Sonic log depth series predictions using machine learning algorithms

by

© Bahare Zare

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Master of Science.

Supervisor: Dr. Hamid Usefi

Department of Computer Science

Memorial University of Newfoundland

May 2024

St. John's, Newfoundland and Labrador, Canada

Abstract

This paper investigates the viability of predicting rock properties within wells using real-time drilling data. Among these properties, the sonic log plays a crucial role in understanding the physical characteristics of subsurface formations and helps geoscientists and drilling engineers interpret the subsurface geology and make informed decisions about well construction, drilling parameters, and reservoir performance. Successfully forecasting sonic logs has the potential to significantly improve the optimization of fracturing processes in wells with similar geological structures. To accomplish this objective, we introduced a well-structured eXtreme Gradient Boosting (XGBoost), LSTM (Long Short-Term Memory), and Random Forest (RF) models that utilize depth-series data for predicting sonic log in the field of oil and gas exploration. The data used in this research was gathered from the drilling project "A Data Analytics Approach to Energy and Safety Improvements" which received funding from the NL Offshore Oil and Gas Industry Recovery Assistance Fund.

Acknowledgments

I wish to convey my most sincere appreciation to Dr. Hamid Usefi for his invaluable mentorship and support throughout this project. His feedback and constructive critique have played a pivotal role in my personal and professional development. I would also like to extend my gratitude to the Department of Computer Science at Memorial University and the Drilling Research Group at the Process Engineering Department, particularly Dr. Lesley James and Dr. Mohammad Mojammel Huque, for their guidance, support and for providing me with the opportunity to pursue this research. Also, the funding was received from the NL Offshore Oil and Gas Industry Recovery Assistance Fund through the effort of Dr. Lesley James.

My heartfelt thanks go to my husband, Naser, for always being a listening ear and for his sense of humor during challenging moments. I also appreciate the sacrifices he has made, enabling me to pursue my master's degree.

I am deeply thankful to my family, especially my younger brother, Mohammadreza, who has been a constant source of support; I am genuinely appreciative of their unwavering encouragement.

In conclusion, I dedicate this work to my family, whose role in shaping the person I am today is indispensable. Their love, support, and encouragement have consistently provided me with strength; for that, I am eternally grateful.

Statement of contribution

The following delineates the contributions made to this project. My contribution involved preprocessing of data and feature selection, implementing analytical approaches, and enhancing machine learning algorithms(*eXtreme Gradient Boosting (*XGBoost*)*, *Random Forest (*RF*)*, and *Long Short-Term Memory (*LSTM*)*) to address sonic log prediction. Dr. Mohammad Mojammel Huque provided the dataset. He helped to identify the most critical features, eliminate the redundant and irrelevant ones, and reduce noise in the signals. Prof. Hamid Usefi, my supervisor, assisted in establishing methodologies and predicting sonic log data. Furthermore, Dr. Usefi identified the issue of shuffled data in time series data splitting. Additionally, he contributed to validating results and engaging in discussions on the outcomes.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN Artificial Neural Network

CNN Convolutional Neural Network

DNN Deep Neural Network

DTCO Delta T sonic Compressional

DTCO-MH-R compressional-wave slowness

DSP Digital Signal Processing

EWMA Exponential Weighted Moving Average

FIR Finite Impulse Response

GBM Gradient Boosting machine

GPM Gallons per Minute

GR Gamma Ray

IQR Interquartile Range

IIR Infinite Impulse Respons

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MLP Multilayer Perceptron

MSE Mean Squared Error

NMR Nuclear Magnetic Resonance

NN Neural Network

PSO Particle Swarm Optimization

RBF Radial Basis Function

RF Random Forest

RNN Recurrent Neural Network

RPM Rotational Speed

ROP Rate Of Penetration

ROP5 Rate Of Penetration Averaged Over the Last 5 foot

RMSE Root Mean Squared Error

SPPA Stand Pipe Pressure

SMSE Surface Mechanical Specific Energy

SVM Support Vector Machine

STOR Surface Torque

SWOB Surface Weight On Bit

TFLO Total Flow Rate

TOB Torque On Bit

TVD True Vertical Depth

WOB weight-on-bit

XGBoost eXtreme Gradient Boosting

R^2 Coefficient of determination

Chapter 1

Introduction

Well logging data is crucial for deducing the physical and chemical characteristics of rock formations. This information enables geologists and engineers to gain insights into the geological composition of the region, facilitating informed decisions about optimal resource extraction methods, such as for oil and gas. Many well logging tools and techniques exist, each designed to measure distinct properties of rock formations. Common well logging measurements include:

- **Sonic Logs**: These logs measure the P-wave travel time versus depth, recorded in microseconds per meter (ms/n), indicating the speed of acoustic waves through rock formations [1].

- **Resistivity Logs**: These electrical logs record the formation's resistivity at shallow, medium, and deep depths, inferring porosity, water saturation, and hydrocarbon presence [1, 2].

- **Density Logs**: These logs measure neutron scattering to determine rock density [3].
- **Porosity Logs**: These logs measure the open space within the rock formation [4].
- **Gamma-Ray Logs**: These logs measure the natural radiation emitted by the rock formation [5].

The primary objective of this research is to develop a predictive model for sonic logs using the extensive drilling parameters available in our dataset. These parameters include resistivity, density, porosity, and gamma-ray logging, which are used to understand formation properties comprehensively. Various drilling parameters are accessible for each well within our dataset. However, sonic logs are selectively acquired and not universally available across all wells. More data is gathered during exploratory drilling activities. Drilling parameters such as penetration rate, weight on bit, bit size, rotational speed, torque, flow rate, and mechanical specific energy are not currently correlated with wireline data such as density, porosity, and sonic logs. By analyzing these parameters, we aim to predict sonic logs accurately, facilitating better understanding and decision-making regarding rock formations. Understanding the travel time of sound waves recorded in sonic logs is essential for comprehending the composition and structure of rock formations. The development of a predictive model using drilling parameters will enhance the ability to make informed decisions about resource extraction, even when direct sonic log data is not available. How-

ever, wireline logs are statistically sensitive to drilling parameters. This results in a non-linear relationship between drilling parameters, density, and porosity. Also, drilling parameters are expected to be highly interactive with each other. In other words, changes in some drilling parameters, such as pump pressure, will affect other drilling parameters, such as flow and penetration rates. Since the relationship between drilling parameters and wireline logs, specifically sonic logs, is non-linear and interactive, we investigate its deep learning potential [6]. Based on other research, it is evident that intelligent systems, including traditional machine learning methods and deep learning-based approaches, have a distinct advantage in solving geophysical problems. Several types of studies have been conducted, from simple *Artificial Neural Network (*ANN*)*, support vector regression models, genetic algorithms, and fuzzy logic, to *Convolutional Neural Network (*CNN*)*, LSTM, time-reversing algorithms, and non-linear autoregressive methods with exogenous inputs [7, 8, 9].

1.1 Workflow

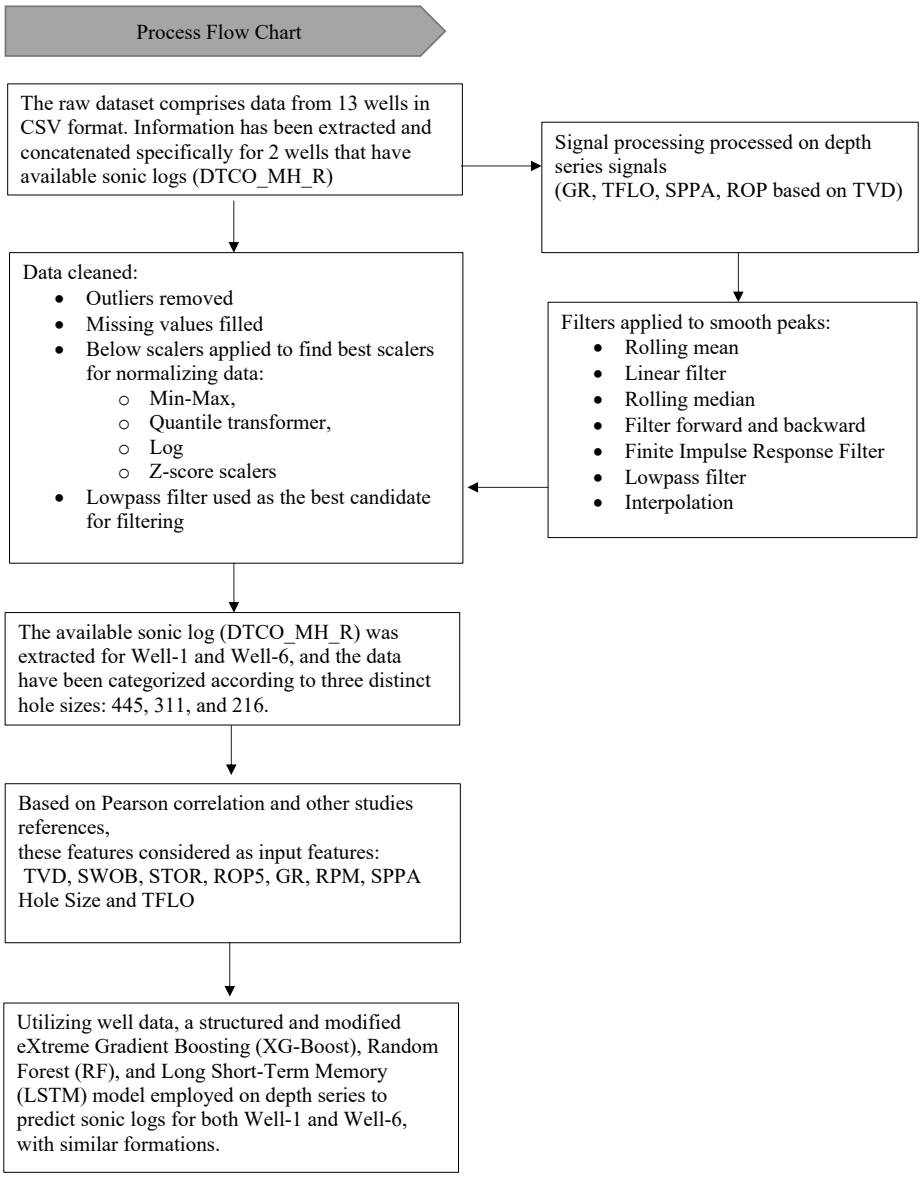


Figure 1.1: Workflow

Features	Description	Features	Description
TVD	True Vertical Depth	A40H	Acoustic Impedance at 40 Hz
DEPT	Depth	BS	Bit Size
GR	Gamma Ray	CHSH_QPINV	Caliper Shale
SPPA	Stand Pipe Pressure	DEVI	Deviation
RPM	Rotations Per Minute	DRHB	Density of Rock Head
STOR	Storage	DTCO_MH_R	Compressional Sonic Travel Time
SWOB	Surface Weight on Bit	DTSH_QPINV	Shear Sonic Travel Time
TFLO	Total Flow	FRQMAX_QPINV	Maximum Frequency
ROP5	Rate of Penetration (5-minute average)	FRQMIN_QPINV	Minimum Frequency
SMSE	Surface Mechanical Specific Energy	P16H	Pressure at 16 Hz
CRPM	Corrected Rotations Per Minute	P28H	Pressure at 28 Hz
TRPM	True Rotations Per Minute	P40H	Pressure at 40 Hz
DHAP	Downhole Axial Pressure	PR	Pressure Ratio
DHAT	Downhole Axial Temperature	ROBB	Rotary Bit
MWTI	Mud Weight In	TICO_MH_R6	Inline Compressional Time
ECD	Equivalent Circulating Density	TISH_QPINV_R6	Inline Shear Time
HoleSize	Size of the Hole	TNPH	Thermal Neutron Porosity
Geological Formation	Geological Formation Information	VPVS	Velocity Ratio of P-wave to S-wave
WellName	Name of the Well	DTSH_MH_R	Shear Sonic Travel Time (Method H)
PEB	Photoelectric Effect	DTCO_INV	Inversion of Compressional Sonic Travel Time
UCS	Unconfined Compressive Strength		

Table 1.1: Well-1 dataset log types and descriptions

Figure 1.1 illustrates the workflow implemented in this study. The dataset comprises 13 wells labeled as Well-1, Well-2, etc. Each dataset consists of 49 features shown in Table 1.1. However, it is important to note that sonic log data is only available for Well-1 and Well-6. In data preprocessing, we undertook a series of crucial operations to refine our dataset. Our initial task involved identifying and excluding outliers or

data points that deviate significantly from the norm. This step was imperative to uphold our data's overall quality and reliability. Subsequently, we performed a thorough quality check on our dataset to confirm its integrity and address any discrepancies. To make our data more consistent and suitable for predictive modeling, we ventured into data normalization and scaling. During this process, we explored various methods to adjust the scale of the data and carefully evaluated their performance. Notably, the Min-Max scaler emerged as the most effective in producing desirable outcomes. The rationale behind implementing scalers is their capacity to enhance our confidence in the data's suitability for predictive and analytical purposes. Towards the final stages of our data preparation, we conducted feature selection and analysis, a crucial step for identifying the most pertinent aspects of our dataset. This process was applied to data samples of Well-1 and Well-6. Following comprehensive analyses involving P-value calculations, Pearson correlation coefficient (discussed in Section 5.2) assessments, and a review of relevant research, it was concluded that the most effective set of attributes for predicting the sonic log comprises ten features. Therefore, the dataset's dimensionality from forty-nine features was reduced to ten. These variables (described in Table 1.2) are GR, TFLO, Hole Size, *Rotational Speed (*RPM*)*, ROP5, SPPA, SWOB, TVD, *Surface Torque (*STOR*)*, and *compressional-wave slowness (*DTCO-MH-R*)*. Importantly, these attributes are present and observed in both Well-1 and Well-6.

Therefore, we will focus on using nine features, including GR, TFLO, Hole Size,

RPM, ROP5, SPPA, SWOB, TVD, and STOR as input features and our target is predicting sonic log, identified as a DTCO-MH-R.

For reference, a comprehensive list of the abbreviations and their corresponding full names of selected features are presented in Table 1.2.

Table 1.2: List of important variables in Well-1 and Well-6.

Features	Description of Features	Units
DEPT	Depth Index/Measured Depth	m
SPPA	Standpipe Pressure	kPa
RPM	Rotational Speed	c/min
STOR	Surface Torque	kN.m
SWOB	Surface Weight On Bit	1000 kgf
ROP5	Rate of Penetration Averaged Over the Last 5 ft	m/h
TFLO	Total Flow Rate of All Active Pumps	L/min
GR	Gamma Ray	gAPI
TVD	True Vertical Depth	m
SMSE	Surface Mechanical Specific Energy	kPa
DTCO	Delta T Sonic Compressional	ms/m

Another critical facet of preprocessing is signal processing, which entails transforming the data and filtering out irrelevant noise. In our study, we concentrated on

the efficacy of different filtering techniques for smoothing peaks in the data.

Our project employed a supervised learning model, specifically XGBoost and RF, alongside LSTM to predict the sonic log. We conducted hyperparameter tuning to optimize accuracy.

It is crucial to note that, when leveraging XGBoost or RF, powerful predictive modeling tools, we ensured that the data remained ordered by depth. This approach is vital as time series algorithms should handle sequential data, preserving the temporal order for accurate analysis and prediction. All components of data analysis and model development were executed using Python.

The remainder of this document adheres to a structured outline. Chapter 2 explores prior research relevant to our work, providing context for the topics discussed in this thesis. Chapter 3 offers a more detailed overview of signal processing, while Chapter 4 delves into data processing. Chapter 5 serves as a methodology section where we examine utilizing the XGBoost, RF, and LSTM models with depth series data in this study. This chapter also reveals the results obtained from the models. Finally, Chapter 6 concludes the thesis and outlines potential directions for future research.

Chapter 2

Background and Related Work

This chapter provides a comprehensive overview of the foundational concepts and prior research relevant to this study. We begin with the objectives of the research, outlining the key goals and motivations driving this work. Following this, we delve into various machine learning algorithms, focusing on those particularly pertinent to our study, such as RF and XGBoost, which are renowned for their robustness and accuracy in handling complex datasets. We also explore LSTM networks, which are effective for sequential data analysis. In the context of our research, we place special emphasis on sonic log data and discuss how previous papers have utilized machine learning techniques to interpret these data types. Additionally, we examine the critical process of hyperparameter tuning, which is essential for optimizing the performance of machine learning models. Finally, we review previous studies, highlighting the methodologies, findings, and gaps in the existing literature that our work

aims to address. This background sets the stage for understanding the current state of knowledge and how our research contributes to advancing the field.

Sonic log prediction is crucial for the petroleum industry's cost-effective and efficient operation. In well logging, electrical or radioactive signals are sent into rock formations; they are continuous records of various physical properties at depth. Log parameters can help evaluate a zone and determine whether a well-completion attempt is warranted. In this work, our focus is on applying machine learning methods to predict the sonic log. The relationship between energy demand prediction and well logging highlights the importance of accurate and reliable data for informed decision-making in the energy industry. Machine learning as a tool proposes promising results in different fields for predicting sonic logs based on important features. Through a review of various articles, it becomes evident that the algorithms most frequently explored and analyzed are *Neural Network (*NN*)*, LSTM, *Recurrent Neural Network (*RNN*)*, and XGBoost. In essence, we have provided a concise introduction to these prominent algorithms.

2.1 Objective

The primary objective of this study is to predict the compressional sonic log using machine learning algorithms, namely LSTM, RF, and XGBoost, utilizing commonly acquired well logs. Additionally, the study aimed to assess the effectiveness, particularly by comparing the performance and errors of LSTM with RF and XGBoost.

The study emphasized the importance of data preprocessing, relevant input parameters, optimized hyperparameter selection, and Grid Search cross-validation to ensure confidence in the predictions made by the ML algorithms [10, 11].

2.2 Machine learning algorithms

A neural network is a robust computational tool characterized by its neurons, activation functions, biases, and corresponding weights. In comparison to regression models, it stands out as a potent model for pattern recognition. By utilizing non-linear functions that iterate within the network, it can effectively capture intricate relationships between input and output variables. Moreover, it exhibits resilience in the presence of missing or imprecise data. ANN model is a widely adopted machine learning algorithm for both linear and non-linear regression tasks. The ANN model comprises three distinct layers; an input layer, one or more hidden layers, and an output layer. These layers are constructed from an array of nodes and neurons. In its early design, the ANN operated as a feedforward neural network, enabling the flow of information exclusively from the input layer to the output layer. The ANN algorithm works to determine an optimal set of weights for each layer of neurons in each interface, ultimately resulting in a refined set of optimized weights [12].

XGBoost is a popular and powerful machine learning algorithm known for its efficiency and effectiveness, particularly in structured data and tabular data tasks. It is based on the gradient boosting framework and has gained prominence in ma-

chine learning competitions, real-world applications, and various data science tasks. It combines multiple decision trees, gradually improving model performance [13]. XGBoost optimizes an objective function through gradient descent, incorporating regularization to prevent overfitting. It offers valuable feature importance insights, handles missing data efficiently, and supports parallel processing. This algorithm is well-suited for structured data, allowing for powerful predictive modeling, and is a top choice in various fields due to its speed and accuracy. It is often the preferred choice for structured data problems, especially in situations where large datasets are involved. XGBoost uses parallel computing to build trees by utilizing all available CPUs during the training process. Unlike traditional methods, XGBoost employs a "max depth" parameter as the stopping criterion and applies tree pruning in a backward direction, resulting in improved computational efficiency and faster processing times compared to other *Gradient Boosting machine (*GBM*)* frameworks [14]. Additionally, XGBoost can automatically learn the best way to handle missing values based on the training loss, allowing it to handle different types of sparsity patterns in the input data more effectively.

Random Forest is an ensemble learning method for classification, and regression that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned [15, 16]. We aim to use the RF regression process, which involves several key steps.

This model randomly samples the training data with replacement (bootstrapping) to create multiple subsets of the data. Each subset is used to train an individual decision tree. At each node of the decision tree, it randomly selects a subset of features from the total set of features. This introduces diversity among the trees and helps prevent overfitting. For each bootstrapped sample and random feature subset, it builds an individual decision tree. The trees are constructed recursively by splitting nodes based on the selected features until a stopping criterion is met (e.g., maximum depth, minimum samples) per leaf. For classification tasks, each tree votes for a class, and the class with the majority of votes is assigned as the final prediction. For regression tasks, the predictions of individual trees are averaged to obtain the final prediction. Then it assesses the importance of each feature by measuring how much each feature contributes to the reduction in impurity or information gain across all trees. Features that are frequently used for splitting nodes are considered more important. The fundamental idea behind Random Forest is that by combining a large number of diverse and slightly overfitting decision trees, the overall model becomes more robust and generalizes well to unseen data. The randomness introduced at both the instance and feature levels helps in decorrelating the trees, reducing the risk of overfitting, and improving the model's accuracy and stability. In summary, this method is known for its flexibility, ease of use, and ability to handle high-dimensional data with many features. It is less prone to overfitting compared to individual decision trees, making it a powerful and widely used algorithm in practice. Additionally, the parallelization

capabilities of Random Forest make it computationally efficient, allowing it to handle large datasets efficiently.

LSTM is a special RNN that specializes in sequential data [17]. Unlike *Deep Neural Network (*DNN*)*, a fully connected network system, RNN has a loop. The loop transfers the information from the previous step to the next step so that the previous information can be used for the next data. RNN outperforms ANN models when it comes to processing sequential data. However, RNNs are known to have a significant decrease in learning ability if the distance between the relevant information in the previous step and the current step increases. LSTM was introduced to deal with this vanishing gradient problem. It has three gate units (input, output, and forget). This model can effectively maintain the long-range sequence information of the input data by using the internal network of cell states.

In our research, we employed XGBoost, Random Forest, and LSTM to forecast the sonic log. We also conducted hyperparameter tuning to optimize our model. To assess the performance, we partitioned the data, with the majority allocated for training (90 percent) and the remainder for validation. This section provides a summary of prior research endeavors that focused on examining the viability of forecasting subsurface rock properties within wells using real-time drilling data. Researchers introduced innovative deep learning or machine learning models, which enabled them to recognize various rock hydrate structures and gauge gas hydrate saturations at varying depths in the wells. Additionally, certain studies adopted more preprocessing steps

to enhance the accuracy of their predictions.

2.3 Previous studies

Chiranth and Ken [18] demonstrated real-world applications of machine learning in drilling engineering, exploring practical techniques such as Random Forest. In their studies, a notable application involved utilizing Random Forest to forecast across diverse formations and lithologies. Through the incorporation of feature engineering, they observed enhanced model performance, resulting in more accurate and improved predictions in drilling operations.

Another application they employed was the optimization of ROP for drilling a well, using machine learning and data analytics to predict ROP while drilling. By modifying surface parameters such as *weight-on-bit (*WOB*)*, rotary speed, and flow rate, they asserted that ROP could increase over different lengths of the well. They effectively demonstrated their approach over 25, 50, and 100 feet.

Also, Chiranth and Ken [19], evaluated different strategies for drilling optimization, emphasizing the importance of considering multiple factors such as ROP, *Torque On Bit (*TOB*)*, and mechanical specific energy. Utilizing a Random Forest algorithm, they developed models for predicting ROP, TOB, and mechanical specific energy using WOB, flow rate, rotary speed, and rock strength as input features. Using mechanical specific energy as an objective function resulted in a balanced improvement in drilling, with an increase in ROP and a reduction in mechanical specific

energy and torque, potentially extending the bit's longevity. Overall, these practical applications of machine learning in drilling engineering demonstrated its potential to enhance drilling performance and efficiency.

Luís Felipe et al. [20] conducted a comprehensive review of the literature on ROP prediction with a particular focus on machine learning techniques. In their review, they discussed various strategies employed to optimize the performance of these models using data from the literature. The authors delved into a range of machine learning techniques and approaches, including non-iterative algorithms, *Support Vector Machine (*SVM*)*, Fuzzy System Inference (FIS), neuro-fuzzy, and ensemble models. They highlighted the importance of ensemble models in enhancing the predictive power of weak learners by combining their predictions. The article also explained the differences between homogeneous and heterogeneous ensemble models and provided examples of each. Finally, the paper briefly mentioned specific algorithms, such as Random Forests and Gradient Boosting Machines, that had been studied in ROP research.

Mohammad et al. [21] introduced a precise predictive model ROP in drilling operations. Constructed through machine learning algorithms, the model utilized data derived from mud logs and wireline logs. The initial phase involved the application of various statistical models, including those fine-tuned with a Genetic Algorithm (GA), to establish baseline predictions. Subsequently, the performance of diverse ANN was evaluated against these baseline predictions. The research revealed that

incorporating petrophysical logs, particularly considering the geological formations in conjunction with drilling parameters, substantially enhanced the predictive capabilities of the machine learning algorithms. Notably, *Multilayer Perceptron (*MLP*)* training technique that utilizes a *Particle Swarm Optimization (*PSO*)* algorithm, the hybrid MLP-PSO, emerged as the most effective model, surpassing the performance of other algorithms in the study. Furthermore, the study demonstrated the superiority of SVR, MLP, radial basis function artificial neural network (RBF-ANN), and the hybrid MLP-PSO model in comparison to Decision Tree and Random Forest algorithms when applied to pre-processed data. The study also highlighted the benefits of employing the Savitzky-Golay (SG) filter as a noise reduction method, leading to improved predictive performance, particularly in the case of the SVR model.

Jiachun et al. [22] introduced an inventive strategy for predicting Shear Wave Velocity (V_s) curves within wells situated in Alaminos Canyon Block 21. They achieved this by harnessing the capabilities of the LSTM neural network. Notably, this investigation was conducted in circumstances where V_s log measurements were limited in availability. The LSTM model was meticulously constructed using input features such as gamma-ray, density, porosity, Compressional Wave Velocity (V_p), and resistivity to forecast V_s. The well-logging curves depicted traditional time or depth sequences, capturing the changes in stratigraphic sequences. Specialized deep learning models, such as RNN and LSTM models designed for handling sequences, were utilized in this context. The results of the study revealed that the LSTM method surpassed

the performance of the Linear Support Vector method in predicting V_s curves. Additionally, the research delved into the concept of transfer learning, an approach that efficiently optimized a new NN model by leveraging knowledge from a pre-trained model. This study further illustrated the effectiveness of using a pre-trained LSTM model and transfer learning to attain highly accurate predictions of V_s values.

Muhammad Ali et al. [23] introduced an innovative technique for the prediction of missing shear sonic logs utilizing machine learning and DNN. This approach involved analyzing patterns of similarity, utilizing metrics such as Jaccard and overlap similarities among wells that shared similar geophysical characteristics. The aim was to achieve precise and detailed predictions for missing log data. Moreover, it is important to highlight that this methodology showed promise for extension to predicting absent density and sonic logs across diverse well locations.

Hany et al. [11] presented a novel approach using machine learning techniques like Random Forest and Decision Tree to predict sonic data based on surface drilling parameters. The research showed that both Random Forest and Decision Tree could accurately predict sonic slowness. The model utilized various drilling parameters, including WOB, SPPA, RPM, and ROP, with the most significant hyperparameters being max-depth and min-samples-split. During the testing and training phases, the Random Forest model proved superior to the Decision Tree model.

Ruizhi et al. [17] underscored the increased utilization of machine learning in the oil and gas sector, particularly in drilling applications. They highlighted that ANN,

SVM, and Bayesian Networks (BN) stood out as the most commonly employed algorithms in well control investigations. The authors also pointed out that drilling operations generated a substantial amount of data, necessitating thorough quality checks, and emphasized the criticality of real-time implementation for tasks like determining drilling fluid properties, optimizing drilling processes, and detecting drilling issues. Input parameters for machine learning models encompassed drilling parameters, fluid properties, well details, and formation properties. The authors cautioned that the reliability of model outcomes might be influenced by noisy drilling data and the random selection of training and testing samples. Furthermore, they illustrated that while ANN was the most frequently used machine learning method, advanced algorithms such as Gradient Boosting Trees and time series machine learning could offer superior performance.

Jongkook's study [12] demonstrated a novel method that combined machine learning techniques to create synthetic sonic logs, known for their high acquisition costs. This involved inputting data from five wireline logs into three supervised machine-learning models, alongside the use of unsupervised learning through data clustering and PSO to improve model accuracy and identify optimal hyperparameters. The study revealed that the hybrid approach proved to be more dependable and effective in generating synthetic logs for petrophysical or geomechanical purposes. It was suggested for practical use based on its favorable outcomes.

