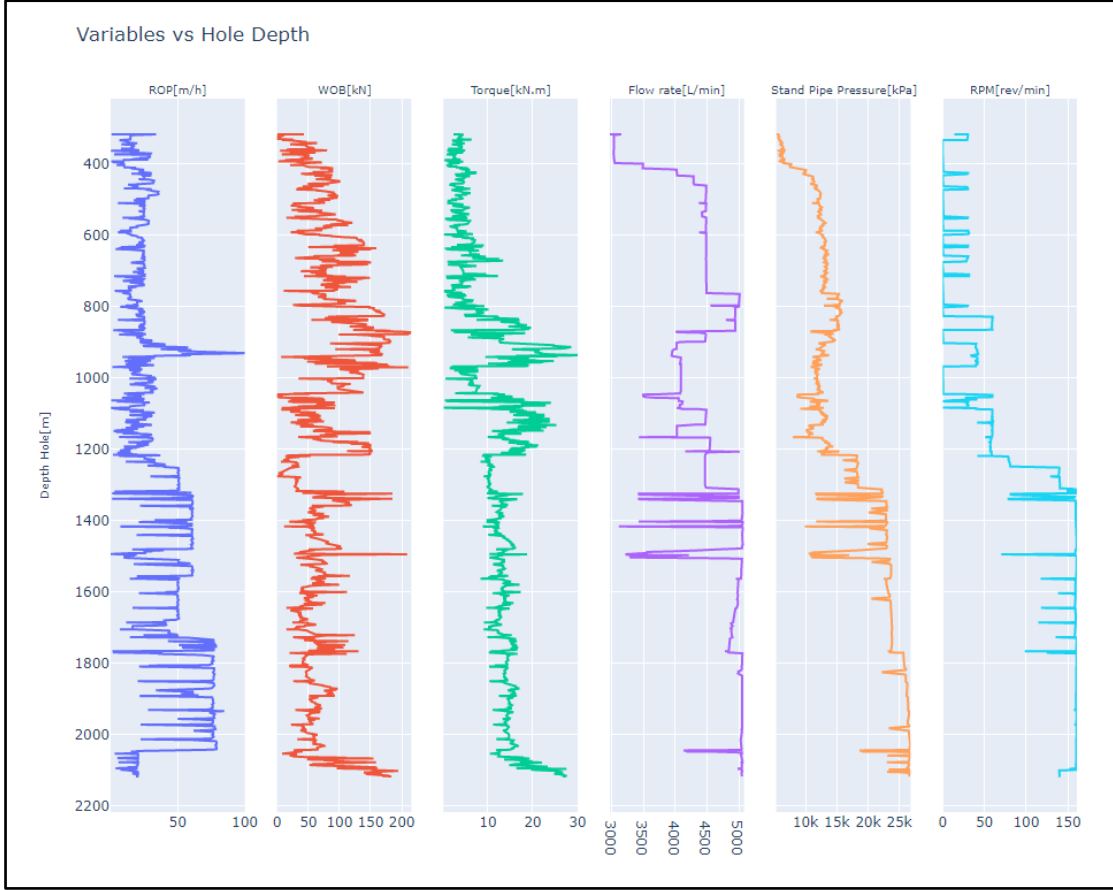


Figure 11: Variables vs. depth of well number 6 after removing outliers manually and using IPR method in well 1.

3.3.4. Removing Noise

To smooth data signals, filtering process was executed. In this study, a moving average filter was used, which formula is defined in the following way, as it is noted by Sui [48]:

$$y(t) = \frac{1}{n} (x_{(t-n+1)} + x_{(t-n+2)} + \dots + x_{(t)}) \quad (55)$$

Applying Fourier transform:

$$x(t - k) \leftrightarrow X(j\omega)e^{-jk\omega} \quad (56)$$

$$Y(j\omega) = \frac{1}{n} X(j\omega) (e^{-j\omega(n-1)} + \dots + e^{-j\omega} + 1) \quad (57)$$

Transfer function becomes:

$$H(j\omega) = \frac{1 - e^{-j\omega n}}{n(1 - e^{-j\omega})} \quad (58)$$

This filter was implemented in python using the syntax rolling, which is showed in the appendix. Results after applying median average filter in well 1 are shown in fig. 12. For more information, the code is available in the appendix.

Radius Neighbours Regressor

Although filtering showed remarkable results in data quality, it is necessary to downsize data to removed noises that could not be removed by the moving average filter and for computational running. The data was initially time-based, but it is easy to handle and understand when it is depth-based. Besides, at this point it was required to separate data in different sections since each section will have a different machine learning model. Therefore, as the frequency of measurements is very low, many points share the same depth and the parameters were averaged and grouped by depth. Furthermore, it was necessary to evenly distribute the depths data points before ML modelling and Radius Neighbours Regressor were implemented. One example is shown in fig. 13. The data was reduced to 90%, but as it is seen, the pattern is the same and it has not been altered drastically. The code is written in the modelling part of the appendix, but it is considered a data cleaning technique, so it is described in this chapter.

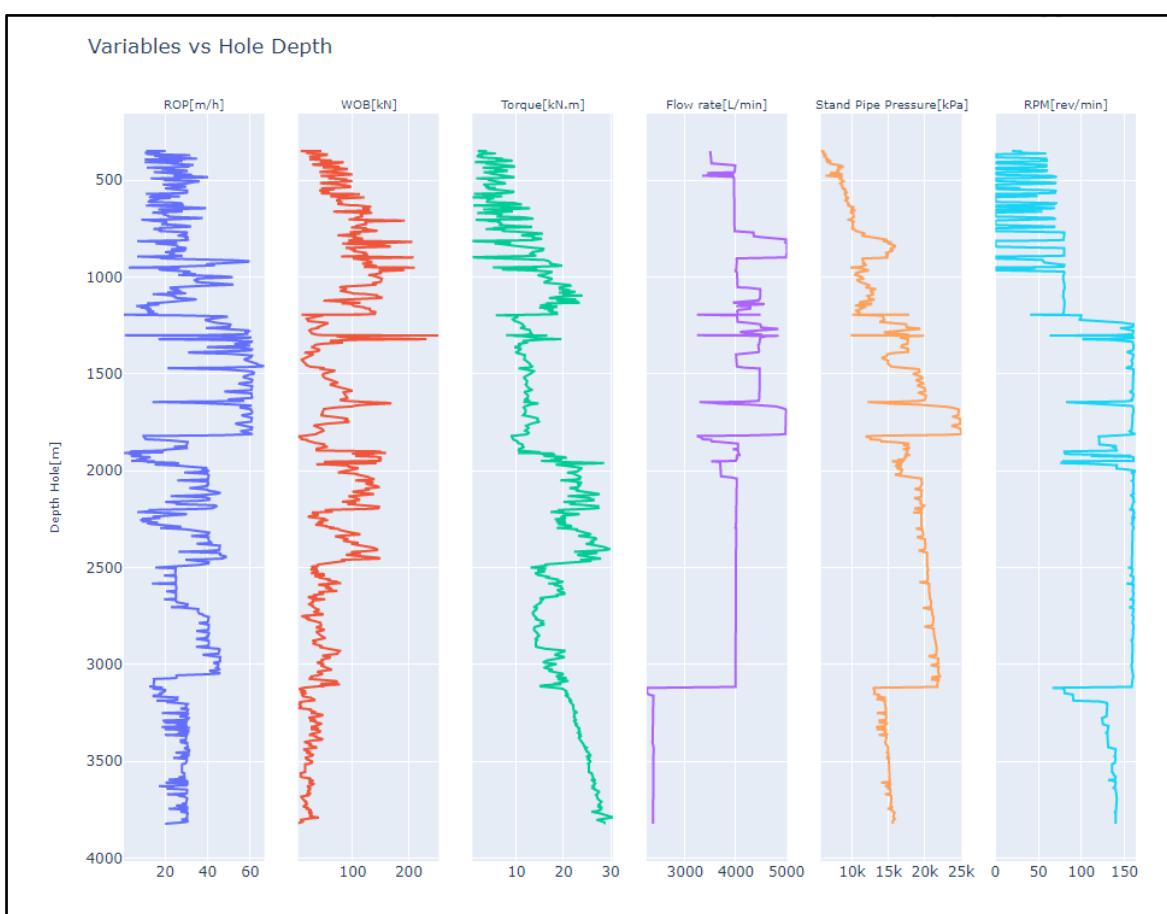


Figure 12: Variables vs. depth of well number 6 after removing noise using median average filter in well 1

Median Filter

As Sui [48] mentions, median filter replaces data signals with the median of neighbours. This is done by providing the code a window size, which is the number of rows the filter takes when it is running. In this project, to remove the last outliers and smooth the signals, a median filter was applied at the end of the cleaning process. In order to perform this operation, the `median_filter` function from SciPy was adopted. The code implementation, along with all the cleaning process is attached in the appendix and is also written in the modelling part as it was executed after Radius Neighbors Regressor.

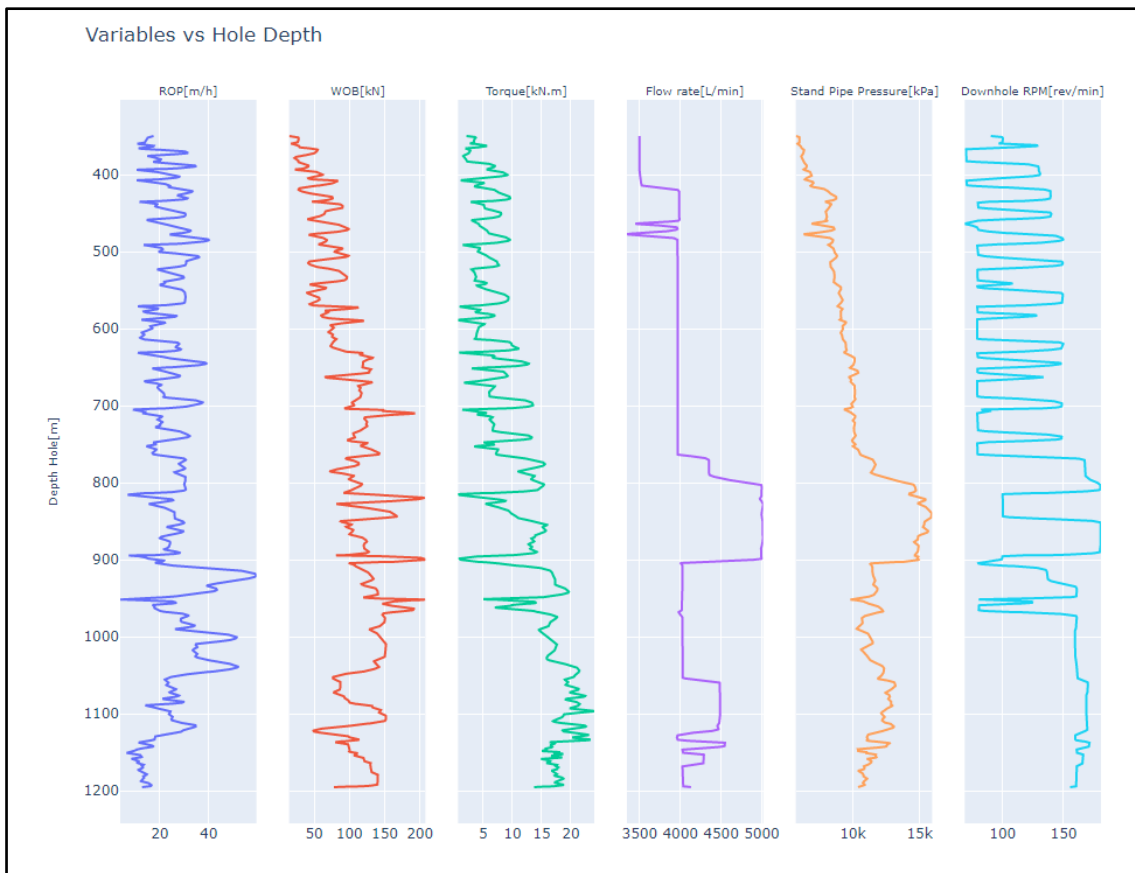


Figure 13: Variables vs. depth of well number 6 after removing noise using median average filter and Radius Neighbors Regressor in well 1 section 26”.

3.4. Data Selection

Data selection was made based on the performance of the different ML models. Many well sections were tested, and only section 26” from well 1 had good performance using Machine Learning with both ROP and Torque models. It is relevant to remark that this final selection was made after modelling all wells sections and not obtaining good results in both models. Considering this project uses ML data driven models, only relevant

data was selected where the ROP was in the same range in each well section. For both ROP and Torque models, Bit Depth (DB), Weight on Bit (WOB), top drive revolutions (RPM), and flow rate (Q) were selected as independent variables and ROP and Torque as dependent variables.

4. Machine Learning Implementation

In this chapter the implementation of distinctive Machine Learning algorithms will be explained. In this study, Python was used as coding language due to the reasons mentioned in Chapter 1 Section 2. Four ML algorithms were used and evaluated in all well sections, where each well section corresponds to a different casing size.

To implement these ML models, Python library Skicit Learn was used. Random Forest (RF), Gradient Boosting (GB), AdaBoost (AB) and K-Nearest Neighbors (KNN) were evaluated in two different scenarios for ROP and Torque models that will be explained in the following section. Table 4.1 shows input features for all models.

| Feature | Units | Data Type |
|---------------|---------|-----------|
| Depth Bit | meters | float |
| Rotary Speed | rev/min | float |
| Weight on Bit | kN | float |
| Flow rate | L/min | float |

Table 3: input features for ROP and Torque models.

4.1. Data Split

Most of the studies conducted in ROP modelling and/or optimization are not clear on how data was split. As Encinas [44] mentioned in his study, data must be separated into three parts to avoid model bias and overfitting: training set, validation set and test set. In this work, data was split into training and test set and was tested using sequential sampling and continuous learning sampling.

As it was noted in Chapter 2 Section 2.3, there are three different types of data sampling: random sampling, sequential sampling, and continuous learning sampling. The first one is based on separate data randomly, for example selecting 70% percent of random data as training set and remaining 30% as test set. The problem with this approach has been described before and will not be extended. To the writer point of view, it is an unrealistic drilling scenario.

Sequential sampling, however, represents a more realistic scenario. Here, data is split in the desired percentage but not randomized. Training data represents the section that has been drilled and the test data is compared with the predicted data, which is the section that is being drilled. For instance, 65% of initial data is selected as training set and the remaining as test set. Fig. 14 can easily explain the difference between the two types of data split.

Continuous training sampling is explained in the next chapter. For the purpose of modelling, this split technique was used to understand data and select sample data for optimization. Usually, when using this method, the performance metrics of the Machine Learning model improves while increasing the training set. Therefore, in the hypothetical case that the best metric performance is reached when 70% of data is selected as training data, 70% of data will be the sample.

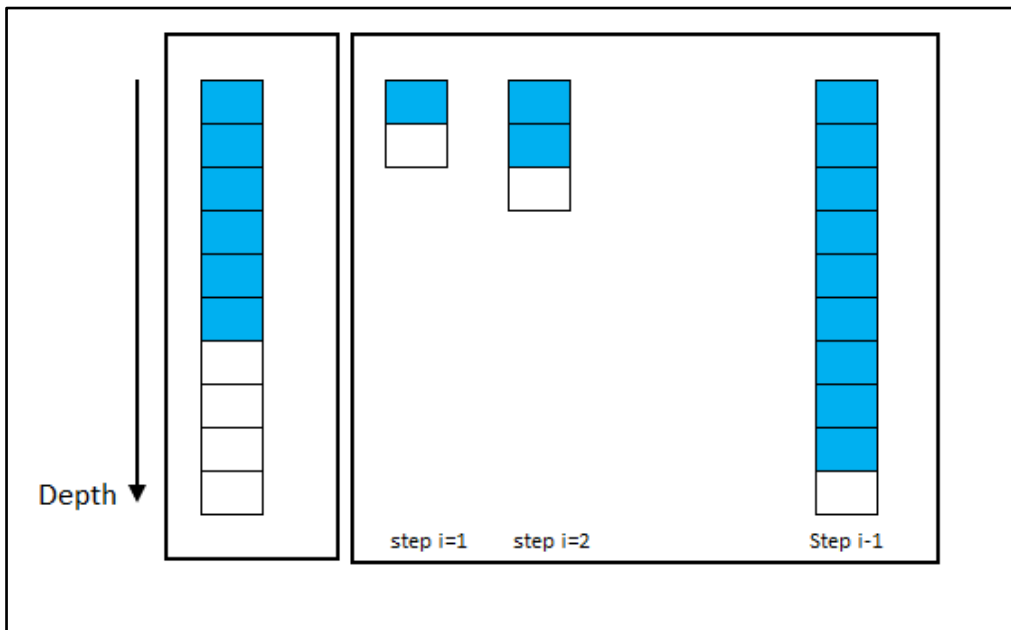


Figure 14: Sequential split (left) and continuous learning split (right). Blue cells represent the training set and white cells represent the test set.

4.2. Performance metrics

To evaluate the performance of each regression model, three different metrics have been chosen: Coefficient of Determination (R^2), Mean Absolute Error (MAE), and standard deviation (std). Since in this study two models were implemented, for each model the same metrics were calculated. These three metrics are available in Python Scikit Library and therefore are very easy to implement in Jupyter Notebooks. These metrics are summarized below:

Coefficient of Determination (R²)

According to many economists and statistics, R² represents the percentage of variation in the dependent variable explained by variation in the independent variables [54]. This can be described by the following formula:

$$R^2 = 1 - \sum_{i=1}^n \frac{(y_i - y_{pi})^2}{(y_i - \bar{y})^2} \quad (59)$$

Where y_i is the real value of the variable that is being modelled (in this case ROP or Torque), \bar{y} is the mean value, and y_{pi} is the predicted value.

Mean Absolute Error (MAE)

Mean Absolute Error is the measured error between predicted values and real values of a predetermine variable. It is calculated as follows:

$$MAE = \frac{\sum_{i=1}^n |y_i - y_{pi}|}{n} \quad (60)$$

Standard Deviation (std)

According to [55], standard deviation measures the dispersion of a dataset relative to its mean. The formula is as follows:

$$std = \sqrt{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n - 1}} \quad (61)$$

4.3. Regressors Implementation

As was mentioned before, all the four ML models were implemented using Scikit Learn package. Although the ideal approach would be to perform hyperparameter tuning when creating all models, the technique was not applied due to run time consumption. A trial using GridSearchCV command from Scikit Learn was performed and it was discarded since this study's goal is not focus on the model's accuracy but on ROP and Torque optimization. Moreover, this parameter tuning took much computer running time with meager improvements. Fig. 15 shows a schematic flow-chart of the process, obtaining the sample that will be used later for modelling. In this case, as it was explained, continuous learning approach is adopted. Both ROP and Torque models were

implemented at the same time, and the process was repeated four times, one for each Regressor. Heat maps of MAE are included in results chapter for a better understanding.

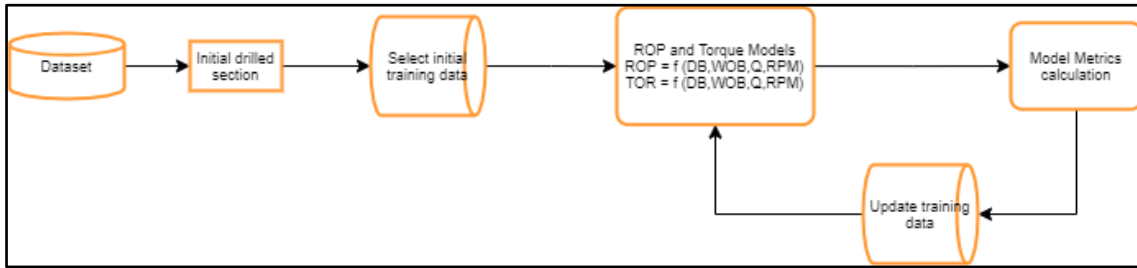


Figure 15: Flow-chart of the continuous learning approach in ROP and Torque modelling.

Once data was selected, it was necessary to create the ML models and evaluate their performance by calculating the metrics. In results chapter, some plots were printed to illustrate ML implementation. Fig. 16 shows a schematic flow-chart of the process, which is very similar to figure 15, but in this case using sequential split and thus, there is no model update since the process is not iterative. Here, ROP and Torque models are created simultaneously, and the process is repeated for each model. As it can be seen in fig. 11, in parts of the plot, RPM and flow rate are constant or there is little variation; thus ML models are not accurate. Only 26" sections are shown in the appendix in modelling results, as the other section models' accuracy was very low.

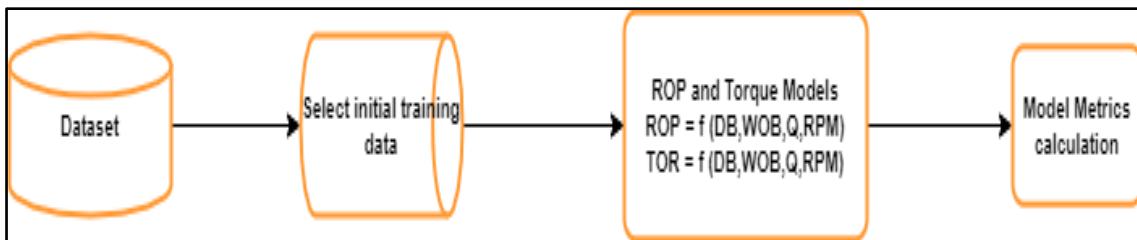


Figure 16: Flow-chart of the sequential split approach in ROP and Torque modelling.

