FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/Specialisation: Computational Engineering	Spring semester, 2022 Open
Author: Sofia Tariq	
Supervisor: Prof. Dan Sui	
Title of master's thesis: Sensitivity Analysis for Drilling data using Machine Learning models	
Credits (ECTS): 30	
Keywords: Sensitivity Analysis Feature Selection Sobol Method Morris Method Monte Carlo Simulation Regressors Drilling data	Number of pages: 122 + supplement material/other : 0 Stavanger, 15th June 2022

Acknowledgments

I would especially like to thank my supervisor, Prof. Dan Sui, for guiding me throughout this work. She was always there to help and guide me when I found myself lost. This thesis work would have never been possible without her.

I am grateful to my parents and my siblings for their constant motivation. I would also like to thank my husband who has been a support system of mine during the tough times.

Abstract

Over the last decade, machine learning models have become highly popular. Without requiring significant human participation, Machine Learning models improve the efficiency and dependability of any system. Machine Learning models like Random Forest and Gradient Booster can attain 99.9% accuracy. Despite all of the advantages of Machine Learning models, it remains a black box. Sensitivity Analysis is one method for deciphering the behavior of Machine Learning models. Sensitivity Analysis is used to study the influence of input parameters on the targeted outputs.

Sensitivity Analysis methods were used to find the most influential input parameters in this thesis. A drilling dataset of the Volve field located in the North sea was used as a test case. The rate of penetration was selected as the output parameter and Sensitivity Analysis methods were implemented to analyze which input parameters have the most effect on the output.

Feature Selection is a pre-processing approach used in Machine Learning models, but it was also used to predict crucial input parameters and compare with Sensitivity Analysis methods.

Four Machine Learning, Random Forest, Gradient Booster, K-Nearest Neighbours, and Local Cascade Ensemble models were implemented with high accuracy and low Root Mean Square error values. These Machine Learning models were combined with methods for feature selection and sensitivity analysis. Recursive Feature Elimination, Pearson's correlation, Mutual Information, Feature Importance utilizing Random Forest, and Permutation were among the five feature selection methods used. Two statistical methods, the Sobol and Morris method and Monte Carlo simulation, were used for sensitivity analysis. In total, seventeen methods were implemented when combined with four Machine Learning models. By selecting the most

essential input parameters and deleting those that were unnecessary, it was possible to reduce the size of the dataset by half using these strategies. The outcomes of all three sensitivity analysis approaches were the same, however, the results of each feature selection method were different. For pre-processing the data, feature selection methods can be coupled with sensitivity analysis, however, they are not efficient enough for sensitivity analysis. The Morris approach was the most efficient and computationally inexpensive method. It can produce accurate results even with small sample size.

Table of Contents

List of Figures

List of Tables

1. Introduction

In today's world, data-driven models are increasingly important in informing and supporting decision-making. Machine Learning (ML) is becoming a popular technology in all fields of modeling. Machine Learning models are becoming efficient and accurate with new research work. While the Machine Learning models are improving, the physical mechanism is unclear. It is unclear how each Machine Learning model uses the dataset for prediction, which parameters are influential, and which are entirely ignored by the model. Because of this, the models created upon Machine Learning models are referred to as "black boxes" (Zhang, 2019). It becomes a matter of curiosity to find a way to learn about the nature of these ML models.

Accuracy and reliability are used to identify the usefulness of a model. No model in this world is perfect. Hence the study related to uncertainty becomes of great interest. Input parameters and the model's efficiency are highly dependent on one another. The motivation behind this study is to interpret the relationship between input parameters and ML models.

One way to identify the relation between the parameters of the dataset and model efficiency is through Sensitivity Analysis (SA). The sensitivity of a model describes the severity of change of the model's output related to the change of a given input value (Tunkiel et al., 2020).

This thesis focuses on the development of reliable models that use a data-driven model and integrate it with SA models to interpret results for the real-operational dataset used. The dataset used for ML analysis is taken from Equinor’s website. It is a drilling dataset from the Volve field located in the central part of the North Sea. In this drilling dataset Rate of penetration is set as a target output, and other parameters are tested against it.

For Sensitivity Analysis, the input parameters are considered independent of each other. Although, there is ongoing research in this field to find the correlation between the input parameters. The input parameters can be highly correlated (Tavakoli et al., 2013). In our study, input parameters are considered independent. This thesis work is divided into three main parts:

- Using the Feature Selection (FS) technique for raking important input parameters
- Integrating ML models with two statistical SA methods
- Using Monte Carlo Simulation (MCS) for sampling along with Gaussian Process Regressor

Feature Selection methods are generally used as pre-processing methods. However, this study aimed to compare the results from Feature Selection techniques with other Sensitivity Analysis methods to identify which Feature Selection technique can give a result closer to sensitivity methods. One wrapper method, recursive method, two filter-based methods, Pearson correlation and Mutual Information (MI), and two embedded methods, Random forest feature importance, and permutation methods, are used. The results from each method are then plugged into four different ML models; Random Forest, Gradient Booster, KNN, and Local Cascade Esenmble (LCE) regressor.

For the second part, the SALib library is used to generate samples from the variance-based Sensitivity Analysis method Sobol and screening method Morris method. The two Sensitivity Analysis methods are used to generate the input samples to be fed into the final models. In this step, two types of models are tested, mathematical and Machine Learning models. Three case studies for mathematical models were developed, and the same four ML models used for Feature Selection were used for integrating Machine Learning models with sensitivity methods.

In the third part, a toolbox Emukit is used to create an emulator using Gaussian Process (GP) with Monte Carlo Simulation (MCS). Gaussian Process Regressor is used for the Machine Learning model and is integrated with MCS to study the sensitivity phenomena.

State of the art includes the work of Fidalgo (2001), Liu et al. (2012), Kamalov (2018), Barraza et al. (2019), and Zhang (2019), focusing on Feature Selection for SA modeling. Sobol and Morris's methods are extensively used for Sensitivity Analysis in all fields. Tavakoli et al. (2013), Pelfrene et al. (2019), ZHENCHAO et al. (2019), and, Tunkiel et al. (2020) have used Sensitivity Analysis for the study of drilling dataset. Some of the few works that have used the SALib Library for the study of SA in drilling datasets include Plúa et al. (2021), Teshale et al. (2020), and Jaxa-Rozen and Kwakkel (2018). Emukit has not been used with the drilling dataset yet. Also, it is the first time that Local Cascade Ensemble, a Machine Learning regressor, has been used to model a drilling dataset.

1.1. Objectives

The objectives of this study are as follows:

- Implement Feature Selection methods and compare the results from all methods
- Use new features as input parameters for ML models and compare the performance
- Use ML models with Sobol and Morris methods using the SALib tool
- Compare results obtained for different ML models
- Implement Gaussian process and Monte Carlo Simulation using Emukit toolbox
- Compare the result with results obtained from Sobol and Morris methods

1.2 Structure

This section is introduced to give an overview of each chapter in this thesis.

- Chapter 2 gives an overview of Sensitivity Analysis. Difference between Local and global sensitivity. Three different methods, the Sobol method, Morris method, and Monte Carlo simulation are discussed.
- Chapter 3 introduces the dataset to be used. A well data from the Volve field in the North sea available on Equinor's website is used. The methods for data pre-processing are discussed. For the outlier removal process, the Interquartile method and DBSCAN are discussed.
- Chapter 4 discusses the theoretical background of the Machine Learning model and discusses the implementation of methods used. Random Forest, Gradient Booster, K-Nearest Neighbors, and a new method Local Cascade Ensemble are used.
- Chapter 5 introduces the methods of Feature Selection for Sensitivity Analysis. The difference between Wrapper, Filter, and Embedded methods is discussed. The most influential input parameters for ROP modeling are identified. The methods used are Recursive Feature Elimination, Pearson's correlation, Mutual Information, Feature Importance, and Permutation method.
- Chapter 6 includes the introduction to a library for Sensitivity Analysis. Three mathematical models and one data-driven model are used as case studies. In this chapter 4 models are proposed that can perform Sensitivity Analysis for data-driven models.
- Chapter 7 proposes a model which uses Monte Carlo Simulation to perform Sensitivity Analysis for the data-driven model. Another toolbox named Emukit is used and its specifications are also discussed.

2. Sensitivity Analysis

2.1 Theory and Concept

Sensitivity analysis is defined by Saltelli et al. (2004) as "The study of how uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input." In simple words, the study of the effect of each input parameter on output is known as sensitivity analysis. The terms' sensitivity' and 'uncertainty' go hand-in-hand. Therefore it is important to distinguish both first. The accuracy of any model depends upon sensitivity and uncertainty illustrated in Figure 2.1.