Ammar et al. [24] presented an innovative data-driven hybrid system designed to

enhance drilling efficiency. This system consisted of two crucial phases. The initial phase involved querying geological data, while the subsequent phase focused on making real-time adjustments to controllable dynamic drilling parameters. The model provided recommendations for optimizing these parameters, including WOB, ROP, and *Gallons per Minute (*GPM*)*, and computed the optimal ROP parameter for each specific well. The connection between ROP and these controllable dynamic drilling parameters was typically represented by a specific equation. Diverging from conventional ROP optimization practices that involved selecting fixed values for each controllable dynamic drilling parameter and adjusting them on a per-section basis, the paper introduced a novel approach to ROP optimization. It leveraged both dynamic and static drilling data available in real time. The authors proposed a comprehensive two-phase, data-driven system that utilized existing data to model optimal drilling practices. These practices were then compared against real-time data from a live well to identify the optimal ROP. Finally, the researchers analyzed the relationship between controllable dynamic parameters such as WOB, RPM, GPM, and ROP. They derived an empirical relationship based on ANN.

Mohsen et al. [25] asserted that the reduction of drilling time and cost was achievable through the optimization of drilling variables and operational parameters. Their investigation focused on ROP as a pivotal determinant in drilling duration. This parameter could be influenced by a range of factors, including drilling mud properties, formation characteristics, rotary speed, and bit attributes. In their research, scholars

put forth models and methodologies encompassing regression analysis and machine learning algorithms, such as ANN, for the prediction of ROP. They harnessed the Generalized Reduced Gradient (GRG) technique and Decision Trees to tackle multi-variable challenges and establish models. Moreover, they introduced the *Radial Basis Function (*RBF*)* neural network as a more straightforward model compared to MLP, employing Gaussian functions for transfer functions. The proposed RBF model featured two spread coefficients and 100 neurons, while optimization algorithms were utilized to enhance drilling efficiency.

In 2023, Liu et al. [10] presented an integrated approach that merged the XGBoost technique with the PSO method. The objective was to predict *Nuclear Magnetic Resonance (*NMR*)* log parameters based on conventional petrophysical logs, with a primary focus on accurately forecasting NMR logging responses using cost-effective logging data. The research utilized data from conventional well logs, including neutron, density, sonic, caliper, gamma ray, and resistivity logs, obtained from sixteen wells in an offshore oilfield located in the Persian Gulf. In this context, the XGBoost-PSO model generated outputs related to free fluid porosity, bound fluid porosity, permeability, and total porosity. The outcomes revealed that the XGBoost-PSO model provided predictions with an accuracy range of 88.5% to 91.4%. Notably, the application of PSO for hyperparameter optimization enhanced the predicted parameters by a minimum of five percent. This study underscored the feasibility of utilizing standard conventional logging data to derive advanced NMR information, thereby

reducing operational costs while retaining the benefits associated with such data.

In 2023, Callistus et al. [26] undertook the task of predicting the compressional sonic log within the Tano basin of Ghana. They employed machine learning algorithms and conducted a comparative analysis of their performances. This study utilized three distinct machine learning algorithms; SVM, RF, and XGBoost. These algorithms were applied to forecast the compressional sonic log using data obtained from commonly collected logs, including gamma-ray, resistivity, density, and neutron-porosity. To gauge the effectiveness of these algorithms, they underwent training and testing using data from two wells. Subsequently, the models were applied to a third well for predicting the sonic log. Evaluation of the algorithm performances was conducted using statistical metrics such as *Coefficient of determination (R^2)*, *Mean Squared Error (*MSE*)*, MAE, and *Root Mean Squared Error (*RMSE*)*. The results unveiled that XGBoost exhibited the highest degree of prediction accuracy, followed by RF, while SVM displayed the lowest level of accuracy. This research significantly contributed to enhancing the understanding of oil and gas fields, especially in regions like the Ghanaian sedimentary basin and the broader West African sub-region, where compressional sonic logs are scarce or entirely unavailable.

Drawing from the insights provided by this comprehensive literature review, it is evident that a wide range of investigations have delved into the significance of employing various combinations of input data, filtering techniques, and hyperparameter tuning. This extensive body of research underscores the critical role these elements play.

In this section, we have showcased the existing works within the realm of oil and gas that leverage machine learning methodologies. We have taken note of the notable endeavors that have already advanced the state of knowledge in this domain. Subsequently, in Chapter 5, by considering the comparison between RF, XGBoost, and LSTM models, we will elucidate a proper model utilizing time series data in oil and gas field to predict sonic log for unseen data more accurately building upon this existing body of work.

The subsequent chapter represents an independent endeavor to minimize noise for upcoming studies. Our goal is to utilize the insights from Chapter 3 to enhance signal smoothness, eliminate certain outliers stemming from oil and gas activities, and improve overall accuracy.

Chapter 3

Signal Processing

Signal processing is a critical step in the analysis and interpretation of data across various domains. It involves techniques that enhance the quality of signals, making it easier to extract meaningful information. This chapter provides a comprehensive overview of fundamental signal processing methods, starting with the rolling mean, which smooths out short-term fluctuations to reveal long-term trends. We then explore linear filters that modify signals by emphasizing certain frequencies, followed by the rolling median, which effectively reduces noise while preserving significant edges in the data. The chapter also covers advanced filtering techniques such as forward and backward filters that prevent phase distortion, and finite impulse response (FIR) filters known for their stability and precise frequency response. Further, we delve into low-pass filters, including the Butterworth filter with its flat passband response and the Fourier transform-based filter, which operates in the frequency domain. In-

terpolation methods, which estimate intermediate data points, are also discussed. Finally, a comparative analysis of these filters helps understand their applications and effectiveness in different scenarios.

Signal processing pertains to the analysis of data frequency over time, with common techniques including filtering to remove unwanted frequencies or noise, compression to reduce signal size, and modulation for encoding information into a signal for transmission. In our investigation, we examine various features, such as gamma rays based on TVD, and observe the impact of different filtering approaches in both short and large intervals.

Data smoothing can be achieved through diverse methods, including randomization, utilizing a random walk, calculating a moving average, or employing various exponential smoothing techniques. Smoothing algorithms are categorized as either global or local, depending on whether we filter noise across the entire series or a smaller segment. This involves summarizing data within a local or global TVD domain, resulting in a smooth estimation of the underlying data. For example, exponential smoothing, employs a simple average calculation with exponentially decreasing weights, starting from the most recent observations.

In this segment of the study, we have conducted preparation steps using Well-7 and Well-10. Leveraging the similarity of these two wells based on formation, a data frame concentrating on a specific portion (indexes 5000 to 10000) was employed. Additionally, in the final section of this chapter, Well-6 is included to compare filters

and discuss the obtained results.

3.1 Rolling mean

A rolling mean, also known as a moving average, is a statistical calculation used to analyze and smooth fluctuations in time series data. It involves calculating the mean of a subset of data points within a moving or rolling window that progresses through the dataset. The filter operates in three sequential steps. Firstly, it employs a moving window approach, where a window of a specified size traverses through the dataset one step at a time. At each step, the data points within the window are utilized to compute the mean. Subsequently, the mean is calculated for each position of the moving window based on the values within that window. This process yields a snapshot of the average value over a specific period. Ultimately, the rolling mean serves to smooth out short-term fluctuations or noise in the data, offering a clearer depiction of the underlying trends or patterns. The pseudocode for this methodology is presented below in Algorithm 1.

Algorithm 1 Rolling Mean Calculation

```
1: function ROLLING_MEAN(data, window_size)
2:     rolling_means ← []
3:     window ← []
4:     for i ← 0 to length(data) – 1 do
5:         append data[i] to window
6:         if length(window) > window_size then
7:             pop the oldest value from window
8:         mean ← calculate_mean(window)
9:         append mean to rolling_means
10:    return rolling_means
11: function CALCULATE_MEAN(values)
12:    sum ← 0
13:    for value in values do
14:        sum ← sum + value
15:    return sum/length(values)
```

The selection of the window size is a crucial decision that relies on the data's characteristics and the specific goals of the analysis. Opting for larger window sizes leads to smoother trends but may potentially mask short-term variations. On the other hand, smaller window sizes capture more detailed variations but might be susceptible

to noise. This filter finds extensive applications across diverse fields such as finance, economics, signal processing, and environmental monitoring. They prove especially valuable in handling time series data, aiding in the identification of trends and patterns while effectively filtering out short-term fluctuations. In Python, libraries like *NumPy* and *pandas* offer functions to calculate rolling means. In this methodology, a rolling mean with varying window sizes, such as 10, 20, 50, and 100, is employed on four columns of the dataset. The vertical axis, which is inverted, corresponds to TVD. The rolling mean is applied to ROP, GR, SPPA, and TFLO. The outcomes of this approach have been graphically represented in Figures 3.1, 3.2, 3.3, and 3.4. Using different window sizes (10, 20, 50, and 100) for the rolling mean filter offers varied detail. Smaller window sizes capture more detailed variations but might be susceptible to noise.

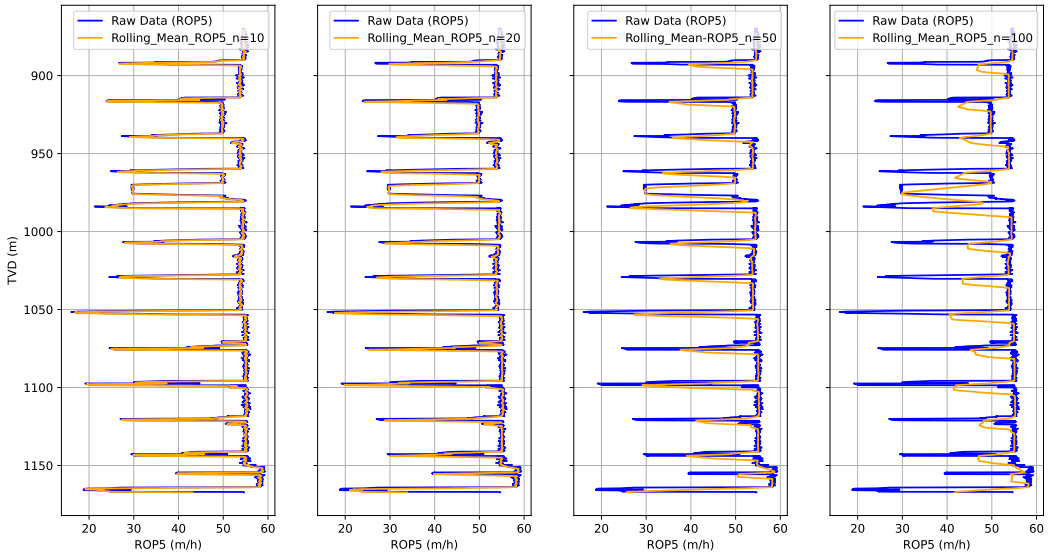


Figure 3.1: Using different window sizes (10, 20, 50, and 100) for the rolling mean filter on ROP, a larger window size results in a smoother feature.

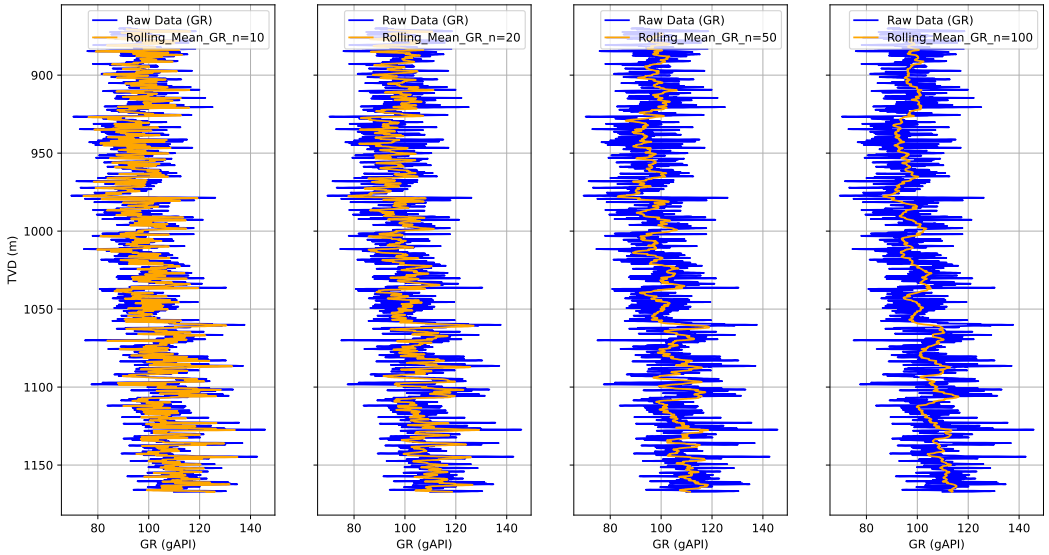


Figure 3.2: Using different window sizes (10, 20, 50, and 100) for the rolling mean filter on GR, a larger window size results in a smoother feature.

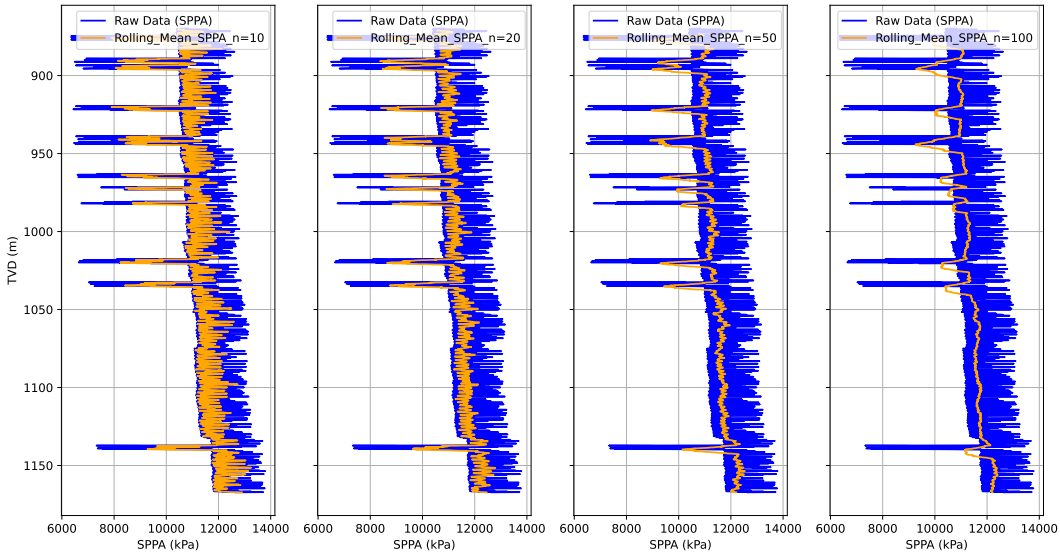


Figure 3.3: Using different window sizes (10, 20, 50, and 100) for the rolling mean filter on SPPA, a larger window size results in a smoother feature.

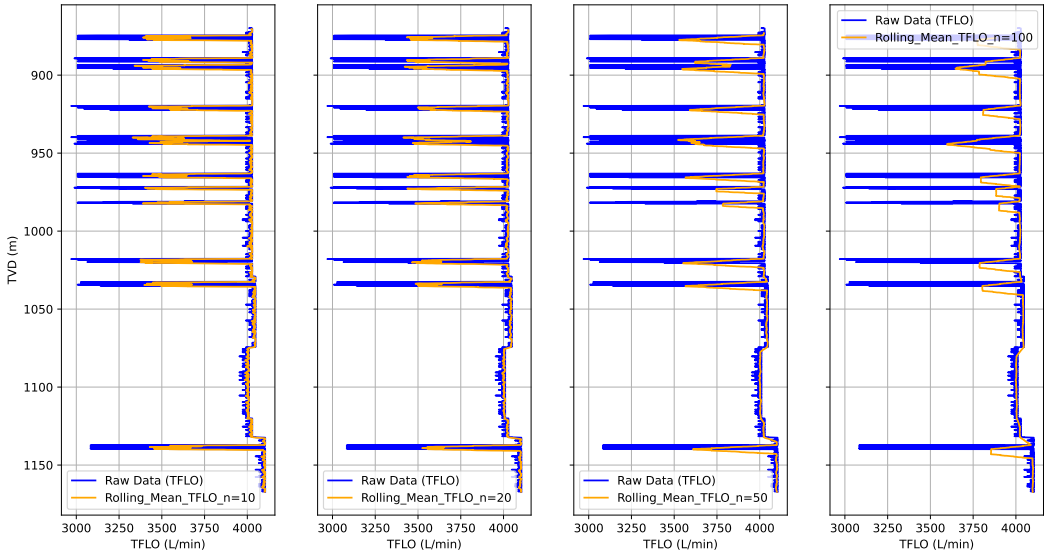


Figure 3.4: Using different window sizes (10, 20, 50, and 100) for the rolling mean filter on TFLO, a larger window size results in a smoother feature.

3.2 Linear filter

Linear filters can be seen as a controlled scaling of the signal components in the frequency domain. Linear filtering entails the manipulation of a signal by passing it through a filter to achieve desired effects, such as smoothing, noise reduction, or extracting specific features [27]. The implementation of the filter is based on the following equation:

$$y[n] = b[0]x[n] + b[1]x[n-1] + \ldots + b[M]x[n-M] - a[1]y[n-1] - \cdots - a[N]y[n-N]$$

where "b" is the numerator coefficient vector in a 1-D sequence, "a" is the denominator coefficient vector in a 1-D sequence, and "x" is an N-dimensional input array. In Python, the "lfilter" function is part of the *scipy.signal* module and is utilized to apply a digital filter to a signal. The pseudocode for this methodology is presented below in Algorithms 2 and 3.

Algorithm 2 The `lfilter` function

```
function LFILTER(coefficients, 1, input_value)
    filtered_value ← 0
    for i ← 0 to length(coefficients) − 1 do
        filtered_value ← filtered_value + coefficients[i] × input_value[i]
    return filtered_value
```

Algorithm 3 Apply lfilter to DataFrame columns

```
function APPLY_LFILTER(df, columns_lfilter)
    for i ← 0 to len(columns_lfilter) – 1 step 2 do
        LFILTER_COLUMN(df, columns_lfilter[i], columns_lfilter[i+1])
function LFILTER_COLUMN(df, input_column_name, output_column_name)
    filter_coefficients ← [1.0/10, 1.0/10, 1.0/10, 1.0/10, 1.0/10,
                           1.0/10, 1.0/10, 1.0/10, 1.0/10, 1.0/10]
    filtered_values ← []
    input_values ← df[input_column_name]
    for each value in input_values do
        filtered_value ← lfilter(filter_coefficients, 1, value)
        append filtered_value to filtered_values
    df[output_column_name] ← filtered_values
```

Using this function, a linear filter is applied to the signal given $b = 0.1$, $a = 1$, and x were the values of Well-7 from the 5000th to 10000th index. Both the filtered and unfiltered sequences are plotted on ROP, GR, SPPA, and TFLO based on TVD as depicted in Figure 3.5.[1]

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html

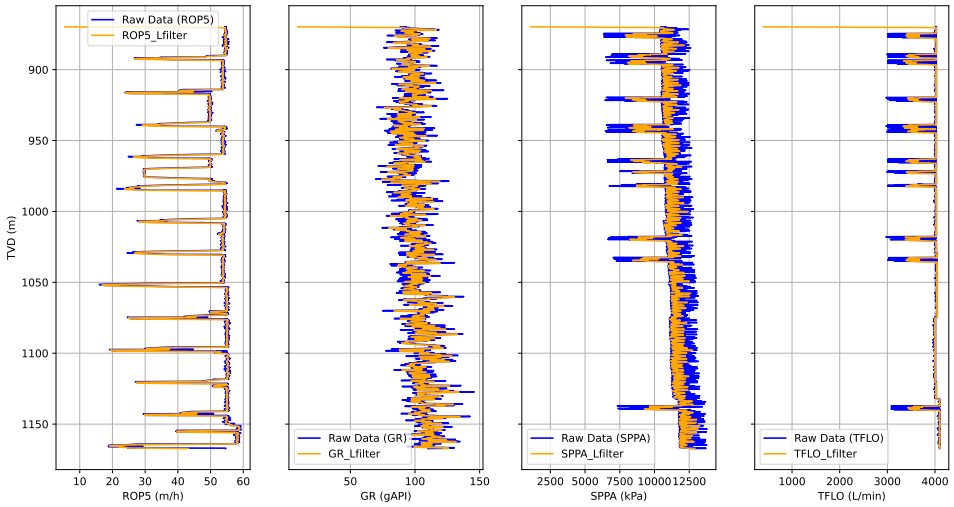


Figure 3.5: Applying linear filter on ROP, GR, SPPA and TFLO. Filtered data is smoother and less noisy.

3.3 Rolling median

The concept of a rolling median is rooted in statistical principles applied to time series analysis and signal processing. It entails computing the median of a subset of data points within a moving or rolling window as it traverses the dataset. The primary objective of employing a rolling median is to smooth out data variations and accentuate underlying trends or patterns[2].

In essence, the rolling median represents the median of a specified number of preceding periods in a time series. The process typically unfolds in three steps. Firstly, a moving window of a specified size progresses through the dataset step by step. At each step, the data points within the window are used to calculate the median. This median calculation considers the values within the window at each position of the moving window. Unlike the mean, which treats all values equally, the median represents the middle value when the data is sorted, making it less sensitive to extreme values or outliers. Ultimately, the rolling median aids in smoothing out short-term fluctuations or noise in the data, offering a clearer perspective on underlying trends or patterns. The pseudocode for this methodology is presented below in Algorithm 4.

[2] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.core.window.rolling.Rolling.median.html

Algorithm 4 Rolling Median Calculation

```
function ROLLING_MEDIAN(data, window_size)
    rolling_medians ← []
    window ← []
    for i ← 0 to length(data) – 1 do
        append data[i] to window
        if length(window) > window_size then
            pop the oldest value from window
        median ← calculate_median(window)
        append median to rolling_medians
    return rolling_medians
function CALCULATE_MEDIAN(values)
    sorted_values ← sort(values)
    middle_index ← length(sorted_values)/2
    if length(sorted_values)  mod 2 = 1 then
        return sorted_values[middle_index]
    else
        return (sorted_values[middle_index – 1]+
sorted_values[middle_index])/2
```

In the below plots, while TVD serves as the inverted y-axis, rolling medians with different window sizes (10, 20, 50, and 100) for ROP, GR, SPPA, and TFLO are applied and depicted in 3.6, 3.7, 3.8 and 3.9.

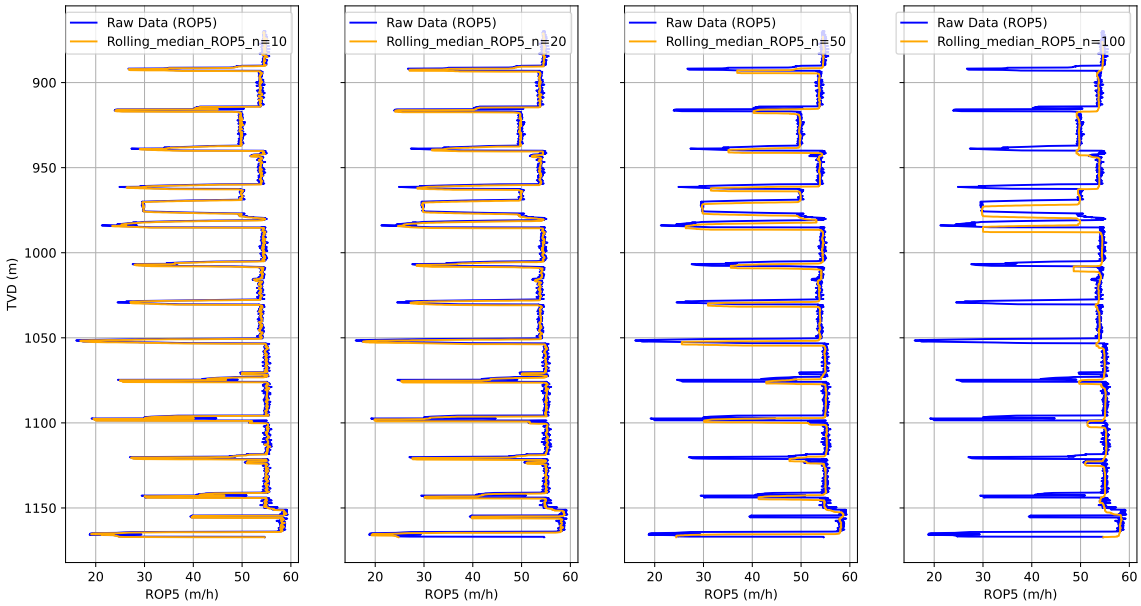


Figure 3.6: Applying rolling-median filter on ROP with window size 10, 20, 50, and 100, a larger window size results in a smoother feature.

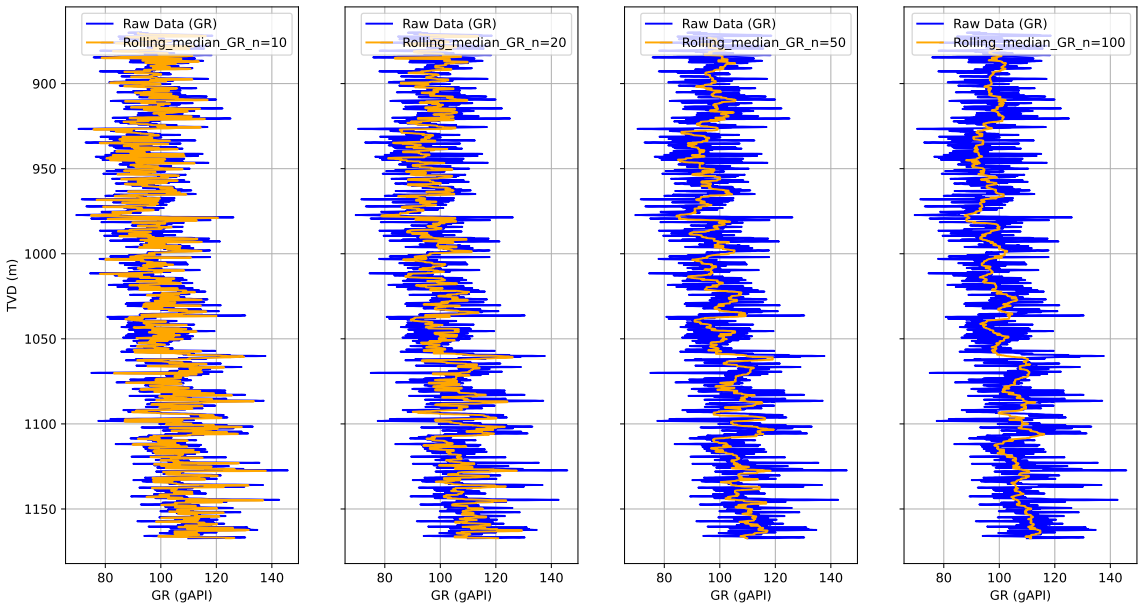


Figure 3.7: Applying rolling-median filter on GR with window size 10, 20, 50, and 100,a larger window size results in a smoother feature.