The regressors were implemented with no tuning and using the default parameters, and the most relevant are mentioned below:

Random Forest:

- `n_estimators`: number of trees in the forest. Default: 100
- `max_depth`: maximum depth of the tree. Nodes are expanded until all leaves contain less than `min_samples`
- `max_depth_split`: minimum number of samples to split a node. Default: 2.

Gradient Boosting:

- loss: Loss function to be optimized. Default is Least square regression ('ls')
- n_estimators: number of boosting stages to perform. Default is 100
- min_samples_split: minimum number of samples required to split an internal node. Default: 2

AdaBoost:

- n_estimators: maximum number of estimators at which boosting is terminated. Default=50.
- loss: loss function to update the weights after each boosting iteration. Default: 'linear'.

K-Nearest Neighbors:

- n_neighbors: number of neighbours to use for k-neighbors queries. Default: 5
- weights: weight function. Default: 'uniform'.
- algorithm: Default: 'auto'.

5. Modelling and Optimization Implementation

In this chapter, the optimization methodology will be explained. Once the modelling was implemented and all metrics were analysed, well number 1 section 26” was chosen due to being the only one with optimistic results in both models. From this section, only the top 60% was selected because of the rock lithology. At 900 m, the bit reached a different formation, so for this part of the study only a shale section of 500 m is considered.

As said before, optimization was carried out along with modelling, where the model is updated each fixed-step and optimization is performed in the next step. As Tunkiel [28] says, continuous learning is a good approach since “it does not require information from reference wells” and “do not suffer problems like difference in equipment used between the wells and changes in logged attributes.”

For a better understanding of this approach, the method's representation is shown in figure 14, where each blue cell represents the training sample and the white cell test sample. To start optimization, a small portion of the dataset for the first training set is needed. This work starts with 10% of the training data, which corresponds to approximately 80 meters of drilled rock. A drilling step of 30 meters is considered based on the length of a joint/stand [32]. Subsequently, new data is gathered and the ML model is updated. This process is iterated until step n-1, where n is the number of iterations the algorithm is run, where the bottom of the optimization selected data is reached. This loop is repeated for each ML technique and for each optimizer, having eight different results to compare. Optimization flow-chart is shown in fig. 17 and relevant part of the code is written in the appendix.

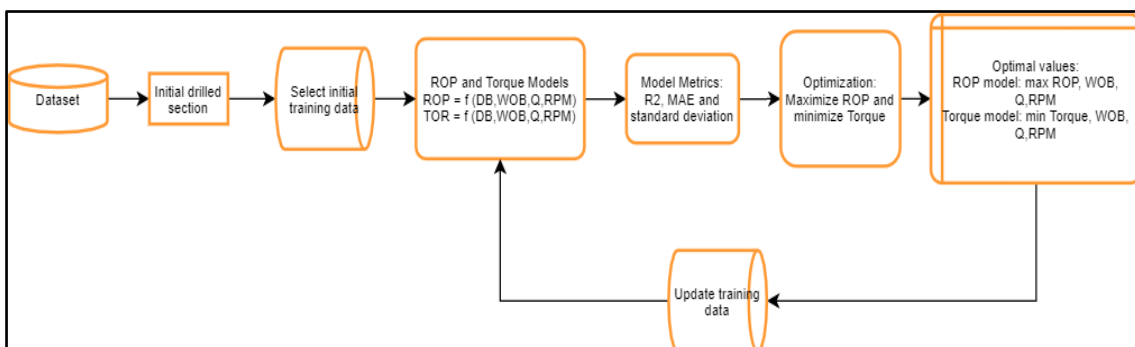


Figure 17: Modelling and Optimization flow-chart

5.1. Optimization Algorithms and constrains

Two optimization algorithms were used in this work: Differential Evolution (DE) and Particle Swarm Optimization (PSO). As described in chapter 2, both are part of metaheuristic optimization algorithms and population-based methods (for more information check Luke's book, pages 54-58 [35]). The implementation of these optimizers is straightforward since both can be found on the internet. However, parameters tuning differs in both cases.

On the one hand, the application is quite simple for DE algorithm as it is included in Scipy library. For the implementation, no parameter tuning was executed and, default values were used. The most relevant parameters are the following:

- maxiter (maximum number of generations): 1000
- popsize (population size): 15
- tol (Relative tolerance for convergence): 0.01
- mutation: 0.5
- recombination (crossover): 0.7

On the other hand, PSO algorithm was implemented using one of the optimizers included in PySwarms library. The optimizer chosen was the global-best Particle Swarm Optimizer, where every particle compares itself with the best-performing particle in the swarm (more details of the code are in the appendix). Due to its nature, PSO needed two functions to be executed. The most relevant parameters are tuned as follows:

- c1 (proportion of the original velocity retained): 0.5
- c2 (proportion of the personal best (location) to be retained): 0.3
- w (proportion of the informants' best location to be retained): 0.9
- n_particles (number of particles): 10
- iters (maximum number of iterations): 100
- dimensions (number of variables, in this case WOB, Q, RPM): 3

In both cases, bounds are the constrain of the problem. It is vital to mention that in this study, hole cleaning and bit vibrations are not considered; therefore, the range of the constraints is wide. For the optimization, the bounds selected were the range of the three variables in the section (WOB, Q, RPM). The ML models used for optimization have four inputs (DB, WOB, RPM, Q) so, when optimizing, it was necessary to give a fix depth value to the function. Consequently, the depth centre of the predicted 30-meter section

was chosen for this purpose. The three controllable parameters bounds were defined as follows:

$$WOB [kN] = [10, 200]$$

$$Q [L/min] = [3500, 5000]$$

$$RPM [rev/min] = [70, 180]$$

6. Results and Discussion

6.1. Modelling Results

As mentioned in Chapter 4, the modelling was performed in two steps. First, a continuous learning split approach was completed to understand data. To illustrate this method, a heat map was chosen, where each Machine Learning performance was evaluated using MAE. From the author's point of view, MAE should decrease while the model gathers more data, and based on this hypothesis, the goal is to find the percentage of dataset that makes this possible. Figures 18 and 19 show ROP and Torque MAEs for each model. The x-axis represents the percentage of the dataset that is evaluated (training set + test set). The model starts with 10% of initial training data and 10% of test data. Then, the model is updated, and 20% of the dataset represents the training data, and the following 10% is the test data. This process is iterated until 90% of the data is trained. Only relevant part of the code is included in the appendix for more information. For heat map colour representation, maximum values in the scale are 10 m/h and 3 kN.m since they represent 15% of the maximum ROP and Torque values.

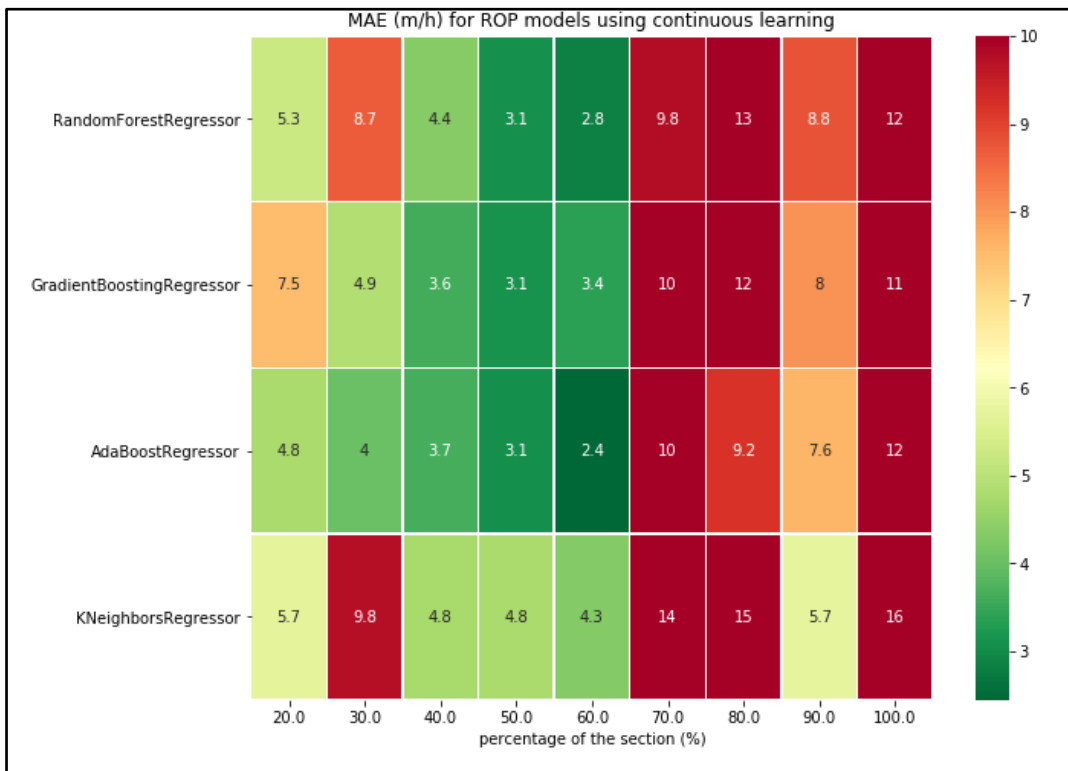


Figure 18: ROP models' MAEs using continuous learning approach

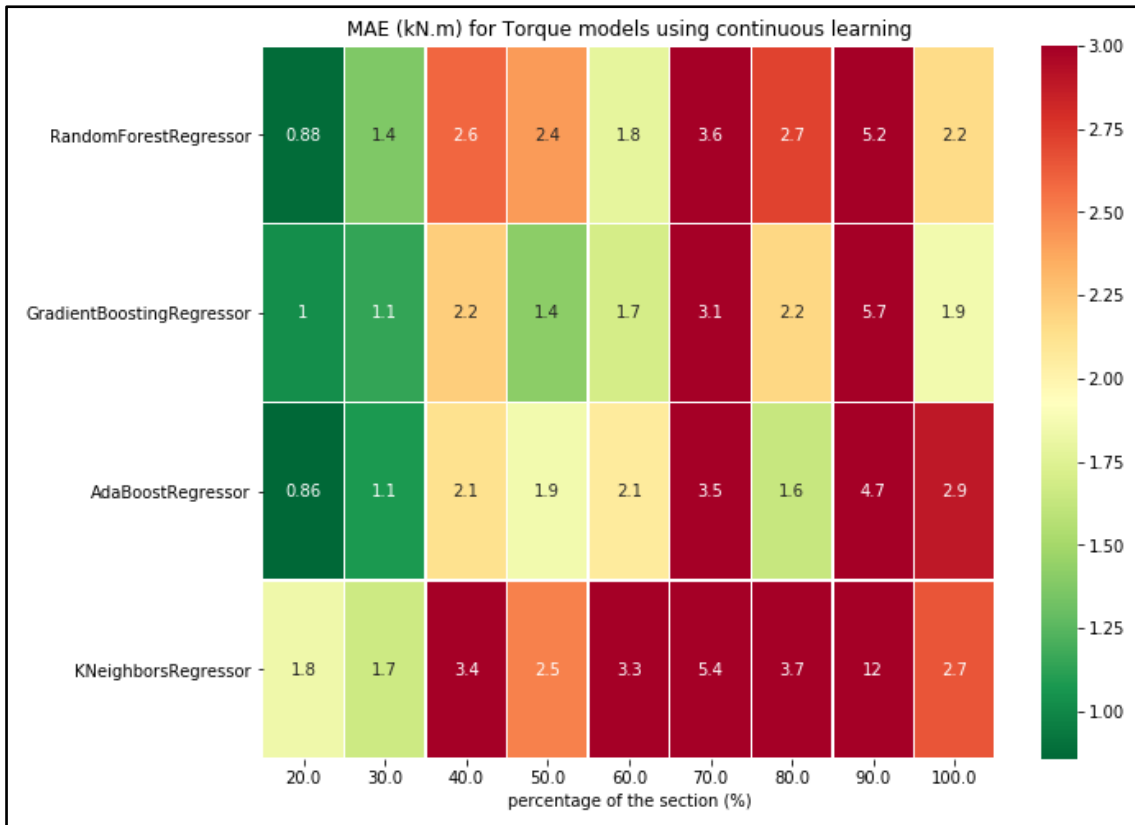


Figure 19: Torque models' MAEs using continuous learning approach from one of the well sections

Afterwards, 60% of the dataset was selected, and the same four ML algorithms were implemented using a sequential split approach in all wells (only plots from well 1 section 26" are shared in this chapter since they are the ones with the best performance metrics). Tables 4 and 5 show the performance metrics evaluated in the different models, yielding some conclusions along with figures 20 and 21. Here it can be seen how good is the prediction using Gradient Boosting algorithm, where test data is compared with predicted data for ROP and Torque models. Since test data in the Torque model is not in the same range as training data (training data is between 1 and 13 kN.m and test data is between 1 and 17 kN.m) the model performance is not quite good but still catch abrupt changes.

| | MAE(kN.m) | R2 | std(kN.m) | | MAE(m/h) | R2 | std(m/h) |
|---------------------------|-----------|----------|-----------|---------------------------|----------|-----------|----------|
| RandomForestRegressor | 2.257744 | 0.509748 | 2.760554 | RandomForestRegressor | 3.831430 | 0.336529 | 5.266654 |
| GradientBoostingRegressor | 1.840149 | 0.680473 | 2.904550 | GradientBoostingRegressor | 3.444714 | 0.466474 | 5.111127 |
| AdaBoostRegressor | 2.253176 | 0.544432 | 2.916015 | AdaBoostRegressor | 2.722208 | 0.616841 | 4.609254 |
| KNeighborsRegressor | 1.608439 | 0.713093 | 2.937487 | KNeighborsRegressor | 5.093230 | -0.102962 | 6.263865 |

Table 4 and Table 5: Performance metrics for Machine Learning models using 60/40 split (Torque model on the left and ROP model on the right) from well 1 section 26".

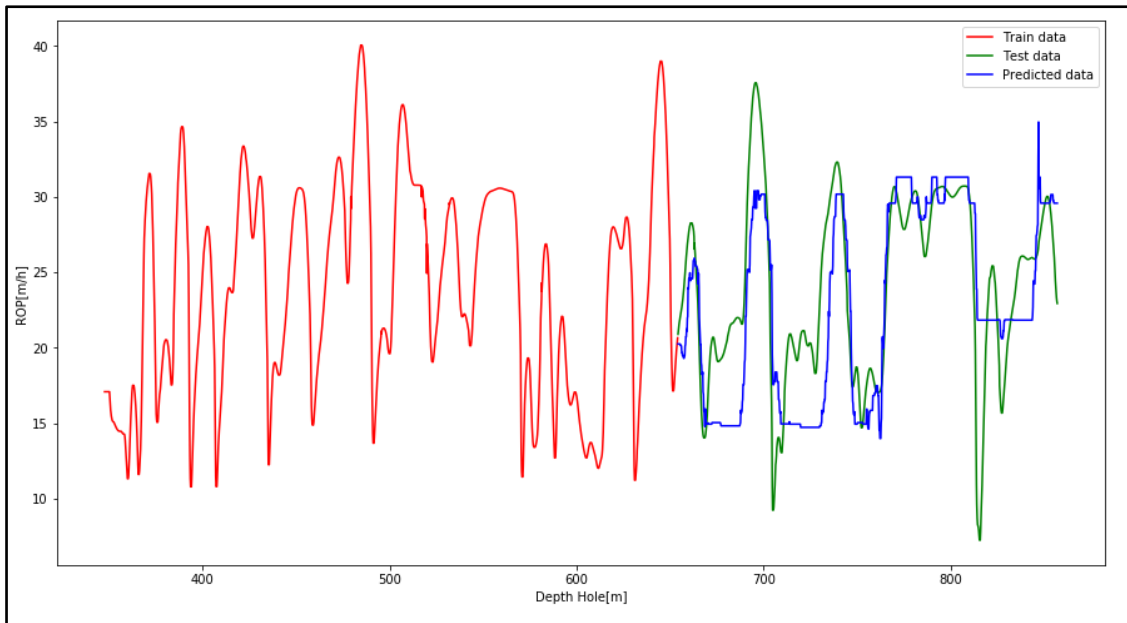


Figure 20: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well1 section 26". Predicted data (blue) is evaluated against test data (green).

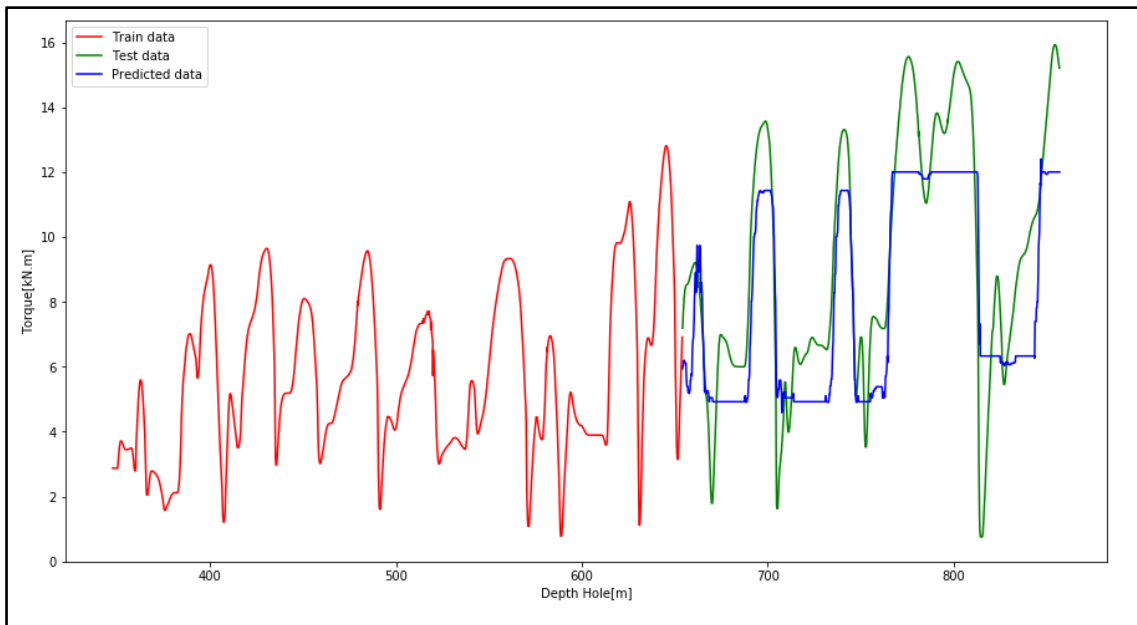


Figure 21: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well1 section 26". Predicted data (blue) is evaluated against test data (green).

In the case of the ROP model, on the other hand, training and test data are in the same range and have an acceptable prediction for optimization purposes.

6.2. Modelling and Optimization Results

In this section, results from part 3 of the study will be exposed. As was mentioned in section 2.3, two optimization algorithms were implemented in this study using continuous learning split approach to simulate drilling conditions. With this technique, only a tiny part of the dataset is known (10%, which is approximately 80 meters), and the model is updated every 30 meters as it is the length of a drill string stand. Therefore, with the parameters mentioned in Chapter 5, the two optimizers were executed using the same four ML models. From left to right in figure 22, the code runtime is illustrated using Random Forest, Gradient Boosting, AdaBoost and K-Nearest Neighbors algorithms.

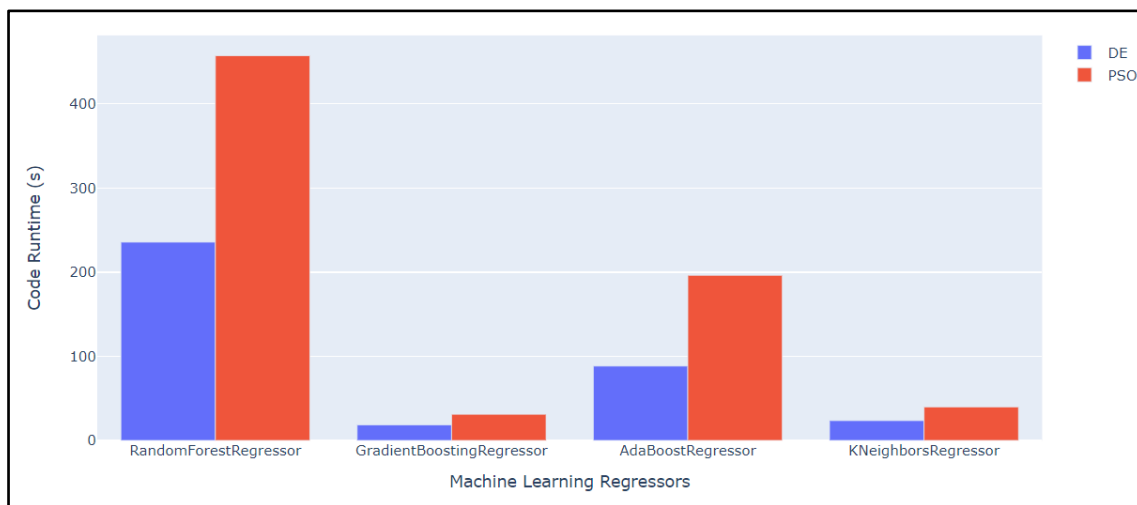


Figure 22: Code Runtime (seconds) for the two optimizers (Differential Evolution in blue and Particle Swarm in red) and using RF, GB, AB, and KN with the same parameters.