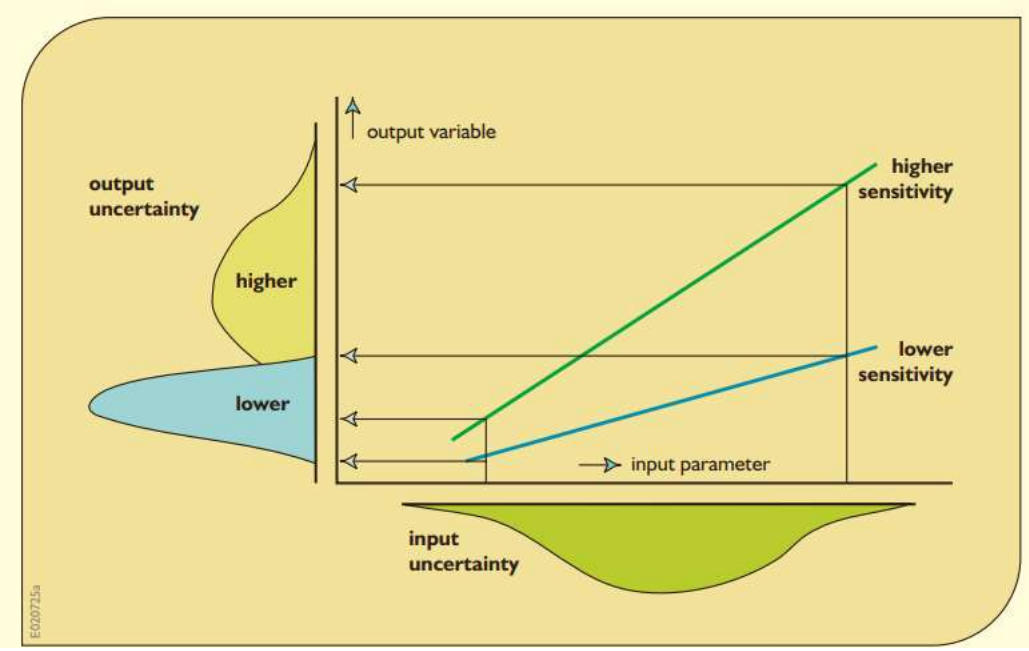


Figure 2.1: Schematic diagram showing the relationship between model input parameter uncertainty and sensitivity to model output variable uncertainty (adapted from Loucks et al., 2005)

Uncertainty analysis can construct probability distributions of model outputs and system performance indexes using probabilistic descriptions of model inputs (Geffray et al., 2019). In contrast, the influence of probable inaccuracies in input data on expected model outputs and

system performance is investigated and quantified using sensitivity analysis methodologies (Loucks et al., 2005). We focus on sensitivity analysis and methods to evaluate it in this work.

By definition, uncertainty analysis gives an overall probability of possible outcomes. However, sensitivity analysis shows how the model will behave when the input changes. Sensitivity analysis is used in scenarios where the aim is to find the most impactful input parameters that affect the model performance. Loucks et al. (2005) describe sensitivity analysis to obtain detailed results that guide research and help model development efforts. Sensitivity analysis helps study the effect of input parameters on output, but it also helps study their interaction with other input parameters (Kiparissides et al., 2009). The workflow of sensitivity analysis is shown in Figure 2.2 with the help of a simple flow chart for a better understanding.

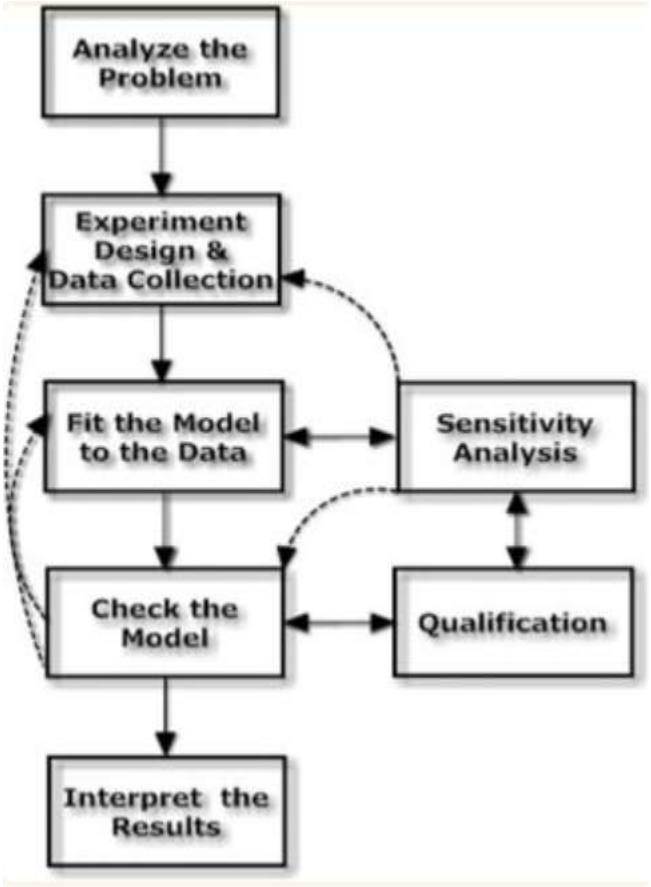


Figure 2.2: Flow chart to demonstrate the steps of data analysis (adapted from Zhang et al., 2015)

Before performing sensitivity analysis, we need to identify the properties of a model. Saltelli et al. (2004) describe models to be of the following types: (1) Diagnostic or prognostic and (2) Data-driven or law-driven. In our case, a data-driven model predicts the system's behavior. The methods applied for sensitivity analysis are based on this information.

2.2 Local vs. Global Sensitivity Analysis

The sensitivity analysis is divided into two categories: Local Sensitivity Analysis and Global Sensitivity Analysis.

2.2.1 Local Sensitivity Analysis

Local sensitivity analysis is used to analyze how a minor perturbation near a value in the input space $x^0 = (x_1^0, \dots\dots\dots, x_2^0)$ influences the value of $y = \emptyset(x_0)$ (Morio, 2011). It analyzes changes in the model outputs concerning variations in a single parameter input (Hamby, 1994). OAT (One At a Time) methods use a local sensitivity analysis approach. Local sensitivity uses differential Equations to calculate the effect of input variables on the output model (Saltelli et al., 2012).

2.2.2 Global Sensitivity Analysis

Global sensitivity analysis (GSA) is used to cut the computational cost. The work of Helton (1993) is considered one of the very early ones to promote the use of global sensitivity analysis. Global sensitivity analysis is concerned with the variance of model output Y and, more specifically, how to input variability affects the variance output (Homma and Saltelli, 1996). All parameters are modified concurrently over the whole parameter space in a global sensitivity analysis. It allows for simultaneous evaluation of each parameter's relative contributions and the interactions between parameters to the model output variance (Zhang et al., 2015).

The most used methods for global sensitivity analysis are a weighted average of local sensitivity analysis, partial rank correlation coefficient, multiparametric sensitivity analysis, Fourier amplitude sensitivity analysis (FAST), and Sobol's method (Saltelli et al., 2000). The global sensitivity analysis can be further classified as OAT (One-At a Time) or AAT (All-At-Once) methods (Pianosi et al., 2016).

- In the One At a Time method, one input is varied at once to observe the effect on the output
- In contrast, in AAT methods, all input parameters are varied together. It gives us detail about the influence of each input on output and gives a detailed description of the interaction between two input parameters.

Sobol Method and Morris methods are engaging in this thesis work and will be discussed in detail.

2.3 Sobol Method

The basis of Sobol's technique is the decomposition of the model output variance into summands of variances of the input parameters in increasing dimensionality (Saltelli et al., 2012). The variance-based effect of input parameters on output can be calculated using Sobol (1993) method (Saltelli et al., 2012). Sobol (2001) and Morio (2011) provide the mathematical explanation of the Sobol method as follows:

Let us consider a function to be investigated is given as:

$$Y = f(\boldsymbol{X}) \tag{2.1}$$

Y is model output, and **X** is a vector representation of input variables. It is assumed that inputs are statistically independent and are uniformly distributed across the range X € [0,1]. The Sobol decomposition for function f can be written as:

$$Y = f_0 + \sum_{i=1}^{d} f_i\,(X_i) + \sum_{i<j}^{d} f_{i,j}\,(X_i, X_j) + \ \ldots\ldots + f_{1,2,\ldots\ldots,d}(X_1, X_2\ , \ldots\ldots X_d) \qquad (2.2)$$

In Equation (2.2) f_0 is a constant term, f_i is a function of X_i and $f_{i,j}$ is a function of $X_{i,}\ and\ X_j$. For Equation (2.2) to be used, the following orthogonality condition in Equation (2.3) must be fulfilled.

$$\int_0^1 f_{i_1,i_2,\ldots\ldots i_d}(X_{i_1}, X_{i_2}, \ldots\ldots, X_{i_d})dX_t\ = 0 \qquad (2.3)$$

$$for\ k = i_1, i_2, \ldots\ldots, i_d$$

If the condition in Equation 2.3 holds, then we can write the functions $f_0, f_i\ and\ f_{i,j}$ as:

$$f_0 = E(Y) \qquad (2.4)$$

$$f_i(X_i) = E(Y|X_i) - f_0 \qquad (2.5)$$

$$f_{ij}(X_{i,}X_j) = E(Y|Xi, X_j) - f_0 - f_i - f_j \qquad (2.6)$$

The higher-order interactions can be studied by using Equation (2.7).

$$Var(Y) = E(Y^2) - E(Y)^2 = \int_0^1 f^2(X)dX - f_0^2 \tag{2.7}$$

Equation (2.7) can be decomposed for variance in the form of Equation (2.8).

$$Var(Y) = \int_0^1 \sum_{i=1}^{d} f_i(X_i)dX_i + \int_0^1 \sum_{i<j}^{d} f_{ij}\left(X_i, X_j\right)dX_i dX_j + \cdots + \int_0^1 f_{1,2,\ldots,d}\left(X_1, X_2, \ldots, X_d\right)dX \tag{2.8}$$

The variance of Y can be decomposed into Equation (2.9), where Vi and Vij can be further decomposed as shown in Equations (2.10) and (2.11).