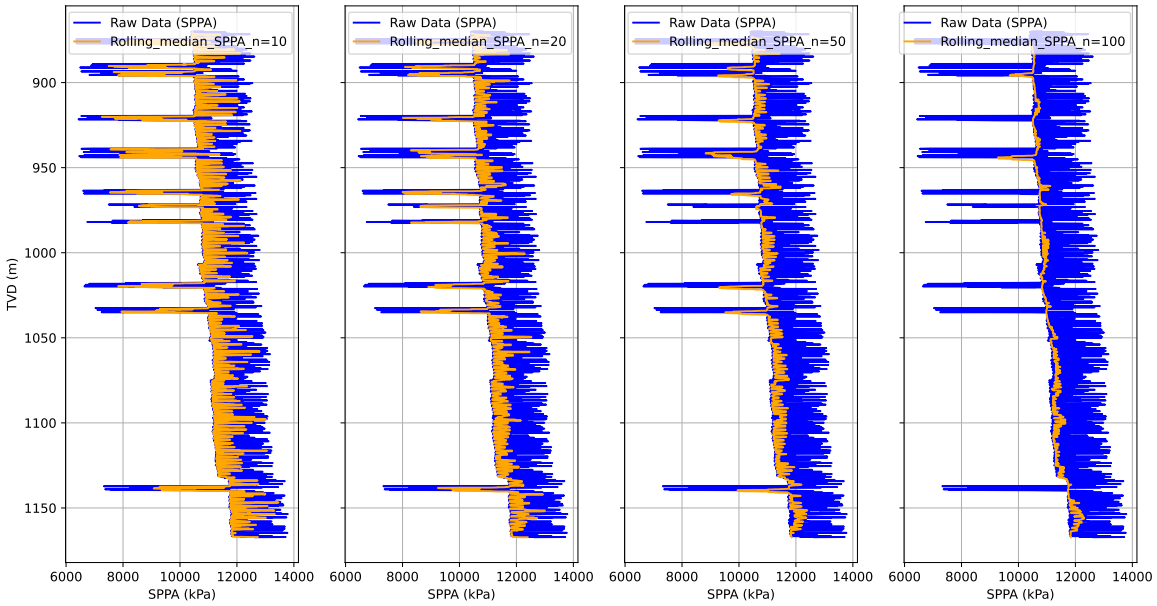


Figure 3.8: Applying rolling-median filter on SPPA with window size 10, 20, 50, and 100,a larger window size results in a smoother feature.

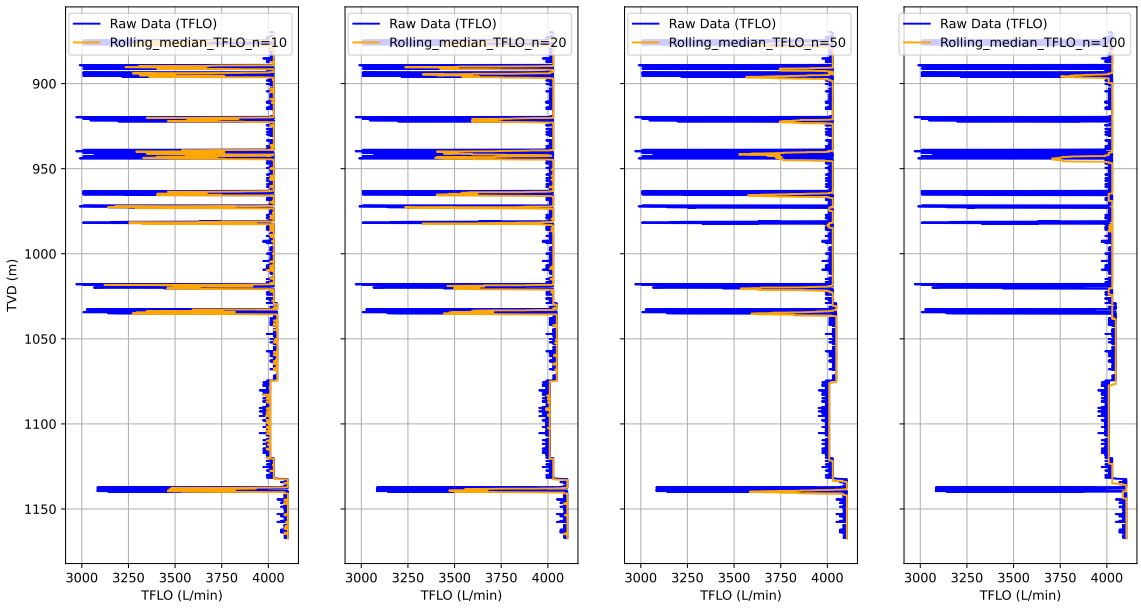


Figure 3.9: Applying rolling-median filter on TFLO with window size 10, 20, 50, and 100, a larger window size results in a smoother feature.

3.4 Forward and Backward filter

The "butter" function within the *scipy.signal* module is a tool for designing Butterworth digital and analog filters. It is capable of designing an Nth-order digital or analog Butterworth filter and provides the filter coefficients. The key parameters include "N", representing the filter order, and "Wn", denoting the critical frequency. For lowpass and highpass filters, "Wn" is a scalar. This function returns two parameters, "b" and "a", representing the numerator and denominator polynomials of the IIR filter, respectively[3].

The "filtfilt" function, also from the *scipy.signal* module, is widely used in signal processing, standing for Filter Forward and Backward. It applies a linear digital filter twice, first forward and then backward, resulting in a combined filter with zero phase and a filter order twice that of the original. This zero-phase filtering is advantageous for preserving temporal accuracy, crucial in applications like time series analysis or event-related data handling.

The input parameters for "filtfilt" are "b" and "a", representing the numerator and denominator coefficient vectors, respectively, along with "x", an N-dimensional input array. The output is the filtered data[4]. The pseudocode for this methodology is presented below in Algorithm 5 and 6.

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html

Algorithm 5 Butterworth Filter Design

function BUTTER(N, Wn, btype='low', analog=False, output='ba', fs=None)
    $Wn\_norm \leftarrow$ normalize(Wn, fs)
    $b, a \leftarrow$ calculate_coefficients$(N, Wn_norm, btype, analog)$
    if $output = 'ba'$ **then**
        return b, a
    else if $output = 'zpk'$ **then**
        $z, p, k \leftarrow$ bilinear_transform(b, a, fs)
        return z, p, k
function NORMALIZE(Wn, fs)
    if $fs = None$ **then**
        return Wn
    else
        return $\frac{2}{fs} \times Wn$
function CALCULATE_COEFFICIENTS(N, Wn, btype, analog)
    if $analog$ **then**
        $b, a \leftarrow$ analog_filter_coefficients$(N, Wn, btype)$
    else
        $b, a \leftarrow$ digital_filter_coefficients$(N, Wn, btype)$
    return b, a

Algorithm 6 Forward-Backward Filtering

1: **function** FILTFILT(b, a, x)

2: $y\_forward \leftarrow$ forward_filtering(b, a, x)

3: $y\_backward \leftarrow$ backward_filtering$(b, a, y_forward)$

4: **return** $y\_backward$

5: **function** FORWARD_FILTERING(b, a, x)

6: $y\_forward \leftarrow []$

7: $y\_forward[0] \leftarrow x[0] \times b[0]$

8: **for** $i \leftarrow 1$ **to** length$(x) - 1$ **do**

9: $y\_forward[i] \leftarrow x[i] \times b[0]$

10: **for** $j \leftarrow 1$ **to** $\min(i, \text{length}(b) - 1)$ **do**

11: $y\_forward[i] \leftarrow y\_forward[i] + x[i-j] \times b[j] - y\_forward[i-j] \times a[j]$

12: **return** $y\_forward$

13: **function** BACKWARD_FILTERING(b, a, x)

14: $y\_backward \leftarrow []$

15: $y\_backward[\text{length}(x) - 1] \leftarrow x[\text{length}(x) - 1] \times b[0]$

16: **for** $i \leftarrow \text{length}(x) - 2$ **downto** 0 **do**

17: $y\_backward[i] \leftarrow x[i] \times b[0]$

18: **for** $j \leftarrow 1$ **to** $\min(\text{length}(x) - i - 1, \text{length}(b) - 1)$ **do**

19: $y\_backward[i] \leftarrow y\_backward[i] + x[i+j] \times b[j] - y\_backward[i+j] \times a[j]$

20: **return** $y\_backward$

In practice, after obtaining the coefficients "b" and "a" from the Butter function with specified parameters $N = 4$ and $Wn = \frac{2}{365}$, they are used with the "filtfilt" function for backward and forward filtering. It is worth noting that backward filtering, as applied in time series, involves predicting future data points, rendering it more suitable for offline processing of recorded signals rather than real-time applications.

In the context of subsequent analysis, "filtfilt" is employed for features ROP, GR, SPPA, and TFLO, and the results are plotted in Figure 3.10.

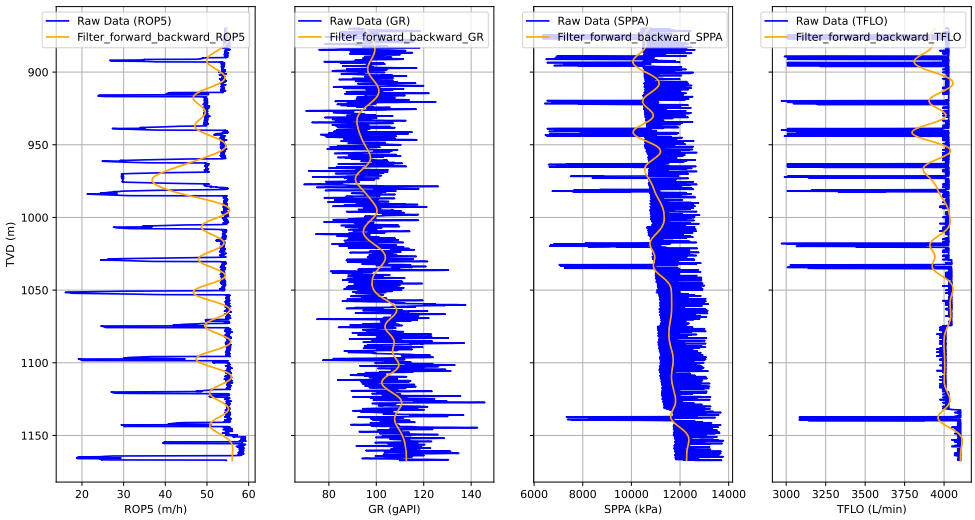


Figure 3.10: Applying forward and backward filter on ROP, GR, SPPA, and TFLO. Filtered data was not Representative of data due to smoothing data points roughly.

3.5 Finite Impulse Response filter

*Finite Impulse Response (*FIR*)* filter is a type of digital filter used in signal processing. Unlike *Infinite Impulse Respons (*IIR*)* filters, FIR filters only have a finite response to an input signal. This means that the output of an FIR filter is determined entirely by a weighted sum of the input samples and does not rely on feedback from previous outputs. The general structure of an FIR filter involves convolving the input signal with a set of coefficients, also known as the filter taps. The coefficients determine the filter's frequency response and are usually determined through filter design techniques.

FIR filters find applications in various areas, including audio processing, image processing, telecommunications, and control systems. They are often used for tasks such as smoothing, noise reduction, equalization, and signal shaping. The procedure of the FIR filter includes a segment that creates a triangular window of length N and then convolves it with the specific feature from DataFrame. Finally, it stores the result of the convolution, filtered by the triangular window, in a new column as FIR filter on the data frame.

The pseudocode for this methodology is presented below in Algorithm 7.

Algorithm 7 Finite Impulse Response filter

```
1: function CONVOLVE(signal, filter)
2:     result ← []
3:     for i ← 0 to length(signal) + length(filter) − 1 do
4:         sum ← 0
5:         for j ← 0 to length(filter) − 1 do
6:             if i − j ≥ 0 and i − j < length(signal) then
7:                 sum ← sum + signal[i − j] × filter[j]
8:         append sum to result
9:     return result
```

In Python, creating a FIR filter can be accomplished by using *Scipy.signal* module. The following figure demonstrates the application of the FIR filter to ROP, GR, SPPA, and TFLO, illustrating the reduction of noise achieved through this procedure.

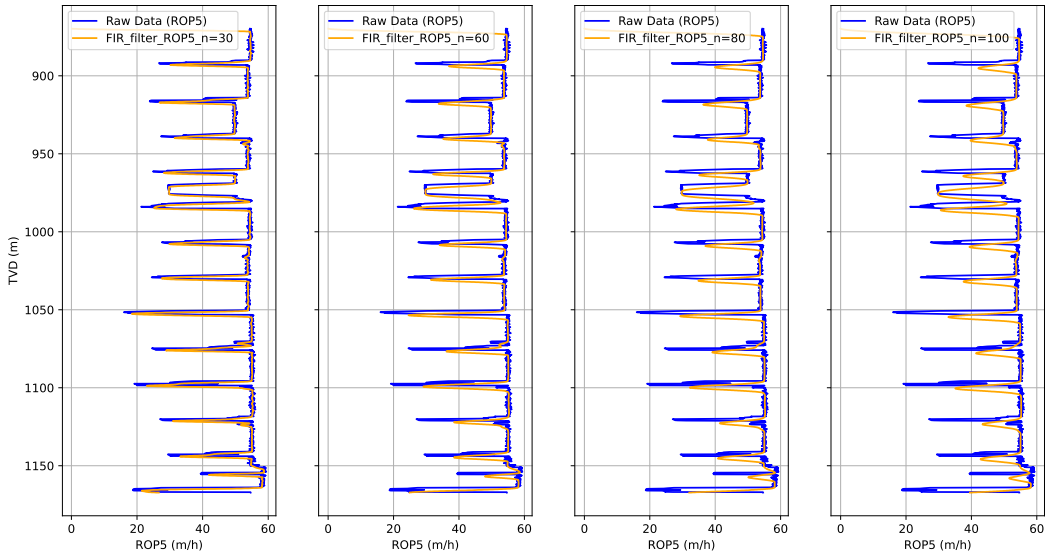


Figure 3.11: Applying Finite Impulse Response filter on ROP with window size 30, 60, 80, and 100, a larger window size results in a smoother feature.

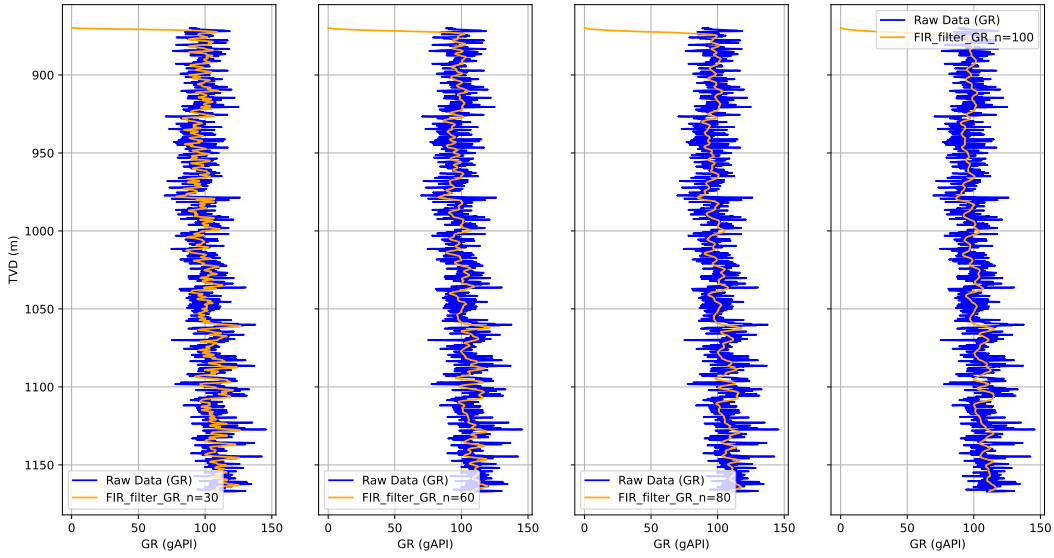


Figure 3.12: Applying Finite Impulse Response filter on GR with window size 30, 60, 80, and 100, a larger window size results in a smoother feature.

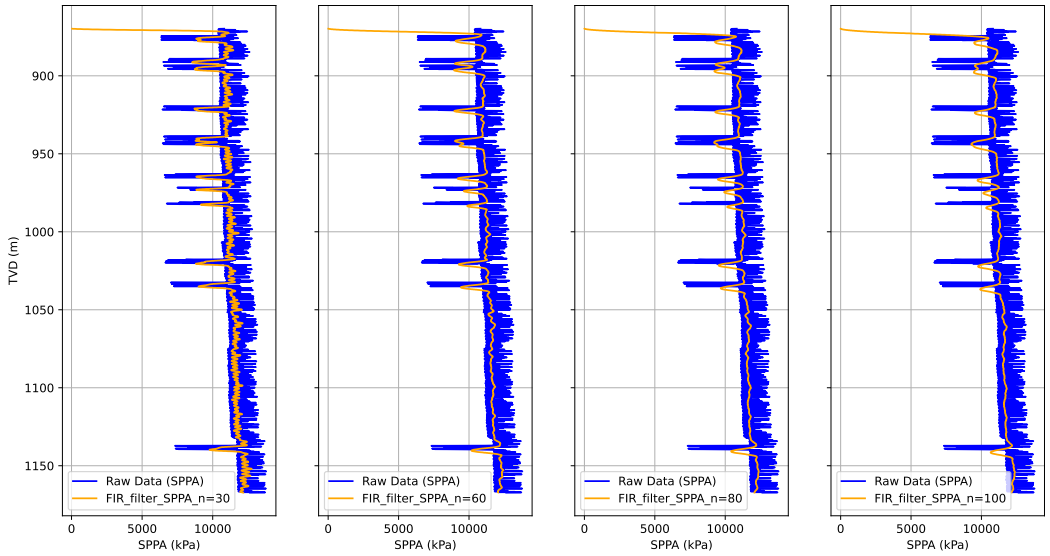


Figure 3.13: Applying Finite Impulse Response filter on SPPA with window size 30, 60, 80, and 100, a larger window size results in a smoother feature.

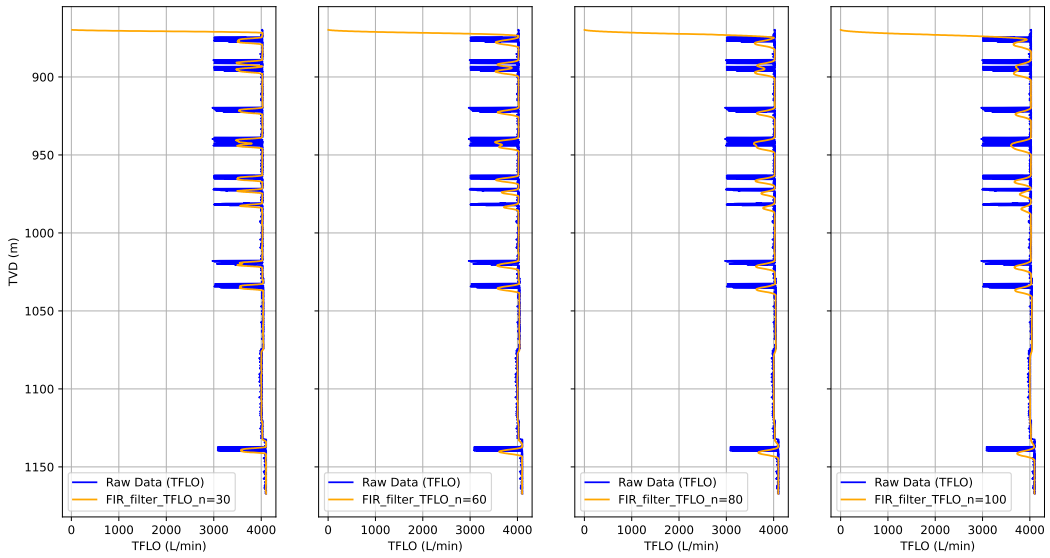


Figure 3.14: Applying Finite Impulse Response filter on TFLO with window size 30, 60, 80, and 100, a larger window size results in a smoother feature.

3.6 Low-pass filter

A low-pass filter is a type of electronic filter that allows signals with a frequency lower than a certain cutoff frequency to pass through while reducing the amplitudes of signals with frequencies higher than the cutoff frequency. In other words, it permits low-frequency components to pass through and blocks or reduces the amplitudes of high-frequency components. For implementing a low-pass filter we used two different methods: Butterworth filter and Fourier transform. We have utilized these filters on features ROP, GR, SPPA, and TFLO and presented the results. However, in the section dedicated to comparing filters, the Fourier transform method is specifically applied.

3.6.1 Butterworth low-pass filter

A low-pass filter is a crucial component in signal processing, communication systems, and audio applications, serving to attenuate higher frequencies beyond a specified cut-off frequency while allowing lower frequencies to pass through. The cutoff frequency marks the point at which the filter begins attenuating the signal, and the transition between the passband and stopband defines the filter's roll-off rate.

These filters, including types like Butterworth, Chebyshev, and elliptic filters, find widespread application in tasks such as smoothing signals and eliminating high-frequency noise or fluctuations.

The *Butterworth low-pass filter* function specifically facilitates the design of a

Butterworth low-pass filter. This function requires inputs such as the cutoff frequency (*cutoff*), sampling frequency (f_s), and filter order (*order*). It employs the *butter* function from the *scipy.signal* module to compute and return the filter coefficients (b, a). This function is instrumental in tailoring low-pass filters to meet the specific requirements of diverse applications. Within our dataset, elliptic filters with a cut-off of 4, a frequency of 2/300, and a threshold of 5000 have been applied. Subsequently, the *butter-lowpass filter* function is utilized to implement the Butterworth filter on the provided input signal. The filter coefficients (b, a) are derived using the *butter-lowpass* function, and the *linear filter* function is then employed to filter the input signal. The pseudocode for this methodology is presented below in Algorithm 8 and 9.

Algorithm 8 Butterworth low-pass filter design

```
function BUTTER_LOWPASS(cutoff, f_s, order=5)
    return BUTTER(order, cutoff, f_s=f_s, btype="low", analog=False)
```

Algorithm 9 Butterworth low-pass filtering

```
function BUTTER_LOWPASS_FILTER(data, cutoff, f_s, order=5)
    (b, a) ← BUTTER_LOWPASS(cutoff, f_s, order=order)
    y ← LFILTER(b, a, data)
    return y
```

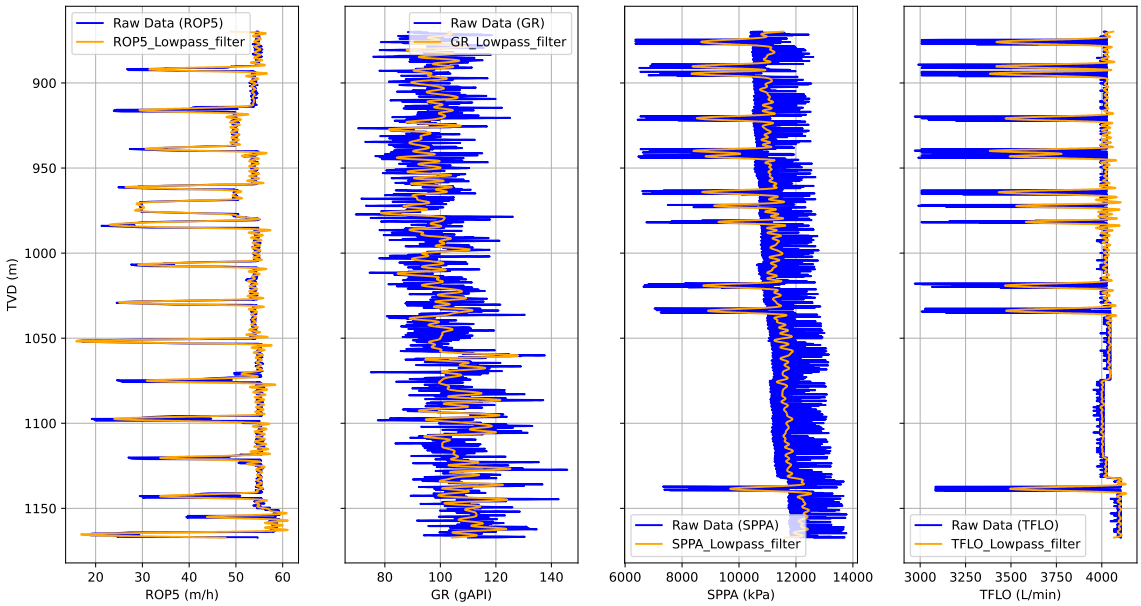


Figure 3.15: Applying butterworth low pass filter for ROP, GR, SPPA, and TFLO. Filtered data is smoother and less noisy.

3.6.2 Fourier transform low-pass filter

The Fourier transform is a mathematical operation that transforms a signal or function from its original domain (typically time or space) into a representation in the frequency domain. It decomposes a complex signal into simpler sinusoidal components, revealing the frequency content of the original signal. The transformed signal provides information about the amplitude and phase of different frequency components. The function *low-pass* defined in Algorithm 10 below, implements a low-pass filter using the Fourier transform. It takes the input signal (s) and a frequency threshold (*threshold*) as inputs. The function first checks if the length of the signal is odd and removes the last element if necessary. It then applies the real-valued *Fast*

Fourier transform (rfft) from *scipy.signal* library to the signal. Frequencies above the specified threshold are set to zero in the Fourier domain. Then the *inverse Fourier transform (irfft)* from *scipy.signal* library applied to obtain the filtered signal. The pseudocode for this methodology is presented in Algorithm 10.

Algorithm 10 Fourier transform low-pass Filter

```
1: function LOW_PASS(s, threshold)
2:     if len(s) mod 2 = 1 then
3:         s ← s[: −1]
4:     fourier ← RFFT(s)
5:     frequencies ← RFFTFREQ(size(s), d = 2e − 2/size(s))
6:     fourier[frequencies > threshold] ← 0
7:     return IRFFT(fourier)
```

3.7 Interpolation

Interpolation is a mathematical technique used to estimate values that fall between known values [28]. It is commonly employed in various fields, including mathematics, computer science, statistics, and signal processing. The primary goal of interpolation is to predict the values of a function or dataset at points that are not explicitly provided but lie within the range of existing data points. Knowing this concept we have applied the *Exponential Weighted Moving Average (*EWMA*)* method which is used for smoothing time series data or for estimating the underlying trend of a time series. In the context of interpolation, EWMA is often used to fill in missing or irregularly sampled data points in a time series.

The basic idea behind EWMA is to assign exponentially decreasing weights to past observations, with more recent observations receiving higher weights. This makes EWMA particularly suitable for capturing trends and reacting quickly to changes in the data.

In this section, necessary libraries for numerical operations (*numpy*), Fourier transforms (*numpy.fft*), signal processing (*scipy.signal*), and plotting (*matplotlib*) have been imported. There are some constants and parameters that need to be defined in applying interpolation including DELTA, a parameter used in outlier removal; LOW-CLIP and HIGH-CLIP, threshold values for clipping data; SPAN value is the number of data points that are used to calculate the average, and SPIKE is the amplitude of a spike in the data. Additional functions are required at this stage and are described

below:

Clip-data: This function takes an unclipped data column and clips values outside the specified range (LOW-CLIP to HIGH-CLIP). It converts the data to an array, applies the clipping conditions, and returns the clipped data as a list. The pseudocode for this function is presented as Algorithm 11.

Algorithm 11 clip_data Function

```
1: function CLIP_DATA(unclipped, high_clip, low_clip)
2:     np_unclipped ← convert_to_numpy_array(unclipped)
3:     for each value in np_unclipped do
4:         if value > high_clip or value < low_clip then
5:             set value to NaN
6:     clipped_data ← convert_to_list(np_unclipped)
7:     return clipped_data
```

Create-sample-data: This function creates a sample data frame using a subset of the original data frame. The pseudocode for this function is presented as Algorithm 12.

Algorithm 12 create_sample_data Function

```
1: function CREATE_SAMPLE_DATA
2:     df ← df[ "GR", "SPPA","TFLO", "SWOB", "ROP" ]
3:     df["y_spikey_df_column"] ← df["df_column"]
4:     return df
```

Ewma-fb: This function applies a forward-backward exponential weighted moving average filter to a given data column. It uses the EWM method from the *pandas* library to calculate forward and backward EWMA separately and then combines them to get the final forward-backward EWMA. The pseudocode for this function is presented as Algorithm 13.

Algorithm 13 ewma_fb Function

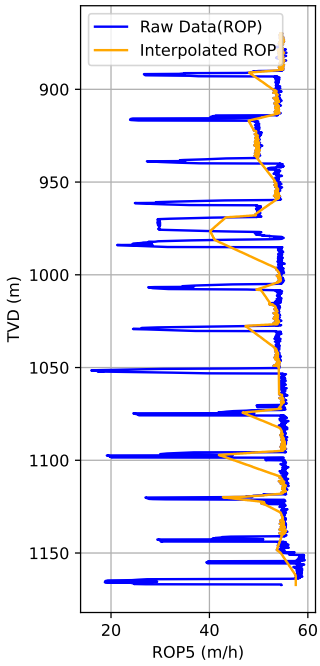
```
1: function EWMA_FB(df_column, span)
2:     fwd ← pd.Series.ewm(df_column, span = span).mean()
3:     bwd ← pd.Series.ewm(df_column[::-1], span = 10).mean()
4:     stacked_ewma ← np.vstack((fwd, bwd[::-1]))
5:     fb_ewma ← np.mean(stacked_ewma, axis = 0)
6:     return fb_ewma
```

Remove outliers: This function removes outliers from a data column (*spikey*) based on a threshold (*delta*) and a reference column. It converts the input data to arrays, applies the outlier condition, and returns the data with outliers replaced by NaN. The pseudocode for this function is presented as Algorithm 14.