It is essential to mention that runtime corresponds to the total time of the process, which is modelling update and optimization of all the selected section. Furthermore, the number of iterations using the PSO algorithm is 100, while DE used 1000. This is not relevant since PSO converged to the solution before 100 iterations, but it was modified because using 1000 iterations with PSO was very time-consuming. The most pertinent analysis is that run time is considerably higher and is more affected by the ML model than optimizer, although DE is faster. As a result, only GB and KN show the most rapid optimization.

Afterwards, Hydromechanical Specific Energy (HMSE) was implemented and calculated for all the section. HMSE and Torque were calculated for all ROP models, using the Torque model's prediction and HMSE equation from Chapter 2.4. Alternatively, the ROP and HMSE were calculated using ROP model's prediction and HMSE equation

for Torque models. It is important to mention that values of the potential core length (L), the distance between the nozzle to the formation (s), and the angle of axially symmetric jet were assumed as 156 in, 8 in, and 3 degrees, respectively, based on values used in Berg's study [56]. The constant k has the same value as in Mohan's paper [6] (k=0.122). First, to avoid confusion, the term growth rate as the percentage change of any variable in terms of the average of the optimized 30-meter section will be defined. The growth rate equation is as follows:

$$GR = \frac{x_{opt} - x_{av}}{x_{av}} \quad (62)$$

Where x_{opt} is the optimal value of ROP, Torque or HMSE and x_{av} the average value of same parameters in the 30-meter section. Eventually, growth rate was calculated for the three parameters in both models to evaluate the optimization. Figures 23 and 24 highlight growth rates using Gradient Boosting Torque model and AdaBoost ROP model, respectively. Values were achieved by adopting Differential Evolution algorithm since it is faster. More plots are available in the appendix.

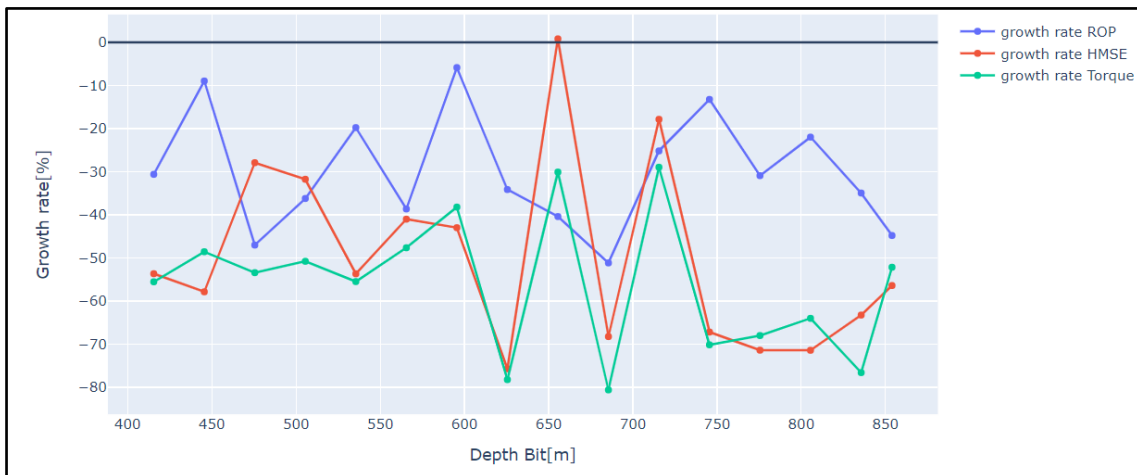


Figure 23: ROP (blue), HMSE (red) and Torque (green) growth rate using GB Torque and ROP model and DE combination. Growth rate is negative because the three parameters decrease.

Finally, a bar chart was created to show the three variables' average growth rate using the four ML models and DE algorithm. Based on these plots, there is an average growth rate nearly 48% when maximizing ROP. However, this approach raises the HMSE closer than 50%, which is not desirable. On the other hand, when minimizing Torque, HMSE decreases on average 50% but ROP also drops but only 20%.

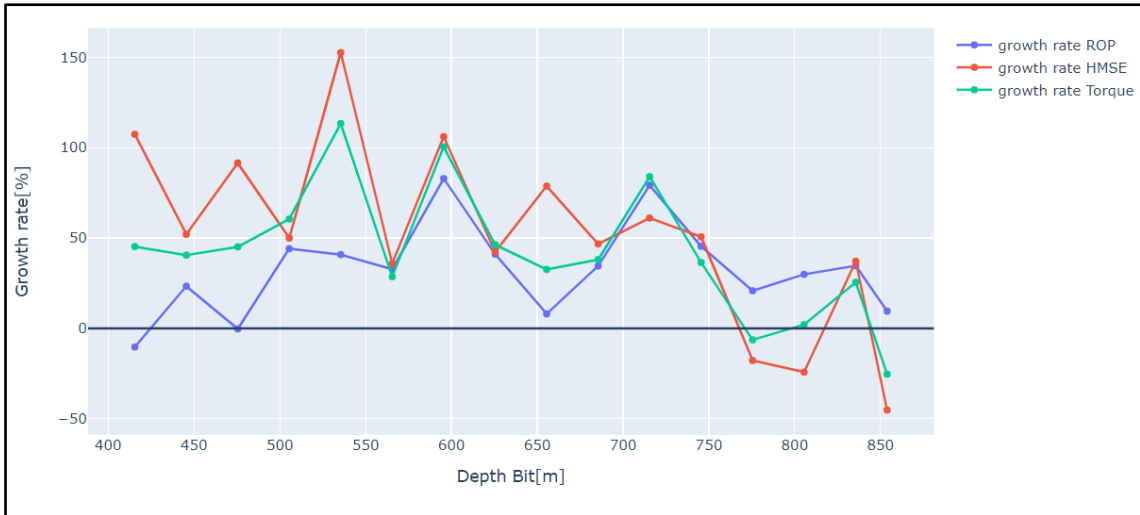


Figure 24: ROP (blue), HMSE (red) and Torque (green) growth rate using AB ROP and Torque ROP model and DE combination.

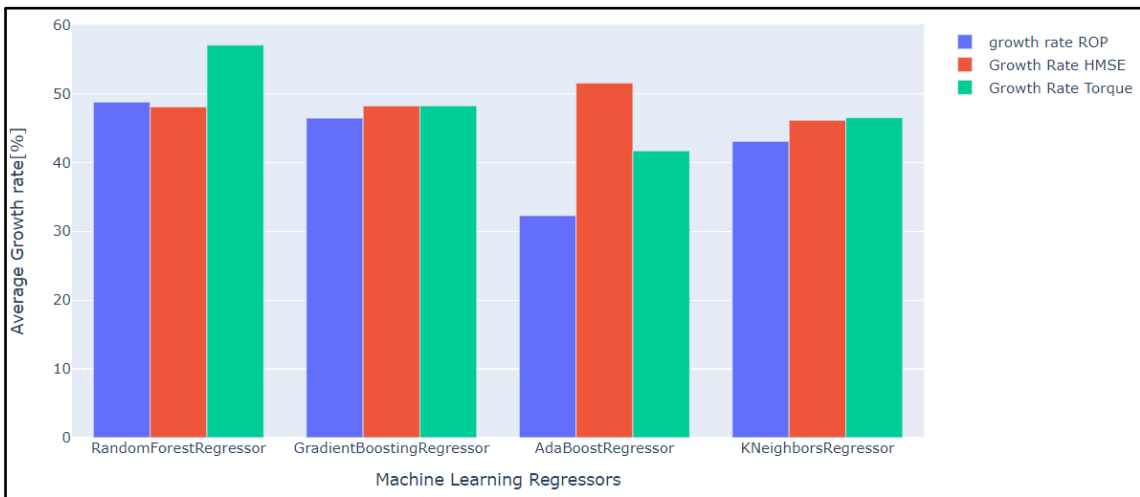


Figure 25: ROP (blue), HMSE (red), and Torque (green) average growth for all ML models for ROP optimization (maximize ROP) using DE algorithm.

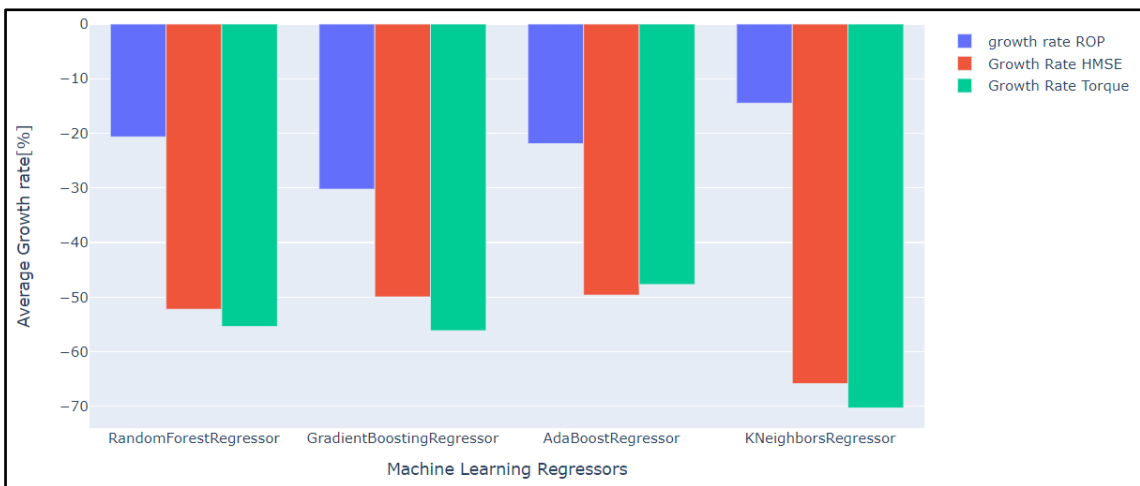


Figure 26: ROP (blue), HMSE (red), and Torque (green) average growth for all ML models for Torque optimization (minimize Torque) using DE algorithm.

6.3. Sensitivity Analysis

In the last part of this study, a sensitivity analysis was conducted to evaluate Machine Learning algorithms' performance. Since it was impossible to perform a field experiment for model evaluation, a sensitivity analysis is an alternative technique to evaluate how sensitive the ML models are to variations on its parameters. Two different techniques were used in this research. Consequently, the four ML models were implemented using default parameters described in Chapter 4 using a 60/40 data split approach and DE as optimizer. Modelling and optimization were performed once per model, and optimal parameters were obtained from the process (maximum ROP and optimal WOB, Q and RPM for the ROP model and minimum Torque and optimal WOB, Q and RPM for the Torque model). The bit depth was set at the centre of the test section. In the first case, each model is evaluated using the optimal parameters from the other models. For instance, when ROP is calculated using the optimal values from RF, ROP using GB model and RF optimal features is calculated. Also, the same process with ROP using AB model and KNN model is repeated. This process is then replicated for the three remaining algorithms. Figures 27 and 28 show a bar chart for Torque and ROP models where this analysis is executed. Here, 4 groups of 4 bars are represented. Each group corresponds to the sensitivity analysis of the ROP using the same optimal parameters from the model indicated in x-axis. For example, in Fig. 6.10 the first group conforms to the Torque values of the four ML models using optimal parameters from Random Forest.

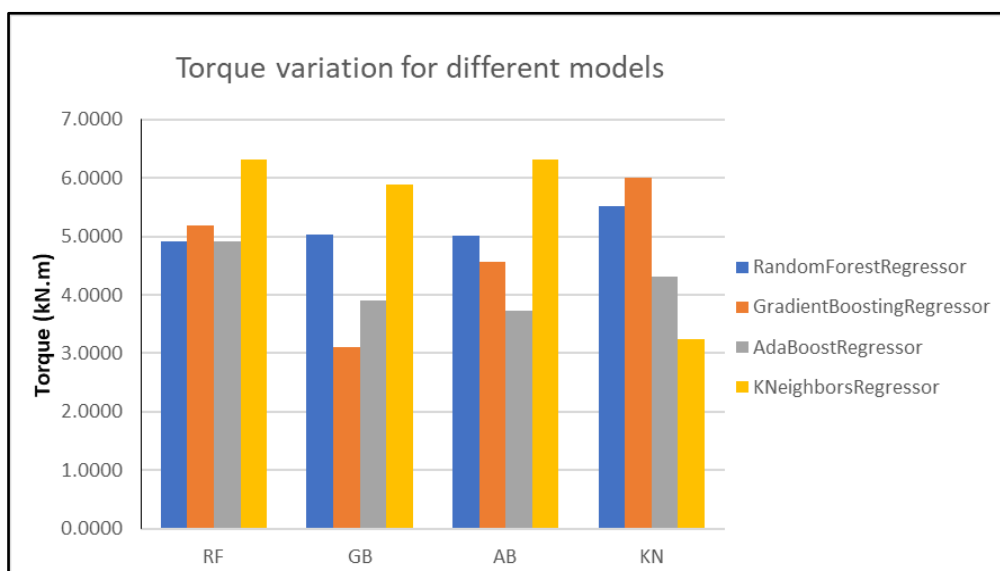


Figure 27: Sensitivity analysis case 1 for Torque models. The x-axis represents the model whose optimal parameters are implemented.

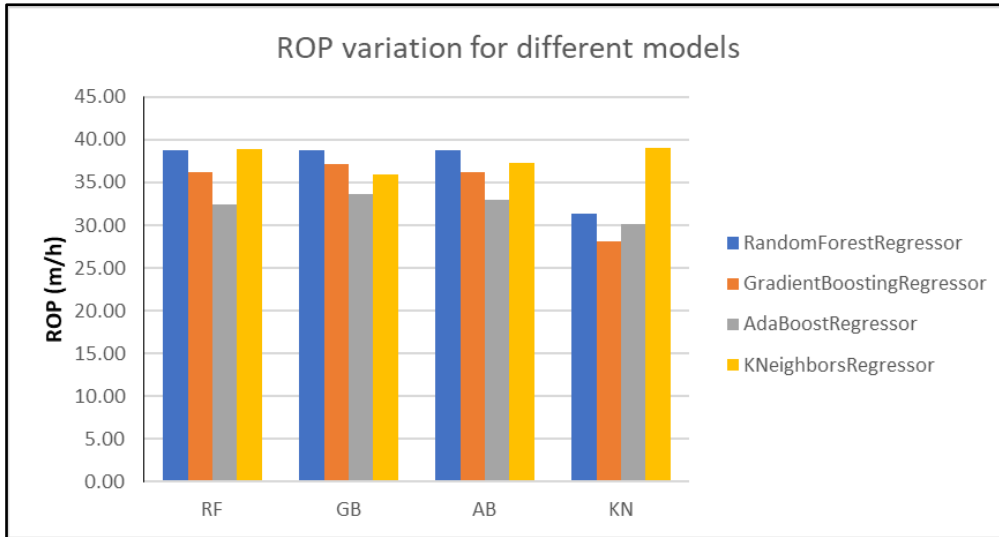


Figure 28: Sensitivity analysis case 1 for ROP models. The x-axis represents the model whose optimal parameters are implemented.

From figures 27 and 28, KNN Torque model is very sensitive to any parameter variation, and RF is not affected by WOB, which is strange. GB and AB are very influenced by parameter changes, but they will be analysed with more detail doing the case 2 analysis. From ROP models, it is clear that WOB plays a relevant role.

In the second case, each model was individually tested by a variation of 20% of its input parameters (Bit Depth, WOB, Q, and RPM), based on optimal values. For instance, GB Torque model was evaluated first by varying only Depth Bit in -20% and 20%. Therefore, it can be assessed how sensitive each model is to the variation of all input parameters. Figures 29 and 30 illustrate this analysis.

From figures 29 and 30, it can be deduced that KNN model is susceptible to small fluctuations, both ROP and Torque model, which makes it no reliable. AB and GB ROP models are unaffected by WOB and bit position variations, which is not associated with the theory. RF ROP model is also unaffected by bit position variations.

On the other hand, all Torque models are very receptive to bit position and rotary speed changes, while they are no responsive to WOB changes.

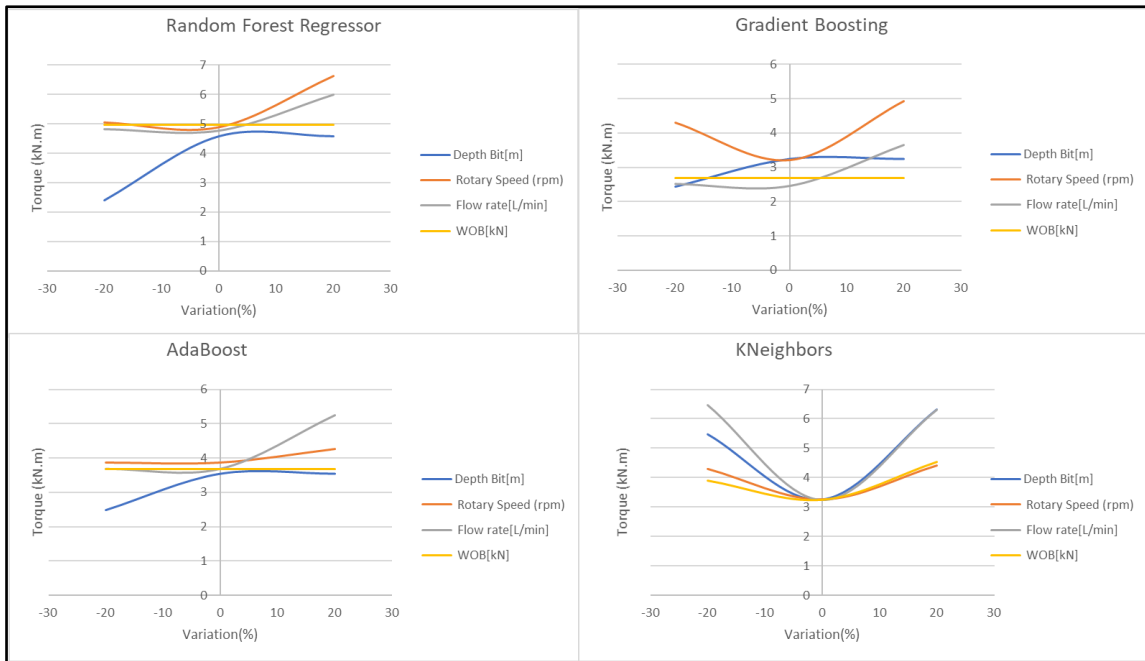


Figure 29: Sensitivity analysis case 2 for Torque models.



Figure 30: Sensitivity analysis case 2 for ROP models.

Lastly, it is crucial to indicate the nature of all ML models, summarized in Chapter 2 Section 4. RF, GB, and AB are based on decision trees, and prediction could have the shape of figure 2 or figure 21, where the predicted variable is sometimes flat, so the input parameter variation does not change the expected value. Alternatively, KNN has no explicit training process, so it is unstable and changes by minor input variations.

6.4. Discussion

The goal of this study is to maximize ROP and/or minimize Torque in real-time. Therefore, the most crucial part of the study is the optimizer performance and runtime, leaving the model development as a second priority. Consequently, this causes that ML models were not as accurate as in other studies. This discussion will focus on three different aspects that were not covered or not successfully achieved by this study.

The first part is the use of deep learning. It is important to remark that the hardware availability (a personal computer with a simple GPU) limited the author in implementing deep learning models. The reason is that deep learning requires a lot of computation. Ahmed, Al-AbdulJabbar, Alali, Brenjkar, and Manta [21] [23] [4] [22] [24] obtained very accurate models using ANN, all with R^2 values over 0.9, which shows how proficient are these ML algorithms for ROP modelling. Encinas and Tunkiel [44] [27] also achieved remarkable ROP prediction using RNN. Thus, using cloud computing, the model development should improve even knowing that it is necessary to update each model for every 30 meters during optimization. The solution proposed in this model is for a simple computer that is usually available in rig operations.

Second part is regarding data quality. When using physics-based ROP models, it is essential to use downhole measurements or downhole corrections. Otherwise, it will produce wrong results. In fact, Encinas [44] showed on his model that the best approach is using a mix of surface and downhole WOB values as inputs for his long short-term memory (LSTM) model. For instance, setting downhole RPM as input in 26" sections instead of surface RPM improved the models. So, calculating downhole Torque and WOB would have undeniably improved ML model development.

Finally, the last topic is regarding the constraints during ROP and Toque optimization. Hedge [7], for instance, included drilling vibrations as a constrain to "ensure that calculated optimal drilling parameters do not induce excessive vibrations." Therefore, additional analysis regarding drilling vibrations, hole cleaning, borehole stability, and pressure variation must be performed to achieve more realistic optimal drilling parameters.

7. Conclusions and Future Work

After completing this study, a real-time data driven and optimization model was achieved using Johan Sverdrup wells. The most important conclusions are summarized below:

A dataset of 6 wells was analysed, with 16 different features. To select the appropriate section was necessary to clean and pre-processed all data.

A continuous learning approach was successfully implemented to select data in the same range in all well sections during model development. Surface data such as controllable parameters like WOB, flow rate and rotary speed and also bit position were used as input in both ROP and Torque models. MAE was calculated and used as filter to measure model's performance. Random Forest (RF), Gradient Boosting (GB), K-Nearest Neighbours (KNN) and AdaBoost ML algorithms were tested.