$$Var\ (Y) = \sum_{i=1}^{d} V_i + \sum_{i<j}^{d} V_{ij} + \cdots \ldots . . + V_{1,2,\ldots d} \tag{2.9}$$

$$V_i = V_{X_i}(E_{X_{-i}}(Y|X_i)) \tag{2.10}$$

$$V_{ij} = V_{X_{ij}}\left(E_{X_{-ij}}\left(Y\middle|X_i, X_j\right)\right) - V_i - V_j \tag{2.11}$$

Equation (2.12) is used for the first order Sobol index. The first-order Sobol index gives the variance value for Y when each input parameter is changed.

$$S_i = \frac{V_i}{Var(Y)} \tag{2.12}$$

The total index is the calculation of combined variance caused to output X_i. Moreover, the variance of each input variable is caused by interaction with each other. The total sensitivity index is given as:

$$S_T = \frac{E_{X_{-i}} \ (Var_{X_i}(Y|\ X_i))}{Var\ Y} \tag{2.13}$$

$$S_T = \frac{1 - Var_{X_i}(\ E_{X_{-i}}\ (Y|\ X_i))}{Var\ Y} \tag{2.14}$$

Before implementing the Sobol method, we are defining which parameters we want to include in the analysis is crucial. After this step, the Sobol sequence is used to generate input parameters. Sobol sequence is generated using quasi-random samples. The generated sample set for the input parameter is fed to the model. As a result, first-order and total sensitivity indices are obtained, as shown in Figure 2.3.

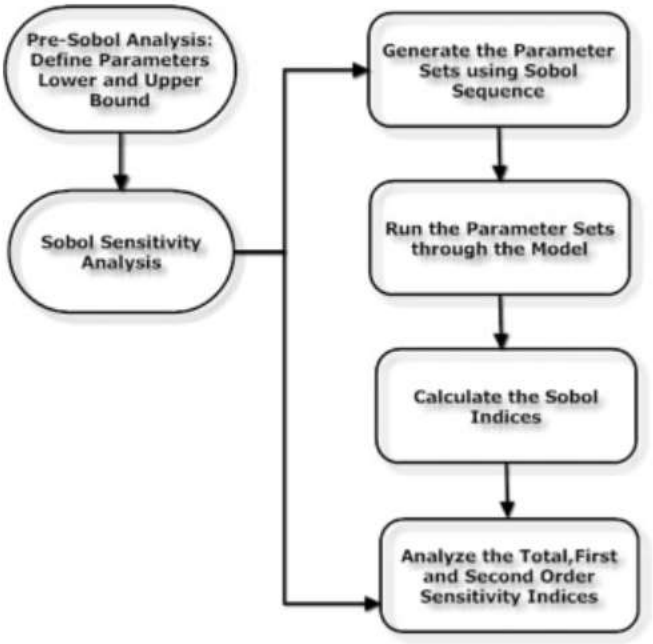


Figure 2.3: The steps showing implementation of the Sobol Method (adapted from Zhang et al., 2015)

2.4 Morris Method

The Morris (1991) method is an example of the OAT (One At a Time) global sensitivity analysis method. It is also called Elementary Effect Method (EEF) (Ikonen, 2016).In Morris's method, the individual effect of inputs is sampled on output by varying the value of one input at a time. The Elementary Effect Test is simply an average of derivatives over the space of factors, and its implementation can be expressed in the following way (Saltelli et al., 2004).

First, we need to consider model k with input parameters X_i. Here the range for i varies from 1 to k; for a given value of X, the elementary effect of the ith input factor is defined as:

$$\frac{EE_i = [Y(X_1, X_2, \dots\dots\dots, X_{-i}, X_i + \Delta, \dots\dots\dots, X_k) - Y(X_1, X_2, \dots\dots\dots, X_k)]}{\Delta} \qquad (2.15)$$

The input range varies across p, where p is the number of levels and Δ is equal to $\frac{1}{p-1}, \dots\dots \frac{1-1}{p-1}$.

Campolongo et al. (2011) describe three parameters to evaluate the input parameter performance. μ_i is the mean, μ_i^* is the absolute mean and $\sigma_i^2$is standard deviation.

$$\mu_i = \frac{1}{N}\sum_{n=1}^{N} EE_i^n \tag{2.16}$$

$$\mu_i^* = \frac{1}{N}\sum_{n=1}^{N} |EE_i^n| \tag{2.17}$$

$$\sigma_i^2 = \frac{1}{N}\sum_{n=1}^{N} (EE_i^n - \mu_i)^2 \tag{2.18}$$

The Morris method is usually used as a screening method, which means it is used as a pre-processor method for other SA methods to reduce the number of input parameters (Ikonen, 2016). This method is computationally cheap than the Sobol method (Saltelli et al., 2004)**.**

2.5 Monte Carlo Estimation

The Monte Carlo method presented by Metropolis and Ulam (1949) uses probability distribution to perform sensitivity analysis. The Monte Carlo method's first step is to find a random set of input values from their probability distribution. Then these values are used to obtain values for output (Loucks et al., 2005). This process is repeated till the sample size is specified. The final result is in the form of a probability distribution. Monte Carlo simulation is most effective in regression and classification models (Pianosi et al., 2016). Figure 2.4 shows a complete process of the Monte Carlo simulation for a hydrological simulation model.

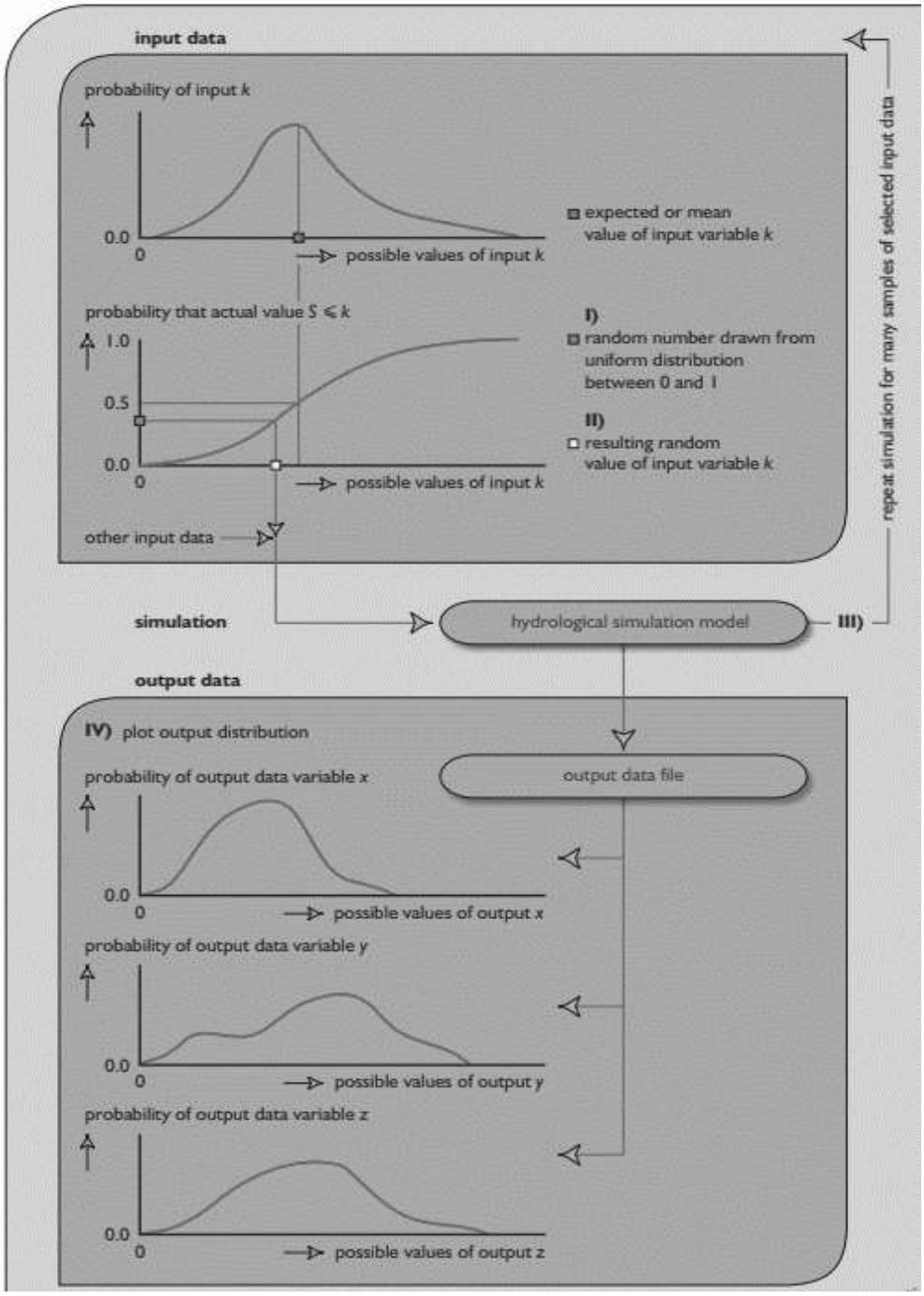


Figure 2.4: Monte Carlo simulation method to find output probability distributions (after Loucks et al., 2005)