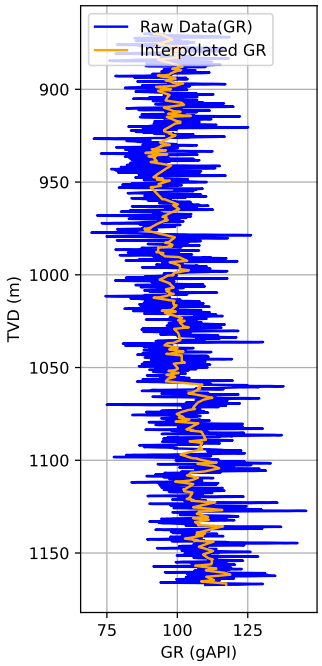
Algorithm 14 remove_outliers Function

```
1: function REMOVE_OUTLIERS(spikey, fbewma, delta)
2:     np_spikey ← np.array(spikey)
3:     np_fbewma ← np.array(fbewma)
4:     cond_delta ← (np.abs(np_spikey – np_fbewma) > delta)
5:     np_remove_outliers ← np.where(cond_delta, np.nan, np_spikey)
6:     return np_remove_outliers
```

It is worth mentioning that each dataset necessitates modifications in LOW-CLIP, HIGH-CLIP, and DELTA for interpolation, even if these adjustments might not be immediately evident in datasets with similar ranges. The figures 3.16 and 3.17 illustrate this.

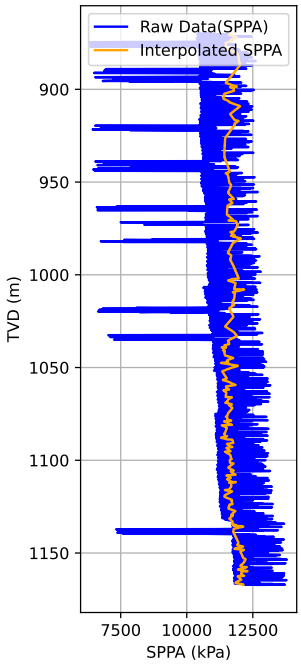


(a) Applying interpolation with DELTA=0.6, SPAN=200, LOW-CLIP=10, HIGH-CLIP=300, and Spike amplitude=20 for ROP in 350 m (850-1200) of ROP for Well-7.

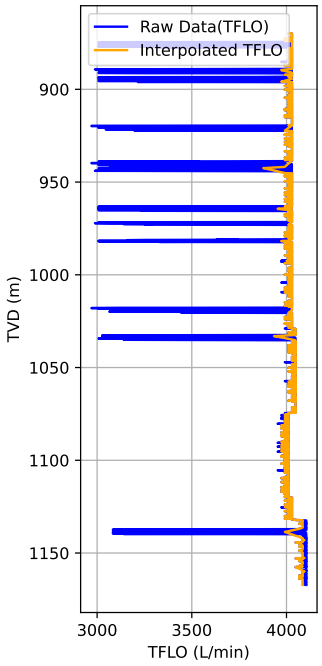


(b) Applying interpolation with DELTA=0.6, SPAN=100, LOW-CLIP=10, HIGH-CLIP=300, and Spike amplitude=20 for GR in 350 m (850-1200) of ROP for Well-7.

Figure 3.16: Adjustments to LOW-CLIP, HIGH-CLIP, and DELTA are necessary for interpolation although it might not be apparent in datasets with identical ranges.



(a) Applying interpolation with DELTA=100, SPAN=100, LOW-CLIP=11000, HIGH-CLIP=13000, and Spike amplitude=20 for SPPA in 350 m (850-1200) of ROP for Well-7.



(b) Applying interpolation with DELTA=30, SPAN=100, LOW-CLIP=3700, HIGH-CLIP=4100, and Spike amplitude=20 for TFLO in 350 m (850-1200) of ROP for Well-7.

Figure 3.17: Adjustments to LOW-CLIP, HIGH-CLIP, and DELTA are necessary for interpolation.

3.8 Comparison of filters

Selecting the most suitable filter for signal processing depends on several factors, including the characteristics of the data, the type of signal, and the specific goals of the analysis. Filters are designed to address various aspects of signal processing, such as noise reduction, smoothing, and feature extraction. One crucial aspect to consider is the frequency response of the filters. Different filters may exhibit distinct effects on high and low-frequency components of the signal. It is imperative to assess whether a filter introduces phase shifts, particularly when preserving the timing of signal features is of utmost importance.

Efficient noise reduction while preserving the signal-to-noise ratio is of paramount consideration. The choice between a smoother output and the preservation of sharp transitions in the signal depends on the specific goals of the analysis. Different filters demonstrate varying degrees of smoothing or sharpness.

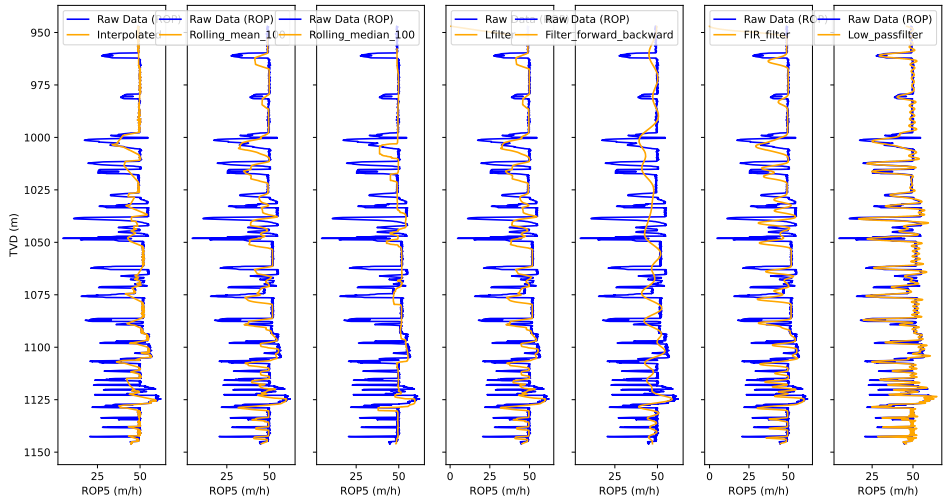
Each type of filter, whether it be Butterworth, elliptic, or Fourier transform, possesses unique characteristics. For instance, Butterworth filters provide a smooth frequency response, while elliptic filters offer a steeper roll-off. Evaluating these characteristics in alignment with the goals of the analysis is imperative.

As we proceed with our analysis, the incorporation of performance metrics becomes pivotal for an objective and comprehensive comparison of filters. Metrics such as mean squared error or signal-to-noise ratio offer valuable insights, enhancing the depth of our assessment. Our study is committed to optimizing cross-correlations

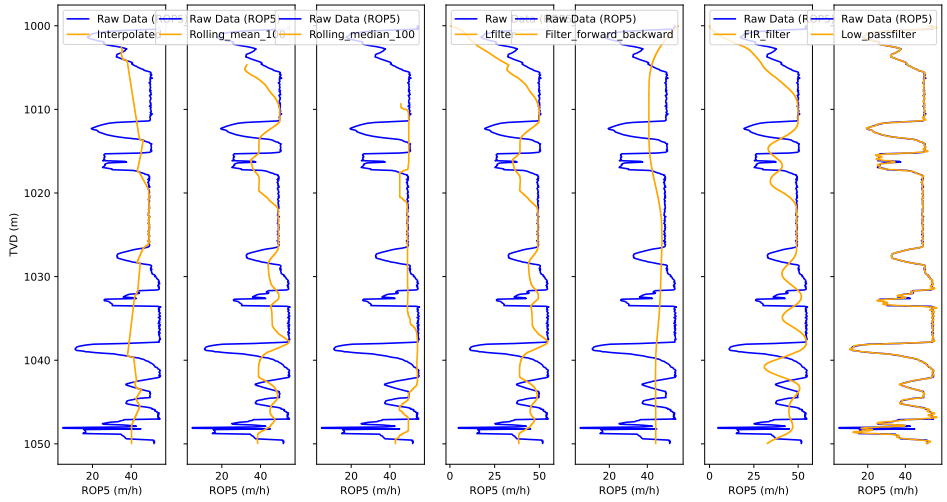
across diverse geological formations, and to achieve this objective, we strategically employ a low-pass filter. This selection is made in consideration of the distinctive advantages offered by low-pass filters in terms of noise reduction and signal clarity. The rationale behind selecting a low-pass filter is rooted in its unique advantages, contributing to enhanced signal clarity and interpretability. The most important advantage of the low-pass filter is noise removal, as high-frequency noise in a signal can be disruptive and impede interpretability. Low-pass filters excel in removing such high-frequency noise, ensuring a clearer and more intelligible signal. The elimination of noise facilitates a more accurate interpretation of the signal's intrinsic features. The second rationale revolves around the augmentation of Signal-to-Noise Ratio (SNR). High-frequency noise tends to undermine the overall signal-to-noise ratio. The application of a low-pass filter results in the removal of high-frequency noise, thereby improving the overall SNR. Improved SNR facilitates the detection and analysis of subtle signals that might otherwise be overshadowed by noise.

As a conclusion, the strategic use of a low-pass filter emerges as a valuable approach in our study, aiming to optimize cross-correlations amidst diverse geological formations. By leveraging the noise removal capabilities and enhancing the signal-to-noise ratio, the low-pass filter becomes an instrumental tool in elucidating meaningful insights from the geological data. The advantages of this filter contribute to a refined and clearer understanding of the underlying signals, fostering more robust and accurate interpretations in the context of cross-correlations. In Figures 3.18a, 3.18b,

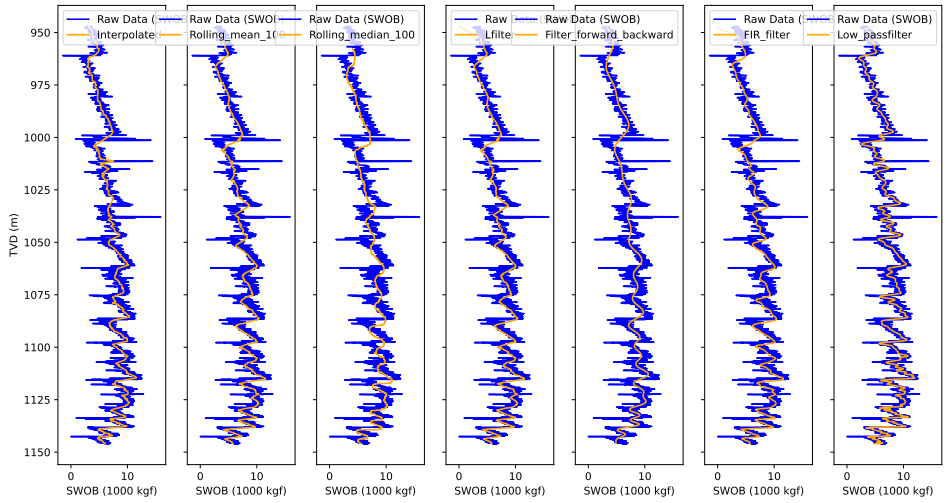
3.19a and 3.19b, the application of various filters on both large and small intervals of TVD respectively for ROP and SWOB features of Well-6 is depicted.



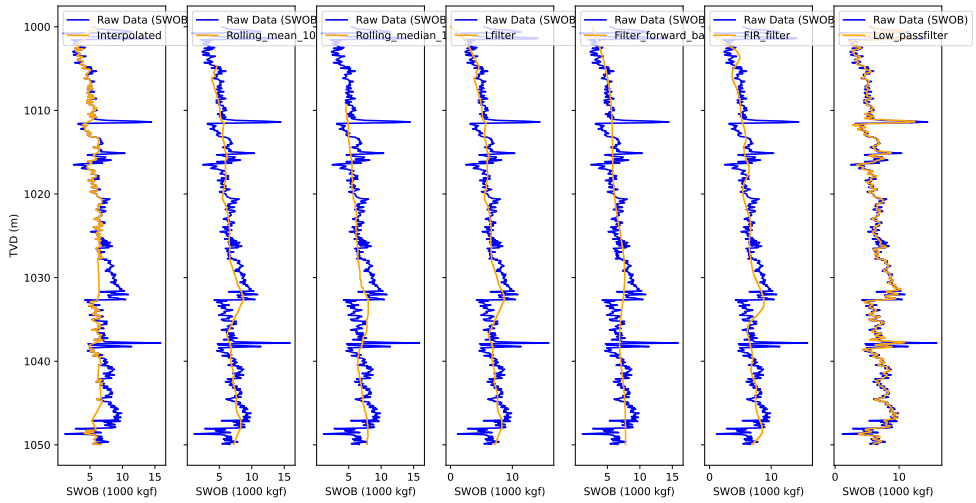
(a) Comparison of the mentioned filters for depth of 200 meters in TVD (950-1150) for ROP of Well-6, a low-pass filter provides a more accurate representation of the data while employing Finite Forward and Backward methods significantly enhances data smoothness.



(b) Comparison of the mentioned filters for depth of 50 meters in TVD (950-1150) for ROP of Well-6, a low-pass filter provides a more accurate representation of the data while employing Finite Forward and Backward methods significantly enhances data smoothness.



(a) Comparison of the mentioned filters for depth of 200 meters in TVD (950-1150) for SWOB of Well-6, a low-pass filter provides a more accurate representation of the data while employing Finite Forward and Backward methods significantly enhances data smoothness.



(b) Comparison of the mentioned filters for depth of 50 meters in TVD (950-1050) for SWOB of Well-6, a low-pass filter provides a more accurate representation of the data while employing other methods significantly enhances data smoothness.

In this chapter, we explored various signal processing techniques, including rolling mean, linear and median filters, FIR filters, low-pass filters (Butterworth and Fourier transform), and interpolation methods. These techniques are essential for enhancing signal quality by reducing noise and highlighting important features. As we move forward, the next chapter will focus on preprocessing, which prepares these refined signals for analysis by addressing scaling, and normalization. Preprocessing transforms the cleaned signals into a format suitable for advanced analysis and modeling, ensuring data integrity and usability in various applications.

Chapter 4

Preprocessing

Preprocessing is the first step for refining raw data into a format suitable for machine learning algorithms. This transformative process involves a series of operations, such as cleaning, feature selection, and transformation. In this chapter, through preprocessing, we streamline the dataset by selecting the most important features, applying scalers and removing outliers, and laying the groundwork for robust and accurate model training.

4.1 Research Server Specifications

My research was conducted using a server with the following specifications:

- **Server:** Lapicque
- **Model Name:** Intel(R) Xeon(R) Gold 6244 CPU @ 3.60GHz

- **CPUs:** 32
- **Cores per Socket:** 8
- **Memory:** 251 GiB (approximately 269.41 GB)

The server's memory usage was monitored during the experiments using the `free -h` command, revealing insights into total, used, and free memory. This research leveraged Python programming language, with key libraries including *NumPy*, *SciPy*, and *TensorFlow*. Additionally, data processing tasks were executed using *Pandas*.

4.2 The dataset

In our study, we analyzed two datasets linked to Well-1 and Well-6, each containing accessible sonic log information. The Well-1 dataset consists of 55,575 rows and 42 features, with significant features displayed in Table 1.2. It is noteworthy that certain parts of the data were missing, resulting in NaN values. Specifically, our focus was on the DTCO-MH-R column, representing the sonic log, which contained 37650 non-NaN values after removing outliers related to hole sizes 311, 445, and 216 (mm). In the dataset, only two variables were categorized as objects, "Well Name" and "Geological formations". The geological formations variable encompassed distinct formations, including Banquereau, Oligocene Sandstone Unit, South Mara Member, Mudstone Member Top, Dawson Canyon, Petrel Member, Nautilus, Ben Nevis, Avalon, and Whiterose Formation. Additionally, we treated the "Hole Size" column

as categorical data due to its non-uniform distribution with three distinct hole size categories. Consequently, separate predictions were conducted for each size category. The Well-6 dataset comprises 68,273 rows and 49 columns. The DTCO-MH-R column, representing the sonic log, contains 44,306 non-NaN entries associated with hole sizes 311, 445, and 216 (mm). The geological formation variable encompasses distinct formations, including Banquereau, Oligocene Sandstone Unit, South Mara Member, Mudstone Member Top, Dawson Canyon, Petrel Member, Nautilus, and Ben Nevis Formation.

The values in the Hole Size column were determined based on TVD and extracted from the raw data. The TVD height range corresponding to each hole size is presented in Table 4.1.

Table 4.1: Corresponding hole sizes for different TVD ranges.

Well	**TVD Range (m)**	**Corresponding Hole Size**
1	523 to 1062	445
1	1062 to 1770	311
1	1766 to 2564	216
6	544 to 1335	445
6	1062 to 1770	311
6	2009 to 2050	216

4.3 Feature selection

Pearson Correlation Coefficient

The Pearson correlation coefficient, denoted as r, is a measure of the linear correlation between two variables X and Y. It quantifies the degree to which a linear relationship exists between these two variables. The value of r ranges from -1 to 1:

- $r = 1$ indicates a perfect positive linear correlation,
- $r = -1$ indicates a perfect negative linear correlation,
- $r = 0$ indicates no linear correlation.

The Pearson correlation coefficient is calculated by taking the covariance of the two variables and dividing it by the product of their standard deviations. Mathematically, it is expressed as:

$$r = \frac{\sum(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum(X_i - \overline{X})^2 \sum(Y_i - \overline{Y})^2}}$$

where:

- X_i and Y_i are the individual sample points,
- $\overline{X}$ and $\overline{Y}$ are the means of X and Y, respectively,
- $\sum$ denotes the summation over all sample points.

The coefficient r provides insight into the strength and direction of the linear relationship between the variables. A value closer to ±1 signifies a stronger linear relationship, while a value closer to 0 signifies a weaker one.

The pseudocode for calculating the Pearson correlation coefficient between two lists of numbers, X and Y is presented as Algorithm 15.

Algorithm 15 Pearson Correlation Coefficient

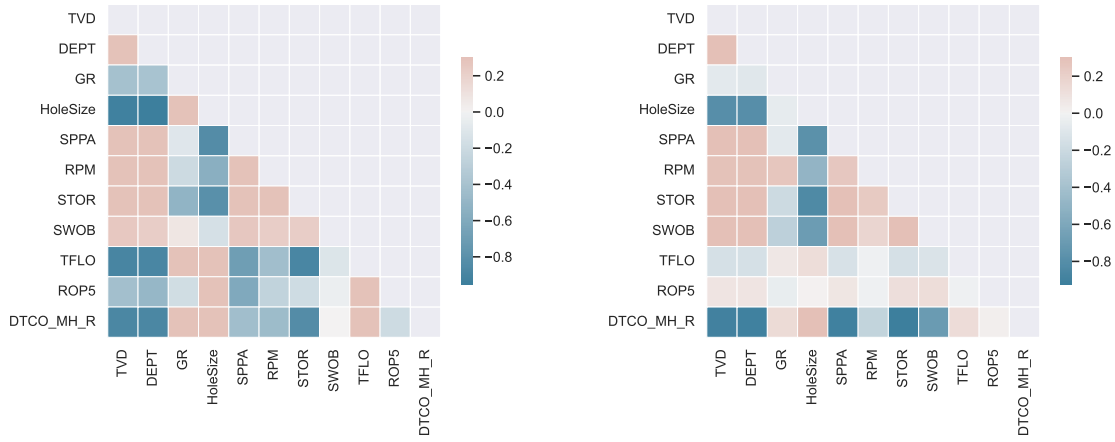
```
1: function PearsonCorrelationCoefficient(X, Y)
2:     mean_X ← mean(X)
3:     mean_Y ← mean(Y)
4:     numerator ← 0
5:     denominator_X ← 0
6:     denominator_Y ← 0
7:     for each i from 0 to length(X) – 1 do
8:         diff_X ← X[i] – mean_X
9:         diff_Y ← Y[i] – mean_Y
10:        numerator ← numerator + diff_X × diff_Y
11:        denominator_X ← denominator_X + diff_X × diff_X
12:        denominator_Y ← denominator_Y + diff_Y × diff_Y
13:    denominator ← √(denominator_X × denominator_Y)
14:    if denominator == 0 then
15:        return 0
16:    else
17:        return numerator/denominator
18: function mean(values)
19:    return ∑(values)/length(values)
```

In order to identify the most impactful contributors among the columns, a regres-

sion analysis was conducted, computing P-values for each predictor variable. The P-value serves as an indicator of the likelihood of observing the data assuming no relationship between the predictor variable and the response variable.

The columns were subsequently arranged in ascending order based on their P-values. A lower P-value suggests stronger evidence against the null hypothesis, indicating a greater potential contribution of the column to the model. A significance threshold of 0.05, a standard choice in statistical analyses, was employed. Columns with p-values below this threshold were deemed statistically significant, signifying their meaningful role in the model.

Upon sorting, columns with P-values below the chosen significance level were identified. These following columns, TVD, GR, SPPA, RPM, Hole Size, STOR, SWOB, TFLO, ROP, and Geological Formation, demonstrated statistically significant contributions, providing valuable insights into explaining the variability in the DTCO-MH-R variable. For consistency between Well-1 and Well-6 datasets, variables available in both were selected. Additionally, considering the high correlation between TVD and DEPT, only TVD was included in the analysis. Through a meticulous assessment of P-values and their significance levels, the most prominent column contributions were discerned and prioritized based on their statistical significance.



(a) Correlation matrix of selected features for Well-1.

(b) Correlation matrix of selected features for Well-6.

Figure 4.1: Heatmap of Well-1 and Well-6.

Figures 4.1a and 4.1b provided the visual representation of a matrix of data using color gradients respectively for Well-1 and Well-6. Therefore, in alignment with prior studies, RPM, ROP, TFLO, SWOB, STOR, GR, TVD, and Hole Size were employed as predictor variables to forecast the sonic log.

Tables 4.2, 4.3, 4.4, 4.5, and 4.6 provide a quick overview of key statistical measures for the numerical attributes within a dataset. These measures, encompassing minimum, maximum, and standard deviation, serve to convey the range and variability of values for the chosen features across different hole sizes in both Well-1 and Well-6.

Table 4.2: Summary of statistical characteristics of selected features for hole size 445 in Well-1.

	TVD	GR	SPPA	RPM	STOR	SWOB	TFLO	ROP5	DTCO_MH_R
count	**202**	**202**	**202**	**202**	**202**	**202**	**202**	**202**	**202**
mean	**1052.616**	**100.739**	**12939.714**	**158.564**	**15.615**	**19.555**	**4148.052**	**52.904**	**416.594**
std	**4.439**	**8.904**	**69.662**	**0.497**	**1.770**	**1.095**	**20.578**	**14.227**	**8.663**
min	**1044.969**	**80.655**	**12735.560**	**158.000**	**11.290**	**16.371**	**4100.345**	**22.404**	**401.844**
25%	**1048.802**	**94.819**	**12893.492**	**158.000**	**14.275**	**18.864**	**4138.136**	**47.476**	**411.014**
50%	**1052.624**	**99.528**	**12939.610**	**159.000**	**15.725**	**19.662**	**4138.136**	**56.633**	**415.066**
75%	**1056.433**	**106.468**	**12992.838**	**159.000**	**16.732**	**20.393**	**4175.928**	**62.303**	**419.680**
max	**1060.235**	**128.062**	**13095.290**	**159.000**	**21.320**	**21.974**	**4175.928**	**70.353**	**460.274**

Table 4.3: Summary of statistical characteristics of selected features for hole size 311 in Well-1.

	TVD	GR	SPPA	RPM	STOR	SWOB	TFLO	ROP5	DTCO_MH_R
count	**16257**	**16257**	**16257**	**16257**	**16257**	**16257**	**16257**	**16257**	**16257**
mean	**1396.011**	**115.353**	**13651.338**	**154.272**	**15.909**	**12.600**	**3710.958**	**24.176**	**371.394**
std	**193.295**	**25.867**	**769.518**	**12.219**	**2.954**	**4.683**	**60.702**	**10.036**	**32.584**
min	**1062.087**	**35.034**	**9572.271**	**111.000**	**6.670**	**1.132**	**3117.774**	**5.402**	**250.784**
25%	**1227.113**	**100.723**	**13144.905**	**149.000**	**13.800**	**9.451**	**3703.538**	**17.537**	**356.222**
50%	**1394.678**	**114.361**	**13686.825**	**149.000**	**14.970**	**12.133**	**3722.433**	**17.784**	**375.825**
75%	**1550.061**	**138.799**	**14197.680**	**160.000**	**18.070**	**15.514**	**3741.329**	**38.886**	**397.873**
max	**1765.971**	**155.089**	**15276.570**	**180.000**	**28.270**	**26.266**	**4138.136**	**49.575**	**466.860**

Table 4.4: Summary of statistical characteristics of selected features for hole size 216 in Well-1.

	TVD	GR	SPPA	RPM	STOR	SWOB	TFLO	ROP5	DTCO_MH_R
count	**21191**	**21191**	**21191**	**21191**	**21191**	**21191**	**21191**	**21191**	**21191**
mean	**2149.576**	**77.210**	**14976.438**	**161.832**	**37.429**	**12.239**	**1991.216**	**29.370**	**279.359**
std	**230.264**	**23.965**	**1118.166**	**14.675**	**8.623**	**3.998**	**91.472**	**14.902**	**38.556**
min	**1766.090**	**29.387**	**9948.147**	**110.000**	**13.230**	**-2.545**	**1662.813**	**4.679**	**148.642**
25%	**1944.349**	**57.309**	**14015.334**	**158.000**	**30.864**	**9.238**	**1984.038**	**16.773**	**260.161**
50%	**2125.299**	**83.554**	**15033.590**	**159.200**	**37.872**	**12.904**	**1991.596**	**27.963**	**280.610**
75%	**2362.843**	**96.482**	**15949.148**	**177.000**	**43.555**	**15.249**	**2002.934**	**40.020**	**305.453**
max	**2539.241**	**127.454**	**17247.270**	**181.000**	**55.090**	**26.231**	**3722.433**	**91.030**	**368.705**

Table 4.5: Summary of statistical characteristics of selected features for hole size 311 in Well-6.

	TVD	GR	SPPA	RPM	STOR	SWOB	TFLO	ROP5	DTCO_MH_R
count	**44289**	**44289**	**44289**	**44289**	**44289**	**44289**	**44289**	**44289**	**44289**
mean	**1632.239**	**87.325**	**28719.066**	**149.512**	**31.171**	**8.214**	**4187.331**	**46.540**	**321.172**
std	**178.125**	**13.408**	**4116.948**	**9.618**	**8.737**	**3.765**	**11.839**	**8.975**	**46.405**
min	**1342.703**	**48.646**	**19813.190**	**108.000**	**7.530**	**0.000**	**4161.300**	**20.639**	**186.292**
25%	**1477.359**	**79.804**	**24604.185**	**139.500**	**21.925**	**5.303**	**4180.215**	**42.140**	**271.473**
50%	**1626.865**	**87.432**	**28720.680**	**150.000**	**30.415**	**7.444**	**4189.672**	**49.580**	**319.500**
75%	**1787.773**	**96.294**	**32122.625**	**158.500**	**40.220**	**10.605**	**4199.130**	**54.345**	**365.084**
max	**2003.019**	**123.372**	**36453.815**	**180.000**	**51.050**	**17.505**	**4218.045**	**69.811**	**421.709**

Table 4.6: Summary of statistical characteristics of selected features for hole size 216 in Well-6.