A sequential split approach was executed using 60/40 split to select the well sections to optimize. Good results were achieved only in well 1 section 26" using WOB, flow rate, RPM, and bit position as inputs for both models and testing RF, AB, GB, and KNN models.

Modelling and optimization were auspiciously completed using the continuous learning approach, updating the model every 30 meters, and predicting and optimizing the next stand. The four ML models were implemented (RF, GB, AB, and KNN) among the two algorithms of stochastic optimization: Differential Evolution (DE) and Particle Swarm Optimizer (PSO). In total, 8 combinations were tested, and the mix of GB and DE was the best based on the metric features and code running time. During optimization, the centre of the predicted next stand was used as input, and the controllable drilling parameters constraints were chosen only to fit in the data range.

Finally, a sensitivity analysis was performed to evaluate Machine Learning algorithms' performance to substitute a field experiment validation. Two scenarios were simulated, one considering each model using the optimal parameters from the other models and the other testing each model by a variation of 20% of its input parameters. Results showed strong dependence on WOB in ROP models and high sensitivity of Torque models in bit position and rotary speed. This analysis also demonstrates the four ML models limitations since three are based on trees (see Chapter 2 section 4, RF, AB and GB theory) and KNN has no explicit training process, therefore its sensitivity to all inputs variations.

This work aims to create a methodology for drilling optimization, either by maximizing ROP or minimizing HMSE (by minimizing Torque) in real-time using data driven models, modifying controllable surface parameters. As mentioned in the discussion, many limitations will be outlined below as future work:

1. Previous studies [21] [23] [4] [22] [44] [27] revealed that Neural Networks achieve remarkable results when using in ROP prediction. The limitation in this model was the hardware used and this problem could be solved using cloud computing. Thus, a more accurate model will reach better predictions and consequently more realistic optimization.
2. Data quality is of vital importance since this study is a data driven approach. Only rotary speed was corrected from surface, and with WOB and Torque corrections the model accuracy would definitively improve.
3. No specific constraints were selected during optimization, which would make a more realistic and complete model. These bounds can be modified based on drilling vibrations, hole cleaning, borehole stability, and pressure variation among other drilling issues.

8. References

- [1] "Investments and operating costs," 2021. [Online]. Available: <https://www.norskpetroleum.no/en/economy/investments-operating-costs/>.
- [2] "Drilling production wells," [Online]. Available: <https://www.npd.no/en/facts/publications/reports2/resource-report/resource-report-2017/recovery/adopt-new-technology/drilling-production-wells/#>.
- [3] C. Soares, H. Daigle and K. Gray, "Evaluation of PDC bit ROP models and the effect of rock strength on model coefficients," *Journal of Natural Gas Science and Engineering*, vol. 3, pp. 1225-1236, 2016.
- [4] A. M. Alali, M. F. Abughaban and B. M. Amanc, "Hybrid Data Driven Drilling and Rate of Penetration Optimization," *Journal of Petroleum Science and Engineering*, 2020.
- [5] C. Hegde, H. Daigle, H. Millwater and K. Gray, "Analysis of rate of penetration (ROP) prediction in drilling using physics-based and data-driven models," *Journal of Petroleum Science and Engineering*, pp. 295-306, 2017.
- [6] K. Mohan, F. Adil and R. Samuel, "Comprehensive Hydromechanical Specific Energy Calculation for Drilling Efficiency," 2014.
- [7] C. Hegde, M. Pyrcz, H. Daigle and K. Gray, "Fully coupled end-to-end drilling optimization model using machine learning," *Journal of Petroleum Science and Engineering*, 2019.
- [8] D. Ertas, J. R. Bailey, L. Wang and P. E. Pastusek, "Drillstring Mechanics Model for Surveillance, Root Cause Analysis and Mitigation of Torsional Vibrations," *SPE Drilling and Completions*, 2014.
- [9] G. Bingham, A new approach to interpreting rock drillability, Tulsa: Petroleum Publishing Company, 1965.
- [10] A. T. Bourgoyne, K. K. Millheim, M. E. Chenevert and F. S. Young Jr., Applied Drilling Engineering, Second Edition ed., vol. 2, Richardson, TX: SPE textbook series, 1991.

- [11] W. W. Winters and E. Onyia, "Roller Bit Model With Rock Ductility and Cone Offset," in *62nd SPE Annual Technical Conference and Exhibition*, 1987.
- [12] H. Motahhari, G. Hareland and J. James, "Improved Drilling Efficiency Technique Using Integrated PDM and PDC Bit Parameters," *Journal of Canadian Petroleum Technology*, 2010.
- [13] L. M. Leao de Barros, "ROP Modeling Chronology, Sensitivity Analyses, and Field Data Comparisons," 2015.
- [14] J. VanderPlas, *Python Data Science Handbook*, O'Reilly Media, Inc, 2016.
- [15] Z.-H. Zhou, "Ensemble Learning," in *Encyclopedia of Biometrics*, Boston, Springer, 2009.
- [16] N. Bagalkot, A. Keprate and R. Orderlø, "Combining Computational Fluid Dynamics and Gradient Boosting Regressor for Predicting Force Distribution on Horizontal Axis Wind Turbine," *MDPI*, vol. 4, pp. 248-262, 2021.
- [17] P. Prettenhofer and G. Louppe, "Gradient Boosted Regression Trees".
- [18] G. Biau, "Analysis of a Random Forests Model".
- [19] M. Yoo, "Variable Importance Assessment in Regression: Linear Regression versus Random Forest," *The American Statistician*, vol. 63, no. 4, pp. 308-319, 2009.
- [20] Z.-H. Zhou, *Ensemble Methods*, Taylor & Francis Group, 2012.
- [21] A. Ahmed, A. Ali, S. Elkatatny and A. Abdulraheem, "New Artificial Neural Networks Model for Predicting Rate of Penetration in Deep Shale Formation," *Sustainability*, vol. 11, no. 6527, 2019.
- [22] E. Brenjkar and E. K. K. Biniaz Delijani, "Prediction of penetration rate in drilling operations: a comparative study of three neural network forecast methods," *J Petrol Explor Prod Technol*, vol. 11, p. 805–818, 2021.
- [23] A. Al-AbdulJabbar, S. Elkatatny, A. A. Mahmoud, T. Moussa, D. Al-Shehri, M. Abughaban and A. Al-Yami, "Prediction of the Rate of Penetration

while Drilling Horizontal Carbonate Reservoirs Using the Self-Adaptive Artificial Neural Networks Technique," 2020.

- [24] B. Mantha and R. Samuel, "ROP Optimization Using Artificial Intelligence Techniques with Statistical regression Coupling," in *SPE Annual Technical Conference and Exhibition*, Dubai, UAE, 2016.
- [25] C. I. Noshi, "Application of Data Science and Machine Learning Algorithms for ROP Optimization in West Texas: Turning Data into Knowledge," in *Offshore Technology Conference*, Houston, Texas, 2019.
- [26] K. Singh, S. S. Yalamarty, M. Kamyab and C. Cheatham, "Cloud-Based ROP Prediction and Optimization in Real Time Using Supervised Machine Learning," in *SPE/AAPG/SEG Unconventional Resources Technology Conference*, Denver, 2019.
- [27] A. T. Tunkiel, D. Sui and T. Wiktorski, "Training-while-drilling approach to inclination prediction in directional drilling utilizing recurrent neural networks," Stavanger, 2020.
- [28] A. T. Tunkiel, D. Sui and T. Wiktorski, "Reference Dataset for Rate of Penetration Benchmarking," 2020.
- [29] C. Gan, W. Cao, K.-Z. Liu, M. Wu, F.-W. Wang and S.-B. Zhang, "A New Hybrid Bat Algorithm and its Application to the ROP Optimization in Drilling Processes," *IEEE Transactions on Industrial Informatics*, 2019.
- [30] E. Wiktorski, A. Kuznetsov and D. Sui, "ROP Optimization and Modelling in Directional Drilling Process," in *SPE Bergen Seminar*, Bergen, Norway, 2017.
- [31] D. Sui and B. S. Aadnøy, "Rate of Penetration Optimization using Moving Horizon Estimation," *Modeling, Identification and Control*, vol. 3, no. 1890-1328, pp. 149-158, 2016.
- [32] C. Hegde and K. Gray, "Evaluation of coupled machine learning models for drilling optimization," *Journal of Natural Gas Science and Engineering*, 2018.

- [33] C. Hegde, H. Daigle and K. Gray, "Performance Comparison of Algorithms for Real-Time Rate-of-Penetration Optimization in Drilling Using Data-Driven Models," *SPE Journal*, vol. 23, pp. 1706-1722.
- [34] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
- [35] S. Luke, *Essentials of Metaheuristics*, Lulu, 2016.
- [36] "A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems," in *Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, 2004.
- [37] "Total Bit Revolution in When Running A Drilling Mud Motor and Rotating At Surface," 2012. [Online]. Available: <https://www.drillingformulas.com/total-bit-revolution-in-when-running-a-drilling-mud-motor-and-rotating-at-surface/>.
- [38] T. Jaeger and K. Doering, *Baker-Hughes-Navi-Drill-motor-handbook-15th-edition-2020*, 2020.
- [39] M. Belayneh, *Advanced Drilling Engineering and Technology*, 2021.
- [40] C. Johancsik, D. Friesen and R. Dawson, "Torque and Drag in Directional Wells-Prediction and Measurement," *J Pet Technol*, vol. 36, p. 987–992, 1984.
- [41] R. Pessier and M. Fear, "Quantifying Common Drilling Problems With Mechanical Specific Energy and a Bit-Specific Coefficient of Sliding Friction," in *SPE Annual Technical Conference and Exhibition*, Washington, D.C., 1992.
- [42] M. Belayneh, "New alternative MSE based ROP Modelling and Analysis with North Sea field Data," *International Journal of Engineering Research and Technology*, vol. 12, no. 10, pp. 1696-1700, 2019.
- [43] G. Hareland and A. Wu, "The Field Tests for Measurement of Downhole Weight on Bit (DWOB) and the Calibration of a Real-time DWOB Model," in *International Petroleum Technology Conference*, Doha, 2014.

- [44] M. Encinas Quisbert, Data Driven ROP Modelling – Analysis and Feasibility Study.
- [45] X. Chen, J. Yang and D. Gao, "Drilling Performance Optimization Based on Mechanical Specific Energy Technologies," *Drilling*, 2018.
- [46] B. S. Aadnoy, M. Fazaelizadeh and G. Hareland, "3D analytical model for wellbore friction," *Journal of Canadian Petroleum Technology*, vol. 49, no. 10, pp. 25-36, 2010.
- [47] E. Wiktorski and D. Sui, "Investigation of Stick-Slip Severity in a Coupled Axial-Torsional Drillstring Dynamics Using a two DOF Finite Element Model," in *ASME 2020 39th International Conference on Ocean, Offshore and Arctic Engineering OMAE 2020*, Fort Lauderdale, Florida, 2020.
- [48] D. Sui, *Drilling Automation and Modeling Compendium*, Springer, 2019.
- [49] R. Teale, "The concept of specific energy in rock drilling," *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts*, vol. 2, no. 1, pp. 57-73, 1965.
- [50] F. E. Dupriest and W. L. Koederitz, "Maximizing Drill Rates with Real-Time Surveillance of Mechanical Specific Energy.," in *SPE/IADC Drilling Conference*, Amsterdam, Netherlands, 2005.
- [51] B. S. Aadnøy, *Modern Well Design*, London: Taylor and Francis Group, 2010, pp. 25-26.
- [52] "Equinor Johan Sverdrup Field," [Online]. Available: <https://www.equinor.com/en/what-we-do/johan-sverdrup.html>. [Accessed 17 March 2021].
- [53] S. A. Hadi, 19 January 2020. [Online]. Available: <https://www.r-bloggers.com/2020/01/how-to-remove-outliers-in-r/>.
- [54] D. Figueiredo, S. Júnior and E. Rocha, "What is R2 all about?," *Leviathan*, vol. 3, pp. 60-68, 2011.
- [55] M. Hargrave, "Standard Deviation," 15 April 2021. [Online]. Available: <https://www.investopedia.com/terms/s/standarddeviation.asp>.

- [56] P. V. Berg and Ø. S. Tveit, "Model for evaluating drilling efficiency based on the Concept of Mechanical Specific Energy," Norwegian University of Science and Technology, 2016.
- [57] M. Bataee and S. Mohseni, "Application of Artificial Systems in ROP Optimization: a Case Study in Shadegan Oil Field," in *SPE Middle East Unconventional Gas Conference*, Muscat, Oman, 2011.
- [58] Y. Guo, S. Guan and Z. Liu, "ROP Optimization Technology for Poor Drillability Formation in Western South China Sea," in *SPE Middle East Oil & Gas Show and Conference*, Manama, Bahrain, 2017.
- [59] Y. Luo, P. A. Bern and B. D. Chambers, "Flow-rate predictions for Cleaning Deviated Wells," in *SPE/IADC Drilling Conference*, New Orleans, Louisiana, 1992.
- [60] E. E. Okoro, A. O. Alaba, S. E. Sanni and E. B. Ekeinde, "Development of an automated drilling fluid selection tool using integral geometric parameters for effective drilling operations," *Heliyon*, vol. 5, no. 5, pp. 1-10, 9 May 2019.
- [61] C. Soares and G. Kenneth, "Real-time predictive capabilities of analytical and machine learning rate of penetration (ROP) models," *Journal of Petroleum Science and Engineering*, vol. 172, p. 934–959, 2019.

9. Appendix

Results Modelling

| | MAE(kN.m) | R2 | std(kN.m) |
|----------------------------------|-----------|-----------|-----------|
| RandomForestRegressor | 0.680682 | -0.193099 | 0.888280 |
| GradientBoostingRegressor | 0.933953 | -1.061390 | 0.883637 |
| AdaBoostRegressor | 0.664994 | -0.102638 | 0.877376 |
| KNeighborsRegressor | 1.581632 | -4.331757 | 1.242992 |

Table 6: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for Torque models from well 1 section 16".

| | MAE(m/h) | R2 | std(m/h) |
|----------------------------------|-----------|------------|-----------|
| RandomForestRegressor | 7.999631 | -1.042186 | 9.203995 |
| GradientBoostingRegressor | 11.505671 | -3.655793 | 11.628632 |
| AdaBoostRegressor | 5.502389 | 0.029350 | 6.880342 |
| KNeighborsRegressor | 20.393470 | -14.722169 | 16.137512 |

Table 7: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for ROP models from well 1 section 16".

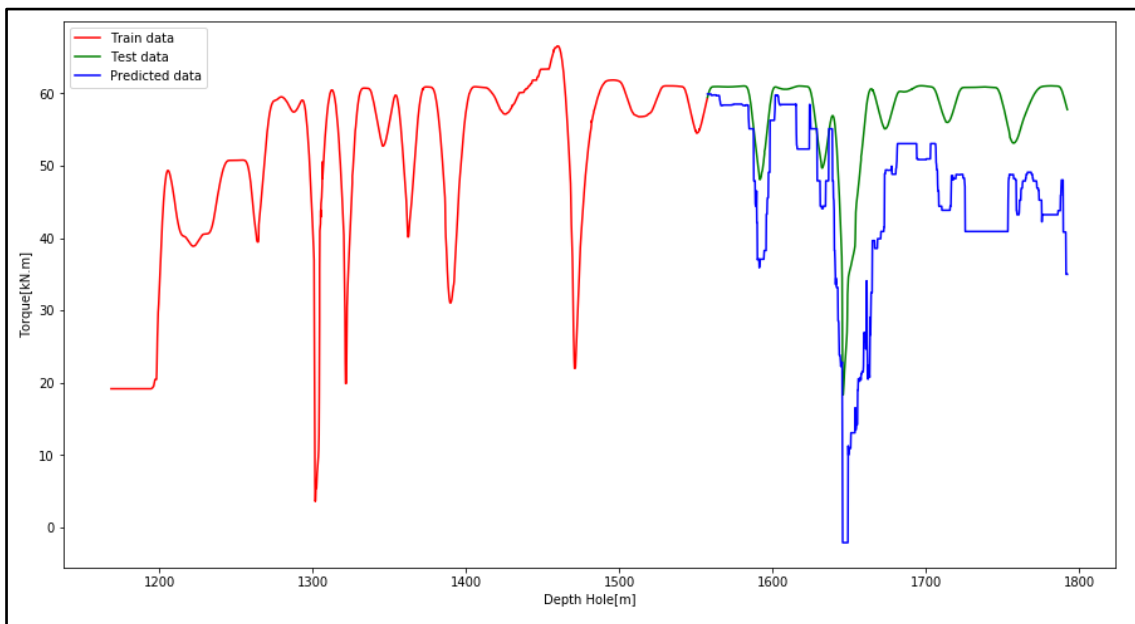


Figure 31: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 1 section 16". Predicted data (blue) is evaluated against test data (green).

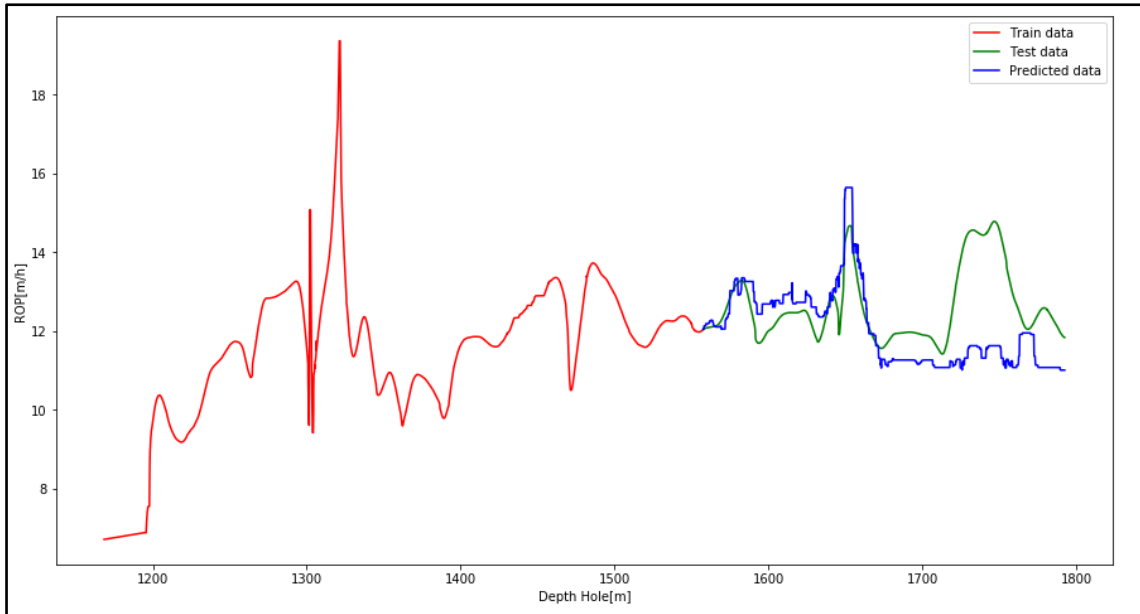


Figure 32: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 1 section 16". Predicted data (blue) is evaluated against test data (green).

| | MAE(kN.m) | R2 | std(kN.m) |
|----------------------------------|-----------|-----------|-----------|
| RandomForestRegressor | 5.346680 | -0.778614 | 4.393865 |
| GradientBoostingRegressor | 4.368783 | -0.202600 | 4.048347 |
| AdaBoostRegressor | 4.532833 | -0.236412 | 4.102374 |
| KNeighborsRegressor | 4.731552 | -0.526818 | 4.090080 |

Table 8: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for Torque models from well 2 section 26".

| | MAE(m/h) | R2 | std(m/h) |
|----------------------------------|----------|-----------|----------|
| RandomForestRegressor | 6.177855 | -0.030271 | 4.721043 |
| GradientBoostingRegressor | 5.884523 | 0.060371 | 4.827522 |
| AdaBoostRegressor | 6.036769 | 0.037804 | 4.448812 |
| KNeighborsRegressor | 5.527145 | 0.111568 | 4.736781 |

Table 9: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for ROP models from well 2 section 26".

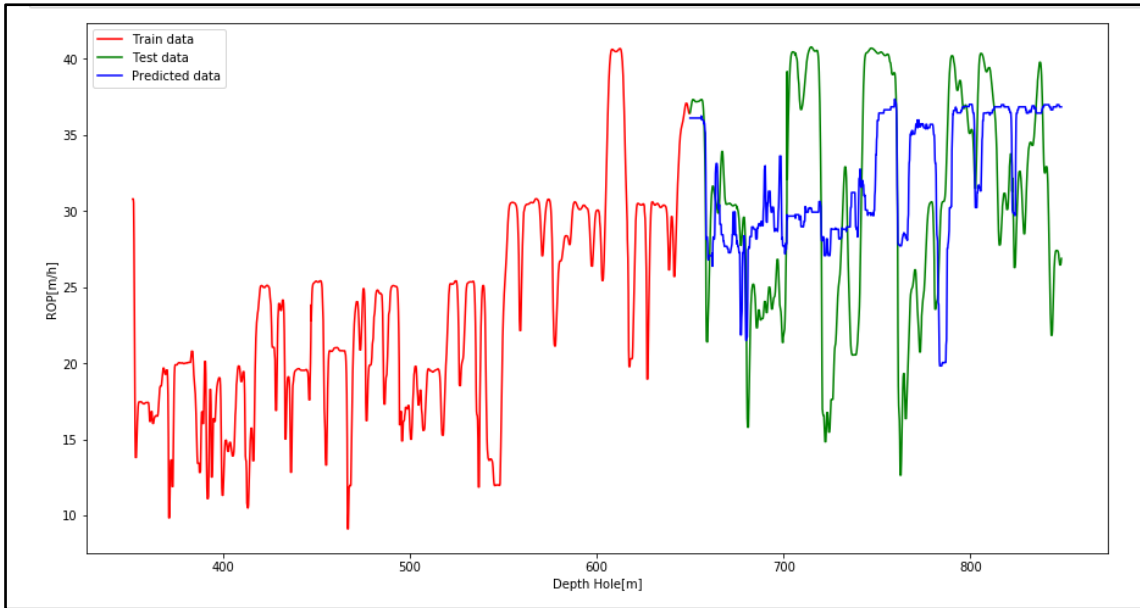


Figure 33: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 2 section 26". Predicted data (blue) is evaluated against test data (green).