Morio (2011) reviewed a method in their work to demonstrate how Sobol indices can be estimated by using Monte Carlo (MC) estimations. Firstly, two N sample size realizations of X are considered:

$$X_k^i = (x_{k_1}^i, \dots\dots, x_{k_p}^i) \tag{2.19}$$

$where\ k = 1, \dots., N;\ and\ i = 1,2$

The first-order sensitivity indices for input can be given as:

$$S_i = \frac{V_i}{V} = \frac{U_i - \emptyset_0^2}{V} \tag{2.20}$$

$\emptyset_0$ is the mean, which can be estimated using Equation (2.21).

$$\emptyset_0 = \frac{1}{N}\sum_{k=1}^{N} \emptyset\ (x_{k_1}^1, \dots\dots, x_{k_p}^1) \tag{2.21}$$

The variance can be estimated as:

$$V = \frac{1}{N}\sum_{k=1}^{N} \emptyset^2\ \left(x_{k_1}^1, \dots\dots, x_{k_p}^1\right) - \emptyset_0^2 \tag{2.22}$$

The term V_i which is variance can be decomposed as:

$$V_i = \frac{1}{N}\sum_{k=1}^{N} \emptyset(x_{k_1}^1, \ldots\ldots, x_{k_{(i-1)}}^1, x_{k_{(i+1)}}^1, \ldots\ldots, x_{k_p}^1) \cdot \emptyset(x_{k_1}^1, \ldots\ldots, x_{k_{(i-1)}}^1, x_{k_{(i+1)}}^1, \ldots\ldots, x_{k_p}^1) \tag{2.23}$$

The total sensitivity index can be estimated as:

$$S_{T_i} = 1 - \frac{V_i - \emptyset^2}{V} \tag{2.24}$$

2.6 Emulators for Sensitivity Analysis

When the model becomes complicated, the best approach is to use an emulator for sensitivity analysis. Emulation (also denoted as metamodelling in the literature) is an important and expanding area of research. It represents one of the significant advances in studying complex mathematical models, with applications ranging from model reduction to sensitivity analysis (Ratto et al., 2012). This modeling theme dates back to the work of Blanning (1975), but with the increasing complexity of computer models, it is becoming an area of interest lately.

Meta models are surrogate models built to improve the computation cost of a simulation model (Saltelli et al., 2004).

Like models are abstractions of some reality, meta-models are abstractions of models (Jeusfeld, 2009). Meta models are also called models of models (Cheng et al., 2019). The meta-model is a model inside a model that helps reduce the original model's burden and cut the computational cost (Saltelli et al., 2004).

Polynomial chaos expansions (PCE) and Gaussian processes (GP) have become highly famous for being used meta-models in recent years (Le Gratiet et al., 2017). Figure 2.5 shows an example of sensitivity analysis using a meta-model (surrogate model).

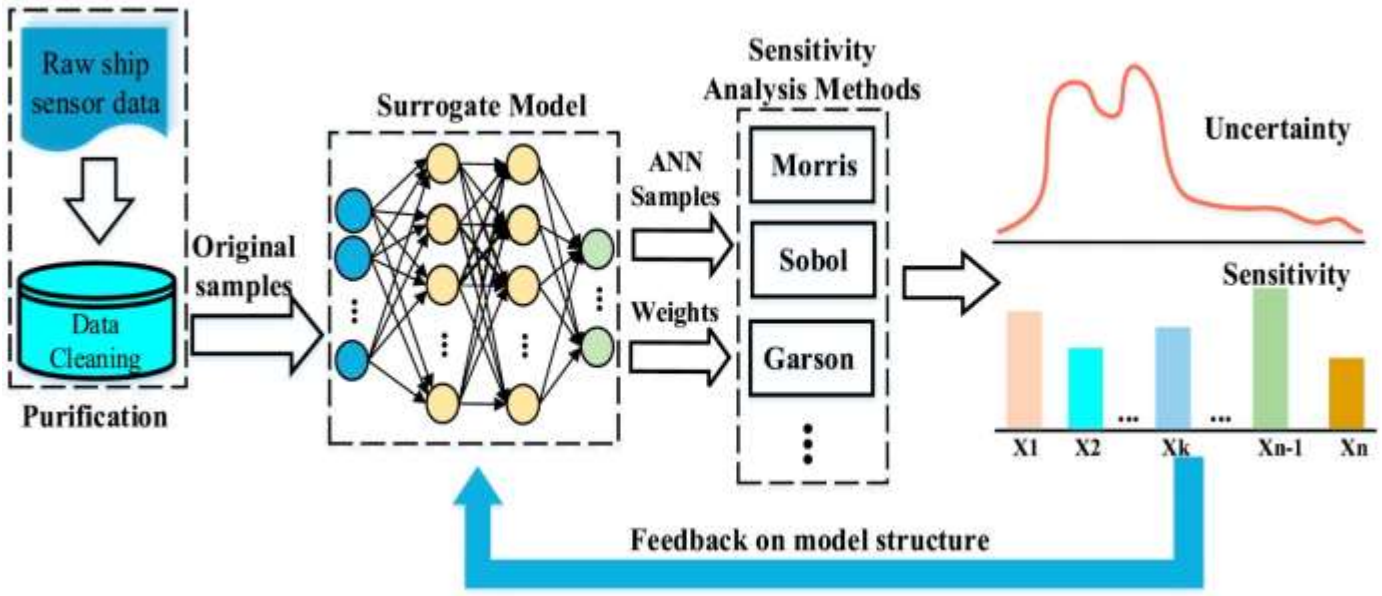


Figure 2.5: Sensitivity Analysis using an ANN surrogate model to predict sensitivity and uncertainty of a ship model

3. ANALYSIS OF DATASET

As mentioned earlier, this work entirely focuses on Sensitivity analysis. For a Machine Learning model to work correctly, it is vital to feed it with clean, corrected, and processed data. As it highly affects the performance of ML models.

The dataset used in this work is Volve field data provided by Equinor on their website. It contains 40,000 data files collected at the North sea oil field. The dataset was collected between 2008 and 2016. Volve dataset used in this is pre-processed by previous UiS students, as it is of great interest for research at UiS. The dataset used was pre-processed and can be located at Volve Field Data Link. Datasets are available for different fields and are divided into time-based and measured depth-based datasets. The dataset used for this study is from well 15/9-F-5 of the Volve field present at the North Sea.

Cleaning the raw data for the machine learning process is essential. The processes performed on this data are discussed below.

3.1 Construction of Dataset

The first step of analyzing data is importing and arranging it. The names of the column were not specified and only numbered. So the name of each column is selected for a better understanding of the results. Labeling data is also crucial for supervised Machine Learning.

The next step is to verify the correct data type of each feature. This step is critical as it is vital to know the data type before feeding it into ML models. This step was vital in the other parts of this work, where Emukit was used to implement the sensitivity analysis method. Knowing the type of features to integrate with the software was significant.

Feature	Unit	Data type
Measured Depth	meters	float64
Weight On Bit	tonnes	float64
Hook load	tonnes	float64
Surface Torque	kN-m	float64
Downhole Torque	kN-m	float64
Downhole Weight on the bit	tonnes	float64
Speed	rpm	float64
Mud Flow	L/min	float64
Standpipe Pressure	kPa	float64
Rate of Penetration	m/h	float64

Table 3. 1: Features selected from well 15/9-F-5

The features of this study are shown in Table 3.1 Measured Depth, Weight On Bit, Downhole Weight on a bit, Surface Torque, Downhole Torque, Standpipe pressure, rpm, Rate of penetration Hookload, and Mudflow.

3.2 Data Cleaning

Data quality is one of the most crucial aspects of Machine Learning models (Gupta et al., 2021). Detecting dirty data and cleaning it to any model is essential before feeding it to any model. If not done, results can become inaccurate and unreliable (Chu et al., 2016).

The dataset used in this work is obtained from the oil field and contains data collected through sensors. It is possible that the raw data is dirty and has outliers, irregularities, and errors. The messy data results in bad results, so cleaning data before feeding it to the ML model is significant.

Data cleaning is the technique used to detect and repair the errors in the dataset (Ilyas and Chu, 2019). The data cleaning process consists of three steps.

- Detecting the error and type of error (i.e., qualitative or quantitative)
- Fixing that error using different techniques
- Evaluating the quality of processed data

Initially, it was checked if any null values were present, but as no null values were current, it was not required to address this issue early. This work focuses on outlier removal, handling missing data (after outlier removal), and data transformation.

3.2.1 Outlier Removal

Aggarwal (2017) describes outliers as "abnormalities, discordant, deviants, or anomalies in the data mining and statistics literature." Removing outliers can be challenging as essential information about the dataset can be lost. Secondly, outlier removal can cause a reduction in data which needs to be handled with care. Outliers can be visualized manually using graphs; then, advanced techniques can be used for removal. In this work, we have used two approaches to determine whether both identify the same number of outliers. The first technique removes outliers through the Interquartile range, and the second is through Density-based spatial clustering of applications with noise (DBSACN) (Ester et al., 1996). Both methods were applied separately first and were able to identify the same number of outliers in the data. However, DBSCAN proved to be computationally costly compared to the IQR method. The box plots were used to observe the outlier in our study before applying any technique, and Figure 3.1 shows the results. The red circles on the graph indicate the outliers for each feature.