	TVD	GR	SPPA	RPM	STOR	SWOB	TFLO	ROP5	DTCO_MH_R
count	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**
mean	**2014.247**	**51.578**	**36116.338**	**158.265**	**41.907**	**9.774**	**4187.447**	**31.572**	**295.273**
std	**3.744**	**3.385**	**58.605**	**2.024**	**1.357**	**3.394**	**13.177**	**7.712**	**11.121**
min	**2010.669**	**48.734**	**36015.300**	**155.000**	**39.230**	**5.404**	**4161.300**	**21.643**	**279.708**
25%	**2010.924**	**48.918**	**36050.810**	**157.000**	**40.920**	**6.322**	**4180.215**	**23.734**	**284.703**
50%	**2012.285**	**50.389**	**36127.500**	**158.000**	**42.090**	**9.891**	**4189.672**	**31.118**	**293.209**
75%	**2018.929**	**54.439**	**36165.810**	**159.000**	**42.920**	**13.970**	**4199.130**	**40.601**	**307.355**
max	**2019.696**	**57.894**	**36191.250**	**162.000**	**43.950**	**14.582**	**4199.130**	**40.717**	**314.810**

As previously mentioned, in the dataset of Well-1 and Well-6, the DTCO-MH-R column consists of many NaN values. We omitted all rows containing NaN values. To address the NaN values in the Geological Formation column, the forward fill method (ffill) was employed. This method propagates the last known value forward to fill in the missing values.

4.4 Sonic log and geological formation

The relationship between geological formations and sonic logs is fundamental in understanding subsurface rock properties. Sonic logs measure the travel time of acoustic waves through the formation, which provides insight into the rock's elastic properties. These properties include porosity, lithology, and the presence of fluids, which are all

critical for geological and reservoir characterization. Sonic logs help in identifying different lithologies (rock types) based on their acoustic properties. Different rocks, such as sandstones, shales, and carbonates, have distinct sonic velocities [29]. The acoustic velocity is influenced by the porosity of the rock. High porosity typically results in slower acoustic velocities due to the presence of fluids within the pores, while low porosity corresponds to higher velocities [30]. Sonic logs can indicate the presence of hydrocarbons. Fluid-filled porosity, particularly with hydrocarbons, affects the acoustic impedance and velocity [31]. Understanding the mechanical properties of the rock, such as Young's modulus and Poisson's ratio, which are derived from sonic log data, is crucial for drilling and reservoir management [32].

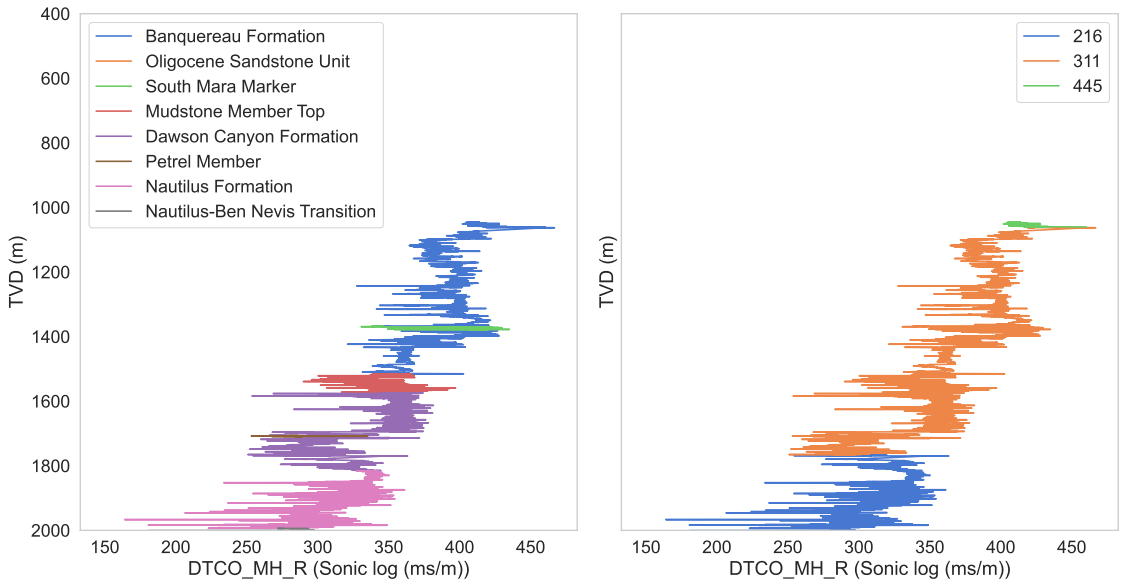


Figure 4.2: Sonic log values attributed to various formations based on TVD and different hole sizes for Well-1.

Figure 4.2, presents the sonic log values attributed to various formations based on TVD and different hole sizes. However, it is noteworthy that in Well-1 there is a small portion of target values for a hole size of 445. As evident from the chart, the initial meters of Well-1 pose a challenge for making predictions due to this missing data. In Well-6 there are no target values for a hole size of 445 as we can see in Figure 4.3.

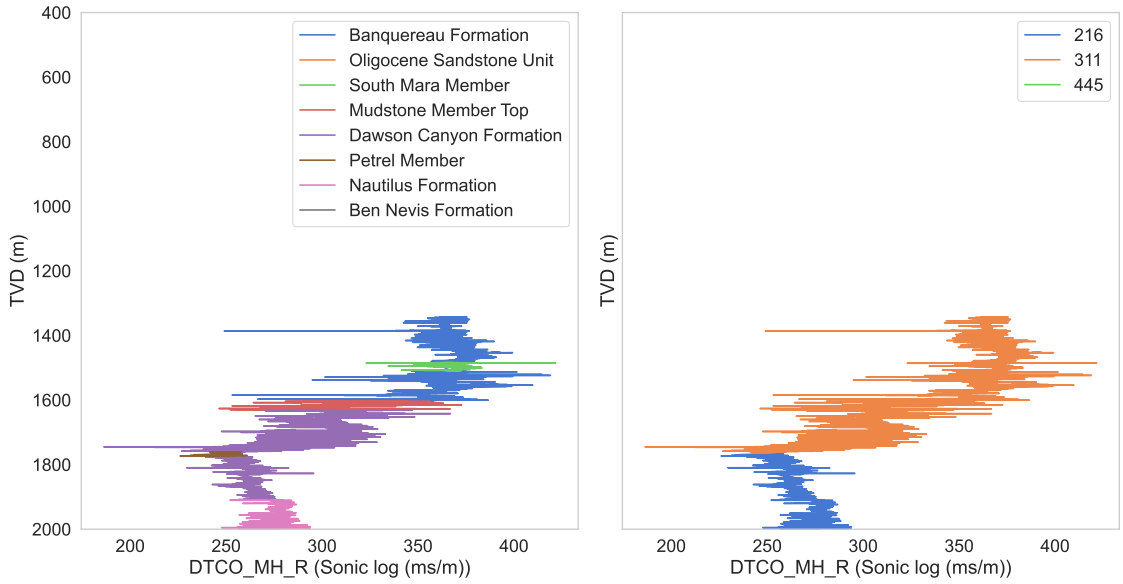


Figure 4.3: Sonic log values attributed to various formations based on TVD and different hole sizes for Well-6.

4.5 Removing outliers

There are two approaches for identifying outliers in the data; one involves utilizing *Interquartile Range (*IQR*)*, and the other involves utilizing standard deviation. In our case, we opted for IQR method to eliminate outliers from the dataset.

To implement this, we calculated the first and third quartiles (representing the 25th and 75th percentiles, respectively) for each column in the input and output features. Subsequently, we computed the IQR by taking the difference between the third quartile (Q3) and the first quartile (Q1). The IQR is a statistical measure that helps assess the dispersion of data, focusing on the middle 50% of the dataset.

Following the IQR calculation, we proceeded to clean the dataset by removing data points (rows) deemed as outliers within the designated columns. This data-cleaning process aids in enhancing the data quality for subsequent analyses or modeling by eliminating data points that exhibit significant deviations from the central tendencies of the dataset.

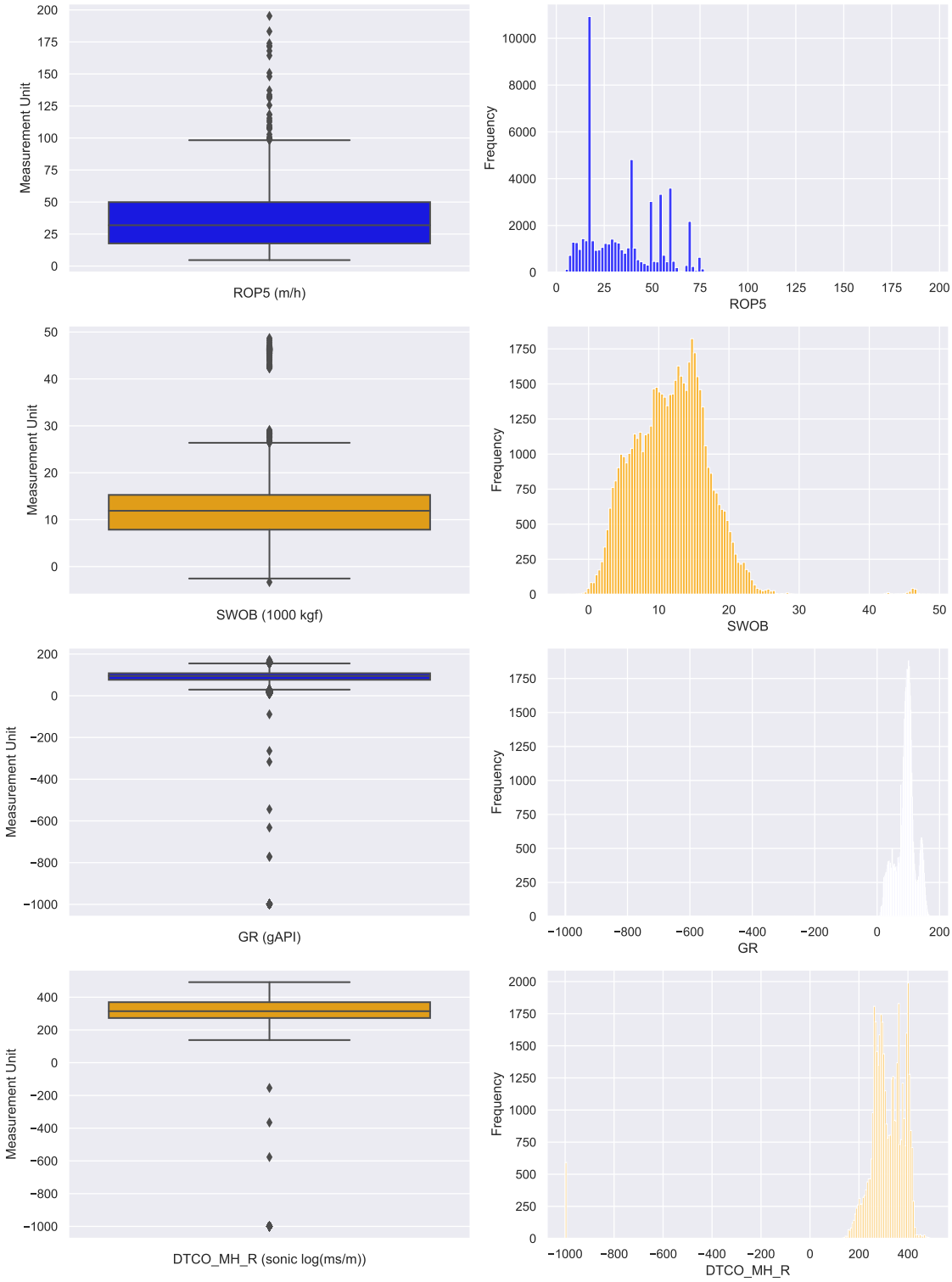


Figure 4.4: Boxplot and histogram of ROP5, SWOB, GR and sonic log before removing outliers of Well-1.

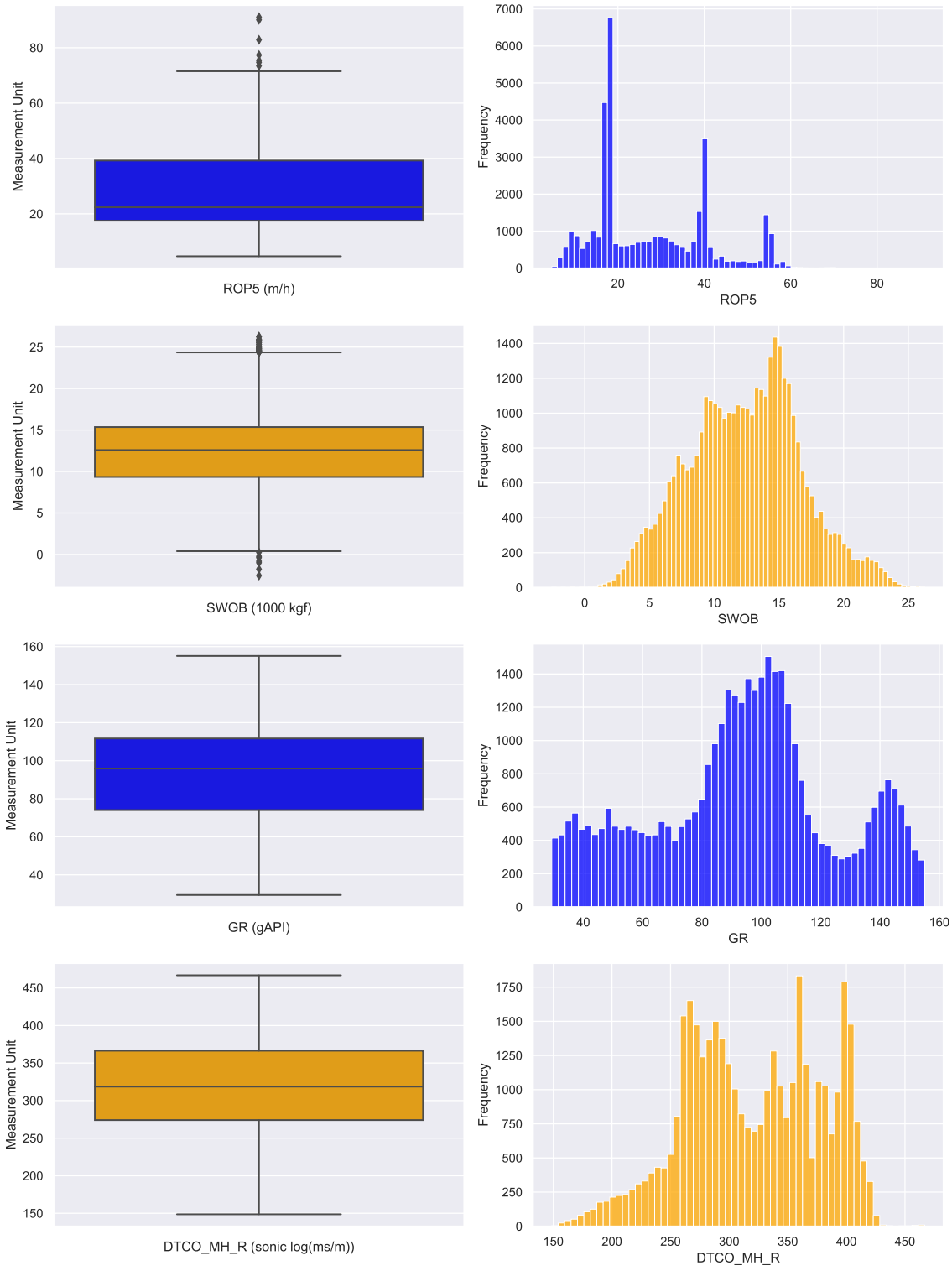


Figure 4.5: Boxplot and histogram of ROP5, SWOB, GR and sonic log after removing outliers of Well-1.

Figures 4.4 and 4.5 showcase box plots and histograms illustrating the distributions of ROP5, SWOB, GR, and the sonic log before and after the removal of outliers for Well-1.

4.6 Scaling

In our models, scaling is employed to reduce the disparities between data points, particularly when the data exhibits significant differences. In simpler terms, scaling aims to create generalized data points in such a way that the distance between them becomes more uniform. Having considered various scalers like quantile transform, log scaling, and Z-score and their applications, the scaling method adopted in our study is the Min-Max scaler. In the following, we briefly discuss alternative scalers and their respective applications.

4.6.1 Min-Max scaler

Min-Max scaling, also referred to as normalization, is a technique employed to rescale the values within a dataset to a predefined range. The primary objective is to standardize all values within the dataset to a common scale, facilitating meaningful comparisons.

The Min-Max scaling formula is expressed as follows:

$$\text{Normalized Value} = \frac{\text{Value} - \text{minimum}}{\text{maximum} - \text{minimum}},$$

where Value is the original value of the data.

This formula scales the values in the dataset such that they are all within the range of 0 to 1. The minimum value in the dataset will be scaled to 0, and the maximum value will be scaled to 1. All other values will be scaled between 0 and 1 based on their position relative to the minimum and maximum values. Min-Max scaling can be useful for bringing all the values in a dataset into the same range for modeling or visualization.[1]

It is important to note that Min-Max scaling can be sensitive to outliers, as the minimum and maximum values will be based on the entire dataset. In our study, we have removed the outliers and then used the scaler on the Well-1 and Well-6. Figure 4.6 shows the density of ROP5, SWOB, GR, and TFLO by applying a Min-Max scaler.

4.6.2 Quantile transform

Quantile transformation is a method used to transform the values of a dataset so that they are distributed evenly across a specified range. This is often done to ensure that the transformed data follows a specific distribution, such as a normal distribution.

There are several ways to perform a quantile transformation, but one common method is to use the quantile function of a probability distribution. For example, to transform a dataset to follow a normal distribution, you could use the quantile

[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

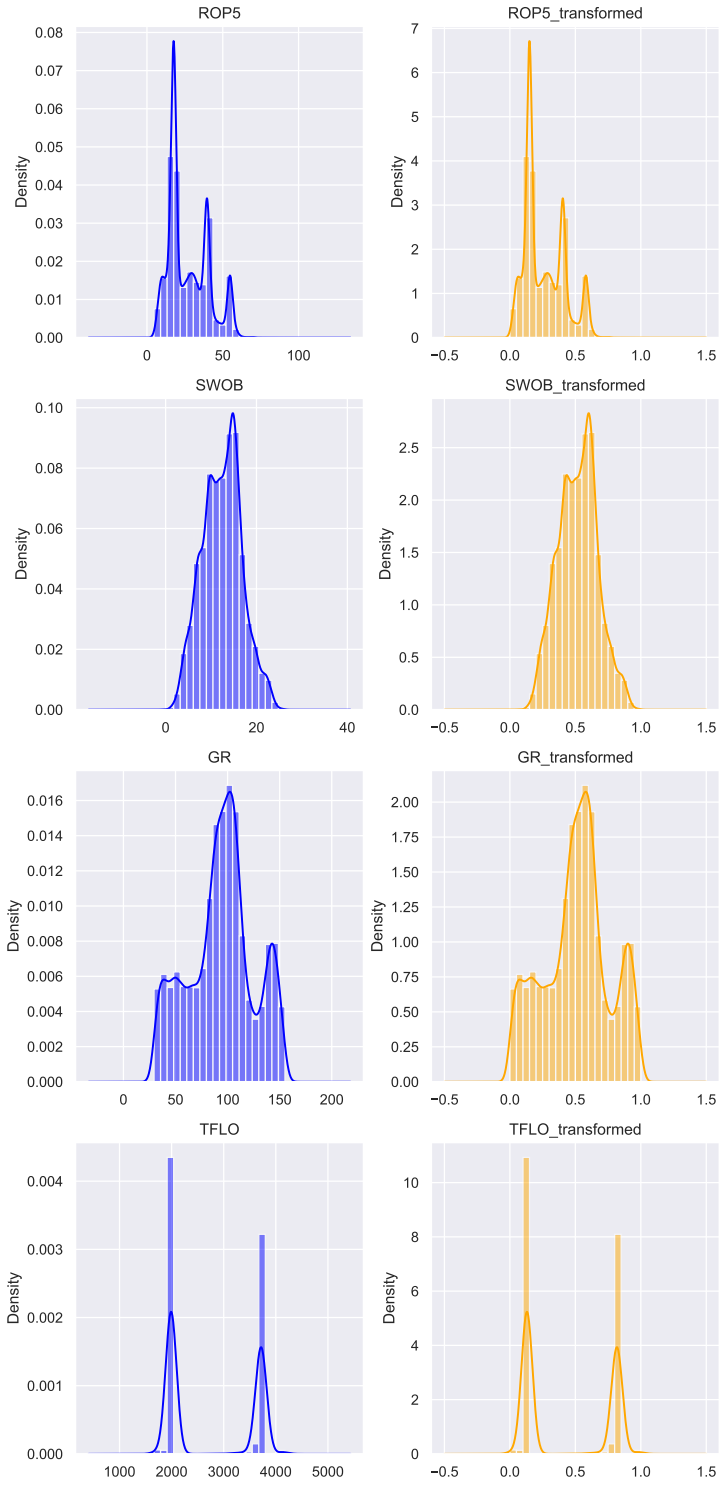


Figure 4.6: Showing the density of ROP5, SWOB, GR and TFLO applying Min-Max scaler.

function of the normal distribution to transform the values of the dataset.

The process of quantile transformation typically involves sorting the values in the dataset from lowest to highest and then mapping each value to a new value based on its quantile within the dataset. For example, if the dataset has 100 values, the value at the 25th quantile (also known as the first quartile) would be mapped to a new value that is 25% of the way through the specified range.

Quantile transformation can be useful for ensuring that the transformed data follows a specific distribution, such as a normal distribution. It can also be useful when you want to bring the values of a dataset into a specific range, such as 0 to 1. It is important to note that quantile transformation is sensitive to the distribution of the original data, as the transformation will be based on the quantiles of the original data. If the original data does not follow the desired distribution, the transformed data may not follow the desired distribution either. In these cases, it may be necessary to use a different method of transformation. This transform also reduces the impact of marginal outliers.[2]

4.6.3 Log scaling

Log scaling is a method used to transform the values of a dataset by taking the logarithm of each value. This can be useful when the values in the dataset span a wide range and you want to bring them into a smaller range for modeling or visualization.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html

Several types of logarithms can be used for log scaling, including the natural logarithm (base e), the common logarithm (base 10), and the logarithm to any other base. The choice of base will depend on the specifics of the dataset and the desired range of the transformed values.

Log scaling can be useful when the values in the dataset span a wide range and you want to bring them into a smaller range for modeling or visualization. It can also be beneficial when the original values are skewed or have a long-tailed distribution, as log scaling can help to bring the values into a more symmetrical distribution.

4.6.4 Z-score

Standardization, also known as Z-score normalization, is a method used to transform the values of a dataset so that they have a mean of 0 and a standard deviation of 1. This is often done to bring the values in a dataset into a common scale for modeling or comparison.

The formula for standardization is as follows:

$$\text{Standardized value} = \frac{\text{Value} - \text{mean}}{\text{standard deviation}},$$

where Value is the original value of the data.

This formula transforms the values in the dataset such that they have a mean of 0 and a standard deviation of 1. All the values in the dataset will be transformed to fall within the range of -3 to 3, with most of the values falling within the range of -1 to 1. Standardization can be useful when we want to bring all the values in a dataset into

the same scale for modeling. It is noteworthy to mention that standardization assumes the original data follows a normal distribution. If the original data does not follow a normal distribution, the transformed data may not follow a normal distribution either. In these cases, it may be necessary to use a different method of scaling, such as Min-Max scaling or quantile transformation.

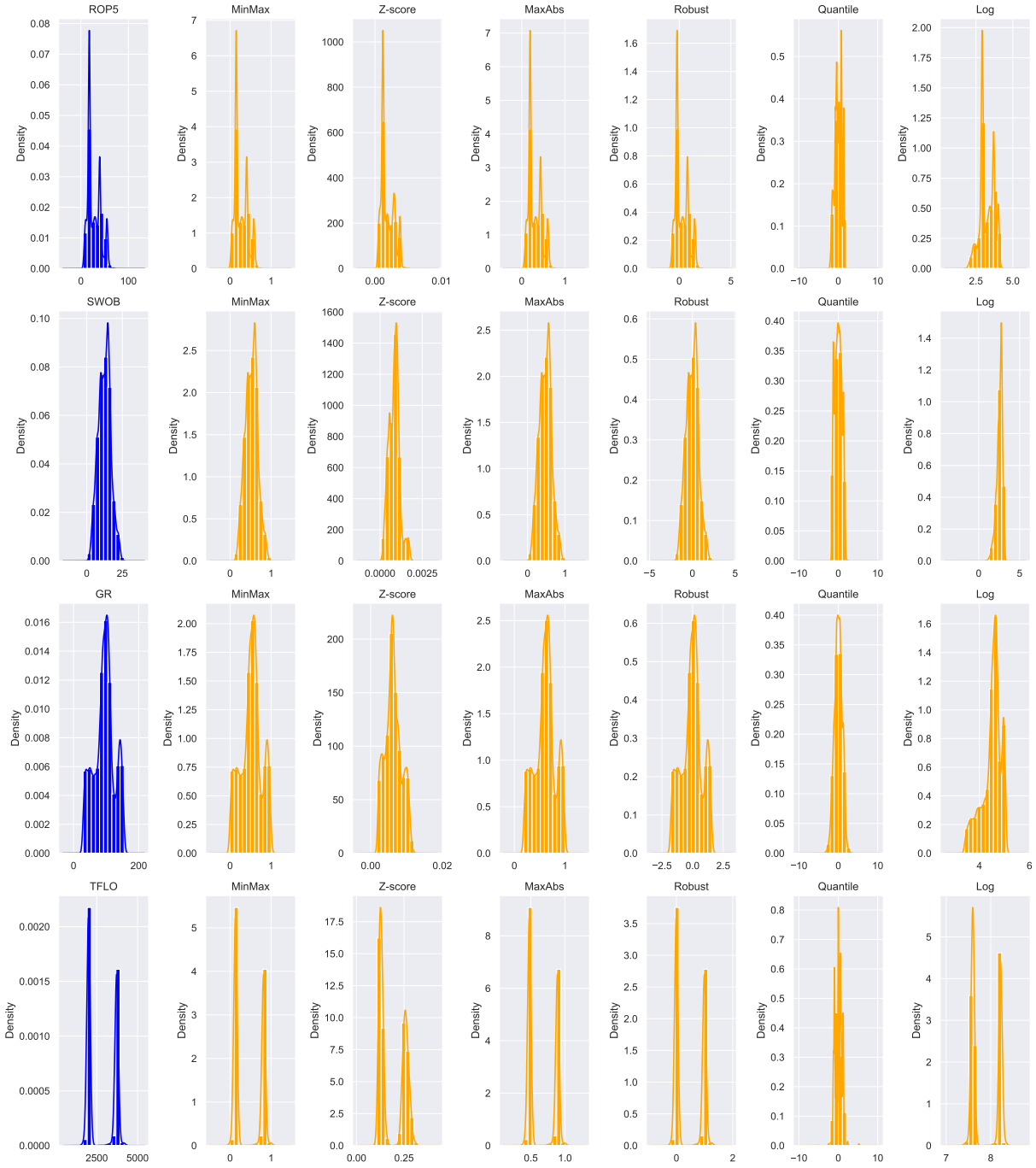


Figure 4.7: Applying Min-Max, Z-score, MaxAbs, Robust, Quantile, and Log scalers on features ROP5, SWOB, GR and TFLO of Well-1.

Figure 4.7 shows the effect of different scalers on features ROP5, SWOB, GR, and TFLO of Well-1.

In our research, we encountered features with varying units or scales. To make the features comparable without losing significant information, we opted for Min-Max scaling. Furthermore, we applied different scalers to the data for modeling and validated our decision by comparing results using performance metrics.

4.7 Accuracy measurements

At the end of this chapter, we aim to provide a short explanation of the accuracy measurements utilized. These measurements will be employed in the subsequent chapter to evaluate the algorithms. In these formulas, y_i is the actual value, $\hat{y}_i$ is the predicted value, $\bar{y}$ is the mean of the actual values and n is the sample size.

Mean Absolute Error (MAE):

MAE measures the average absolute difference between the actual and predicted values. It provides a straightforward representation of the average error and is calculated by taking the mean of the absolute differences between each actual and predicted value. The formula is:

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

Mean Squared Error (MSE):

MSE calculates the average of the squared differences between the actual and predicted values. Squaring the errors emphasizes larger errors, making it sensitive to

outliers. The formula is:

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

Root Mean Squared Error (RMSE):

RMSE is the square root of MSE, providing a measure of the average magnitude of errors in the same units as the target variable. It is often preferred when the errors are expected to be normally distributed. The formula is:

$$RMSE = \sqrt{MSE}$$

Coefficient of determination or R-squared (R^2):

R^2 score represents the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, with 1 indicating a perfect fit. It is a relative measure that assesses the goodness of fit of a regression model. The formula is:

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

In this chapter, we have cleaned our dataset through various techniques, such as outlier removal and scaling. With the data now prepared, we transition to the methodology phase. In the upcoming chapter, we will detail the methodologies employed for analysis, encompassing advanced algorithms and fundamental statistical approaches.