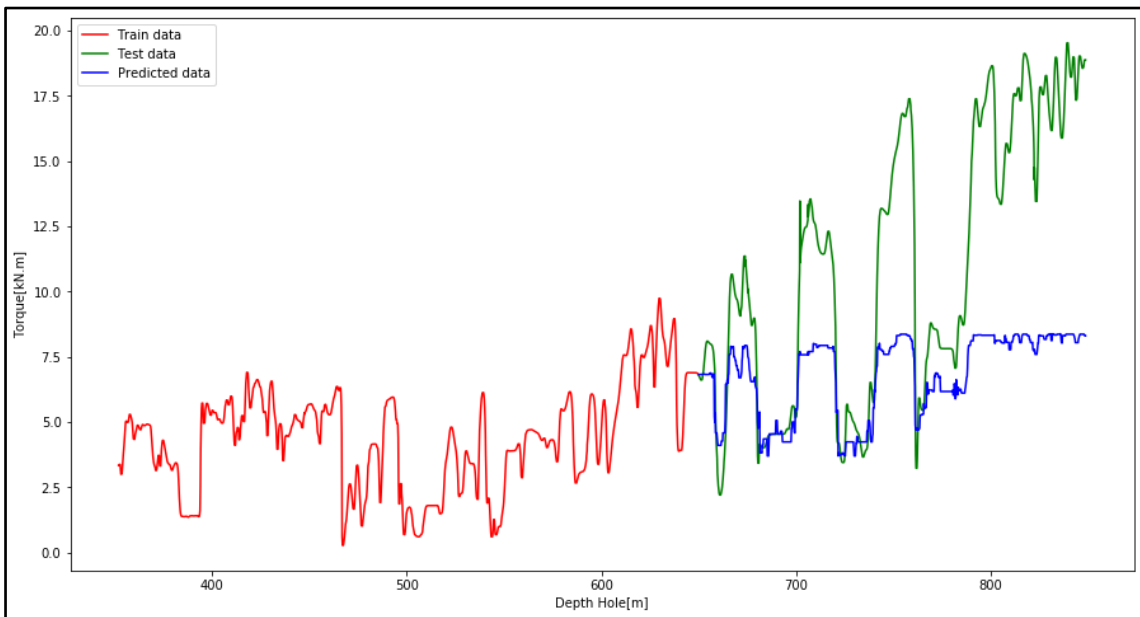


Figure 34: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 2 section 26". Predicted data (blue) is evaluated against test data (green).

| | MAE(kN.m) | R2 | std(kN.m) |
|----------------------------------|-----------|-----------|-----------|
| RandomForestRegressor | 3.028029 | -0.061816 | 3.542976 |
| GradientBoostingRegressor | 2.865264 | 0.076040 | 3.412709 |
| AdaBoostRegressor | 2.990533 | -0.013373 | 3.555454 |
| KNeighborsRegressor | 3.316821 | -0.361803 | 3.556617 |

Table 10: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for Torque models from well 3 section 26".

| | MAE(m/h) | R2 | std(m/h) |
|----------------------------------|----------|-----------|----------|
| RandomForestRegressor | 5.112361 | -0.560077 | 2.781780 |
| GradientBoostingRegressor | 5.103229 | -0.594032 | 2.972947 |
| AdaBoostRegressor | 6.642902 | -1.671430 | 3.631268 |
| KNeighborsRegressor | 6.227955 | -1.379596 | 4.045204 |

Table 11: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for ROP models from well 3 section 26".

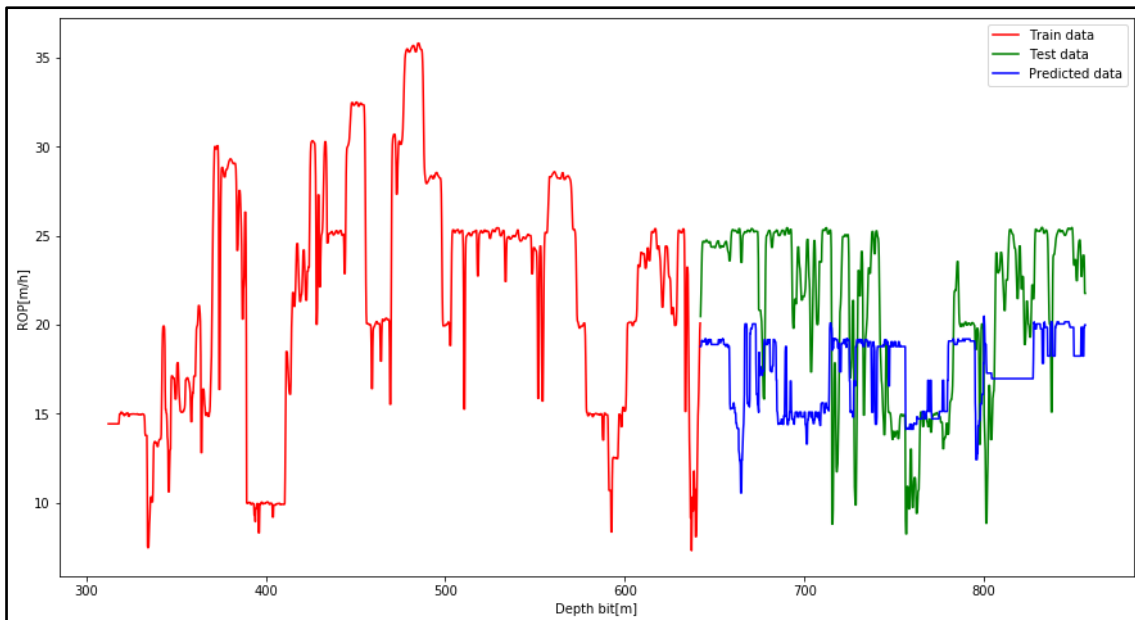


Figure 35: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 3 section 26". Predicted data (blue) is evaluated against test data (green).

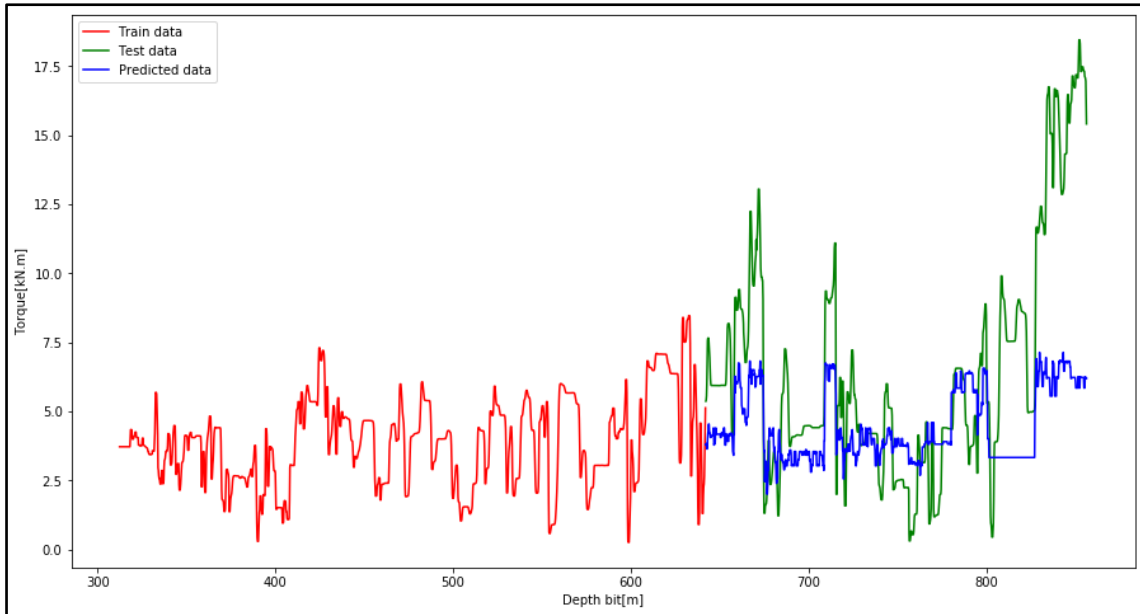


Figure 36: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 3 section 26". Predicted data (blue) is evaluated against test data (green).

| | MAE(kN.m) | R2 | std(kN.m) |
|----------------------------------|-----------|----------|-----------|
| RandomForestRegressor | 1.475561 | 0.796919 | 2.812511 |
| GradientBoostingRegressor | 1.754518 | 0.726743 | 2.545251 |
| AdaBoostRegressor | 1.704237 | 0.741897 | 2.511074 |
| KNeighborsRegressor | 2.660413 | 0.169979 | 2.574100 |

Table 12: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for Torque models from well 4 section 26".

| | MAE(m/h) | R2 | std(m/h) |
|----------------------------------|----------|-----------|----------|
| RandomForestRegressor | 8.801240 | -1.168768 | 6.759197 |
| GradientBoostingRegressor | 8.422941 | -0.893038 | 6.703809 |
| AdaBoostRegressor | 7.861795 | -0.817840 | 6.387726 |
| KNeighborsRegressor | 8.101674 | -0.865303 | 6.176790 |

Table 13: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for ROP models from well 4 section 26".

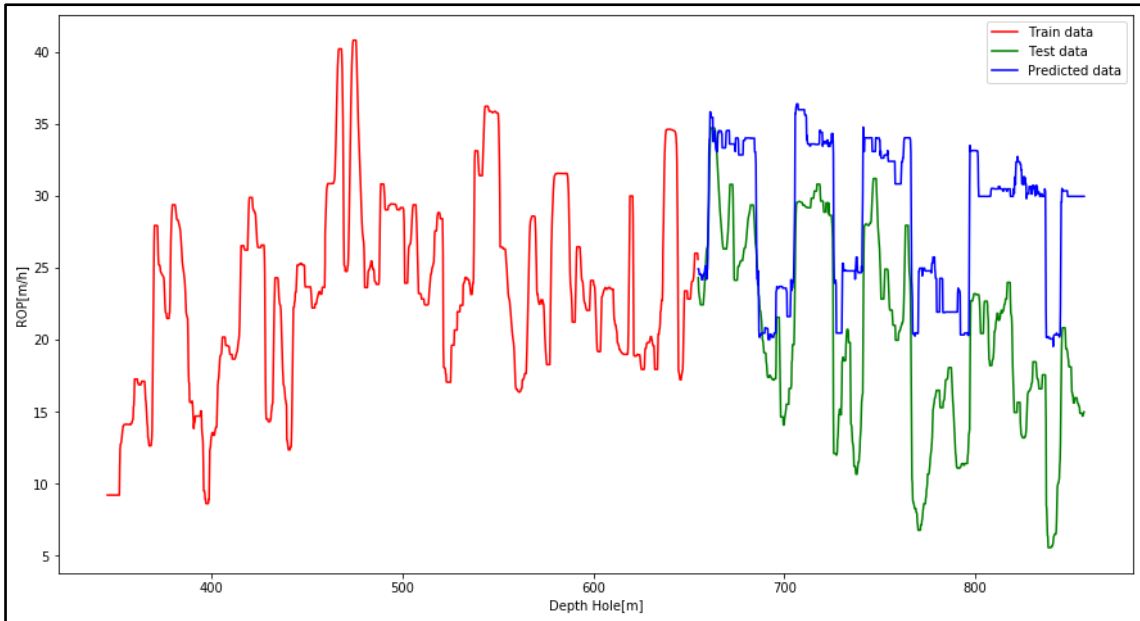


Figure 37: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 4 section 26". Predicted data (blue) is evaluated against test data (green).

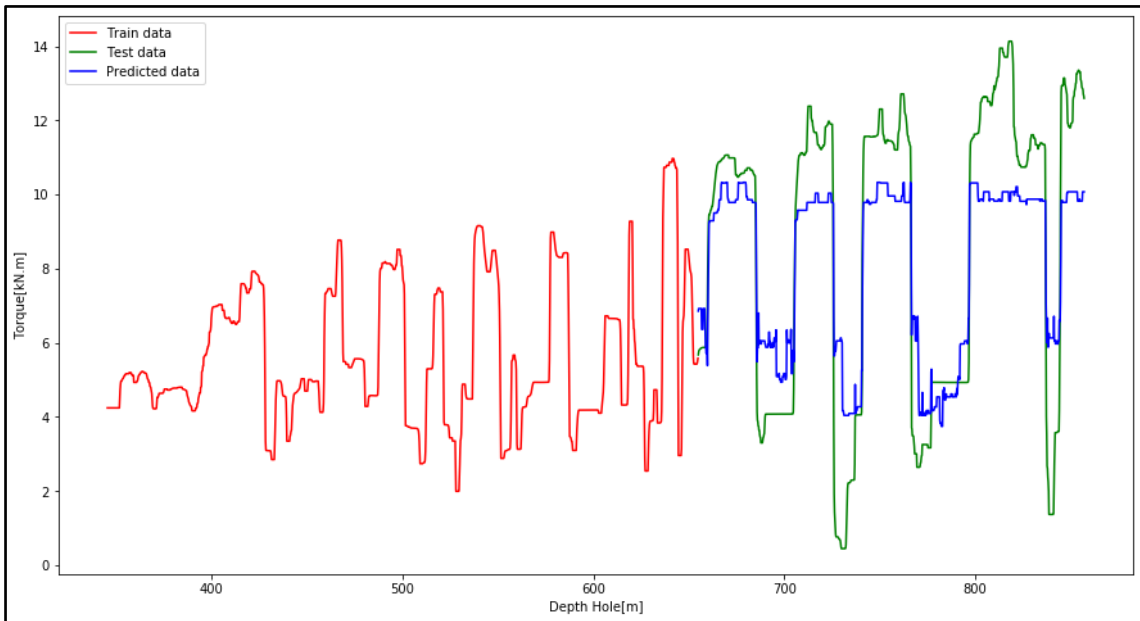


Figure 38: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 4 section 26". Predicted data (blue) is evaluated against test data (green).

| | MAE(kN.m) | R2 | std(kN.m) |
|----------------------------------|-----------|----------|-----------|
| RandomForestRegressor | 1.522331 | 0.599464 | 2.714751 |
| GradientBoostingRegressor | 1.426338 | 0.624032 | 2.683018 |
| AdaBoostRegressor | 1.854912 | 0.500111 | 2.899079 |
| KNeighborsRegressor | 2.007140 | 0.319283 | 2.636681 |

Table 14: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for Torque models from well 5 section 26''

| | MAE(m/h) | R2 | std(m/h) |
|----------------------------------|----------|----------|----------|
| RandomForestRegressor | 5.927597 | 0.084249 | 5.664106 |
| GradientBoostingRegressor | 5.449038 | 0.273782 | 5.765587 |
| AdaBoostRegressor | 5.916922 | 0.124623 | 6.414889 |
| KNeighborsRegressor | 5.842076 | 0.061529 | 5.735481 |

Table 15: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for ROP models from well 5 section 26''

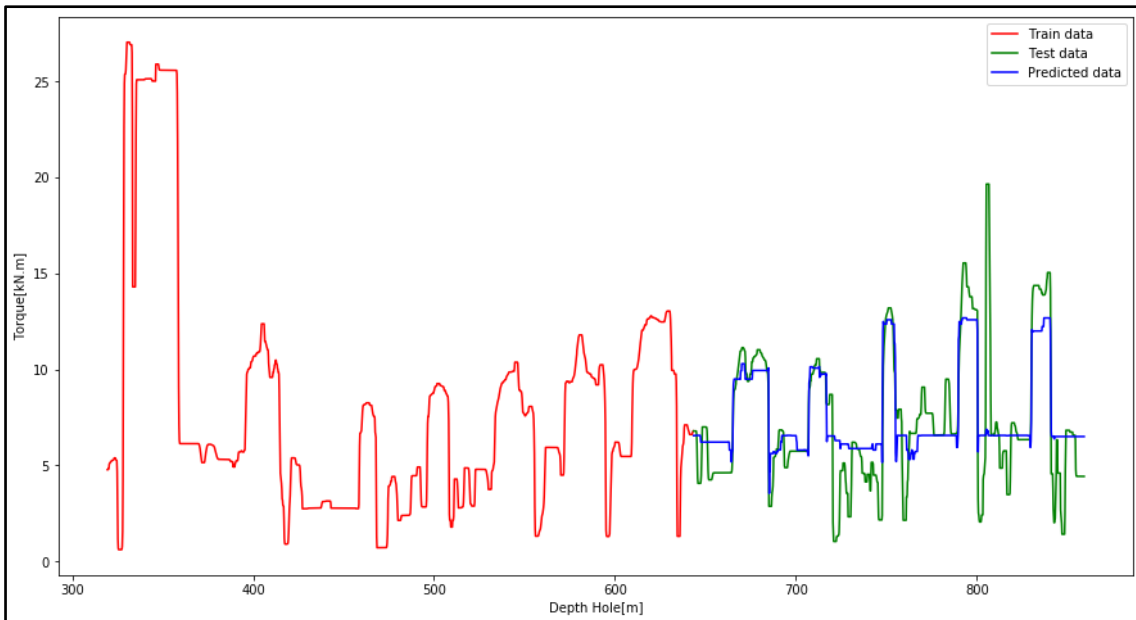


Figure 39: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 5 section 26''. Predicted data (blue) is evaluated against test data (green).

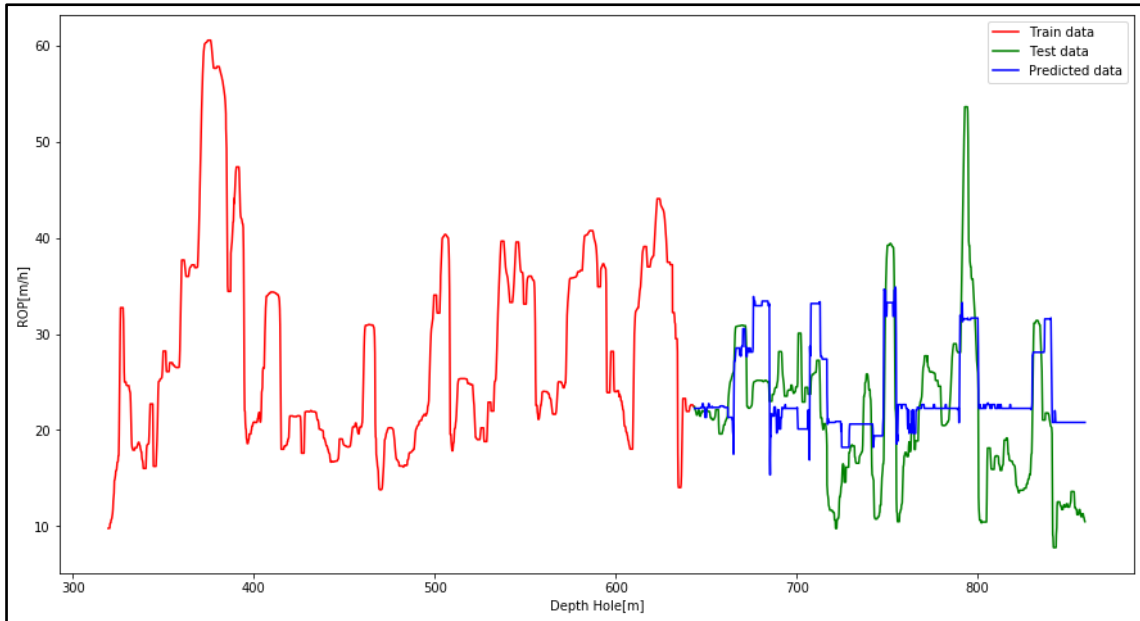


Figure 40: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 5 section 26". Predicted data (blue) is evaluated against test data (green).

| | MAE(kN.m) | R2 | std(kN.m) |
|----------------------------------|-----------|-----------|-----------|
| RandomForestRegressor | 2.541876 | -0.883403 | 1.939043 |
| GradientBoostingRegressor | 2.512490 | -0.823308 | 1.917233 |
| AdaBoostRegressor | 2.873014 | -1.367623 | 1.873093 |
| KNeighborsRegressor | 3.416860 | -2.559180 | 1.938739 |

Table 16: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for Torque models from well 6 section 26".

| | MAE(m/h) | R2 | std(m/h) |
|----------------------------------|----------|-----------|----------|
| RandomForestRegressor | 4.335182 | -0.255038 | 4.360897 |
| GradientBoostingRegressor | 4.132021 | -0.114473 | 4.202282 |
| AdaBoostRegressor | 4.167678 | -0.062695 | 3.772744 |
| KNeighborsRegressor | 5.036684 | -0.401059 | 4.414932 |

Table 17: Mean Absolute Error (MAE), coefficient of determination (R^2) and standard deviation (std) for ROP models from well 6 section 26".