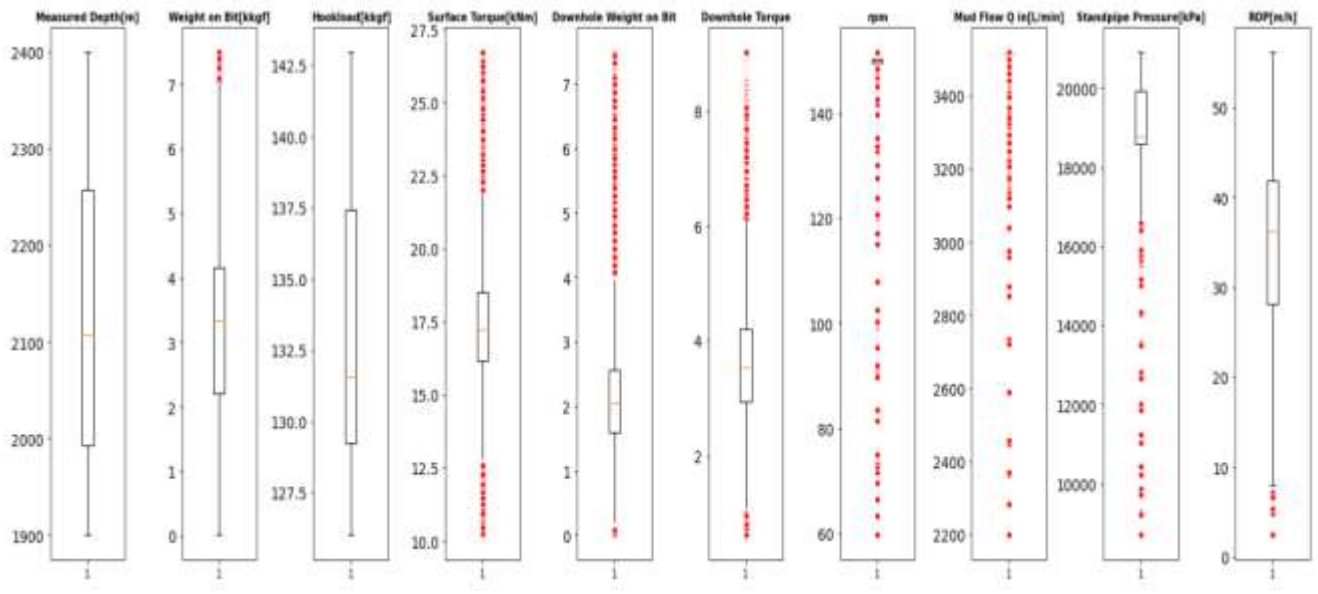


Figure 3.1: Outliers identification using box plots

Interquartile Range: IQR

The interquartile method removes data points located away from the median. IQR defines the propagation of distribution (Sainis et al., 2018). Whaley III (2005) describes the theory and methodology of interquartile range in detail. The difference between the third and first quartiles is the interquartile range (Miyazaki et al., 2021). The following Equation (3.1) gives it.

$$IQR = Q3 - Q1 \tag{3.1}$$

The third quartile is three quarters or seventy-five percent of any dataset, and the first quartile is the first quarter or twenty-five percent of data. One way to visualize the outliers is through box plots can be used. Figure 3.2 shows the distribution of an interquartile range of a data set using a box plot.

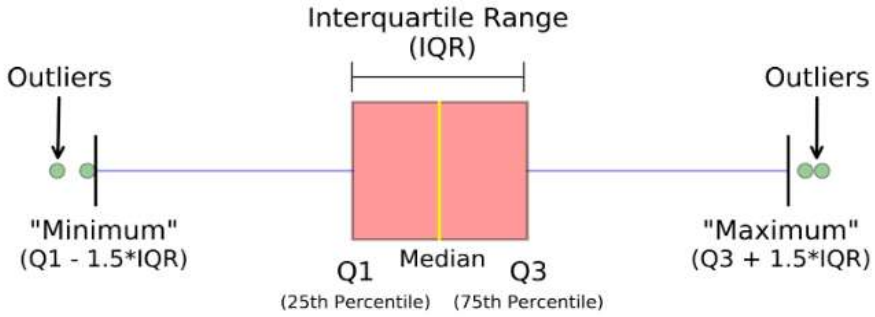


Figure 3.2: Distribution of data in a box plot (after towardsdatascience.com)

The minimum and maximum of the data are given by Equation (3.2) and Equation (3.3). Outliers are data points that fall outside of this range.

$$Minimum = Q1 - 1.5 \cdot IQR \tag{3.2}$$

$$Maximum = Q3 + 1.5 \cdot IQR \tag{3.3}$$

DBSCAN

DBSCAN is a Density-Based Spatial Clustering of Applications with Noise that uses a pioneer density-based algorithm to form the clustering of spatial datasets (Khan et al., 2014). It forms clusters of many different shapes and sizes in a dataset based on the two-parameter values; epsilon and minimum samples. Any data point not a part of these clusters is identified as an outlier. Epsilon is the maximum distance between two points of the same cluster, whereas minimum samples are the number of data points that can become a part of each cluster. The minimum sample value identifies whether a data point is a core point or not. DBSCAN is highly sensitive to these two values (Khan et al., 2014).

This study used the scikit learn library with default values for epsilon and minimum samples for outlier removal.

3.2.2 Handling Null values

After applying outlier techniques, 3884 data points became null values. It is a considerable number, and this issue needs to be solved. The interpolation technique was used to fill in the gaps for these values. The interpolation method uses the already known data points to fill the null values gap (Steffensen, 2006). Interpolation can be linear, quadratic, cubic, or polynomial. We have used linear interpolation using the pandas data frame. After applying the interpolation technique, there were no more null values in the data set.

3.3 Data Standardization

Data normalization refers to bringing all data in the same form. In any dataset, there can be various units for different input parameters. Through normalization, these values are changed to the same range. Data ranges between 0 and 1 or -1 and 1 after normalization (Ali et al., 2014).

We have used the Min-Max scaler from scikit learn to normalize the data. The formula for Min-Max normalization is given as follows (Singh and Singh, 2020):

$$x'_{i,n} = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)} (nMax - nMin) + nMin \quad (3.4)$$

nMax and nMin are lower and upper bounds to rescale the data, whereas min and max are the minima and maximum values for the i_{th} feature.

4. Machine Learning Models

Machine learning is an evolving branch of computational algorithms designed to emulate human intelligence by learning from the surrounding environment (El Naqa and Murphy, 2015). Machine learning techniques are classified as Supervised and Unsupervised Learning. The learning technique is chosen based on the dataset available and the problem to be solved.

4.1 Supervised Learning

In Supervised Learning, the new outcomes depend upon the externally supplied instances to predict the fate of future instances (Singh et al., 2016). For Supervised Learning, the training data must be adequately labeled. It uses a mapping function to construct a mathematical model for the labeled training data, which is then used for the outcome prediction for future data instances. Supervised Learning uses feedback to check the accuracy (Nasteski, 2017). These mathematical models are built by utilizing either classification or regression. Choosing a classifier or regressor depends upon the dataset's type of input and output. Classifiers are used when categorical or discrete data, whereas regressors are used when information is continuous in a given range (Müller and Guido, 2016).

4.2 Machine Learning Algorithms

The dataset used in this study is continuous. Both inputs and target output are known for our study; therefore, regression models are used. The machine learning regression models used in this study are

- Random Forest
- Gradient Boosting
- K-Nearest Neighbor

- Local Cascade Ensemble

4.2.1 Random Forest

Random forest (RF) algorithms are a type of ensemble learning algorithm resulting from an assemblage of Decision Trees. Breiman (2001) proposed this method, which predicts the output by creating several decision trees from the training dataset. Decision Trees (DT) use a stepwise approach to split the provided training dataset Figure 4.1. DT follows a top-down approach; it starts from the start at the top of the tree and descends by selecting a path based on desired characteristics or features of the outcome. The root node recursively applies a binary split on the most predictive feature to create new nodes in every step. This process continues till the purest node is achieved.

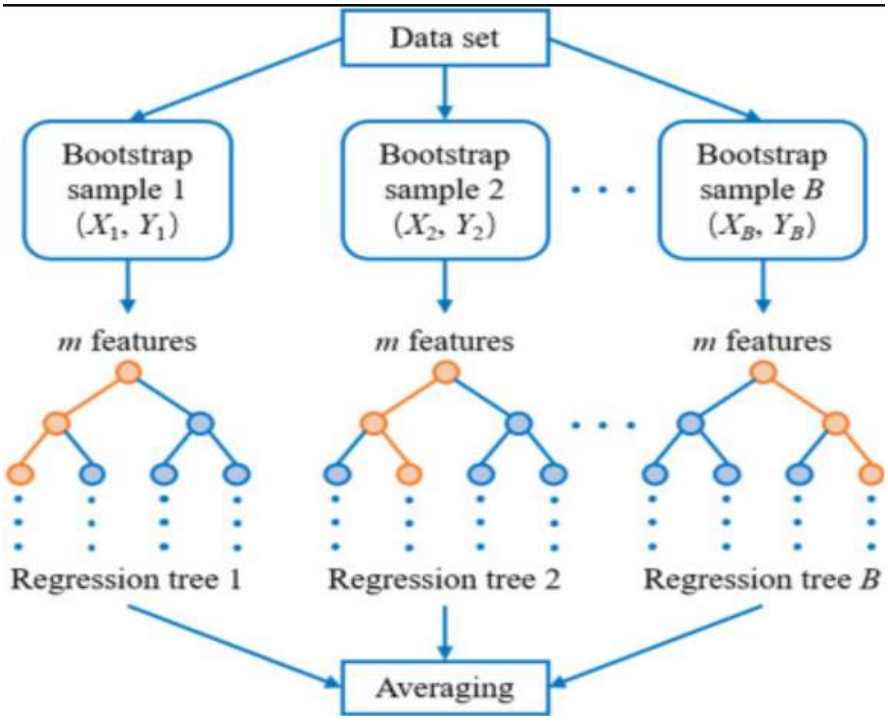


Figure 4.1: Workflow of random forest algorithm (Wang et al., 2022)

Unlike DT, Random Forest does not select all features but selects a random set of features to create decision trees. These decision trees are then assigned a weight according to prediction accuracy. The outcome is predicted based on the DT with higher accuracies.