Chapter 5

Methodology

The methodology involved several steps to achieve the study's objectives. This study employed LSTM, RF, and XGBoost algorithms for predictive modeling and ensured optimized hyperparameter selection and grid search with cross-validation to validate the models' performance. Cross plots of predicted versus actual compressional wave travel times on blind data were generated to validate the predictions made by LSTM, XGBoost, and RF algorithms. The study concluded that while all three algorithms successfully predicted the compressional sonic log, RF and XGBoost exhibited superior generalization abilities compared to LSTM model.

In the subsequent section, we present the outcomes of our model evaluation for predicting the sonic log.

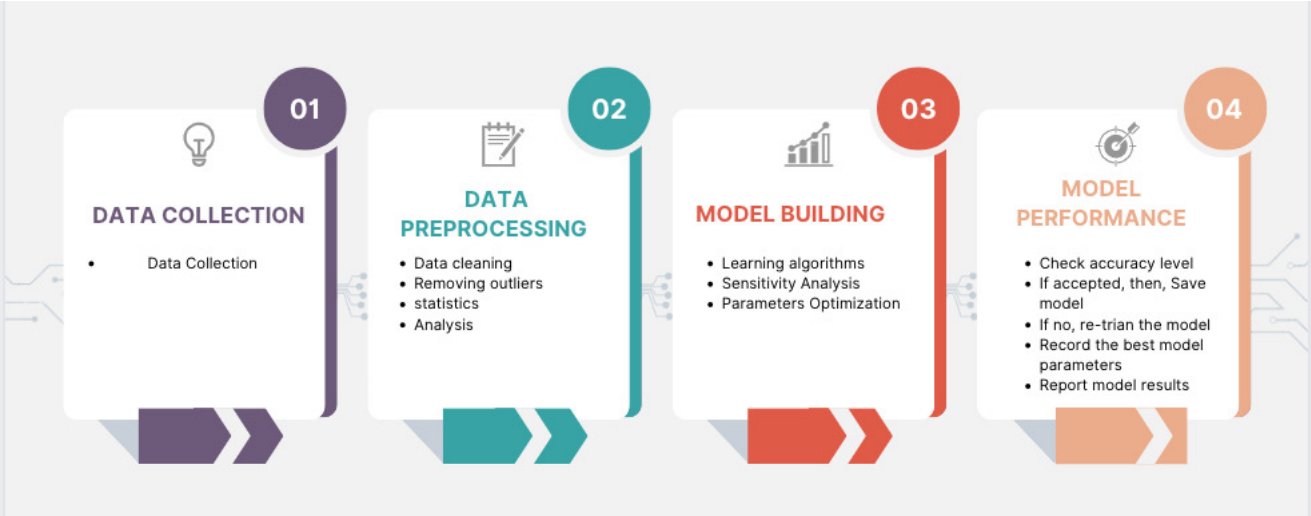


Figure 5.1: Working process

5.1 Working process

The general structure of the work has three basic steps following data collection illustrated in Figure 5.1[1] :

- Data preparation
- Feeding the machine learning model with the training data
- Evaluating the model on test, compute R^2 score, MSE , RMSE and MAE.

5.2 Data exploration and feature engineering

The dataset utilized in this study comprises numerical observations collected in the drilling project for Well-1 and Well-6. In total, the dataset encompasses 42 distinct

[1]Template adapted from www.canva.com

features with 55,574 observations for Well-1 and 49 features for 68,272 observations for Well-6. Notably, certain features are defined based on TVD.

To distill the essential features from this dataset, we adopted a comprehensive approach. Initially, we employed a correlation matrix to identify features demonstrating a strong linear relationship with the target variable. Through the application of the Pearson correlation method, we successfully identified and removed highly correlated features, achieving two key objectives: the preservation of crucial features and the improvement of model interpretability. In addition, we conducted a thorough review of relevant research literature to explore the potential inclusion of other pertinent features. Finally, we considered the availability of variables in the selection process, ensuring a pragmatic approach to feature inclusion. Through the integration of these methodologies, we effectively pinpointed the key features that provide comprehensive insights into explaining the target variable.

The selected input features are as follows:

- TVD: True Vertical Depth (m)
- RPM: Rotational Speed (c/min)
- TFLO: Total Flow rate of all active pumps (L/min)
- SWOB: Surface Weight On Bit (1000 kgf)
- GR: Gamma Ray (gAPI)
- Hole size: Classical variable including 445, 311, or 216 values (mm)

- STOR: Surface Torque (kN.m)
- ROP5: Rate Of Penetration averaged over the last 5 feet (m/h)

Our target variable is sonic log which is denoted as

- DTCO-MH-R (ms/m)

It is crucial to emphasize that within our model, we executed specific data preprocessing procedures. These particular steps play a pivotal role in ensuring the attainment of dependable and uniform results across a variety of models and methodologies. Ultimately, they empower us to make well-founded decisions in practical drilling applications.

To elaborate further, it is worth noting that we undertook the following actions. We removed rows containing NaN values from the dataset to maintain data integrity and completeness. To ensure consistency, we standardized column names, with particular attention to the Hole Size feature. This enhances clarity and coherence in the dataset. Outliers, which could negatively impact model performance, were systematically eliminated using the interquartile method. This ensures that the model is trained on data free from extreme values that might skew predictions and cells with a value of -999 were identified and dropped within the dataset.

By executing these essential data refinement measures and standardizing the dataset, we established a foundation where our model is trained on high-quality, dependable data. This, in turn, enables us to furnish precise predictions for the sonic

log. Such an approach is imperative for achieving dependable and uniform outcomes across the model, ultimately equipping us to make informed decisions in practical drilling scenarios. Figures 5.2, 5.3 and 5.4 illustrate the processed data corresponding to TVD for various hole sizes in Well-1. Additionally, Figure 5.5 depicts the data for hole size 311 in Well-6.

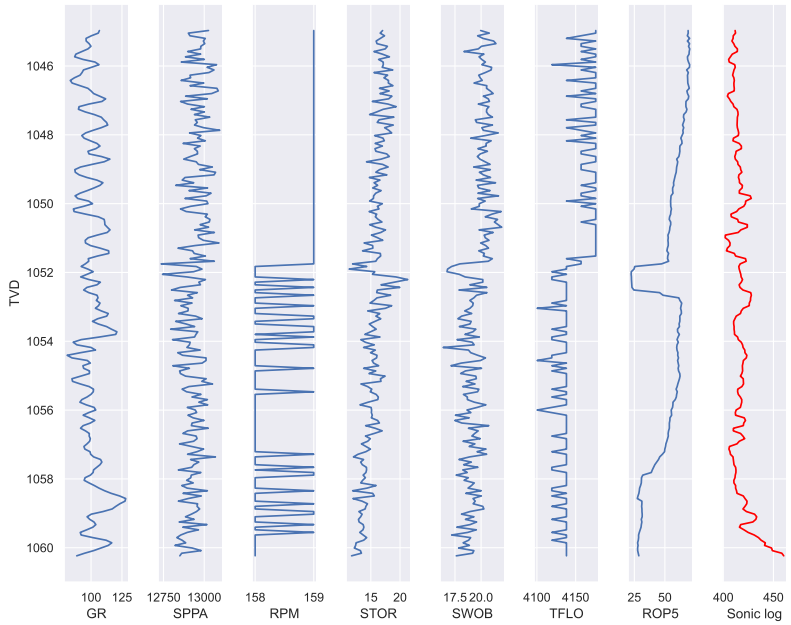


Figure 5.2: The values of selected features at various TVDs for Well-1, with a hole size of 445.

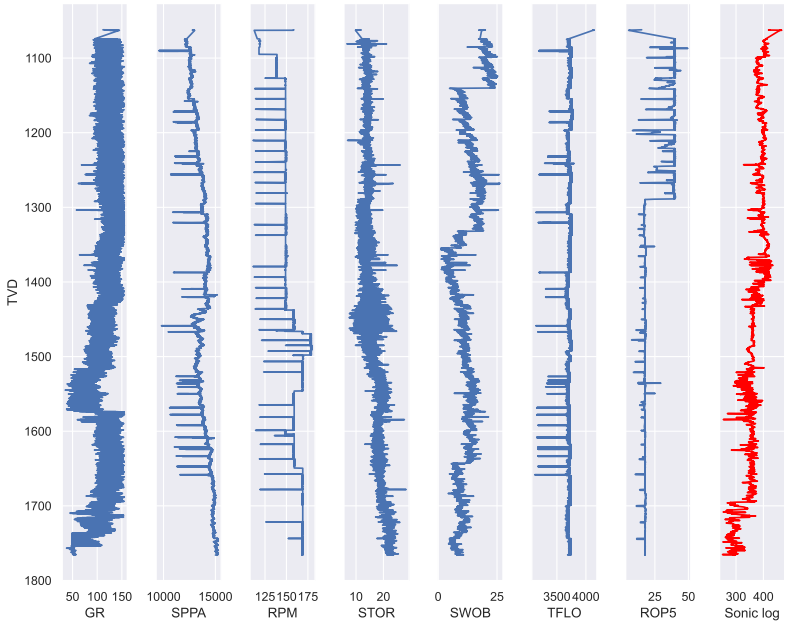


Figure 5.3: The values of selected features at various TVDs for Well-1, with a hole size of 311.

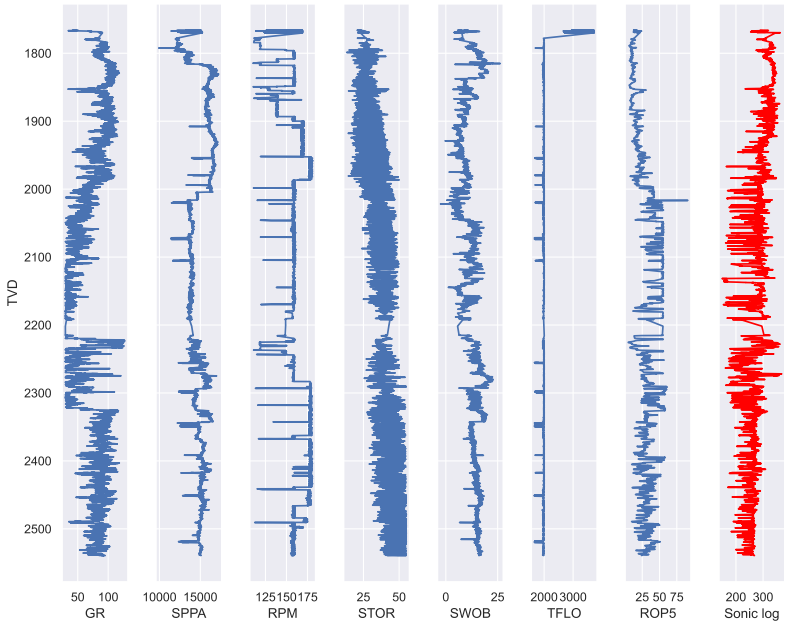


Figure 5.4: The values of selected features at various TVDs for Well-1, with a hole size of 216.

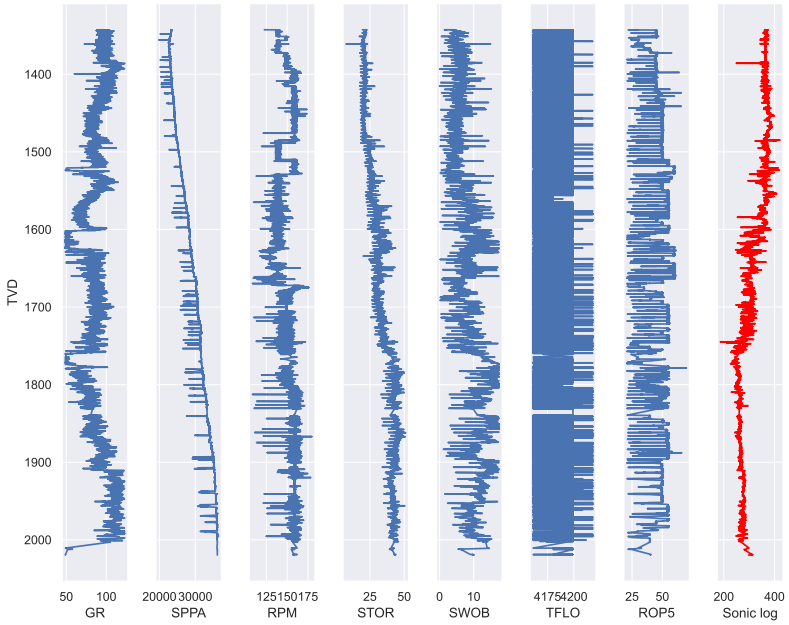


Figure 5.5: The values of selected features at various TVDs for Well-6, with a hole size of 311.

5.3 Data management

Data management is the systematic and efficient process of acquiring, organizing, storing, and utilizing data. In our models, where the signal of variables follows a sequential pattern based on TVD, it is crucial to maintain the sequence integrity of train and test sets. This can not be achieved by utilizing the *train-test split* function from the *sklearn* library which shuffles the signals.

The primary objective of our study is to optimize a machine learning model for continuous sonic log predictions as the depth increases across diverse formations. In this context, the training set comprises data up to a certain depth, whereas the test

set includes data beyond that depth. For example, in Well-1, TVD spans from 1062 to 1770 meters, providing data for a depth of 708 meters for hole size 311. The training data consists of observations (data points) ranging from 0 to 14631 for this hole size, while the test set extends from observations 14631 to 16256. This meticulous approach ensures that the model generalizes effectively to new data, facilitating practical application in real-world scenarios without encountering data leakage or disrupting sequences. Furthermore, our study incorporates a blind prediction strategy. We utilized information from a particular segment of Well-1 to predict the behavior of a similar well, Well-6, from a similar geological formation by considering TVD ranges. Similarly, we employed a segment of Well-6 as training data to forecast a specific section of Well-1.

This methodology empowers us to make precise predictions with constrained information, proving beneficial in practical scenarios where data availability is restricted. Through harnessing the existing data, our objective is to construct a robust model capable of accurately predicting the target variable for novel, unseen data instances.

5.4 Data splitting

For XGBoost, LSTM, or RF models, we divided the dataset into a training set and a test set. The initial 90% of the data were utilized as training data, while the remaining 10% were set aside for testing purposes. This division was consistently applied to all three distinct hole sizes for Well-1. Tables 5.1 and 5.2 illustrate the

sizes of training and test sets for Well-1 and Well-6, respectively. As we can see, there is no data for hole size 445 of Well-6, and there are limited data points for hole size 216 of Well-6.

Table 5.1: Description of data corresponding to Well-1 and the number of data points for three different hole sizes.

Hole size	445	311	216
Well-1	201	16256	21190
Train size	181	14631	19071
Test size	20	1625	2119

Table 5.2: Description of data corresponding to Well-6 and the number of data points for three different hole sizes.

Hole size	445	311	216
Well-6	0	44288	16
Train size	0	39860	15
Test size	0	4428	1

As highlighted earlier, employing a random allocation method for dataset division was not feasible due to the crucial role of event sequencing in time series analysis, especially in our scenario where the sequence order holds paramount importance.

Consequently, we had to be cautious when partitioning the data into training and test sets to maintain the accurate sequencing of events, thereby upholding the reliability of our results. Therefore the function "Train test split", is designed to split a dataset into training and testing sets. This function takes two parameters as input. The first parameter, *data*, represents the dataset that we need to split into training and testing sets.

The second parameter, *n_test*, is an integer that specifies the number of data points to reserve for testing. These data points will be used to evaluate the performance of a predictive model. At the end, the function returns two sets of data. The first set includes all data points from the beginning of the data array up to (but not including) the last *n_test* data points. This portion of the data is typically used for training the predictive model. The second set includes the last *n_test* data points from the data array. This portion of the data is reserved for testing the model's predictions. This type of data split is common when working with time series data to assess how well a model generalizes to unseen data. Algorithm 16 illustrates the pseudocode which shows the functioning of the "Train test split" function in more detail.

5.5 Preparing series for supervised learning

We have implemented a function designed to convert time series data into a format suitable for supervised learning tasks. This function takes sequence data as input and rearranges it to make it usable for training and evaluating XGBoost, RF, and

Algorithm 16 Train test split

Require: *data*: The entire dataset

Require: *n_test*: Number of test instances

function TRAINTESTSPLIT(*data*, *n_test*)

    train ← *data*[: −*n_test*, :]

    test ← *data*[−*n_test* :, :]

    return *train*, *test*

LSTM models. By specifying the number of lag observations to use as input features, the number of future observations to predict, and ensuring that the data is devoid of NaN values, the function outputs the transformed data in the form of a *NumPy* array. This format is ideal for conducting supervised learning, enabling the utilization of past observations as features to predict future ones. The "series-to-supervised" function is designed to handle diverse input parameters, catering to multiple formats of time series data. It primarily accepts input data as either a list, a 2D *NumPy* array, or a data frame featuring multiple features, as is pertinent to our particular study.

There are some parameters in this function, *n_in* is the number of lag observations as input (default is 30) and *n_out* is the number of observations to predict as output (default is 10). Also, we add a boolean flag that determines whether to drop rows with NaN values (default is True). Then, the function checks the type of the data variable to determine the number of variables. If data is a list, it sets *n_vars* to 1; otherwise, it sets it to the number of columns in the data and converts data to a

data frame (df). It converts the input data into a data frame using the data frame constructor. This is typically done for easier manipulation of the data. By studying the features and correlation between the selected features and output, our dataset is multivariate. Now the function creates lag features for the input data. It does this by looping through the range of *n_in* (the number of lag observations) and appending lagged versions of the data to the columns list. This effectively creates columns that represent past observations of time series data.

Next, the function creates forecast features. The function appends forecast features to the columns list by shifting the data in the opposite direction for a range of *n_out* values.

In the end, the function combines lag and forecast features. It concatenates the columns in the columns list along the horizontal axis to create a single data frame. This data frame now contains both lagged and forecasted features. It also handles NaN values, if the *dropnan* is set to True, it removes rows containing NaN values from the data frame. Finally, the function returns the values of the data frame as a *NumPy* array. Algorithm 17 illustrates the pseudocode of the "Convert time series to supervised learning" function.

Algorithm 17 Convert time series to supervised learning

Require: *data*: time series data

Require: *n_in*: Number of lag observations as input (default: 30)

Require: *n_out*: Number of future observations to predict (default: 10)

Require: *dropnan*: Drop rows with NaN values (default: True)

function SeriesToSupervised($data, n\_in, n\_out, dropnan$)

    $n\_vars \leftarrow$ number of features ▷ Assuming data is multivariate

    $df \leftarrow$ DataFrame($data$)

    $cols \leftarrow$ empty list

    ▷ Input sequence (t-n, ..., t-1)

    for $i \leftarrow n\_in$ to 1 by -1 **do**

        Append df.shift(i) to $cols$

    ▷ Forecast sequence (t, t+1, ..., t+n)

    for $i \leftarrow 0$ to $n\_out - 1$ **do**

        Append df.shift($-i$) to $cols$

    $agg \leftarrow$ Concatenate($cols$, axis=1)

    print(agg)

    ▷ Drop rows with NaN values

    if *dropnan* **then**

        agg.dropna(*inplace=True*)

    return agg.values

5.6 XGBoost model

XGBoost is an open-source software library that provides a gradient-boosting framework for Python and other programming languages. It was developed by Tianqi Chen and is commonly used for supervised learning problems like regression, classification, and ranking [13].

One of the reasons why XGBoost is a popular choice for many data science problems is because it provides high performance, and scalability, and offers several tuning hyperparameters that can be adjusted to suit the specific problem at hand.

To use XGBoost for regression, we need to import the *XGBoost* library and create an XGBRegressor object. The fit method is then used to train the regressor on the data, while the predict method is used to make predictions on new data. To improve the performance of the XGBoost regressor, several hyperparameters can be tuned. Some of the most important ones include the learning rate, the maximum depth of the tree, the number of trees in the forest, and the subsample rate.

In this research, our workflow commenced with data loading, followed by data preprocessing and the subsequent division of the dataset into training and testing subsets. We experimented with various combinations of hyperparameters in our XGBoost regressor, based on the *GridSearchCV* from the *scikitlearn* library; the most effective hyperparameters were number of estimators and learning rate. Specifically, we tested with 1000, 2000, 5000, and 10000 estimators, along with learning rates of 0.01, 0.05, 0.1, and 0.3. Ultimately, we initialized the XGBoost regressor with a

specific configuration: 1000 estimators and a learning rate of 0.05. In Table 5.3, a comparison between different hyperparameters based on MAE measurement is provided. After tuning the hyperparameters, the model was trained on the training dataset, and predictions were generated for the test dataset.

Table 5.3: A comparison between different hyperparameters based on MAE measurement.

Number of estimators	Learning Rate	MAE
1000	0.01	3.6029
1000	0.05	3.1066
2000	0.05	3.0993
5000	0.1	3.4400
10000	0.3	3.8363

Based on Table 5.3, the ideal configuration is 2000 estimators with a learning rate of 0.05. However, due to limitations in training time, we chose the second-best option: 1000 estimators with a learning rate of 0.05, which only slightly affects the results. In this study, we carried out separate training and testing using Well-1 and Well-6 applying XGBoost, RF, and LSTM models. Additionally, at the end of this chapter, we performed a blind prediction by training the LSTM model on 60 meters of the dataset from Well-1 on hole size 311 and testing it on 40 meters of data

corresponding to hole size 311 of Well-6. Furthermore, we repeated this process by using 60 meters of Well-6 for training and predicted 40 meters of Well-1 to explore the performance from a different perspective. The detailed presentation of the procedure and results is provided in Section 5.10.3.1.

Algorithm 18 illustrates the pseudocode of the "XGBoost forecast" function.

Algorithm 18 XGBoost forecast

Require: $train$: Training data

Require: $testX$: Test input data

1: **function** XGBoostForecast($train, testX$)

2: $train \leftarrow$ Array($train$)

3: $trainX, trainy \leftarrow train[:, :-1], train[:, -1]$

4: $model \leftarrow$ XGBRegressor($n\_estimators$ = 1000, $learning\_rate$ = 0.05, min_child_weight = $[default$ = 1], reg_lambda = $[default$ = 1], reg_alpha = $[default$ = 0], $subsample$ = $[default$ = 1], $colsample\_bytree$ = $[default$ = 1], max_depth = $[default = 6]$)

5: $model$.fit($trainX, trainy$)

6: $\hat{y} \leftarrow model$.predict(Array($[testX]$))

7: **return** $\hat{y}[0]$

5.7 Walk-forward validation for XGBoost

Before proceeding to the model evaluation, the following preparatory steps were undertaken. An empty list was initialized to capture model predictions and store forecasted values. Subsequently, the dataset was effectively split into training and testing subsets, employing the data splitting method outlined in Section 5.4. Notably, the *train test split* method from the *sklearn* library was not utilized, and shuffling was avoided.

A systematic iteration was conducted, encompassing each depth step within the test set. During each iteration, relevant input features, including lag observations, and the actual target value were considered. The trained XGBoost model was then utilized to generate one-step predictions. This involved making predictions using historical data and input features, with the predicted value stored in a variable labeled $\hat{y}$.

These predictions were systematically recorded, and the actual observation was appended to the historical data. Subsequently, evaluation metrics, such as MAE, R^2, MSE, and RMSE were computed to assess the model's performance. Model predictions were generated through the invocation of the "XGBoost forecast" function from Section 5.6. Algorithm 19 delineates the operation of the "Walk forward validation for XGBoost model" function.

Algorithm 19 Walk forward validation for XGBoost model

Require: $data$: The entire dataset

Require: n_test: Number of test instances

function WalkForwardValidation($data, n\_test$)

  $predictions \leftarrow$ empty list

  $train, test \leftarrow$ TrainTestSplit($data, n\_test$)

  $history \leftarrow$ copy of $train$

  for $i \leftarrow 1$ to $len(test)$ **do**

    $testX, testy \leftarrow$ SplitTestRow($test[i]$)

    $\hat{y} \leftarrow$ XGBoostForecast($history, testX$)

    Append $\hat{y}$ to $predictions$

    Append $test[i]$ to $history$

    Print((expected, predicted) = $(testy, \hat{y})$)

  $MAE \leftarrow$ MeanAbsoluteError($test[:, -1], predictions$)

  $R^2 \leftarrow$ R_Squared($test[:, -1], predictions$)

  $RMSE \leftarrow \sqrt{\text{MeanSquaredError}(test[:, -1], predictions)}$

  $MSE \leftarrow$ MeanSquaredError($test[:, -1], predictions$)

  return $MAE, R^2, RMSE, MSE, test[:, -1], predictions$

The outcomes of the XGBoost model are presented in Tables 5.4 and 5.5 for Well-1 and Well-2, respectively.

Table 5.4: Summary of results of sonic log prediction using XGBoost model for Well-1 on the test datasets. The XGBoost model shows varying performance across different test datasets for Well-1. The model achieves the highest accuracy and best fit on the third dataset, as evidenced by the lowest MAE, $RMSE$, and MSE values and a high R^2 value, which demonstrates more sample data can result in more accurate results.

XGBoost				
Well-1	MAE	R^2	$RMSE$	MSE
Hole size: 445	4.8554	0.7950	5.7426	32.9779
Hole size: 311	2.3419	0.9889	3.4052	11.5958
Hole size: 216	1.4439	0.9811	2.3901	5.7126

Table 5.5: Summary of results of sonic log prediction using XGBoost model for Well-6 on the test dataset.

XGBoost				
Well-6	MAE	R^2	$RMSE$	MSE
Hole size: 311	0.4973	0.9764	1.0851	1.1775

In Figures 5.6a, 5.6b, 5.6c, and 5.6d, the expected and predicted values, derived from varying numbers of lag observations for 30 future observations using the

XGBoost model, are depicted. Among these figures, Figure 5.6a featuring 30 lag observations exhibited superior predictive performance as evidenced by the lowest MAE, $RMSE$, and MSE values and a high R^2 value in Table 5.9 and demonstrated a more consistent trend during the sonic log value increments, resulting in smoother predictions devoid of fluctuations.

Table 5.6: Summary of results of sonic log prediction using XGBoost model for Well-1 on the test dataset.