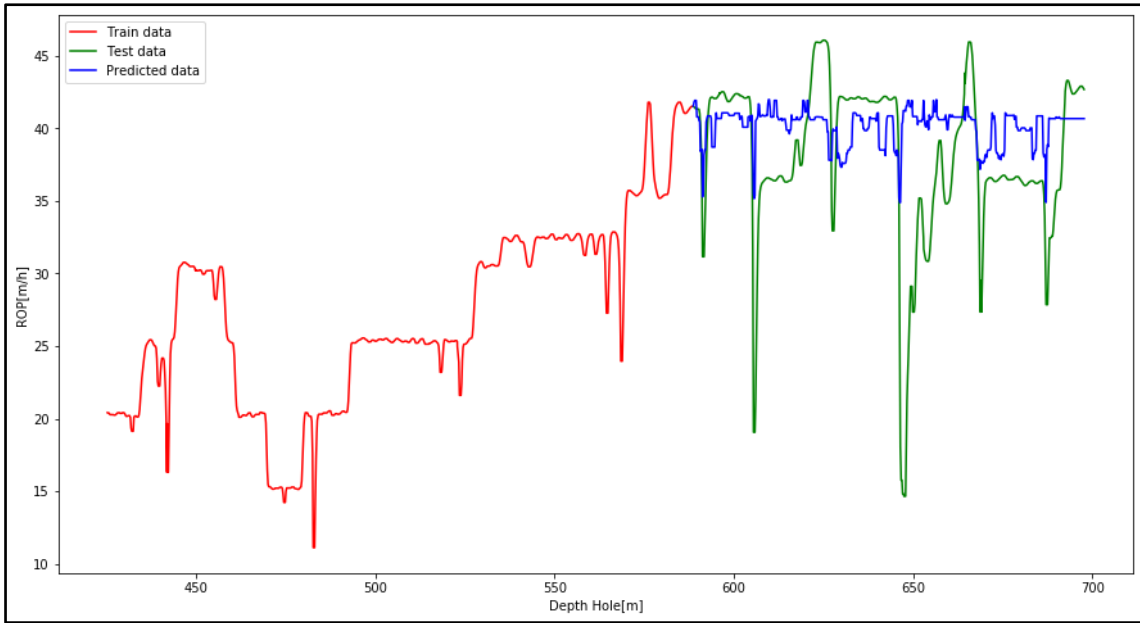


Figure 41: Gradient Boosting algorithm plot ROP vs. depth using 60/40 split from well 6 section 26". Predicted data (blue) is evaluated against test data (green).

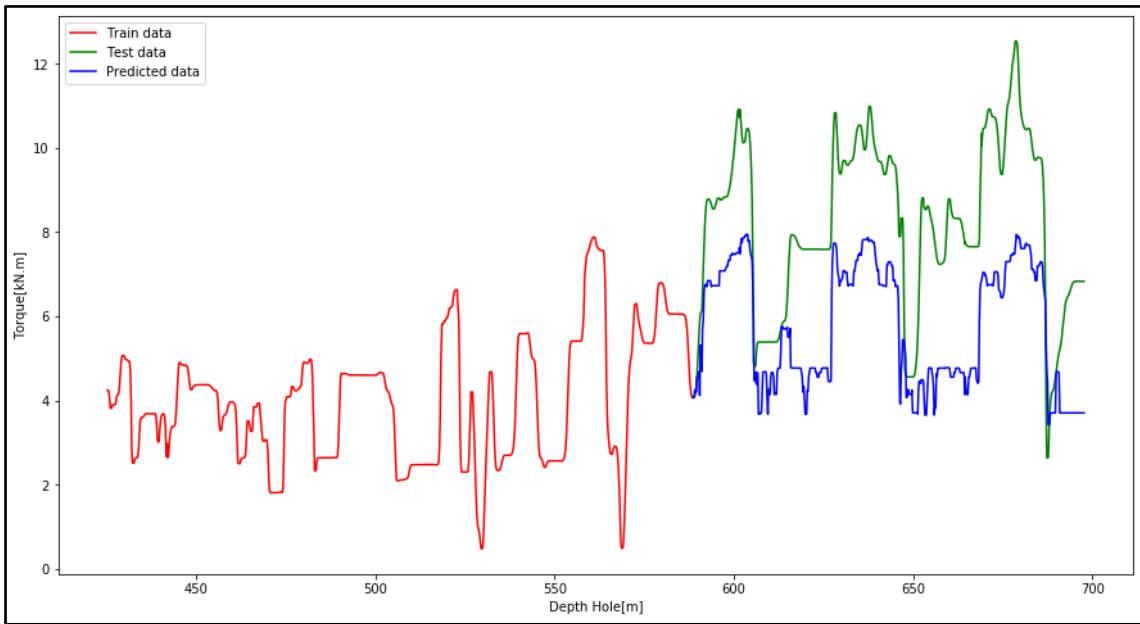


Figure 42: Gradient Boosting algorithm plot Torque vs. depth using 60/40 split from well 6 section 26". Predicted data (blue) is evaluated against test data (green).

Results Optimization PSO

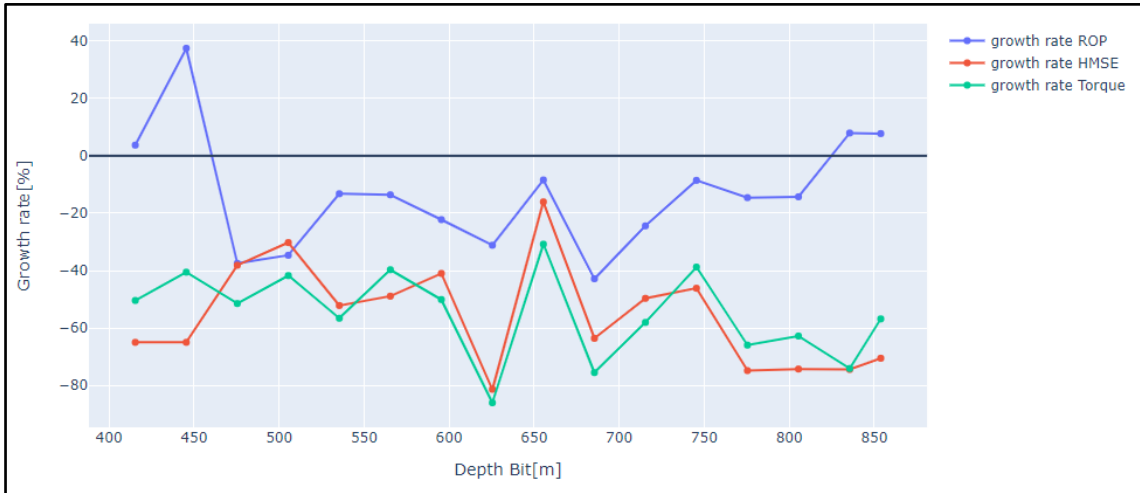


Figure 43: ROP (blue), HMSE (red) and Torque (green) growth rate using GB Torque model and PSO combination. Growth rate is negative because the three parameters decrease.

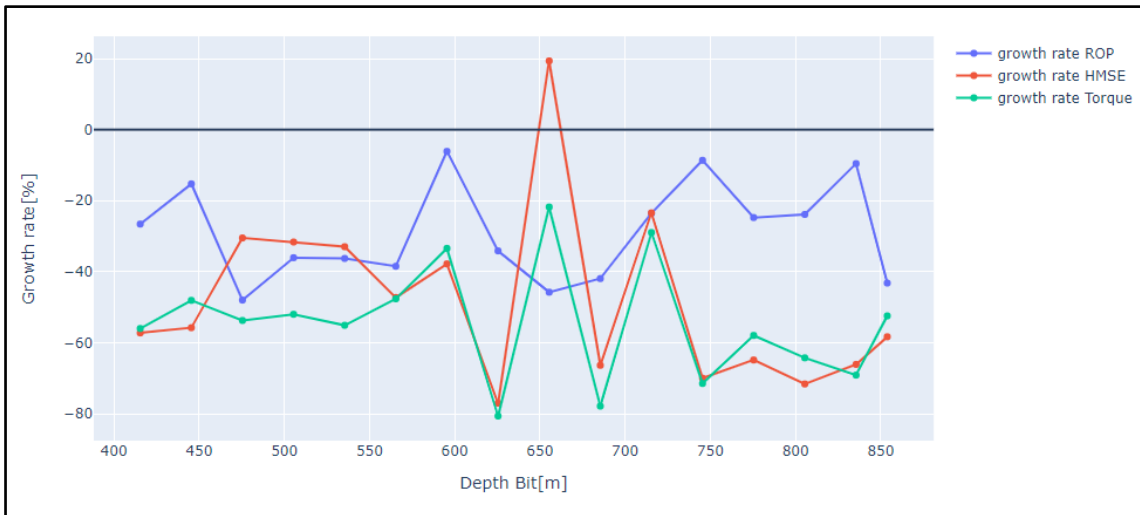


Figure 44: ROP (blue), HMSE (red) and Torque (green) growth rate using GB ROP model and PSO combination.

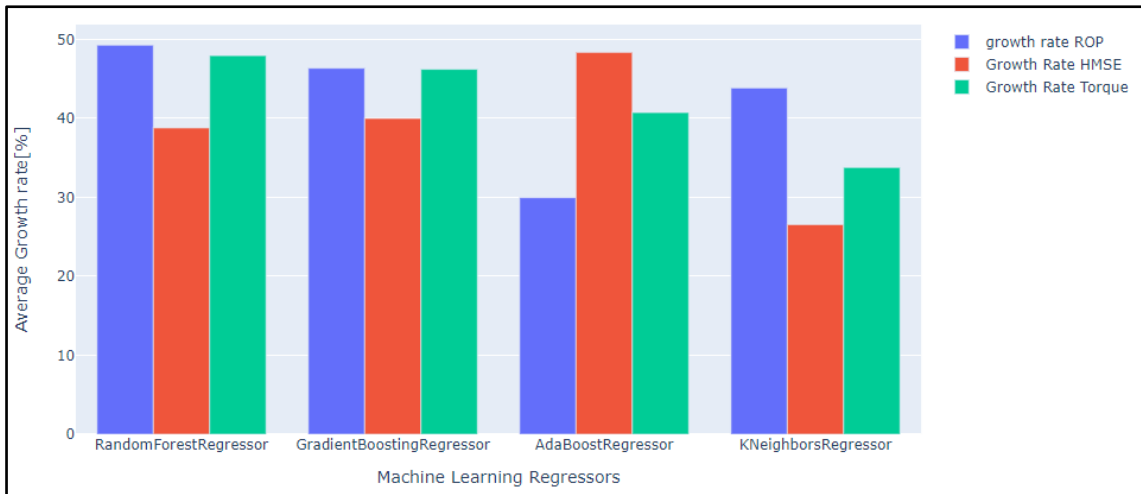


Figure 45: ROP (blue), HMSE (red) and Torque (green) average growth rate for all ML models for ROP optimization (maximize ROP) using PSO algorithm.



Figure 46: ROP (blue), HMSE (red) and Torque (green) average growth rate for all ML models for ROP optimization (maximize ROP) using PSO algorithm.

Python Code

Cleaning Part

Import all the libraries:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor
import matplotlib.pyplot as plt
from scipy.optimize import differential_evolution, least_squares
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import seaborn as sns
from matplotlib.dates import DateFormatter
from datetime import datetime, timedelta, date
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

Convert in datetime and to numeric

```
df['time_DB'] = pd.to_datetime(df['time_DB'])
df['time_DH'] = pd.to_datetime(df['time_DH'])
df['time_RPM'] = pd.to_datetime(df['time_RPM'])
df['time_TOR'] = pd.to_datetime(df['time_TOR'])
df['time_WOB'] = pd.to_datetime(df['time_WOB'])
df['time_ROP'] = pd.to_datetime(df['time_ROP'])
df['time_FR'] = pd.to_datetime(df['time_FR'])
df['time_SPP'] = pd.to_datetime(df['time_SPP'])

df['Depth Bit[m]'] = pd.to_numeric(df['Depth Bit[m]'])
df['Depth Hole[m]'] = pd.to_numeric(df['Depth Hole[m]'])
df['RPM[rev/s]'] = pd.to_numeric(df['RPM[rev/s]'])
df['Torque[N.m]'] = pd.to_numeric(df['Torque[N.m]'])
df['WOB[N]'] = pd.to_numeric(df['WOB[N]'])
df['ROP[m/s]'] = pd.to_numeric(df['ROP[m/s]'])
df['Flow rate[m3/s]'] = pd.to_numeric(df['Flow rate[m3/s]'])
df['Stand Pipe Pressure[Pa]'] = pd.to_numeric(df['Stand Pipe Pressure[Pa]'])
```

Create dataframe for each parameter:

```
df_DB = pd.concat([df['time_DB'], df['Depth Bit[m]']], join = 'outer', axis = 1)
df_DH = pd.concat([df['time_DH'], df['Depth Hole[m]']], join = 'outer', axis = 1)
df_RPM = pd.concat([df['time_RPM'], df['RPM[rev/s]']], join = 'outer', axis = 1)
df_TOR = pd.concat([df['time_TOR'], df['Torque[N.m]']], join = 'outer', axis = 1)
df_WOB = pd.concat([df['time_WOB'], df['WOB[N]']], join = 'outer', axis = 1)
```

```
df_ROP = pd.concat([df['time_ROP'], df['ROP[m/s]']], join = 'outer', axis = 1)
df_FR = pd.concat([df['time_FR'], df['Flow rate[m3/s]']], join = 'outer', axis = 1)
df_SPP = pd.concat([df['time_SPP'], df['Stand Pipe Pressure[Pa]']], join = 'outer', axis = 1)
```

using Dropna, delete all rows with missing values:

```
df_DB = df_DB.dropna()
df_DH = df_DH.dropna()
df_RPM = df_RPM.dropna()
df_TOR = df_TOR.dropna()
df_WOB = df_WOB.dropna()
df_ROP = df_ROP.dropna()
df_FR = df_FR.dropna()
df_SPP = df_SPP.dropna()
```

Check duplicates:

```
print('no. of duplicates DB', df_DB.duplicated().sum(axis=0))
print('no. of duplicates DH', df_DB.duplicated().sum(axis=0))
print('no. of duplicates RPM', df_RPM.duplicated().sum(axis=0))
print('no. of duplicates Torque', df_TOR.duplicated().sum(axis=0))
print('no. of duplicates WOB', df_WOB.duplicated().sum(axis=0))
print('no. of duplicates ROP', df_ROP.duplicated().sum(axis=0))
print('no. of duplicates FR', df_FR.duplicated().sum(axis=0))
print('no. of duplicates SPP', df_SPP.duplicated().sum(axis=0))
```

Create function using plotly to plot variables vs. depth:

```
from plotly.subplots import make_subplots
def plotly_vars_depth(df_sample, section):
    if section == "section 26":
        Titles = ['ROP[m/h]', 'WOB[kN]', 'Torque[kN.m]', 'Flow rate[L/min]', 'Stand Pipe
        Pressure[kPa]', 'Downhole RPM[rev/min]']
    else:
        Titles = ['ROP[m/h]', 'WOB[kN]', 'Torque[kN.m]', 'Flow rate[L/min]', 'Stand Pipe
        Pressure[kPa]', 'RPM[rev/min]']
    fig = make_subplots(rows=1, cols=6, subplot_titles=Titles, shared_yaxes = True)
    for i in range(6):
        fig.add_trace(go.Scatter(x=df_sample[Titles[i]], y=df_sample["Depth Hole[m]"]),row=1,
        col=i+1)
    for i in fig['layout']['annotations']:
        i['font']['size'] = 10

    fig.update_yaxes(autorange="reversed")
    fig.update_yaxes(title_text = 'Depth Hole[m]',title_font_size=10, row =1, col=1)
    fig.update_layout(height=800, width=1000, showlegend=False, title="Variables vs. Hole
    Depth",)
    fig.show()
```

```
# First, we change the name of column in all datasets:
```

```
df_DB = df_DB.rename(columns={'time_DB':'time'})
df_DH = df_DH.rename(columns={'time_DH':'time'})
df_RPM = df_RPM.rename(columns={'time_RPM':'time'})
df_ROP = df_ROP.rename(columns={'time_ROP':'time'})
df_WOB = df_WOB.rename(columns={'time_WOB':'time'})
df_TOR = df_TOR.rename(columns={'time_TOR':'time'})
df_FR = df_FR.rename(columns={'time_FR':'time'})
df_SPP = df_SPP.rename(columns={'time_SPP':'time'})
```

```
# Then we merge the dataframes:
```

```
df1 = pd.merge_asof(df_DH,df_DB,on='time', tolerance = pd.Timedelta('0.1s'))
df2 = pd.merge_asof(df_ROP, df_RPM, on='time', tolerance = pd.Timedelta('0.1s'))
df3 = pd.merge_asof(df_WOB, df_TOR, on='time', tolerance = pd.Timedelta('0.1s'))
df4 = pd.merge_asof(df_FR, df_SPP, on='time', tolerance = pd.Timedelta('0.1s'))
df5 = pd.merge_asof(df1, df2, on='time', tolerance = pd.Timedelta('0.1s'))
df6 = pd.merge_asof(df3, df4, on='time', tolerance = pd.Timedelta('0.1s'))
df_merged = pd.merge_asof(df5, df6, on='time', tolerance = pd.Timedelta('0.1s'))
```

```
# Print all histograms:
```

```
fig2 = plt.figure(figsize=(15, 8))
grid = plt.GridSpec(4, 2, hspace=0.5)
ROP_hist = fig2.add_subplot(grid[0,0])
WOB_hist = fig2.add_subplot(grid[1, 0])
RPM_hist = fig2.add_subplot(grid[2, 0])
DB_hist = fig2.add_subplot(grid[0, 1])
DH_hist = fig2.add_subplot(grid[1, 1])
Tor_hist = fig2.add_subplot(grid[2, 1])
FR_hist = fig2.add_subplot(grid[3, 0])
SPP_hist = fig2.add_subplot(grid[3, 1])

ROP_hist.hist(df_ROP['ROP[m/s]'])
ROP_hist.set(title="ROP[m/s]")
WOB_hist.hist(df_WOB['WOB[N]'])
WOB_hist.set(title="WOB[N]")
RPM_hist.hist(df_RPM['RPM[rev/s]'])
RPM_hist.set(title="RPM[rev/s]")
DB_hist.hist(df_DB['Depth Bit[m]'])
DB_hist.set(title="Depth Bit[m]")
DH_hist.hist(df_DH['Depth Hole[m]'])
DH_hist.set(title="Depth Hole[m]")
Tor_hist.hist(df_TOR['Torque[N.m]'])
Tor_hist.set(title="Torque[N.m]")
FR_hist.hist(df_FR['Flow rate[m3/s]'])
FR_hist.set(title="Flow rate[m3/s]")
SPP_hist.hist(df_SPP['Stand Pipe Pressure[Pa]'])
SPP_hist.set(title="Stand Pipe Pressure[Pa]")
```

```
plt.show()
```

```
# Interpolation:
```

```
df_test = df
df_test['ROP[m/s]'] = df_test['ROP[m/s]'].interpolate()
df_test['WOB[N]'] = df_test['WOB[N]'].interpolate()
df_test['Torque[N.m]'] = df_test['Torque[N.m]'].interpolate()
df_test['RPM[rev/s]'] = df_test['RPM[rev/s]'].interpolate()
df_test['Depth Bit[m]'] = df_test['Depth Bit[m]'].interpolate()
df_test['Flow rate[m3/s]'] = df_test['Flow rate[m3/s]'].interpolate()
df_test['Stand Pipe Pressure[Pa]'] = df_test['Stand Pipe Pressure[Pa]'].interpolate()
```

```
# Convert in desirable units
```

```
df.columns = ['time', 'Depth Hole[m]', 'Depth Bit[m]', 'ROP[m/h]', 'RPM[rev/min]', 'WOB[kN]',
'Torque[kN.m]', 'Flow rate[L/min]', 'Stand Pipe Pressure[kPa]']
df["ROP[m/h]"] = df["ROP[m/h]"]*3600
df['RPM[rev/min]'] = df['RPM[rev/min]']*60
df['WOB[kN]'] = df['WOB[kN]']*0.001
df['Torque[kN.m]'] = df['Torque[kN.m]']*0.001
df['Flow rate[L/min]'] = df['Flow rate[L/min]']*60000
df['Stand Pipe Pressure[kPa]'] = df['Stand Pipe Pressure[kPa]']*0.001
```

```
# Removing outliers
```

```
Q1_ROP = df['ROP[m/h]'].quantile(0.25)
Q3_ROP = df['ROP[m/h]'].quantile(0.75)
IQR_ROP = Q3_ROP - Q1_ROP
```

```
Q1_RPM = df['RPM[rev/min]'].quantile(0.25)
Q3_RPM = df['RPM[rev/min]'].quantile(0.75)
IQR_RPM = Q3_RPM - Q1_RPM
```

```
Q1_TOR = df['Torque[kN.m]'].quantile(0.25)
Q3_TOR= df['Torque[kN.m]'].quantile(0.75)
IQR_TOR = Q3_TOR - Q1_TOR
```

```
Q1_WOB = df['WOB[kN]'].quantile(0.25)
Q3_WOB = df['WOB[kN]'].quantile(0.75)
IQR_WOB = Q3_WOB - Q1_WOB
```

```
Q1_SPP = df['Stand Pipe Pressure[kPa]'].quantile(0.25)
Q3_SPP = df['Stand Pipe Pressure[kPa]'].quantile(0.75)
IQR_SPP = Q3_SPP - Q1_SPP
```

```
Q1_FR = df['Flow rate[L/min]'].quantile(0.25)
Q3_FR= df['Flow rate[L/min]'].quantile(0.85)
IQR_FR = Q3_FR - Q1_FR
```

```

print('ROP:', Q3_ROP+1.5*IQR_ROP, Q1_ROP-1.5*IQR_ROP)
print('RPM:', Q3_RPM+1.5*IQR_RPM, Q1_RPM-1.5*IQR_RPM)
print('Torque:', Q3_TOR+1.5*IQR_TOR, Q1_TOR-1.5*IQR_TOR)
print('WOB:', Q3_WOB+1.5*IQR_WOB, Q1_WOB-1.5*IQR_WOB)
print('SPP:', Q3_SPP+1.5*IQR_SPP, Q1_SPP-1.5*IQR_SPP)
print('FR:', Q3_FR+1.5*IQR_FR, Q1_FR-1.5*IQR_FR)