4.2.2 Gradient Boosting

Gradient Boosting also falls in the category of ensemble method. Friedman (2001) proposed to modify the weak learner algorithm into a better one. Gradient booster iteratively combines the weak learner into a stronger one. It is a powerful model that can find a non-linear relationship between input features and target output. The Gradient Boosting Regression algorithm predicts the continuous value of the model. Unlike Random forest, Gradient Boosting has a fixed number of Decision Trees.

It uses a technique that repeatedly adds decision trees so that the following decision tree corrects the previous decision tree error. Gradient Boosting algorithms are highly sensitive to hyperparameters for accurate outcomes.

4.2.3 K-Nearest Neighbors (KNN)

K-nearest neighbor (KNN) is a non-parametric algorithm for classification and regression problems. It uses feature similarity to identify the new data points. The values are assigned to the new points based on the similarity with the points in the training dataset.

The "similarity" between the points depends upon the "distance." The points with the smallest distance are declared neighbors or similar data points. In regression models, this distance is measured using Euclidean Distance (Zhang and Zhou, 2007)**.**

4.2.4 Local Cascade Ensemble (LCE)

Local Cascade Ensemble (LCE) (Fauvel et al., 2022) is a new machine learning method that combines the features of Random forest and XGBoost. LCE combines the two main features of bagging and boosting to get the best results. LCE uses cascade generalization, where a set of predictors are used to sequentially add new attributes to the input dataset at each stage. It uses a decision tree that splits further based on bias reduction across the node. It is done using boosting predictors. It recursively occurs, and weights are assigned similarly to the XGBoost algorithm. Bagging is used to overcome the overfitting issue in each tree, similar to Random forest. The outcome depends upon the majority votes from highly accurate trees.

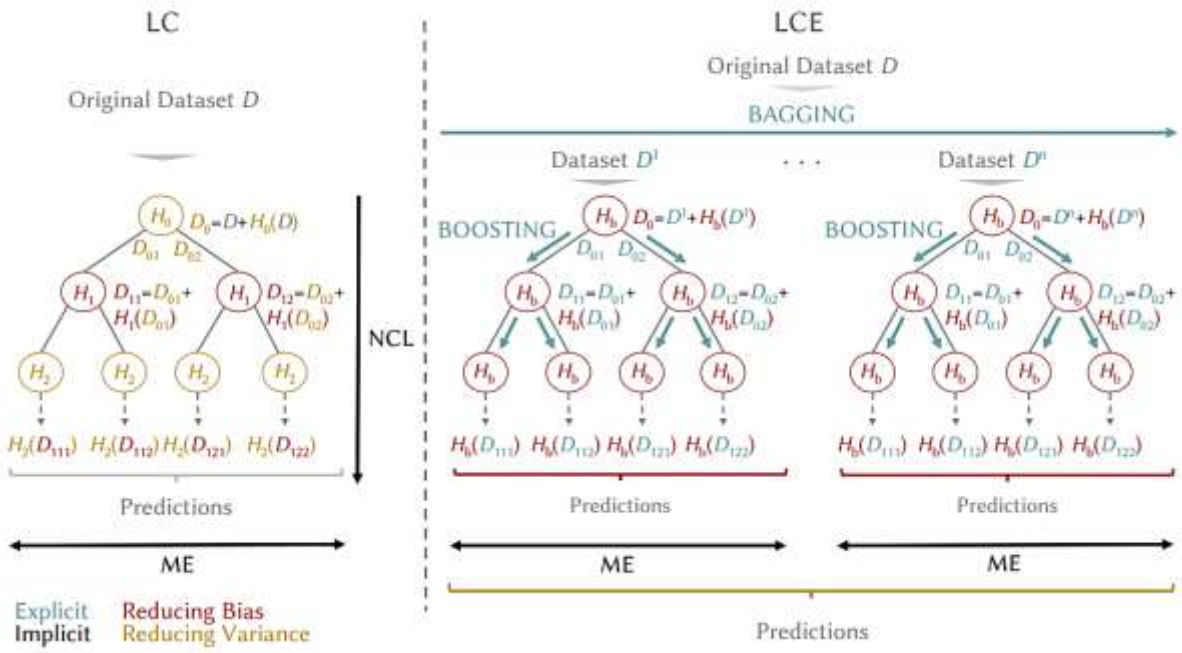


Figure 4.2: Local cascade vs. Locala Cascade ensemble (Fauvel et al., 2022)

4.3 Regression Metrics:

To study the performance of regression models, it is essential to define some regression metrics. The two regression metrics used in this thesis, Coefficient of Determination (R2) and Root Mean Square Error (RMSE), are used for the performance evaluation of each ML model.

4.3.1 Coefficient of Determination (R2 score):

Coefficient of Determination or R2 score is used in ML regression models to evaluate performance. It is the difference between data points and predictions of the machine learning model. The formula to calculate the R2 score is given as follows by Mann (2010):

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y})^2}{(y_i - \bar{y})^2} \tag{4.1}$$

N = Total number of observations

y_i = Data point at i^{th} observation

Y = second or subsequent data point to y_i

4.3.2 Root Mean Square Error (RMSE)

Root Mean Square Error is another metric that calculates error. It is highly used in ML models for performance evaluation. Its formula is given as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted\ i - Actual\ i)^2}{N}} \tag{4.2}$$

N = Total number of observations

Predicted i = Predicted value for i^{th} observation

Actual i = Observed value for i^{th}observation

4.4 Implementation of Machine Learning models:

Now we shall discuss the implementation of ML models. The first step in implementing ML models is splitting the dataset. The dataset is split randomly into the test, train, and validation datasets. The model is trained using the training set to observe and learn from. The test set is used to see how well a model generalizes to new scenarios and should be utilized after the training and validation sets have been completed. In our study, the test data is used for validation. The data is split into 60% training and 40% test datasets.

The next crucial step is hyperparameter tuning. For ML models to perform well, tuning correct hyperparameters is important (Hutter, 2014). We have used the Grid Search method for hyper tuning. It was done by using GridSearchCV in Scikit learn.

Table 4.1 shows the R2 score and RMSE for the Machine Learning models.

Machine Learning models	R2 score	Root Mean Square Error RMSE
Random Forest	0.99	0.0001
Gradient Boosting	0.98	0.0006
K- Nearest Neighbours	0.99	0.0001
Local Cascade Ensemble	0.93	0.0028

Table 4.1: Comparison of Machine Learning models using R2 and RMSE scores

5. Feature Selection

Feature selection is a technique for identifying the most crucial input parameters from a dataset. The input variables that are fed to machine learning models are called features. Feature selection is a process that allows an automatic or manual selection of the features that have the most effect on the model output. Through feature selection number of input, variables can be limited to only those most critical, which further helps reduce the computational cost of the model (Saranya and Pravin, 2021). It helps in reducing training times and helps in handling dimensionality. Besides maximizing model performance, other benefits of applying feature selection include the ability to build more straightforward and faster models using only a subset of all features, as well as gaining a better understanding of the processes described by the data by focusing on a selected subset of features (Guyon and Elisseeff, 2003), (Wang, 2010). Unlike feature extraction, feature selection returns the subset of essential features rather than creating new features from the original dataset (Chandrashekar and Sahin, 2014).

Feature extraction projects the original high-dimensional features to a new feature space with low dimensionality. The newly constructed feature space is usually a linear or non-linear combination of the original features. On the other hand, Feature selection directly selects a subset of relevant features for model construction (Liu et al., 2010).

It has already been mentioned what feature selection is. Now the main question arises why do we need feature selection? In machine learning models, the dataset plays an important role. ML models perform computation on any data. We get bad results if we feed the ML model with imperfect quality data. Similarly, it is imperative to feed the model with good quality data to get

good results (Kalakech et al., 2011),(Zhao et al., 2012). Also, learning models tend to overfit many features, which may cause performance degradation on unseen data (Li et al., 2017).

In today's world, big datasets are getting significant, and they are also crucial for ML models because they help them learn better (Park et al., 2018). However, most of the information in these datasets is irrelevant to the user's needs. To remove an irrelevant parameter, a feature selection criterion is required to measure each feature's relevance with the output class/labels (Chandrashekar and Sahin, 2014). Therefore, selecting the most important features for each modeling scenario becomes necessary. In this thesis, feature selection is treated as a sensitivity analysis method.

It is tested whether feature selection can predict the sensitive input parameters correctly. It is also observed which feature selection methods give accurate results. It was also tested whether traditional SA methods can be replaced by feature selection.

5.1 Categorization of Feature Selection Algorithms

According to the availability of supervision (such as class labels in classification problems), feature selection can be broadly classified as supervised, unsupervised, and semi-supervised methods (Li et al., 2017). Chandrashekar and Sahin (2014), Li et al.(2017), and Hallajian et al. (2022) provide detailed literature about feature selection models.