XGBoost					
Well-1	lag observations	MAE	R^2	$RMSE$	MSE
Hole size: 311	30	4.4835	0.7785	5.6538	31.9658
Hole size: 311	40	4.6961	0.7592	5.8954	34.7563
Hole size: 311	60	4.6130	0.7559	5.9352	35.2275
Hole size: 311	80	4.6939	0.7478	6.0331	36.3993

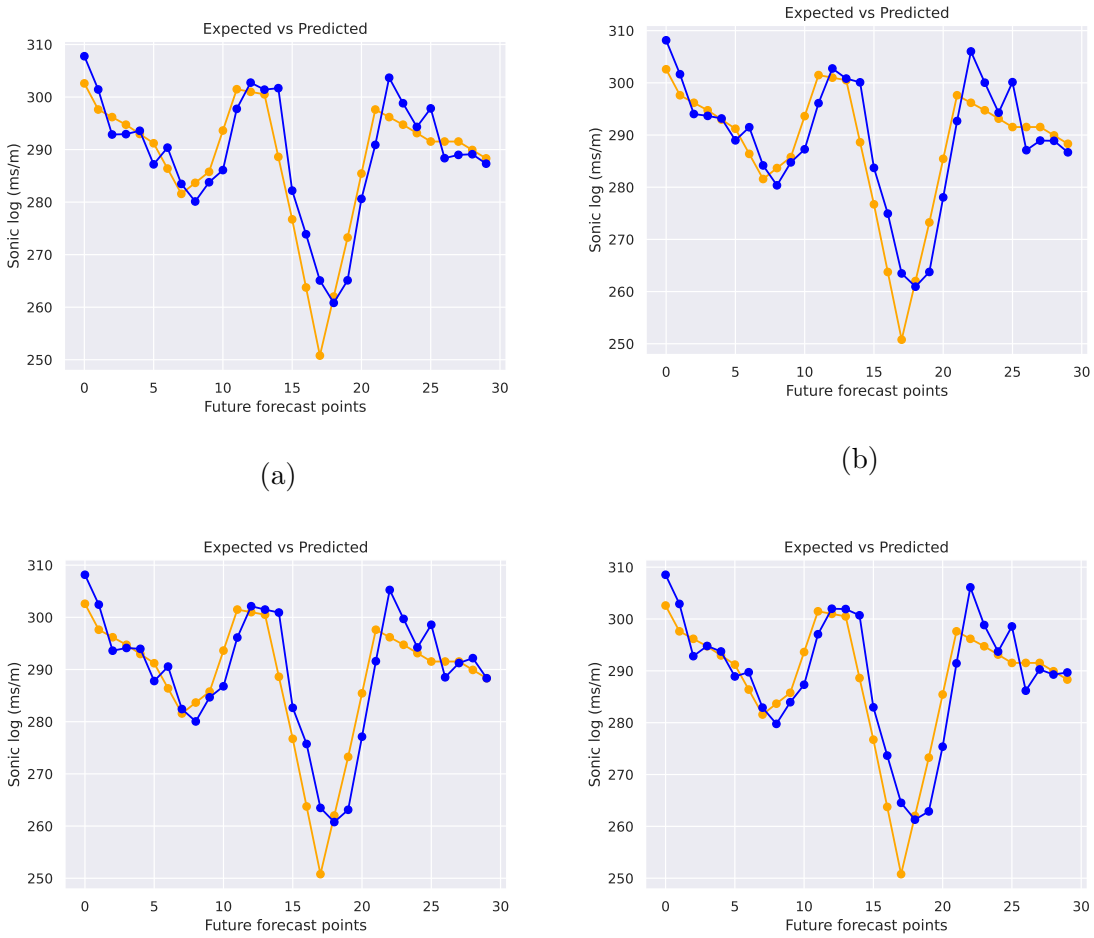


Figure 5.6: Comparison of expected and predicted outcomes using varying numbers of lag observations for the same future observations, generated by the XGBoost model for hole size 311 of Well-1. (a) 30 lag observation and future forecast for 30 values, (b) 40 lag observation and future forecast for 30 values, (c) 60 lag observation and future forecast for 30 values, (d) 80 lag observation and future forecast for 30 values. Based on metric measurements in Figure 5.9, 30 lag observation has the best result.

5.8 Random Forest

Random Forest is an ensemble learning technique that builds multiple decision trees during training and combines their outputs for improved accuracy and robustness. It works by training each tree on a random subset of the training data and then combining their predictions during the testing phase. Known for its versatility and effectiveness in handling complex datasets, RF is widely used for classification and regression tasks in machine learning. [15, 16, 11]

The procedures, like splitting the dataset into training and testing sets and transforming it into depth series, follow the same approach as with the XGBoost model. In Sections 5.4 and 5.5, we introduced "Train test split" and "Convert time series to supervised learning" functions, and thorough explanations were provided. The primary difference lies in the application of the RF model. Within the "Walk forward validation" function, the technique employed for training and adding each trained value to the test set is RF model, accompanied by its unique set of hyperparameters.

To utilize RF for regression, we need to import the necessary libraries, such as *Scikit-learn*, and create a Random Forest Regressor object. The fit method is then employed to train the regressor on the data, while the predict method is utilized to make predictions on new data.

In the performance of the Random Forest regressor, we used n-estimators (fixed 100). Algorithm 20 illustrates the pseudocode of RF model.

Algorithm 20 Random Forest forecast

Require: $train$: Training data

Require: $testX$: Test input data

1: **function** RandomForestForecast($train$, $testX$)

2: $train \leftarrow \text{Array}(train)$

3: $trainX, trainy \leftarrow train[:, :-1], train[:, -1]$

4: $model \leftarrow$ RandomForestRegressor($n\_estimators = 100$, $random\_state = 42$)

5: $model$.fit($trainX, trainy$)

6: $\hat{y} \leftarrow model$.predict(Array($[testX]$))

7: **return** $\hat{y}[0]$

5.9 Walk forward validation applying Random Forest

Before moving on to model evaluation, several preparatory steps were carried out. First, an empty list was initialized to capture model predictions and store forecasted values. Subsequently, the dataset underwent an effective split into training and testing subsets using the data splitting method described in Section 5.4. Notably, the *Sklearn* library's train test split method was not employed, and shuffling was deliberately avoided.

A systematic iteration was conducted, covering each step within the test set. In

each iteration, relevant input features, including lag observations, and the actual target value were taken into account. The trained RF model was then employed to generate one-step predictions. This involved making predictions using historical data and input features, with the predicted value stored in a variable labeled $\hat{y}$.

These predictions were systematically recorded, and the actual observation was appended to the historical data. Subsequently, evaluation metrics, such as MAE, R^2, MSE, and RMSE were computed to assess the model's performance. Model predictions were generated through the invocation of the "Random Forest" function. The provided pseudocode in Algorithm 21 outlines the operation of the "Walk forward validation for Random Forest model" function. The outcomes of RF model for different hole sizes of Well-1 and Well-6 are presented in Tables 5.7 and 5.8 respectively.

Algorithm 21 Walk forward validation for Random Forest

Require: $data$: The entire dataset

Require: n_test: Number of test instances

function WalkForwardValidation($data, n\_test$)
    $predictions \leftarrow$ empty list
    $train, test \leftarrow$ TrainTestSplit($data, n\_test$)
    $history \leftarrow$ copy of $train$
    for $i \leftarrow 1$ to $len(test)$ **do**
        $testX, testy \leftarrow$ SplitTestRow($test[i]$)
        $\hat{y} \leftarrow$ RandomForestForecast($history, testX$)
        Append $\hat{y}$ to $predictions$
        Append $test[i]$ to $history$
        Print((expected, predicted)=(testy, $\hat{y}$))
    $MAE \leftarrow$ MeanAbsoluteError($test[:, -1], predictions$)
    $R^2 \leftarrow$ R_Squared($test[:, -1], predictions$)
    $RMSE \leftarrow \sqrt{\text{MeanSquaredError}(test[:, -1], predictions)}$
    $MSE \leftarrow$ MeanSquaredError($test[:, -1], predictions$)
    return MAE, R^2, $RMSE$, MSE, $test[:, -1]$, $predictions$

Table 5.7: Summary of results for sonic log prediction using Random Forest model for Well-1 on the test datasets. The RF model shows varying performance across different test datasets for Well-1. The model achieves the highest accuracy and best fit on the hole size 216, as evidenced by the lowest MAE, $RMSE$, and MSE values and a high R^2 value.

Random Forest				
Well-1	MAE	R^2	$RMSE$	MSE
Hole size: 445	7.487	0.543	8.579	73.607
Hole size: 311	2.063	0.991	3.046	9.278
Hole size: 216	1.420	0.981	2.389	5.708

Table 5.8: Summary of results of log prediction using Random Forest model for Well-6 on the test dataset.

Random Forest				
Well-6	MAE	R^2	$RMSE$	MSE
Hole size: 311	0.725	0.983	1.377	1.896

In Figures 5.7a, 5.7b,5.7c and 5.7d, the expected and predicted values, derived from varying numbers of lag observations for 30 future observations using the RF model, are depicted. Among these figures, Figure 5.7b featuring 40 Lag observations

shows the best performance across MAE, R^2, and $RMSE$, making it the top choice. Figure 5.7a featuring 30 Lag observations performs well overall and has the lowest MSE, making it the second-best option. 30 Lag observations is our choice due to being comparable with XGBoost on exact dimensions with good performance metrics and the lowest MSE.

Table 5.9: Summary of results of sonic log prediction using RF model for Well-1 on the test dataset.

RF					
Well-1	lag observations	MAE	R^2	$RMSE$	MSE
Hole size: 311	30	4.3001	0.7474	6.0386	31.9658
Hole size: 311	40	4.2861	0.7514	5.9902	35.8828
Hole size: 311	60	4.5144	0.7362	6.1706	38.769
Hole size: 311	80	4.4742	0.7460	6.0547	36.6604

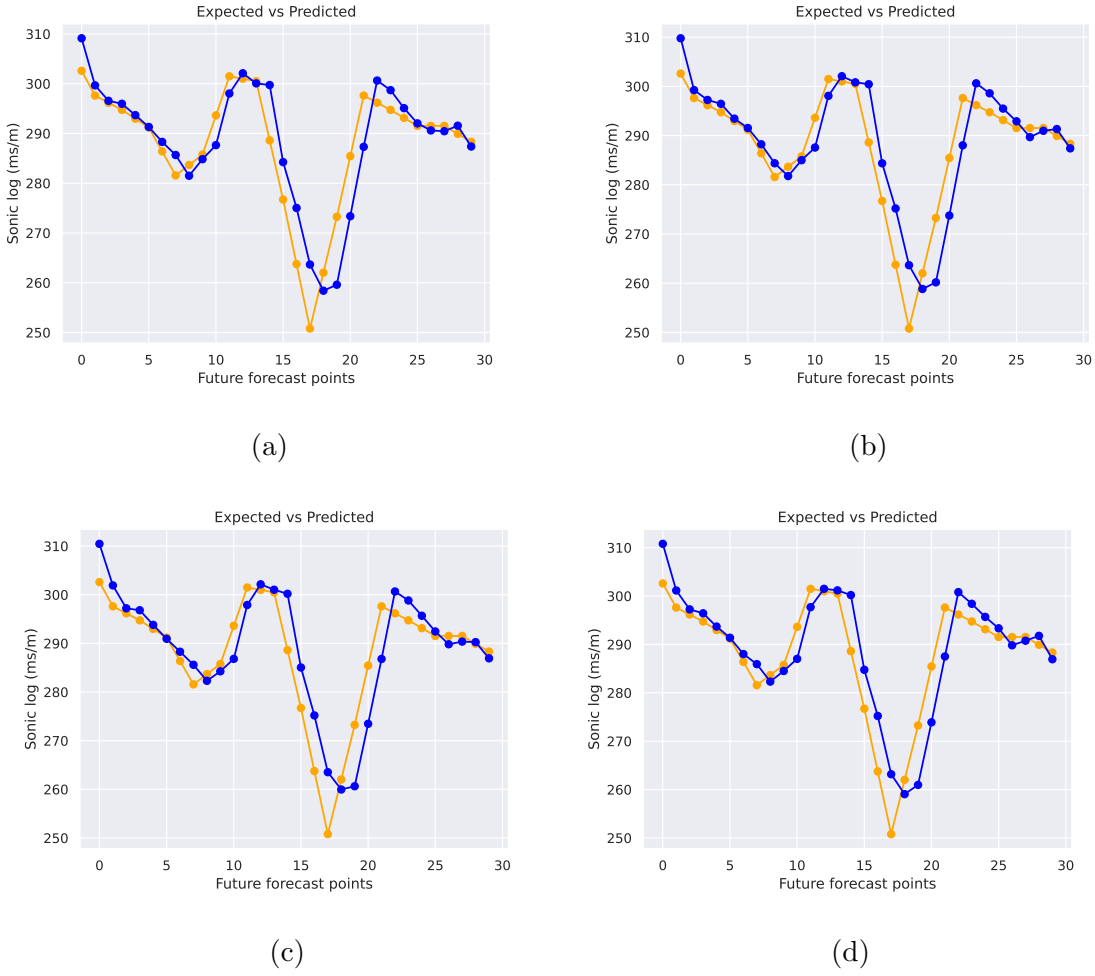


Figure 5.7: Comparison of expected and predicted outcomes using varying numbers of lag observations for the same future observations, generated by the RF model for hole size 311 of Well-1. (a) 30 lag observation and future forecast for 30 values, (b) 40 lag observation and future forecast for 30 values, (c) 60 lag observation and future forecast for 30 values, (d) 80 lag observation and future forecast for 30 values. Based on metric measurements in Figure 5.9, 30 lag observation has the best result

In summary, the comparison between anticipated and forecasted outcomes in Figures 5.6 and 5.7 highlights the influence of different lag observations on the subsequent 30 observations, specifically concentrating on hole size 311 of Well-1. While the RF model demonstrates fewer fluctuations compared to the XGBoost model, both models exhibit a delay in accurately predicting actual values.

5.10 LSTM model

The LSTM model employed in this code is a specialized type of RNN designed to effectively capture long-term dependencies within sequential data. In the specific context of this code, the LSTM model was developed and trained, corresponding to different hole sizes 311, 445, and 216. In the following sections, we delve into a comprehensive overview of the architecture, training process, and evaluation aspects.

5.10.1 Model architecture, compilation and evaluation

The model's architecture is established using the sequential API from *TensorFlow* and *Keras*, popular libraries for building and training neural networks. The initial layer is a bidirectional LSTM layer with 200 units, strategically designed to capture temporal dependencies in both forward and backward directions. This bidirectional nature enhances the model's ability to discern patterns in sequential data. Subsequent dense layers with 20 units and hyperbolic tangent (tanh) activation functions introduce non-linearity to the model. To prevent overfitting, a dropout layer with a dropout rate of 0.25 is incorporated. The final layer is a dense layer with a single unit, serving as the output layer for regression.

The model is compiled using the Adam optimizer, a popular optimization algorithm for training neural networks, and MSE is employed as the loss function. MSE is particularly suitable for regression tasks, aligning to predict continuous sonic log values.

Training involves feeding the model with historical data, where input sequences for each of the hole sizes are associated with target values for that hole size. The training process spans 50 epochs with a batch size of 72, and the model's performance is monitored on a validation set during training.

Figure 5.8 presents the training and validation loss over 50 epochs and Algorithm 22 illustrates the pseudocode of the functioning of the LSTM model.

Algorithm 22 Create and compile LSTM Model

1: *lstm_model* ← **Sequential** [

Bidirectional(**LSTM**(200),

Dense(20, *activation=tanh*),

Bidirectional(**LSTM**(150)),

Dense(20, *activation=tanh*),

Dense(20, *activation=tanh*),

Dropout(0.25),

Dense(1)])

2: *lstm_model*.compile(*optimizer=adam, loss=mse*)

3: *lstm_model*.summary()

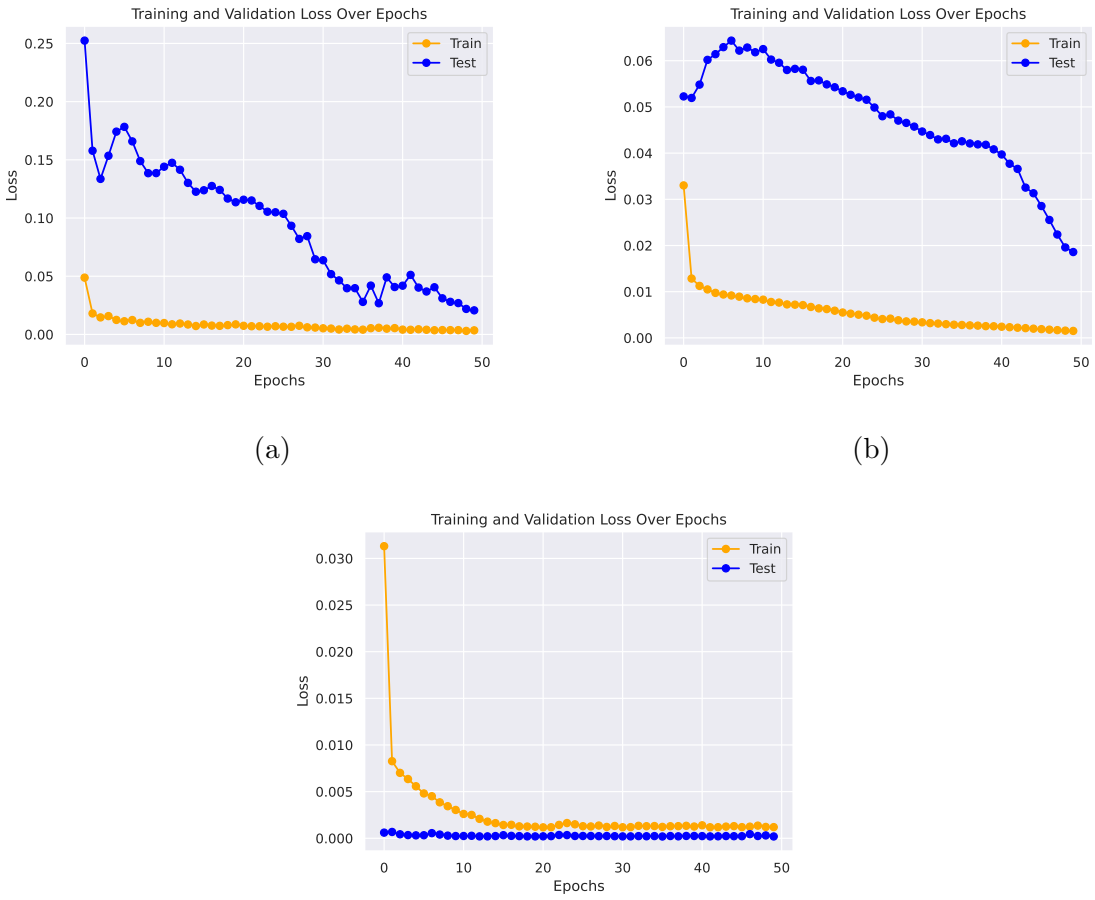


Figure 5.8: Record of training and validation loss values collected during the training process for over 50 epochs by LSTM model for different hole sizes of Well-1. (a) Training history for hole size 445, (b) Training history for hole size 311, (c) Training history for hole size 216. The loss values for both training and validation datasets showed a similar decreasing trend as observed for all hole sizes, demonstrating the model's robustness and ability to adapt to different hole sizes.

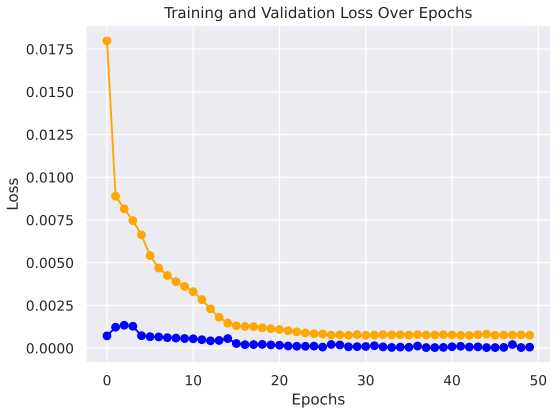


Figure 5.9: Record of training and validation loss values collected during the training process for over 50 epochs by LSTM model for hole size 311 of Well-6.The loss values for both training and validation datasets showed a similar decreasing trend, demonstrating the model's robustness.

5.10.2 Comparison of prediction and actual values

To compare the forecasted and actual values and to visualize the LSTM model's behavior, the following steps were executed. The test dataset, representing the last 10% of the sequence for each hole size, underwent reshaping to conform to the necessary format for prediction. Subsequently, both the predicted and actual values underwent an inverse scaling operation. For the predicted values, the feature columns from the reshaped test dataset were combined with the predictions. The scaling transformation, previously applied during preprocessing using the Min-Max scaler, was then reversed to extract the forecasted values. Similarly, for the actual values, the target data was reshaped and merged with the feature columns of the reshaped test input

data. Inverse scaling was then applied to isolate the actual values. This process allows for a direct comparison between the forecasted and actual values, thereby facilitating the evaluation of the LSTM model's performance. The sensitivity of MAE is typically not sensitive to scaling because it measures the average magnitude of errors without considering their direction. R^2 is sensitive to scaling, especially when the scaling affects the variance of the target variable. RMSE is sensitive to scaling because it calculates the square root of the mean squared errors. Like RMSE, MSE is sensitive to scaling because it measures the average of squared errors. In summary, R^2, RMSE, and MSE are more sensitive to scaled data compared to MAE. In our LSTM model, performance ranked up by using scalers in Section 4.6.

The model's effectiveness is assessed using metrics such as MSE, RMSE, MAE, and R^2 on both the training and validation sets. Tables 5.10 and 5.11 show the result of accuracy measurements for Well-1 and Well-2. In the following, Figures 5.10, 5.11 and 5.12 illustrate the actual and predicted values for different hole sizes of Well-1 and Figure 5.13 shows the actual and predicted values of hole size 311 for Well-6.

Table 5.10: Summary of results in sonic log prediction using LSTM model for Well-1 on the test datasets.The LSTM model shows varying performance across different test datasets for Well-1. The model achieves the highest accuracy and best fit on the hole size 216, as evidenced by the lowest MAE, $RMSE$, and MSE values and the highest R^2 value.

LSTM				
Well-1	MAE	R^2	$RMSE$	MSE
Hole size: 445	25.5841	0.5527	31.657	1002.1938
Hole size: 311	24.7066	0.1701	30.001	900.0620
Hole size: 216	2.0314	0.9691	9.3511	3.058

Table 5.11: Summary of results in sonic log prediction using LSTM model for Well-6 on the test dataset.

LSTM				
Well-6	MAE	R^2	$RMSE$	MSE
Hole size: 311	0.2456	0.9225	0.294	0.0862

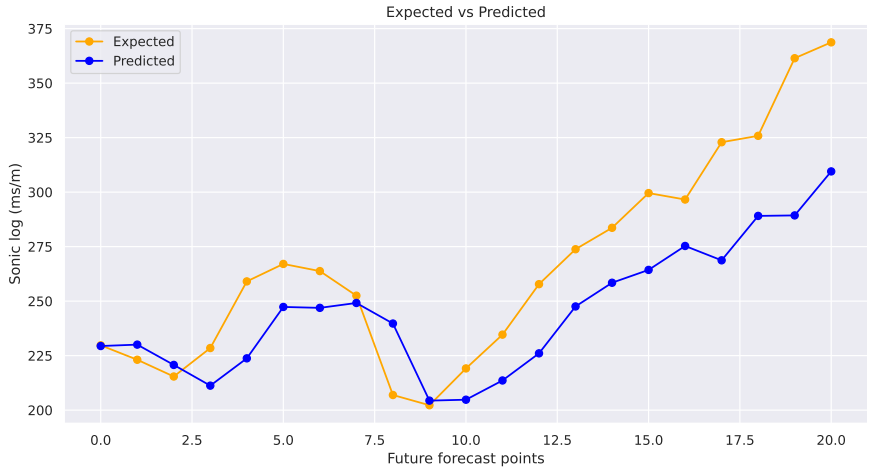


Figure 5.10: Comparison of LSTM predictions and actual values for hole size 445 of Well-1.

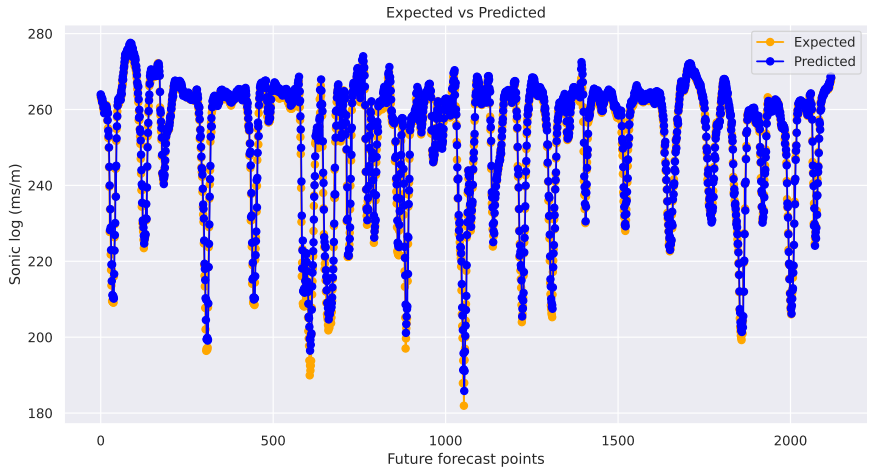


Figure 5.11: Comparison of LSTM predictions and actual values for hole size 216 of Well-1. The LSTM model's predictions closely follow the trend of the actual values, indicating that the model has learned the underlying patterns in the data effectively.

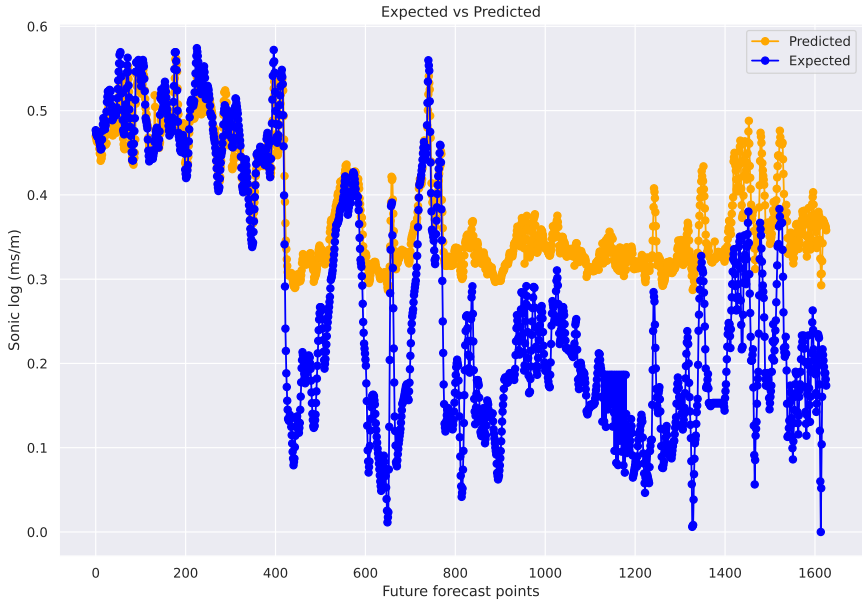


Figure 5.12: Comparison of LSTM predictions and actual values for hole size 311 of Well-1, fFollowing the 400th index, there's a notable jump causing the predicted values to diverge from the expected values significantly.

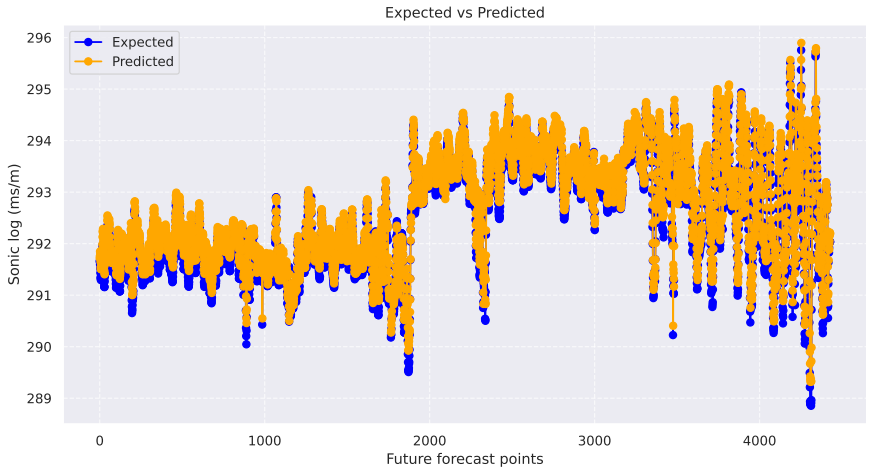


Figure 5.13: Comparison of LSTM predictions and actual values for hole size 311 of Well-6.

As illustrated in Figure 5.12 for Well-1, at around index 400, a significant sudden change is evident in the data for hole size 316. Upon dividing the data into training (90%) and testing (10%) segments, it became apparent that the LSTM model encountered challenges in accurately predicting subsequent data points for hole size 311 of Well-1 following significant noise; leading to deviations in the model's predictions. Post index 400, a marked disparity between predicted and actual values became evident. To address this issue, we segmented the 1620 test data indices into two subplots, comprising the first 400 indices and the subsequent 400 to 1625 indices, as illustrated in Figures 5.14 and 5.15. This subdivision was implemented to enhance clarity. To mitigate this issue, we reconfigured the LSTM model by allocating 95% of the data for training and the remaining 5% for testing. This adjustment allowed the model to comprehensively learn from the training data, resulting in closely aligned actual and predicted values, as evidenced in Figure 5.16. Furthermore, under this configuration, the coefficient of determination R^2 exhibited an improvement. Figure 5.17 illustrates the actual values for the last 10% of data, along with its corresponding prediction. Additionally, it presents the prediction of the last 5% of data after adding noisy data into the training set and the reduction of the test size by 5%. Table 5.12 shows the accuracy measurements for Well-1, considering 95% as a train and 5% as a test by LSTM model.

Table 5.12: Summary of results in sonic log prediction using LSTM model for Well-6 on the test dataset, considering 95% as train and 5% as test test.