```

```

df_test_out = df[(Q1_ROP-1.5*IQR_ROP)<=df['ROP[m/h]']]
df_test_out = df[df['ROP[m/h]']<=(Q3_ROP+1.5*IQR_ROP)]

```

```

df_test_out = df[(Q1_RPM-1.5*IQR_RPM)<=df['RPM[rev/min]']]
df_test_out = df[df['RPM[rev/min]']<=(Q3_RPM+1.5*IQR_RPM)]

```

```

df_test_out = df[(Q1_TOR-1.5*IQR_TOR)<=df['Torque[kN.m]']]
df_test_out = df[df['Torque[kN.m]']<=(Q3_TOR+1.5*IQR_TOR)]

```

```

df_test_out = df[(Q1_WOB-1.5*IQR_WOB)<=df['WOB[kN]']]
df_test_out = df[df['WOB[kN]']<=(Q3_WOB+1.5*IQR_WOB)]

```

```

df_test_out = df[(Q1_SPP-1.5*IQR_SPP)<=df['Stand Pipe Pressure[kPa]']]
df_test_out = df[df['Stand Pipe Pressure[kPa]']<=(Q3_SPP+1.5*IQR_SPP)]

```

```

df_test_out = df[(Q1_FR-1.5*IQR_FR)<=df['Flow rate[L/min]']]
df_test_out = df[df['Flow rate[L/min]']<=(Q3_FR+1.5*IQR_FR)]

```

Delete rows when ROP is 0

```

df = df[df['ROP[m/s]']>0]
df = df[df['ROP[m/s]'] < 0.02]

```

Remove outliers manually:

```

df_test_out = df_test_out[df_test_out['WOB[kN]']>0]
df_test_out = df_test_out[df_test_out['Flow rate[L/min]']> 2000]
df_test_out = df_test_out[df_test_out['Stand Pipe Pressure[kPa]']> 5000]

```

Noise reduction (window = 30):

```

df = df_test_out
df_test2 = df
df_test2['ROP[m/h]'] = df['ROP[m/h]'].rolling(window = 30, center=True).mean()
df_test2['RPM[rev/min]'] = df['RPM[rev/min]'].rolling(window = 30, center=True).mean()
df_test2['WOB[kN]'] = df['WOB[kN]'].rolling(window = 30, center=True).mean()
df_test2['Torque[kN.m]'] = df['Torque[kN.m]'].rolling(window = 30, center=True).mean()
df_test2['Flow rate[L/min]'] = df['Flow rate[L/min]'].rolling(window = 30, center=True).mean()
df_test2['Stand Pipe Pressure[kPa]'] = df['Stand Pipe Pressure[kPa]'].rolling(window = 30,
center=True).mean()
df_test2.dropna(inplace = True)

```

```

df = df_test2

```

```
df.to_csv(r'Cleaned wells/NO162D6-4.csv', index = False, header=True)
```

Modelling Part

```
# Import all the libraries:
```

```
import numpy as np
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor
import matplotlib.pyplot as plt
from scipy.optimize import differential_evolution, least_squares
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import seaborn as sns
from matplotlib.dates import DateFormatter
from datetime import datetime, timedelta, date
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.preprocessing import MinMaxScaler
```

```
# Conver to datetime
```

```
df['time'] = pd.to_datetime(df['time'])
```

```
# Create time difference column to get diferent sections
```

```
df['timeDiff'] = df['time'].diff().dt.total_seconds()
```

```
# Plot time difference and time vs. depth to visually get sections:
```

```
fig = make_subplots(rows=1, cols=2, subplot_titles=("time vs. Depth", "timeDiff vs. Depth"))
fig.add_trace(
    go.Scatter(x=df["time"], y=df["Depth Hole[m]"]),
    row=1, col=1
)
fig.add_trace(
    go.Scatter(x=df["timeDiff"], y=df["Depth Hole[m]"]),
    row=1, col=2
)
fig.update_yaxes(autorange="reversed")
fig.update_layout(height=600, width=800, title_text="Find sections", showlegend=False)
fig.show()
```

```
# Get sections (vary from each well since not all wells have the same sections)
```

```
sections = df[df['timeDiff']>100000]
```

```
Index = sections.index.tolist()
```

```
section_26 = df.iloc[:Index[0]] #section 26"  
section_16 = df.iloc[Index[0]:Index[1]] #section 16"  
section_12 = df.iloc[Index[1]:Index[2]] #section 12 1/4"  
section_8 = df.iloc[Index[2]:] # open hole section 8 1/2"  
Sections = [section_26, section_16, section_12, section_8]
```

```
# Add column Downhole RPM for section 26"
```

```
StF = 0.02 # speed to flow ratio [rev/l]  
section_26['Downhole RPM[rev/min]'] = StF * section_26['Flow rate[L/min]'] +  
section_26['RPM[rev/min]']
```

```
# Create list of columns of the table:
```

```
ROP_list = []  
MAE_list = []  
std_list = []  
params_list = []  
R2_list = []
```

```
ROP_list_RANDOM = []  
MAE_list_RANDOM = []  
std_list_RANDOM = []  
params_list_RANDOM = []  
R2_list_RANDOM = []
```

```
# Create regressor list:
```

```
regs = [  
    RandomForestRegressor(),  
    GradientBoostingRegressor(),  
    AdaBoostRegressor(),  
    KNeighborsRegressor(),  
]  
reg_name = [  
    "RandomForestRegressor",  
    "GradientBoostingRegressor",  
    "AdaBoostRegressor",  
    "KNeighborsRegressor"  
]
```

```
# Drop timediff column
```

```
df1 = Sections[0].drop(columns=['timeDiff'])  
df2 = Sections[1].drop(columns=['timeDiff'])  
df3 = Sections[2].drop(columns=['timeDiff'])  
df4 = Sections[3].drop(columns=['timeDiff'])
```

```
# Density and viscosity values
```

```
rho = 1.39 # g/cm3
```



```
mu = 28 # Mpa.s
```

```
# Define different functions for plotting:
```

```
def plotly_vars_depth(df_sample, section):  
    if section == "section 26":  
        Titles = ['ROP[m/h]', 'WOB[kN]', 'Torque[kN.m]', 'Flow rate[L/min]', 'Stand Pipe  
Pressure[kPa]', 'Downhole RPM[rev/min]']  
    else:  
        Titles = ['ROP[m/h]', 'WOB[kN]', 'Torque[kN.m]', 'Flow rate[L/min]', 'Stand Pipe  
Pressure[kPa]', 'RPM[rev/min]']  
    fig = make_subplots(rows=1, cols=6, subplot_titles=Titles, shared_yaxes = True)  
    for i in range(6):  
        fig.add_trace(go.Scatter(x=df_sample[Titles[i]], y=df_sample["Depth Hole[m]"]),row=1,  
col=i+1)  
    for i in fig['layout']['annotations']:  
        i['font']['size'] = 10  
  
    fig.update_yaxes(autorange="reversed")  
    fig.update_yaxes(title_text = 'Depth Hole[m]',title_font_size=10, row =1, col=1)  
    fig.update_layout(height=800, width=1000, showlegend=False, title="Variables vs. Hole  
Depth",)  
    fig.show()
```

```
def plot_ROP_vs._depth_ML(train, test, pred):  
    fig = go.Figure()  
    fig.add_trace(go.Scatter(name = 'Training data',y= DH_train , x= train))  
    fig.add_trace(go.Scatter(name = 'Test data',y= DH_test, x= test))  
    fig.add_trace(go.Scatter(name = 'Predicted data',y= DH_test , x= pred))  
  
    # Update axis  
    fig.update_xaxes(title_text="ROP[m/h]")  
    fig.update_yaxes(autorange="reversed")  
    fig.update_yaxes(title_text = 'Depth[m]')  
    fig.update_layout(height=600, width=800)  
    fig.show()
```

```
def plot_ROP_vs._depth_ML_RN(train, test,pred, model_phase):  
  
    fig,axis =plt.subplots(figsize =(15,8))  
    sns.lineplot(DB_train , train, ax = axis, label ="Train data",color ='red', ci = None)  
    sns.lineplot(DB_test, test, ax = axis, label ="Test data", color ='green',ci = None)  
    sns.lineplot(DB_test, pred, ax = axis, label ="Predicted data",color ='blue', ci = None)  
  
    if model_phase==2:  
        axis.set(ylabel='ROP[m/h]', xlabel='Depth Hole[m]')  
    else:  
        axis.set(ylabel='Torque[kN.m]', xlabel='Depth Hole[m]')
```

```

plt.show()

# Filter using Radius Neighbors

df1_test = df1.groupby(['Depth Hole[m]']).mean()
df1_test = df1_test.reset_index()

# Convert variable to numpy:
sample = df1_test.copy()
from sklearn.neighbors import RadiusNeighborsRegressor

x = sample["Depth Hole[m]"].to_numpy().reshape(-1,1)
Y = sample.iloc[:,1:].to_numpy()

neigh = RadiusNeighborsRegressor(radius=0.2, weights= 'distance')
neigh.fit(x,Y)

X_test = np.arange(x[0:],x[-1:],0.2).reshape(-1,1) # can change the distance to reduce noise
Y_pred = neigh.predict(X_test)
values_data = np.concatenate((X_test,Y_pred),axis=1)

df1_test2 = pd.DataFrame(values_data)
df1_test2.dropna(inplace=True)

df1_test2.columns = ['Depth Hole[m]', 'Depth Bit[m]', 'ROP[m/h]', 'RPM[rev/min]', 'WOB[kN]',
'Torque[kN.m]', 'Flow rate[L/min]', 'Stand Pipe Pressure[kPa]', 'Downhole RPM[rev/min]']

# Median Filter

df1_test3 = df1_test2.copy()
columns_names = ['ROP[m/h]', 'RPM[rev/min]', 'WOB[kN]', 'Torque[kN.m]', 'Flow rate[L/min]',
'Stand Pipe Pressure[kPa]']
for i in columns_names:
    df1_test3[i] = ndimage.median_filter(df1_test3[i], size=5)

# Continuous Learning modelling
# First model:
# Define independent and dependent variables(x_1 and y_1):
y_1 = df1_test3["Torque[kN.m]"].to_numpy().reshape(-1,1)
x_1 = df1_test3[['Depth Bit[m]', 'WOB[kN]', 'Flow rate[L/min]', 'Downhole RPM[rev/min]']
].to_numpy()

# Second model:
# Define independent and dependent variables(x_2 and y_2):
y_2 = df1_test3["ROP[m/h]"].to_numpy().reshape(-1,1)
x_2 = np.copy(x_1)

```

Function for continuous learning modelling. This is implemented on each ML model:
(Then, heatmap was plot for ROP and Torque models, which is not showed in this appendix,
only relevant part of the code is included)

```
def modelling_cont_learning_2phase(reg_num, num_rows):
```

```
    reg = regs[reg_num]
```

```
    end = num_rows
```

```
    init = int(0.1*end)
```

```
    step = init
```

```
    MAE_2 = []
```

```
    std_2 = []
```

```
    R2_2 = []
```

```
    perc = []
```

```
    MAE_1 = []
```

```
    std_1 = []
```

```
    R2_1 = []
```

```
    for i in range(init, end ,step):
```

```
        # First model:
```

```
            # Define train and test samples:
```

```
            X_train_1 = x_1[:i]
```

```
            y_train_1 = y_1[:i]
```

```
            X_test_1 = x_1[i:i+step]
```

```
            y_test_1 = y_1[i:i+step]
```

```
            # Fit model
```

```
            reg.fit(X_train_1, y_train_1.ravel())
```

```
            # Prediction
```

```
            y_pred_1 = reg.predict(X_test_1)
```

```
            y_train_fit_1 = reg.predict(X_train_1)
```

```
            # Second model:
```

```
            # Define train and test samples:
```

```
            X_train_2 = x_2[:i]
```

```
            y_train_2 = y_2[:i]
```

```
            X_test_2 = x_2[i:i+step]
```

```
            y_test_2 = y_2[i:i+step]
```

```
            # Fit model
```

```
            reg.fit(X_train_2, y_train_2.ravel())
```

```
            # Prediction
```

```
            y_pred_2 = reg.predict(X_test_2)
```

```
            y_train_fit_2 = reg.predict(X_train_2)
```

```
            # variables from the first model:
```

```

MAE_y_1 = MAE(y_test_1, y_pred_1)
R2_y_1 = r2_score(y_test_1, y_pred_1)
error_1 = np.abs(y_test_1 - y_pred_1)
std_y_1 = np.std(error_1)
perc.append((y_test_1.shape[0]+y_train_1.shape[0])/end*100)
# Update lists:
MAE_1.append(MAE_y_1)
std_1.append(std_y_1)
R2_1.append(R2_y_1)

# variables from the second model:
MAE_y_2 = MAE(y_test_2, y_pred_2)
R2_y_2 = r2_score(y_test_2, y_pred_2)
error_2 = np.abs(y_test_2 - y_pred_2)
std_y_2 = np.std(error_2)

# Update lists:
MAE_2.append(MAE_y_2)
std_2.append(std_y_2)
R2_2.append(R2_y_2)

# From the first model
MAE_1 = np.array(MAE_1)
R2_1 = np.array(R2_1)
perc = np.array(perc)
std_1 = np.array(std_1)

# From the second model
MAE_2 = np.array(MAE_2)
R2_2 = np.array(R2_2)
std_2 = np.array(std_2)

return R2_1, MAE_1, std_1, R2_2, MAE_2, std_2, perc

```

Barchart Plot function:

```

def plot_bar_MAE(i,MAE_26, plot_title):
    fig = plt.figure()
    long_title = '{} for the {} model'

    ax = fig.add_axes([0,0,1,1])
    ax.bar(perc,MAE_26, width =5)
    ax.set_title(long_title.format(reg_name [i],plot_title))
    ax.set_xlabel('percentage(%)')
    if plot_title == 'First':
        ax.set_ylabel('MAE(KN.m)')
    else:
        ax.set_ylabel('MAE(m/h)')
    plt.show()

```

Sequential Spit

```

n = int(0.6*df1_test3.shape[0])

```

```
df_sample = df1_test3.head(n)
df_sample.shape
```

```
# First model:
```

```
# Define independent and dependent variables(x_1 and y_1):
```

```
y_1 = df_sample["Torque[kN.m]"].to_numpy().reshape(-1,1)
x_1 = df_sample[["Depth Bit[m]", 'WOB[kN]', 'Flow rate[L/min]', 'Downhole RPM[rev/min]']
].to_numpy()
```

```
# Second model:
```

```
# Define independent and dependent variables(x_2 and y_2):
```

```
y_2 = df_sample["ROP[m/h]"].to_numpy().reshape(-1,1)
x_2 = x_1.copy()
```

```
# Sequential Split function. Used only one with 60/40 split. Then models are plotted and
performance metrics are calculated:
```

```
def modelling_sequential(test_size):
```

```
    MAE_2 = []
    std_2 = []
    R2_2 = []
    perc = []
    MAE_1 = []
    std_1 = []
    R2_1 = []
```

```
    for i in range(4):
```

```
        # First model:
```

```
        reg=regs[i]
```

```
        # Define train and test samples:
```

```
X_train_1,X_test_1,y_train_1,y_test_1=train_test_split(x_1,y_1,test_size=test_size,random_st
ate=0,shuffle=False)
```

```
    # Fit model
```

```
    reg.fit(X_train_1, y_train_1.ravel())
```

```
    # Prediction
```

```
    y_pred_1 = reg.predict(X_test_1)
```

```
    y_train_fit_1 = reg.predict(X_train_1)
```

```
    # Second model:
```

```
    # Define train and test samples:
```

```
X_train_2,X_test_2,y_train_2,y_test_2=train_test_split(x_2,y_2,test_size=test_size,random_st
ate=0,shuffle=False)
```

```
    # Fit model
```

```
    reg.fit(X_train_2, y_train_2.ravel())
```

```

# Prediction
y_pred_2 = reg.predict(X_test_2)
y_train_fit_2 = reg.predict(X_train_2)

# variables from the first model:
MAE_y_1 = MAE(y_test_1, y_pred_1)
R2_y_1 = r2_score(y_test_1, y_pred_1)
error_1 = np.abs(y_test_1 - y_pred_1)
std_y_1 = np.std(error_1)

# Update lists:
MAE_1.append(MAE_y_1)
std_1.append(std_y_1)
R2_1.append(R2_y_1)

# variables from the second model:
MAE_y_2 = MAE(y_test_2, y_pred_2)
R2_y_2 = r2_score(y_test_2, y_pred_2)
error_2 = np.abs(y_test_2 - y_pred_2)
std_y_2 = np.std(error_2)

# Update lists:
MAE_2.append(MAE_y_2)
std_2.append(std_y_2)
R2_2.append(R2_y_2)

# Create marix for plotting
if i==0:
    y_pred_1_matrix = y_pred_1.reshape(-1,1)
    y_pred_2_matrix = y_pred_2.reshape(-1,1)
else:
    y_pred_1_matrix = np.concatenate((y_pred_1_matrix,y_pred_1.reshape(-1,1)), axis = 1)
    y_pred_2_matrix = np.concatenate((y_pred_2_matrix,y_pred_2.reshape(-1,1)), axis = 1)

# From the first model
MAE_1 = np.array(MAE_1)
R2_1 = np.array(R2_1)
perc = np.array(perc)
std_1 = np.array(std_1)

# From the second model
MAE_2 = np.array(MAE_2)
R2_2 = np.array(R2_2)
std_2 = np.array(std_2)

```

```

return R2_1, MAE_1, std_1, R2_2, MAE_2, std_2, y_pred_1_matrix, y_pred_2_matrix,
y_train_2,y_test_2,y_train_1,y_test_1, X_train_1, X_test_1

```

```

# New file is created to use for optimization and make the optimization faster

```

```

df1_test3.to_csv('file location', index = False, header=True)

```

Optimization part:

```

# Density and viscosity values

```

```

rho = 1.39 # g/cm3

```

```

mu = 28 # mPa.s

```

```

# Plotting and MSE function

```

```

def plotly_vars_depth(df_sample, section):

```

```

    if section == "section 26":

```

```

        Titles = ['ROP[m/h]', 'WOB[kN]', 'Torque[kN.m]', 'Flow rate[L/min]', 'Stand Pipe
Pressure[kPa]', 'Downhole RPM[rev/min]']

```

```

    else:

```

```

        Titles = ['ROP[m/h]', 'WOB[kN]', 'Torque[kN.m]', 'Flow rate[L/min]', 'Stand Pipe
Pressure[kPa]', 'RPM[rev/min]']

```

```

        fig = make_subplots(rows=1, cols=6, subplot_titles=Titles, shared_yaxes = True)

```

```

        for i in range(6):

```

```

            fig.add_trace(go.Scatter(x=df_sample[Titles[i]], y=df_sample["Depth Hole[m]"]),row=1,
col=i+1)

```

```

        for i in fig['layout']['annotations']:

```

```

            i['font']['size'] = 10

```

```

        fig.update_yaxes(autorange="reversed")

```

```

        fig.update_yaxes(title_text = 'Depth Hole[m]',title_font_size=10, row =1, col=1)

```

```

        fig.update_layout(height=800, width=1000, showlegend=False, title="Variables vs. Hole
Depth",)

```

```

        fig.show()

```

```

def MSE(WOB,ROP,N, T, Dbit):

```

```

    # The inputs are:

```

```

    # ROP: rate of penetration [m/s]

```

```

    # N: rotary speed [rev/s]

```

```

    # Dbit: bit diameter [m]

```

```

    # T: Torque [N.m]

```

```

    # WOB: weight on bit [N]

```

```

    # The result MSE is in [j/m3]

```

```

    return (4*WOB*ROP+8*np.pi*N*T)/(np.pi*Dbit**2*ROP)