5.1.1 Supervised models

Supervised feature selection refers to the method which uses the output label class for feature selection (Di Mauro et al., 2021). In this model, it is vital to identify the input and output variables beforehand. The data can be classified as numerical or categorical data. Supervised

models use the target variables to identify the variables that can increase the model's efficiency and remove the irrelevant ones. Such models are used for classification and regression.

Supervised methods select the subset of features that can estimate the regression targets. This thesis only focuses on the regression model. Therefore supervised methods will be assessed for ML regressors.

5.2 Feature selection Techniques

The models discussed above are based on a "supervised selection perspective." The following paragraphs describe feature methods based on a "strategy perspective."

Based on selection strategy, feature selection methods are categorized as (1) Wrapper method (Kohavi and John, 1997), (2) Filter method (Liu and Setiono, n.d.), and (3) Embedded methods

5.2.1 Wrapper Method

The wrapper method follows a greedy search approach by evaluating all the possible combinations of features present during evaluating ML models. The feature selection process in the wrapper method depends upon a particular machine learning algorithm that attempts to fit a given dataset. This method is iterative, and after each iteration, it is decided to add or remove some features from the feature subset used in the previous iteration based on the model performance in that iteration, as shown in Figure 5.1. The iteration stops when the algorithm produces a feature subset with the desired number of features. During the process, either all features can be included, or the number of features can be limited by specifying them in the algorithm.

The final ranking of essential features is just a performance measure that varies depending on the problem. The evaluation criterion for regression can be p-values, R-squared, and Adjusted R-

squared; similarly, the evaluation criterion for classification might be accuracy, precision, recall, and f1-score (El Aboudi and Benhlima, 2016). Because it chooses the feature combination that produces the best results for the machine learning algorithm, it can also cause overfitting. These methods are computationally expensive for data with many features (Kohavi and John, 1997).

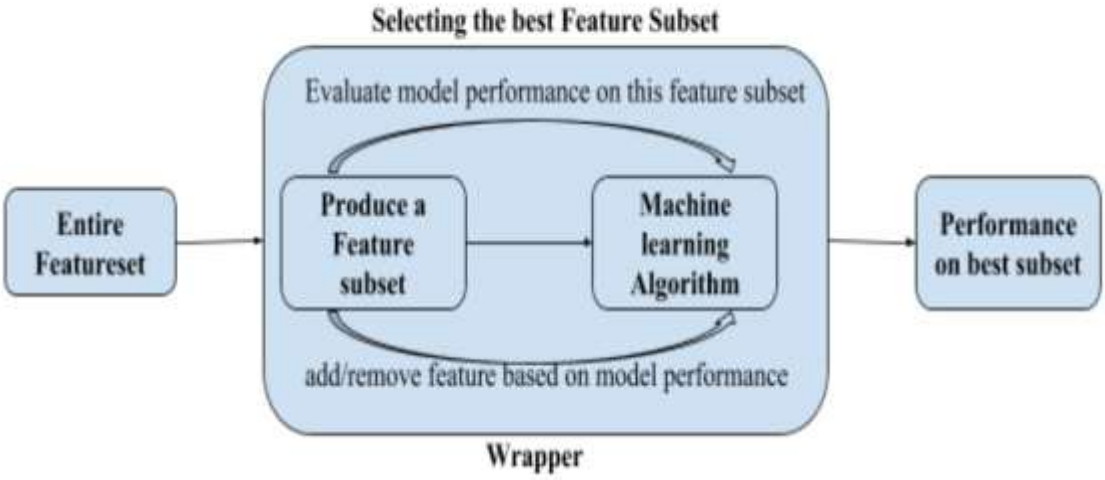


Fig 5.1: Working of wrapper method for feature selection (El Aboudi and Benhlima, 2016)

Recursive Filter Elimination:

RFE is the wrapper method used in this work; therefore, brief literature is shared here. Recursive Feature Elimination, also known as RFE, is the method that selects features based on their effect on the model's overall performance. It removes the unwanted features until the desired number of features is reached. Recursive feature elimination is a backward selection of the predictors (Guyon et al., 2002). RFE continuously performs iterations and every time selects the best and worst-performing features. Once the number of desired features is reached, it stops; the working of the RFE algorithm is shown in Figure 5.2. The number of desired features can be specified in the code or left unspecified, but it can be expensive computationally.RFE is flexible and easy to use as it can be used with any classifier or regressor with an embedded feature importance method. Some models show better results than others when combined with RFE. When the

number of predictors exceeds the number of samples, some models cannot be employed because RFE mandates that the initial model use the entire predictor set (Qi et al., 2018).

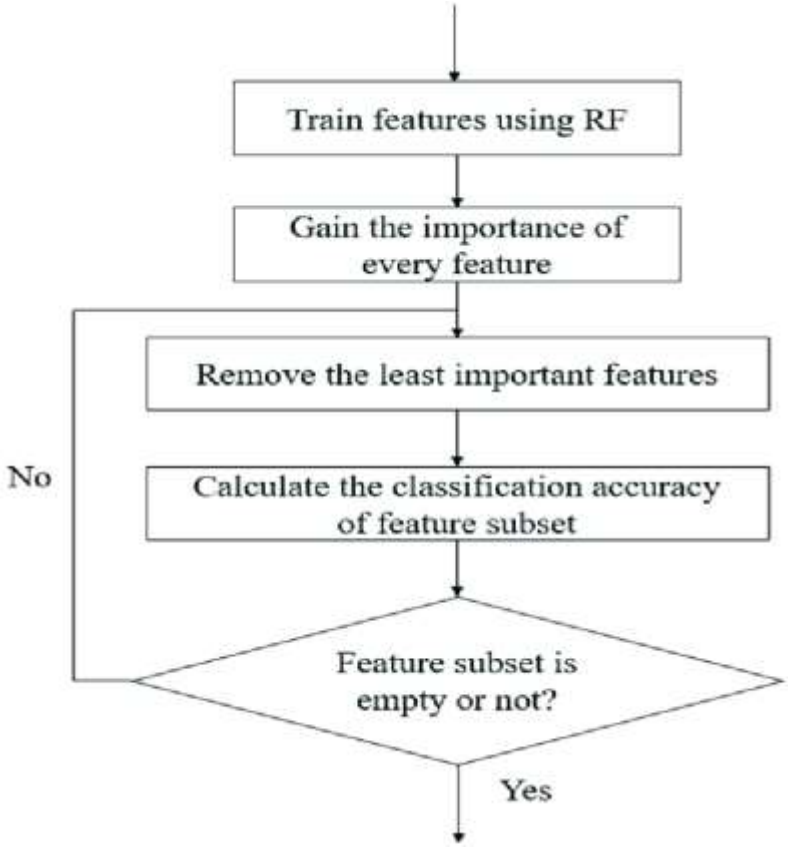


Figure 5.2: Flow chart describing the working of RFE (Qi et al., 2018)

5.2.2 Filter Method:

The filter method serves as a filter to remove irrelevant and redundant features. It works independently of a learning algorithm (Almuallim and Dietterich, 1994). The filter method uses a statistical approach to find the relation between input and targeted output variables. As a result, it provides a ranking for each feature. The filter method is computationally less expensive, but it might not be able to provide the best feature subsets. Therefore, it is used as a pre-processing step for the wrapper method to decrease the number of irrelevant features. Figure 5.3 shows the working of the filter method.

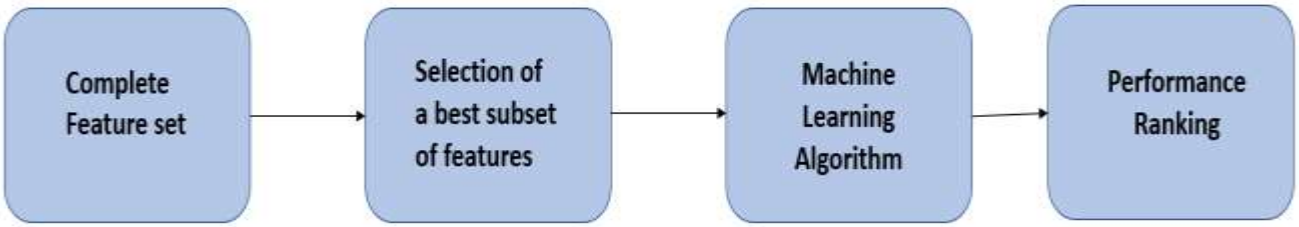


Fig 5.3: Steps of the filter method

Pearson Correlation:

Correlation is a statistical approach used to determine the linear relation between two variables. Taylor (1990) describes the interpretation of correlation values. All of the features can have values between -1 and +1. If the value for correlation is zero, it shows no relation between the feature and output targeted variable. Negative values represent the inverse relation between the chosen input and the targeted output variables. It means that if the value of the input feature increases, the output variable value will decrease. Conversely, positive values represent a direct relationship that increases the input and output variable values (Nettleton, 2014). More detailed information regarding Pearson's correlation can be found in Bollen (1984).

Mutual Information:

Duncan (2006) explains the idea of mutual information to find out the relation between two variables. Mutual Information is a statistical model that finds the relation between two

variables by entropy estimation. The entropy of a random variable is a measure of its uncertainty and the average amount of information required to describe the random variable (Learned-Miller, 2013).

The values obtained for features are always greater than zero. If a feature has a value of mutual information equal to zero, the two-variable do not have any relation and are independent of each other. Also, the greater the value of mutual information more substantial the relation between the input feature and the output target variable.