LSTM				
Well-6	MAE	R^2	$RMSE$	MSE
Hole size: 311	10.771	0.371	12.115	146.774

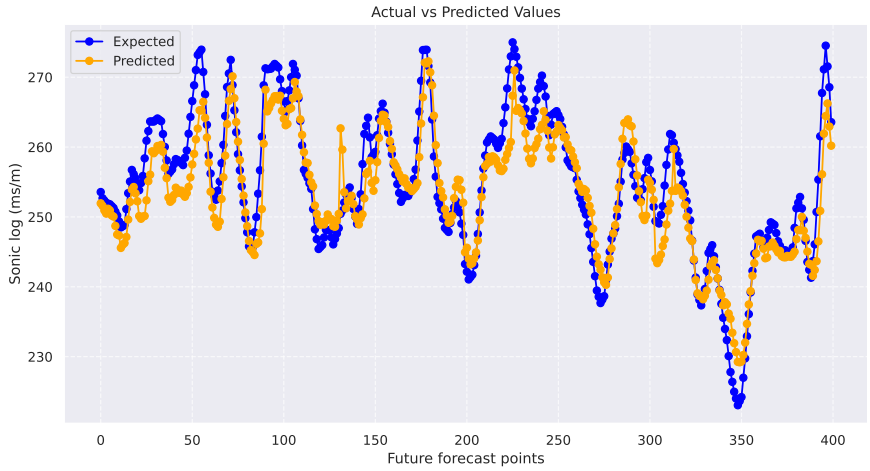


Figure 5.14: Comparing LSTM predictions with actual values on test data, analyzing index values 0 to 400 for hole Size 311 in Well-1.

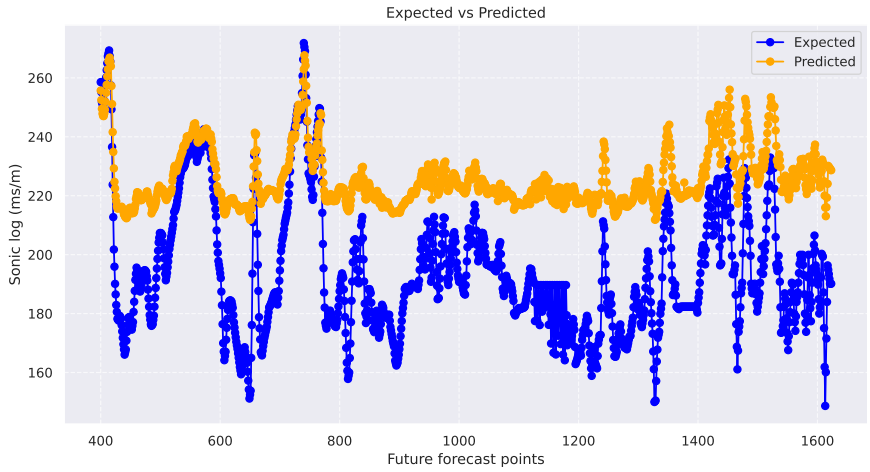


Figure 5.15: Comparing LSTM predictions with actual values on test data, analyzing index values 400 to 1625 for hole Size 311 in Well-1.

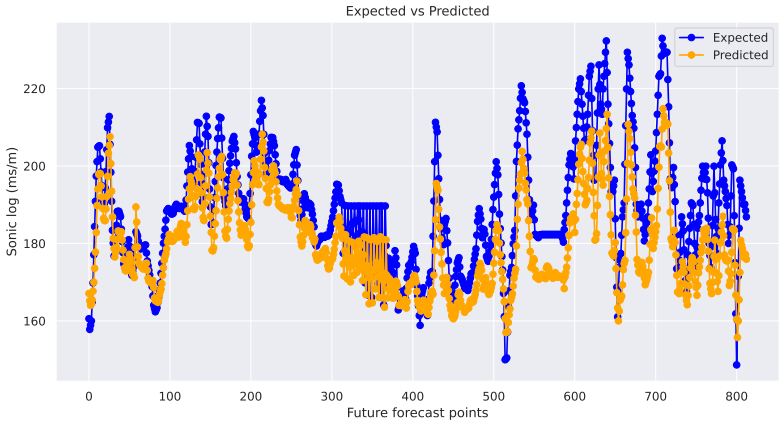


Figure 5.16: Comparison of predictions and actual values for the hole size 311 in Well-1 for the last 5% of sequence as a test dataset. Compared to Figure 5.12, the predicted values are closer to expected values.

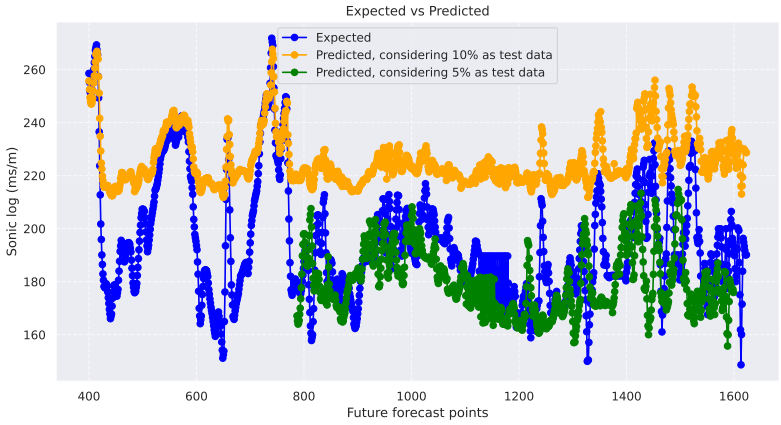


Figure 5.17: Comparison of actual values for the last 10% of the data as test data, together with their respective predictions. Furthermore, the predictions for the last 5% of the data after incorporating noisy data into the training set and reducing the test size by 5%.

5.10.3 Generalization

In this section, the LSTM model is tailored for time series prediction, showcasing a sophisticated bidirectional architecture with multiple layers to capture intricate patterns and dependencies in the data associated with a hole size of 311 to predict sonic log values. Ensuring that the model can generalize to new, unseen data is a crucial consideration. The goal is to establish a model that not only performs well on the training set but also demonstrates robustness in making accurate predictions on data it has not encountered during training.

5.10.3.1 Blind prediction

In our method of blind prediction, we used a specific portion of Well-1 as the training data, while another designated portion of Well-6 was used as the test set for sonic log prediction. Before employing the models (XGBoost, RF, and LSTM), both datasets underwent preprocessing steps to handle NaN values and outliers, ensuring alignment with the algorithmic procedures outlined in this chapter. For this particular scenario, we chose data with TVD ranging from 1460 to 1520 meters for hole size 311 of Well-1, covering a depth of 60 meters, as the training dataset. Subsequently, the models were applied to predict the sonic log for the 40-meter depth segment in Well-6, specifically with TVD ranging from 1470 to 1510. Table 5.13 presents the results obtained from the XGBoost, RF, and LSTM models. In the case of the LSTM model, a bidirectional architecture is utilized. The training and test loss history for

over 50 epochs is depicted in Figure 5.18, while the actual and predicted values using the LSTM model are illustrated in Figure 5.19.

Similarly, we considered data with TVD ranging from 1460 to 1520 meters of hole size 311, with a depth of 60 meters from Well-6, as the training set and the models were then employed to forecast the sonic log for the 40-meter depth segment in Well-1, specifically TVD between 1470 and 1510. In this scenario, the training and testing loss history for the LSTM model over 30 epochs is depicted in Figure 5.20, and the actual and predicted values using the LSTM model are shown in Figure 5.21.

Table 5.14 presents the results obtained from the XGBoost, RF, and LSTM models for this partition of blind prediction.

Table 5.13: Summary of results of blind prediction, using 60 meters of Well-1 (TVD ranging from 1460 to 1520) for training and predicting 40 meters of Well-6 as a test set (TVD ranging from 1470 to 1510).

Blind prediction results				
Models	MAE	R^2	$RMSE$	MSE
XGBoost	0.196	0.994	0.254	0.649
RF	0.078	0.996	0.221	0.048
LSTM	3.577	0.633	29.287	5.412

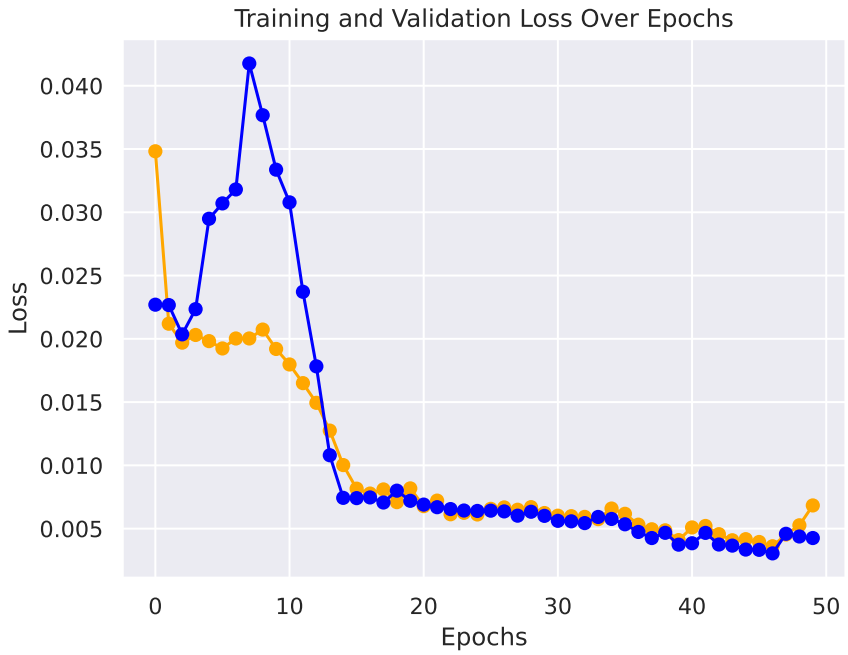


Figure 5.18: Loss history for hole size 311 by LSTM as a blind prediction over 50 epochs, 60 meters of Well-1 is used for training, 40 meters of Well-6 is considered as a test set.

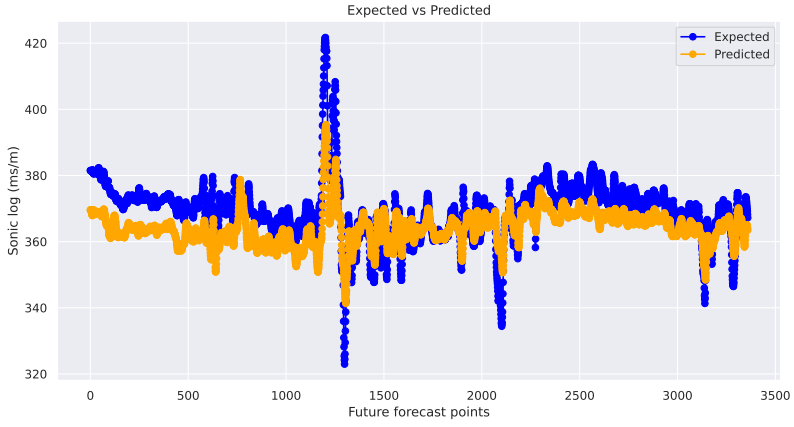


Figure 5.19: Expected and predicted values using 60 meters of Well-1 (TVD from 1460 to 1520) for training and predicting 40 meters of Well-6 (TVD from 1470 to 1510) in hole size 311 as a test set.

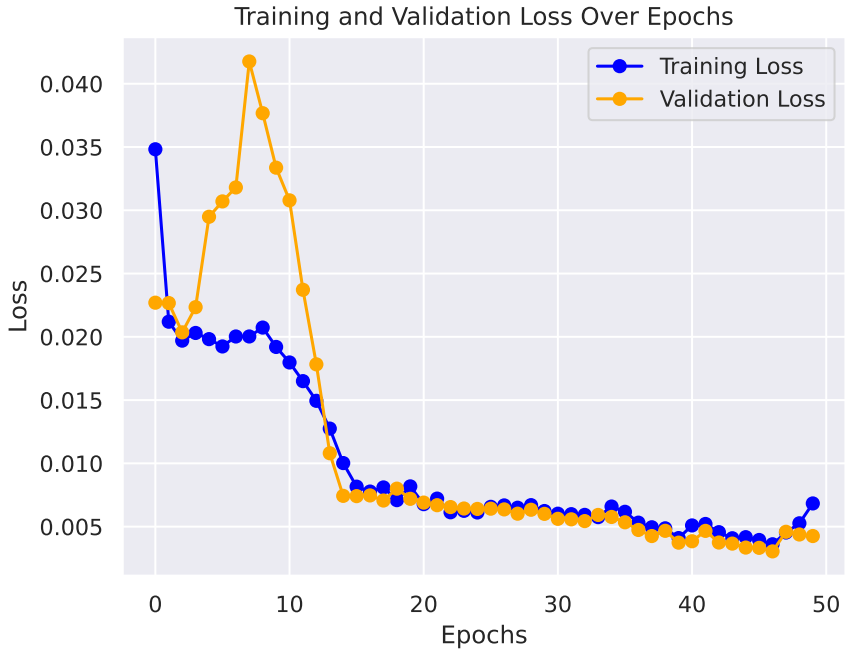


Figure 5.20: Loss history for hole size 311 by LSTM as a blind prediction over 30 epochs, 60 meters of Well-6 is used for training, 40 meters of Well-1 is considered as a test set.

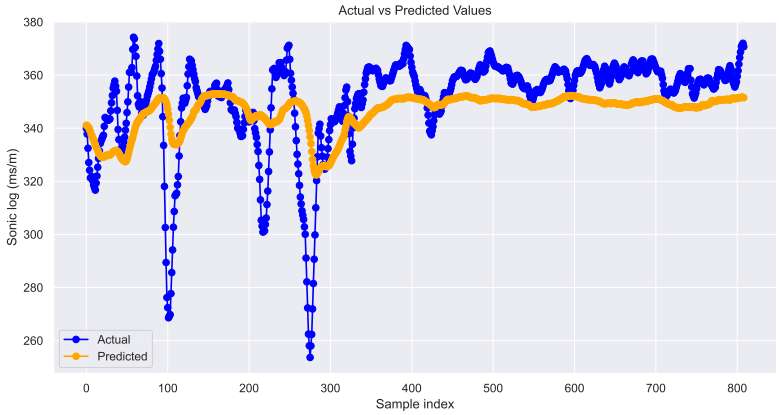


Figure 5.21: Expected and predicted values using 60 meters of Well-6 (TVD from 1460 to 1520) for training and predicting 40 meters of Well-1 (TVD from 1470 to 1510) in hole size 311 as a test set.

Table 5.14: Summary of results of blind prediction, using 60 meters of Well-6 (TVD ranging from 1460 to 1520) for training and predicting 40 meters of Well-1 as a test set (TVD ranging from 1470 to 1510).

Blind prediction results				
Models	MAE	R^2	$RMSE$	MSE
XGBoost	0.838	0.967	1.311	1.719
RF	0.976	0.951	1.452	2.10
LSTM	4.2505	0.4827	6.443	41.5112

Considering Table 5.13, while utilizing a 60-meter segment of Well-1 as training data to predict a 40-meter segment of Well-6, it is evident that both XGBoost and RF models outperform LSTM in terms of MAE, R^2, RMSE, and MSE. Conversely, for Well-6 considering a segment of it as a train, based on Table 5.14, it seems while XGBoost and RF models still perform better than LSTM, the performance gap between XGBoost and LSTM is wider compared to considering Well-1 as a training set, indicating that the LSTM model struggles more with data from Well-6 as a training set.

Chapter 6

Conclusion

In this study, we applied three different machine learning algorithms (XGBoost, RF, and LSTM) to predict the sonic log. We also used Well-1 and Well-6 interchangeably to predict part of each in a blind prediction scenario. Each algorithm brought its unique strengths and considerations to the table. The study highlighted the importance of ML techniques in enhancing reservoir characterization and exploration efforts, potentially reducing costs and environmental impact associated with traditional methods.

XGBoost, a gradient boosting algorithm, demonstrated robust performance with respectable MAE and R^2 across different hole sizes. It showcased its ability to capture complex relationships within the data and adapt well to the blind prediction setting.

Random Forest, an ensemble learning method, provided competitive results, showcasing its effectiveness in handling the provided features. The model demonstrated

good predictive power, making it a valuable contender for sonic log predictions in real-world scenarios while acknowledging that the XGBoost model exhibited even better performance.

Although the LSTM model demonstrates strong performance for predicting sonic log in Well-6, it encounters challenges in accurately forecasting the target value for Well-1, specifically for hole sizes 311 and 445. The limited availability of data for hole size 445 was a contributing factor, but in the case of hole size 311, it had poor performance. Even after expanding the training dataset, although there were improvements in R^2 and RMSE, the model still lags behind XGBoost and RF models.

To compare the performance of XGBoost, RF and LSTM models based on the provided metrics MAE, R^2, RMSE and MSE for different hole sizes of Well-1, Table 6.1 presents a cohesive view of the data.

Table 6.1: Performance comparison of XGBoost, RF and LSTM for Well-1 in all hole sizes, considering last 10% of data as test set.

Algorithm	**Metric**	**Hole Size: 445**	**Hole Size: 311**	**Hole Size: 216**
RF	MAE	7.487	2.063	1.420
	R^2	0.543	0.991	0.981
	$RMSE$	8.579	3.046	2.389
	MSE	73.607	9.278	5.708
XGBoost	MAE	4.8554	2.3419	1.4439
	R^2	0.7950	0.9889	0.9811
	$RMSE$	5.7426	3.4052	2.3901
	MSE	32.9779	11.5958	5.7126
LSTM	MAE	25.5841	24.7066	2.0314
	R^2	0.5527	0.1701	0.9591
	$RMSE$	31.657	30.001	9.3511
	MSE	1002.1938	900.0620	3.058

According to the table, when comparing XGBoost, RF, and LSTM for Well-1 across all hole sizes, particularly considering the last 10% of data as the test set, several observations can be made.

XGBoost consistently demonstrates the lowest MAE across all hole sizes, followed by RF. LSTM exhibits the highest MAE values among the three algorithms,

indicating that XGBoost tends to produce predictions that are closer to the actual values on average.

XGBoost and RF generally perform well in terms of R^2, with XGBoost slightly outperforming RF for smaller hole sizes but showing slightly worse performance for the largest hole size (445). LSTM lags behind in R^2 values compared to XGBoost and RF.

Similar to MAE, XGBoost consistently shows lower $RMSE$ values compared to RF and LSTM across all hole sizes. This suggests that XGBoost's predictions are closer to the actual values on average, with LSTM having the highest RMSE values.

XGBoost also exhibits lower MSE values compared to RF and LSTM for all hole sizes, further emphasizing its better prediction accuracy.

In summary, XGBoost generally outperforms RF and LSTM in terms of prediction accuracy for all hole sizes in Well-1, as evidenced by lower MAE, $RMSE$, and MSE values. However, RF shows slightly better R^2 values for the largest hole size. Also, Table 6.2 presents a comprehensive view of the performance metrics for hole size 311 of Well-6.

Table 6.2: Performance comparison of XGBoost, RF and LSTM for Well-6 in hole size 311, considering last 10% of data as a test set.

Algorithm	**Metric**			
	MAE	R^2	$RMSE$	MSE
Random Forest	0.725	0.983	1.377	1.896
XGBoost	0.4973	0.9764	1.0851	1.1775
LSTM	0.2456	0.9225	0.294	0.0862

Considering Table 6.2, LSTM achieves the lowest MAE, indicating better accuracy in predicting the sonic log values compared to XGBoost and RF. In comparing the performance metrics for Well-6, we observe variations across different models. Specifically, RF emerges with the highest R^2 value, indicating a better fit to the data, followed by XGBoost and LSTM. Conversely, for RMSE and MSE, LSTM demonstrates superior performance, with lower values compared to XGBoost and RF. These lower RMSE and MSE values signify better accuracy of the LSTM model in predicting sonic log values for Well-6. Thus, while RF excels in terms of R^2, LSTM outperforms in terms of MAE, RMSE and MSE, underscoring the importance of considering various metrics when evaluating model performance. The study's outcome proves that the LSTM, RF and XGBoost algorithms can successfully predict the compressional sonic log. Despite the superior performance of the XGBoost, the study cannot confirm that the XGBoost algorithm will always outperform.

6.1 Study Limitations

There are several limitations inherent in this study. Firstly, the sonic log data, our target for prediction, was only available for 2 out of 13 wells. Even for those wells available, data was not available for all hole sizes. The second limitation arises from treating the XGBoost and RF algorithms as supervised learning models and adapting them into time series algorithms. This adaptation necessitates numerous iterations to integrate each time step with the preceding ones and requires the utilization of powerful servers equipped with high GPU and CPU capabilities to facilitate more in-depth research. The third limitation was time constraints; exploring a broad spectrum of hyperparameters and conducting an exhaustive search for numerous iterations was unfeasible. Additionally, determining the suitability of an LSTM algorithm for sequential data relies on identifying the optimal sequence length and frequency, which can be challenging, particularly when dealing with non-uniform frequencies. Consequently, delving deeper into certain concepts proved to be excessively resource-intensive.

6.2 Future Work

Subsequent research endeavors in this domain could explore the integration of signal processing techniques into algorithms with guidance from drilling engineers to oversee and detect sequence noise arising from alterations in drilling projects. The

implementation of signal processing will result in cleaner data, enabling algorithms to identify more optimal frequencies for predicting well performance with unseen data. This involves confirming that such noise constitutes outliers occurring during the project, thereby enhancing data preprocessing. Moreover, fine-tuning relevant hyperparameters for RF and XGBoost algorithms can notably enhance their performance and predictive accuracy. Refining hyperparameters like the number of trees (or estimators), maximum tree depth, minimum samples per leaf, and feature selection criteria can contribute to better generalization or performance on unseen data, while also improving computational efficiency. However, these processes involve significant computational expenses, necessitating the use of powerful servers equipped with high GPU and CPU capabilities to facilitate more comprehensive research. Incorporating larger datasets comprising sequences for all hole sizes and across more than two wells would augment the methodology's generalizability.

Bibliography

[1] Jonathan Evenick. Introduction to well logs & subsurface maps. 2008.

[2] George B Asquith, Daniel Krygowski, and Charles R Gibson. *Basic well log analysis*, volume 16. American Association of Petroleum Geologists Tulsa, 2004.

[3] Jun Li, Wei Zhang, and Xiao Liu. Bulk Density Response and experimental Study of Pulsed Neutron-Gamma Density Logging. *Frontiers*, 2021.

[4] SPE PetroWiki. Density logging. *PetroWiki*, 2013.

[5] Robert L. Hawley and Evan M. Morris. Borehole optical stratigraphy and neutron-scattering density measurements at summit, greenland. *Journal of Glaciology*, 2008.

[6] Rayan Kanfar, Obai Shaikh, Mehrdad Yousefzadeh, and Tapan Mukerji. Real-time well log prediction from drilling data using deep learning. In *International Petroleum Technology Conference*. OnePetro, 2020.

[7] Mojtaba Rajabi, Bahman Bohloli, and Esmaeil Gholampour Ahangar. Intelli-

gent approaches for prediction of compressional, shear and stoneley wave velocities from conventional well log data: A case study from the Sarvak carbonate reservoir in the Abadan Plain (Southwestern iran). *Computers & Geosciences*, 36(5):647–664, 2010.

[8] Shahoo Maleki, Ali Moradzadeh, Reza Ghavami Riabi, Raoof Gholami, and Farhad Sadeghzadeh. Prediction of shear wave velocity using empirical correlations and artificial intelligence methods. *NRIAG Journal of Astronomy and Geophysics*, 3(1):70–81, 2014.

[9] Zeeshan Tariq, Salaheldin Elkatatny, Mohamed Mahmoud, and Abdulazeez Abdulraheem. A new artificial intelligence based empirical correlation to predict sonic travel time. In *International Petroleum Technology Conference*. OnePetro, 2016.

[10] Bo Liu, Auref Rostamian, Mahdi Kheirollahi, Seyyedeh Forough Mirseyed, Erfan Mohammadian, Naser Golsanami, Kouqi Liu, and Mehdi Ostadhassan. NMR log response prediction from conventional petrophysical logs with XGBoost-PSO framework. *Geoenergy Science and Engineering*, 224:211561, 2023.

[11] Hany Gamal, Ahmed Alsaihati, and Salaheldin Elkatatny. Predicting the rock sonic logs while drilling by random forest and decision tree-based algorithms. *Journal of Energy Resources Technology*, 144(4), 2022.

[12] Jongkook Kim. Synthetic shear sonic log generation utilizing hybrid machine learning techniques. *Artificial Intelligence in Geosciences*, 3:53–70, 2022.

[13] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[14] Nimrabanu Memon, Samir B Patel, and Dhruvesh P Patel. Comparative analysis of artificial neural network and XGBoost algorithm for PolSAR image classification. In *International Conference on Pattern Recognition and Machine Intelligence*, pages 452–460. Springer, 2019.

[15] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[16] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[17] Ruizhi Zhong, Cyrus Salehi, and Ray Johnson Jr. Machine learning for drilling applications: A review. *Journal of Natural Gas Science and Engineering*, page 104807, 2022.

[18] Chiranth Hegde and KE Gray. Use of Machine Learning and Data Analytics to increase drilling efficiency for nearby wells. *Journal of Natural Gas Science and Engineering*, 40:327–335, 2017.

[19] Chiranth Hegde and Ken Gray. Evaluation of coupled machine learning models for drilling optimization. *Journal of Natural Gas Science and Engineering*, 56:397–407, 2018.

[20] Luís Felipe FM Barbosa, Andreas Nascimento, Mauro Hugo Mathias, and Joao Andrade de Carvalho Jr. Machine learning methods applied to drilling rate of penetration prediction and optimization-A review. *Journal of Petroleum Science and Engineering*, 183:106332, 2019.

[21] Mohammad Sabah, Mohsen Talebkeikhah, David A Wood, Rasool Khosravanian, Mohammad Anemangely, and Alireza Younesi. A machine learning approach to predict drilling rate using petrophysical and mud logging data. *Earth Science Informatics*, 12:319–339, 2019.

[22] Jiachun You, Junxing Cao, Xingjian Wang, and Wei Liu. Shear wave velocity prediction based on LSTM and its application for morphology identification and saturation inversion of gas hydrate. *Journal of Petroleum Science and Engineering*, 205:109027, 2021.

[23] Muhammad Ali, Ren Jiang, Huolin Ma, Heping Pan, Khizar Abbas, Umar Ashraf, and Jar Ullah. Machine learning-A novel approach of well logs simi-

larity based on synchronization measures to predict shear sonic logs. *Journal of Petroleum Science and Engineering*, 203:108602, 2021.

[24] Ammar M Alali, Mahmoud F Abughaban, Beshir M Aman, and Sai Ravela. Hybrid data driven drilling and rate of penetration optimization. *Journal of Petroleum Science and Engineering*, 200:108075, 2021.

[25] Mohsen Riazi, Hossein Mehrjoo, Reza Nakhaei, Hossein Jalalifar, Mohammadhadi Shateri, Masoud Riazi, Mehdi Ostadhassan, and Abdolhossein Hemmati-Sarapardeh. Modelling rate of penetration in drilling operations using RBF, MLP, LSSVM, and DT models. *Scientific Reports*, 12(1):11650, 2022.

[26] Callistus Nero, Akwasi Acheampong Aning, Sylvester Kojo Danuor, and Victor Mensah. Prediction of compressional sonic log in the western (Tano) sedimentary basin of ghana, west africa using supervised machine learning algorithms. *Heliyon*, 2023.

[27] I.N. Bankman. *Handbook of Medical Imaging: Processing and Analysis*. Academic Press series in biomedical engineering. Academic Press, 2000.

[28] B.P. Lathi and R.A. Green. *Signal Processing and Linear Systems*. The Oxford series in electrical and computer engineering. Oxford University Press, 2021.

[29] Darwin V Ellis and Julian M Singer. *Well Logging for Earth Scientists*. Springer Science & Business Media, 2007.

[30] Li Zhang, Dong Han, and Sen Hu. The Relationship between sonic velocity and Rock Properties in a Sandstone Formation. *Journal of Petroleum Science and Engineering*, 65(1-2):64–72, 2009.

[31] Schlumberger. *Log Interpretation Principles/Applications*. Schlumberger Educational Services, 1998.

[32] C. Fairhurst. *Rock Mechanics and Engineering*. CRC Press, 1996.