```

```

# Calculate MSE

```

```

Dbit = 26*0.0254

```

```
df['MSE [J/m3]'] = MSE(df['WOB[kN]']*1000,df['ROP[m/h]']/3600,df['Downhole
RPM[rev/min]']/60, df['Torque[kN.m]']*1000, Dbit)
```

```
# HMSE function:
```

```
Dn = BHA_data.iloc[0, 3]*0.0254 # Nozzle diameter [m]
```

```
Av = 0.15*Dbit**2/(4*Dn**2) # available fluid area[-]
```

```
L = 6*Dbit # Length of potential core
```

```
phi = 3 # angle of axially symmetric jet [degrees]
```

```
s = 8* 0.0254 # distance of nozzle to hole bottom [m]
```

```
M = (Dn+2*L*np.tan(np.radians(phi/2)))/(Dn+s*np.tan(np.radians(phi/2))) # Loss correction
factor[-]
```

```
k = 0.122
```

```
theta = (1-Av**(-k))/(M**2) # Dummy factor for energy reduction
```

```
TFA = 1.362 # Total Flow Area [in2]
```

```
def Fj(rho,Q): # impact force of nozzles to calculate effective WOB [lb]
```

```
    # Q: flow rate[gallons/min]
```

```
    # Vn: average nozzle velocity[ft/s]
```

```
    # rho: mud density[lb-gal]
```

```
    Vn = 0.75*0.3206*Q/(np.pi*0.625**2/4)+0.25*0.3206*Q/(np.pi*0.75**2/4)
```

```
    return 0.000516*rho*Q*Vn
```

```
Fjx = Fj(rho*8.33,df['Flow rate[L/min]']*0.26417) # Jet force [lb]
```

```
WOBe = df['WOB[kN]']*1000*0.225-theta*Fjx # Effective WOB [lb]
```

```
delta_Pn = rho*1000*(df['Flow rate[L/min]']/60000)**2/(2*(TFA*0.00064516)**2*0.95**2) #
```

```
Nozzle pressure loss [Pa]
```

```
def HMSE(WOB,ROP,N, T, Dbit,theta,delta_Pn,Q): # hydraulic Mechanic specific energy
```

```
    # The inputs are:
```

```
    # ROP: rate of penetration [m/h]
```

```
    # N: rotary speed [rev/min]
```

```
    # Dbit: bit diameter [m]
```

```
    # T: Torque [lb-ft]
```

```
    # WOB: weight on bit [N]
```

```
    # theta: Dummy factor for energy reduction
```

```
    # Delta_Pn: Pressure drop in the bit nozzles [psi]
```

```
    # Q: flow rate [gallon/min]
```

```
    # The result MSE is in [psi]
```

```
    return 4*(WOB*ROP+2*60*np.pi*N*T+1155*theta*delta_Pn*Q)/(np.pi*Dbit**2*ROP)
```

```
# We calculate HMSE in J/m3:
```



```
df['HMSE [J/m3]'] = 6894.76*HMSE(WOBe,df['ROP[m/h]']*3.28,df['Downhole RPM[rev/min]'],
df['Torque[kN.m]']*1000*0.738, Dbit/0.0254,theta, delta_Pn/6894.76,df['Flow
rate[L/min]']*0.24617)
```

```
n = int(0.6*df.shape[0])
df_sample =df.head(n)
```

Plot MAE barchart function:

```
def plot_bar_MAE(i,MAE_26, plot_title):
    fig = plt.figure()
    long_title = '{} for the {} model'

    ax = fig.add_axes([0,0,1,1])
    if plot_title =='First':
        if i ==0:
            ax.bar(df_opt_TOR_RF['Depth Bit[m]'],MAE_26, width =5)
        elif i ==1:
            ax.bar(df_opt_TOR_GB['Depth Bit[m]'],MAE_26, width =5)
        elif i ==2:
            ax.bar(df_opt_TOR_AB['Depth Bit[m]'],MAE_26, width =5)
        else:
            ax.bar(df_opt_TOR_KN['Depth Bit[m]'],MAE_26, width =5)
    else:
        if i ==0:
            ax.bar(df_opt_ROP_RF['Depth Bit[m]'],MAE_26, width =5)
        elif i ==1:
            ax.bar(df_opt_ROP_GB['Depth Bit[m]'],MAE_26, width =5)
        elif i ==2:
            ax.bar(df_opt_ROP_AB['Depth Bit[m]'],MAE_26, width =5)
        else:
            ax.bar(df_opt_ROP_KN['Depth Bit[m]'],MAE_26, width =5)
    ax.set_title(long_title.format(reg_name [i],plot_title))
    ax.set_xlabel('Depth Bit[m]')
    if plot_title =='First':
        ax.set_ylabel('MAE(KN.m)')
    else:
        ax.set_ylabel('MAE(m/h)')
    plt.show()
```

Modelling and Optimization variables:

```
# First model:
# Define independent and dependent variables(x_1 and y_1):
y_1 = df_sample["Torque[kN.m]"].to_numpy().reshape(-1,1)
x_1 = df_sample[['Depth Bit[m]','WOB[kN]','Flow rate[L/min]','Downhole RPM[rev/min]']
].to_numpy()
```

```
# Second model:
```

```

# Define independent and dependent variables(x_2 and y_2):
y_2 = df_sample["ROP[m/h]"].to_numpy().reshape(-1,1)
x_2 = x_1

y_HMSE = df_sample["HMSE [J/m3]"].to_numpy().reshape(-1,1)

```

Modelling and Optimization function using DE algorithm:

```

def modelling_optimization(reg_num, num_rows):

```

```

    # measure time consumed:

```

```

    start_time = time.time()

```

```

    # Choose regressor:

```

```

    reg = regs[reg_num]

```

```

    # Lists:

```

```

    ROP_list = []

```

```

    TOR_list = []

```

```

    DB_list = []

```

```

    params_list_ROP = []

```

```

    params_list_TOR = []

```

```

    av_TOR_list = []

```

```

    av_ROP_list = []

```

```

    av_HMSE_list = []

```

```

    end = num_rows

```

```

    init = int(0.1*end)

```

```

    step = int(stop/0.2)

```

```

    MAE_2 = []

```

```

    std_2 = []

```

```

    R2_2 = []

```

```

    perc = []

```

```

    MAE_1 = []

```

```

    std_1 = []

```

```

    R2_1 = []

```

```

    for i in range(init, end ,step):

```

```

        # First model:

```

```

            # Define train and test samples:

```

```

            X_train_1 = x_1[:i]

```

```

            y_train_1 = y_1[:i]

```

```

            X_test_1 = x_1[i:i+step]

```

```

            y_test_1 = y_1[i:i+step]

```

```

            # Fit model

```

```

            reg.fit(X_train_1, y_train_1.ravel())

```

```

            # Prediction

```

```

            y_pred_1 = reg.predict(X_test_1)

```

```

            y_train_fit_1 = reg.predict(X_train_1)

```

```

# Optimization Torque (minimize):
DB = (X_test_1[0,0]+X_test_1[-1,0])/2

def minimize_me_TOR(my_vars):

    WOB = my_vars[0]
    Q = my_vars[1]
    RPM = my_vars[2]

    return reg.predict([[DB,WOB, Q, RPM]])[0]

bounds_TOR = [(10, 200), (3500,5000), (70,180)]
result = differential_evolution(minimize_me_TOR, bounds_TOR)

# variables from the optimization:
TOR_list.append(result.fun) # TOR
params_list_TOR.append(result.x) # WOB,Bit Depth,Torque

# Second model:

# Define train and test samples:
X_train_2 = x_2[:i]
y_train_2 = y_2[:i]
X_test_2 = x_2[i:i+step]
y_test_2 = y_2[i:i+step]

# Fit model
reg.fit(X_train_2, y_train_2.ravel())

# Prediction
y_pred_2 = reg.predict(X_test_2)
y_train_fit_2 = reg.predict(X_train_2)

# variables from the first model:
MAE_y_1 = MAE(y_test_1, y_pred_1)
R2_y_1 = r2_score(y_test_1, y_pred_1)
error_1 = np.abs(y_test_1 - y_pred_1)
std_y_1 = np.std(error_1)
perc.append((y_test_1.shape[0]+y_train_1.shape[0])/end*100)

av_TOR = np.average(y_test_1)
av_HMSE = np.average(y_HMSE[i:i+step])

# Update lists:
MAE_1.append(MAE_y_1)
std_1.append(std_y_1)
R2_1.append(R2_y_1)

```

```

av_TOR_list.append(av_TOR)
av_HMSE_list.append(av_HMSE)

# variables from the second model:
MAE_y_2 = MAE(y_test_2, y_pred_2)
R2_y_2 = r2_score(y_test_2, y_pred_2)
error_2 = np.abs(y_test_2 - y_pred_2)
std_y_2 = np.std(error_2)

av_ROP = np.average(y_test_2)

# Update lists:
MAE_2.append(MAE_y_2)
std_2.append(std_y_2)
R2_2.append(R2_y_2)

av_ROP_list.append(av_ROP)

# Optimization
DB = (X_test_2[0,0]+X_test_2[-1,0])/2
def minimize_me_ROP(my_vars):

    WOB = my_vars[0]
    Q = my_vars[1]
    RPM = my_vars[2]

    return -reg.predict([[ DB,WOB, Q, RPM]])[0]

bounds_ROP = [(10, 200), (3500,5000), (70,180)]
result = differential_evolution(minimize_me_ROP, bounds_ROP)

# variables from the optimization:
ROP_list.append(result.fun) # ROP
params_list_ROP.append(result.x) # WOB,Bit Depth,Torque
DB_list.append(DB)

# From the first model
MAE_1 = np.array(MAE_1)
R2_1 = np.array(R2_1)
perc = np.array(perc)
std_1 = np.array(std_1)

av_TOR = np.array(av_TOR_list)
av_HMSE = np.array(av_HMSE_list)

# From the second model
MAE_2 = np.array(MAE_2)
R2_2 = np.array(R2_2)
std_2 = np.array(std_2)

av_ROP = np.array(av_ROP_list)

```

```

# From optimization:
TORs = np.array(TOR_list)
ROPs = np.array(ROP_list) * (-1)
params_ROP = np.array(params_list_ROP)
params_TOR = np.array(params_list_TOR)
DB = np.array(DB_list)

end_time = time.time()
code_time = end_time - start_time

return R2_1, MAE_1, std_1, R2_2, MAE_2, std_2, perc, ROPs, TORs, params_ROP,
params_TOR, DB, av_TOR, av_ROP, av_HMSE, code_time

# Data frames were created with optimal values of Depth Bit, WOB, Q and RPM, 4 for different
ML ROP model and other 4 for Torque models. Then HMSE was implemented.

# Modelling and Optimization function using PSO algorithm:
def modelling_optimization(reg_num, num_rows):
    # measure time consumed:
    start_time = time.time()
    # Choose regressor:
    reg = regs[reg_num]

    # Lists:
    ROP_list = []
    TOR_list = []
    DB_list = []
    params_list_ROP = []
    params_list_TOR = []
    av_TOR_list = []
    av_ROP_list = []
    av_HMSE_list = []

    end = num_rows
    init = int(0.1*end)
    step = int(stop/0.2)
    MAE_2 = []
    std_2 = []
    R2_2 = []
    perc = []
    MAE_1 = []
    std_1 = []
    R2_1 = []

    for i in range(init, end ,step):
        # First model:

        # Define train and test samples:
        X_train_1 = x_1[:i]
        y_train_1 = y_1[:i]

```

```

X_test_1 = x_1[i:i+step]
y_test_1 = y_1[i:i+step]

# Fit model
reg.fit(X_train_1, y_train_1.ravel())

# Prediction
y_pred_1 = reg.predict(X_test_1)
y_train_fit_1 = reg.predict(X_train_1)

# Optimization Torque (minimize):
DB = (X_test_1[0,0]+X_test_1[-1,0])/2

def minimize_me_TOR(my_vars):

    WOB = my_vars[0]
    Q = my_vars[1]
    RPM = my_vars[2]

    return reg.predict([[DB,WOB, Q, RPM]])[0]

def minimize_me_TOR_helper(x):

    results_TOR = []

    for rows in x:

        results_TOR.append(minimize_me_TOR(rows))

    return results_TOR

# Create bounds
max_bound = np.array([200,5000,180])
min_bound = np.array([10,3500,70])
bounds_TOR = (min_bound, max_bound)
# Initialize swarm
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}

# Call instance of PSO with bounds argument
optimizer = ps.single.GlobalBestPSO(n_particles=10, dimensions=3, options=options,
bounds=bounds_TOR)

# Perform optimization
cost_TOR, pos_TOR = optimizer.optimize(minimize_me_TOR_helper, iters=100, verbose =
2)

# variables from the optimization:
TOR_list.append(cost_TOR) # TOR
params_list_TOR.append(pos_TOR) # WOB, Bit Depth, Torque

```

```

# Second model:

# Define train and test samples:
X_train_2 = x_2[:i]
y_train_2 = y_2[:i]
X_test_2 = x_2[i:i+step]
y_test_2 = y_2[i:i+step]

# Fit model
reg.fit(X_train_2, y_train_2.ravel())

# Prediction
y_pred_2 = reg.predict(X_test_2)
y_train_fit_2 = reg.predict(X_train_2)

# variables from the first model:
MAE_y_1 = MAE(y_test_1, y_pred_1)
R2_y_1 = r2_score(y_test_1, y_pred_1)
error_1 = np.abs(y_test_1 - y_pred_1)
std_y_1 = np.std(error_1)
perc.append((y_test_1.shape[0]+y_train_1.shape[0])/end*100)

av_TOR = np.average(y_test_1)
av_HMSE = np.average(y_HMSE[i:i+step])

# Update lists:
MAE_1.append(MAE_y_1)
std_1.append(std_y_1)
R2_1.append(R2_y_1)

av_TOR_list.append(av_TOR)
av_HMSE_list.append(av_HMSE)

# variables from the second model:
MAE_y_2 = MAE(y_test_2, y_pred_2)
R2_y_2 = r2_score(y_test_2, y_pred_2)
error_2 = np.abs(y_test_2 - y_pred_2)
std_y_2 = np.std(error_2)

av_ROP = np.average(y_test_2)

# Update lists:
MAE_2.append(MAE_y_2)
std_2.append(std_y_2)
R2_2.append(R2_y_2)

av_ROP_list.append(av_ROP)

```

```

# Optimization
DB = (X_test_2[0,0]+X_test_2[-1,0])/2
def minimize_me_ROP(my_vars):

    WOB = my_vars[0]
    Q = my_vars[1]
    RPM = my_vars[2]

    return -reg.predict([[ DB,WOB, Q, RPM]])[0]

def minimize_me_ROP_helper(x):

    results_ROP = []

    for rows in x:

        results_ROP.append(minimize_me_ROP(rows))

    return results_ROP

# Create bounds
bounds_ROP = bounds_TOR
# Initialize swarm
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}

# Call instance of PSO with bounds argument
optimizer = ps.single.GlobalBestPSO(n_particles=10, dimensions=3, options=options,
bounds=bounds_ROP)

# Perform optimization
cost_ROP, pos_ROP = optimizer.optimize(minimize_me_ROP_helper, iters=100, verbose =
2)

# variables from the optimization:
ROP_list.append(cost_ROP) # ROP
params_list_ROP.append(pos_ROP) # WOB,Bit Depth,Torque
DB_list.append(DB)

# From the first model
MAE_1 = np.array(MAE_1)
R2_1 = np.array(R2_1)
perc = np.array(perc)
std_1 = np.array(std_1)

av_TOR = np.array(av_TOR_list)
av_HMSE = np.array(av_HMSE_list)

# From the second model
MAE_2 = np.array(MAE_2)
R2_2 = np.array(R2_2)
std_2 = np.array(std_2)

```



```

av_ROP = np.array(av_ROP_list)

# From optimization:
TORs = np.array(TOR_list)
ROPs = np.array(ROP_list) * (-1)
params_ROP = np.array(params_list_ROP)
params_TOR = np.array(params_list_TOR)
DB = np.array(DB_list)

end_time = time.time()
code_time = end_time - start_time

return R2_1, MAE_1, std_1, R2_2, MAE_2, std_2, perc, ROPs, TORs, params_ROP,
params_TOR, DB, av_TOR, av_ROP, av_HMSE, code_time

# HMSE implementation:

def modelling_HMSE(test_size, i, data_TOR, data_ROP):

    start_time = time.time()

    MAE_2 = []
    std_2 = []
    R2_2 = []
    MAE_1 = []
    std_1 = []
    R2_1 = []

    # First modell (Torque):
    reg=regs[i]
    # Define train and test samples:

X_train_1,X_test_1,y_train_1,y_test_1=train_test_split(x_1,y_1,test_size=test_size,random_st
ate=0,shuffle=False)
    # Fit model
    reg.fit(X_train_1, y_train_1.ravel())

    # Prediction (No prediction of Torque here)
    y_pred_1 = reg.predict(X_test_1)
    # y_train_fit_1 = reg.predict(X_train_1)

    # HMSE and TOR value of Optimized values
    data_TOR['Torque[kN.m]'] = reg.predict(data_TOR[['Depth Bit[m]', 'WOB[kN]', 'Flow
rate[L/min]', 'Downhole RPM[rev/min]' ]]).to_numpy()
    Fji = Fj(rho*8.33,data_TOR['Flow rate[L/min]']*0.26417) # Jet force [lb]
    WObE = data_TOR['WOB[kN]']*1000*0.225-theta*Fji # Efective WOB [lb]

    delta_Pn = rho*1000*(data_TOR['Flow
rate[L/min]']/60000)**2/(2*(TFA*0.00064516)**2*0.95**2) # Nozzle pressure loss [Pa]

```

```

data_TOR['HMSE [J/m3]'] = 6894.76*HMSE(WOBe,data_TOR['Optimal
ROP[m/h]']*3.28,data_TOR['Downhole RPM[rev/min]'],
data_TOR['Torque[kN.m]']*1000*0.738, Dbit/0.0254,theta, delta_Pn/6894.76,data_TOR['Flow
rate[L/min]']*0.24617)
data_TOR['Average HMSE[J/m3]'] = av_HMSE
data_TOR['Average Torque[kN.m]'] = data_TOR['Average Torque[kN.m]']
data_TOR['HMSE rate[%]'] = (data_TOR['HMSE [J/m3]']-data_TOR['Average
HMSE[J/m3]'])/data_TOR['Average HMSE[J/m3]']*100
data_TOR['Torque rate[%]'] = (data_TOR['Torque[kN.m]']-data_TOR['Average
Torque[kN.m]'])/data_TOR['Average Torque[kN.m]']*100

# Second model (ROP):

# Define train and test samples:

X_train_2,X_test_2,y_train_2,y_test_2=train_test_split(x_2,y_2,test_size=test_size,random_st
ate=0,shuffle=False)
# Fit model
reg.fit(X_train_2, y_train_2.ravel())

# Prediction (No prediction of Torque here)
y_pred_2 = reg.predict(X_test_2)
# y_train_fit_2 = reg.predict(X_train_2)

# HMSE and TOR value of Optimized values
data_ROP['ROP[m/h]'] = reg.predict(data_ROP[['Depth Bit[m]','WOB[kN]','Flow
rate[L/min]','Downhole RPM[rev/min]' ]].to_numpy())
Fji = Fj(rho*8.33,data_ROP['Flow rate[L/min]']*0.26417) # Jet force [lb]
WOBe = data_ROP['WOB[kN]']*1000*0.225-theta*Fji # Effective WOB [lb]

delta_Pn = rho*1000*(data_ROP['Flow
rate[L/min]']/60000)**2/(2*(TFA*0.00064516)**2*0.95**2) # Nozzle pressure loss [Pa]
data_ROP['HMSE [J/m3]'] =
6894.76*HMSE(WOBe,data_ROP['ROP[m/h]']*3.28,data_ROP['Downhole RPM[rev/min]'],
data_ROP['Optimal Torque[kN.m]']*1000*0.738, Dbit/0.0254,theta,
delta_Pn/6894.76,data_ROP['Flow rate[L/min]']*0.24617)
data_ROP['Average HMSE[J/m3]'] = av_HMSE
data_ROP['Average ROP[m/h]'] = data_TOR['Average ROP[m/h]']
data_ROP['HMSE rate[%]'] = (data_ROP['HMSE [J/m3]']-data_ROP['Average
HMSE[J/m3]'])/data_ROP['Average HMSE[J/m3]']*100
data_ROP['ROP rate[%]'] = (data_ROP['ROP[m/h]']-data_ROP['Average
ROP[m/h]'])/data_ROP['Average ROP[m/h]']*100

# variables from the first model:
MAE_y_1 = MAE(y_test_1, y_pred_1)
R2_y_1 = r2_score(y_test_1, y_pred_1)
error_1 = np.abs(y_test_1 - y_pred_1)
std_y_1 = np.std(error_1)

# Update lists:

```

```

MAE_1.append(MAE_y_1)
std_1.append(std_y_1)
R2_1.append(R2_y_1)

# Variables from the second model:
MAE_y_2 = MAE(y_test_2, y_pred_2)
R2_y_2 = r2_score(y_test_2, y_pred_2)
error_2 = np.abs(y_test_2 - y_pred_2)
std_y_2 = np.std(error_2)

# Update lists:
MAE_2.append(MAE_y_2)
std_2.append(std_y_2)
R2_2.append(R2_y_2)

# From the first model
MAE_1 = np.array(MAE_1)
R2_1 = np.array(R2_1)
std_1 = np.array(std_1)

# From the second model
MAE_2 = np.array(MAE_2)
R2_2 = np.array(R2_2)
std_2 = np.array(std_2)

end_time = time.time()
code_time = end_time - start_time

return code_time

```

Then increments were plotted as it is showed in the results section. This part of the code is not available in this ap