5.2.3 Embedded Method:

Embedded methods differ from other feature selection methods in how feature selection and learning interact (Lal et al., 2006). Embedded methods are hybrid methods as they combine filter and wrapper methods. These methods combine the learning algorithm and selection of feature subset parts. Embedded methods proceed more efficiently by directly optimizing a two-part objective function with a goodness-of-fit term and a penalty for many variables (Guyon and Elisseeff, 2003). Algorithms with built-in feature selection methods are used to implement it.

These methods use the machine learning model's inherent structure to select a feature at each phase of the model creation process. The embedded method stops the evaluation when all features have been covered or the model's performance does not appear to be improving. The following Figure 5.4 represents the working of embedded methods.

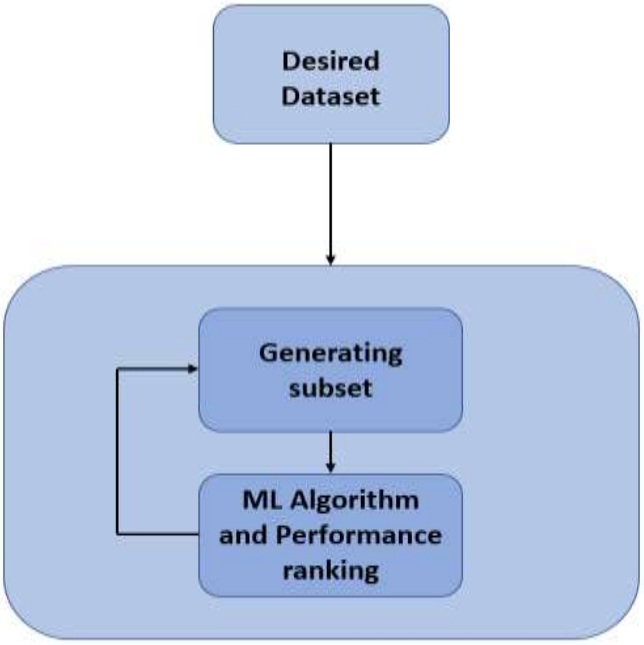


Fig 5.4: Working of embedded methods

Random Forest Feature Importance:

Embedded methods use built-in functions of regressors or classifiers to identify the most critical features from a dataset. It can be implemented using the built-in "feature_importance" function for Random Forest. Random forest uses feature importance to identify the best performing features using Gini importance, also known as mean decrease accuracy. Uddin and Uddiny (2015) explain the working of feature importance in Random forests.

Each Random forest is a set of multiple decision trees, and each decision tree is further divided into nodes and leaves. Based on the Gini index, features are selected, which are used to split the trees further. Gini index is given as follows (Uddin and Uddiny, 2015):

$$Gini\ index(v) = 1 - \sum_{i=1}^{I} f_i\,(1 - f_i) \tag{5.1}$$

Where

v = node

fi = fraction of i

i = records ar node v

The criteria to rank features are based on the decrease in impurity on each node. The feature that most decreases impurity is ranked as the best performing feature.

Permutation

The permutation is another built-in function used to identify best-performing features. It can work with any ML model. This approach will shuffle each feature randomly and calculate the difference in model performance (Huang et al., 2016).

5.3 Implementation of Feature Selection Techniques

In this part, the dataset used to evaluate each feature technique mentioned in the previous section consists of 10 columns. ROP is selected as the targeted output variable, and all other columns are considered inputs. This study uses one wrapper, two filters, and two embedded methods. The dataset used had all numerical values. Both input and output values are continuous numerical values; therefore, ML regressors are used for each method.

It is observed from the results that the type of regressor dramatically affects the final results. Also, many selected features remain almost the same in all cases other than Mutual Information.

The ranking of input parameters is done by using column numbers therefore Table 5.1, represents column numbers associated with each parameter. This table will help in interpreting the results.

Column number	Feature
0	Measured Depth
1	Weight On Bit
2	Hook load
3	Surface Torque
4	Downhole Torque
5	Downhole Weight on the bit
6	Speed
7	Mud Flow
8	Standpipe Pressure
9	Rate of Penetration

Table 5.1: Column numbers for parameters of used drilling dataset

5.3.1 RFE Method: Wrapper

Recursive Feature Elimination is the type of wrapper method used for the study. As mentioned earlier, RFE can be coupled with any ML regressor or classifier, so in this case study Decision tree, Gradient booster, and Random forest regressors are used.

The limit to the number of desired features was not mentioned because there were only nine features, and all were included in the evaluation. Setting a max limit for the desired feature can help reduce computational costs if there are many features.

RFE with Decision Tree:

When RFE is implemented with a Decision tree following results are obtained.

```
Column: 0, Selected True, Rank: 1.000
Column: 1, Selected False, Rank: 2.000
Column: 2, Selected True, Rank: 1.000
Column: 3, Selected False, Rank: 3.000
Column: 4, Selected True, Rank: 1.000
Column: 5, Selected True, Rank: 1.000
Column: 6, Selected False, Rank: 5.000
Column: 7, Selected False, Rank: 6.000
Column: 8, Selected False, Rank: 4.000
```

Figure 5.5: Feature ranking using RFE method with Decision Tree

RFE selects four features that contribute the most to model implementation in this case. The result shows that features from Columns 0, 2,4, and 5 are marked as selected true. As expected, RFE tested all the features, but only four were selected as the ones that affected the output the most. According to these results, Depth, Hookload, Downhole WOB, and Downhole Torque strongly influence ROP.

RFE with GBA:

In the case of GBA following results were obtained:

```
Column: 0, Selected True, Rank: 1.000
Column: 1, Selected True, Rank: 1.000
Column: 2, Selected False, Rank: 3.000
Column: 3, Selected False, Rank: 4.000
Column: 4, Selected True, Rank: 1.000
Column: 5, Selected True, Rank: 1.000
Column: 6, Selected False, Rank: 6.000
Column: 7, Selected False, Rank: 5.000
Column: 8, Selected False, Rank: 2.000
```

Figure 5.6: Feature ranking obtained from RFE with Gradient Booster

From the results, four features were ranked as critical. However, interestingly, three features are the same as selected by the decision tree. Instead of Hookload, WOB is selected as an essential feature.

RFE with Random Forest:

The results for random forest are as follows:

```
Column: 0, Selected True, Rank: 1.000
Column: 1, Selected False, Rank: 4.000
Column: 2, Selected False, Rank: 2.000
Column: 3, Selected False, Rank: 3.000
Column: 4, Selected True, Rank: 1.000
Column: 5, Selected True, Rank: 1.000
Column: 6, Selected False, Rank: 6.000
Column: 7, Selected False, Rank: 5.000
Column: 8, Selected True, Rank: 1.000
```

Figure 5.7: Feature ranking using the RFE method with Random Forest

It is worth noting that the number of most critical features remains four no matter which ML regressor is used. In the case of Random forest, three chosen features are the same, but the fourth is Standpipe pressure.

The idea of implementing RFE with different regressors was to observe the behavior of the technique. It is safe to say that number of features selected was the same in each case. Secondly, Depth, Downhole WOB, and Downhole torque were selected in each case.

5.3.2 Pearson's correlation: Filter

The bar plot below shows the predicted features with Pearson correlation.

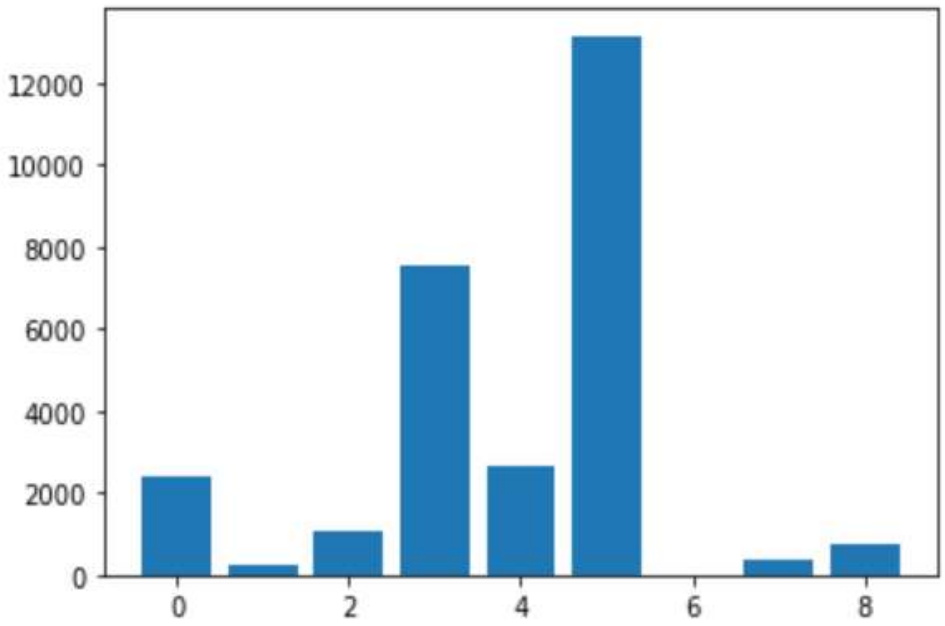


Figure 5.8: Features selected through Pearson's correlation

According to Pearson Correlation four features, Depth, Surface torque, Downhole WOB, and Downhole torque have the highest correlation with output ROP.

5.3.3 Mutual Information Method: Filter

Results obtained by Mutual Information are represented in Figure 5.9

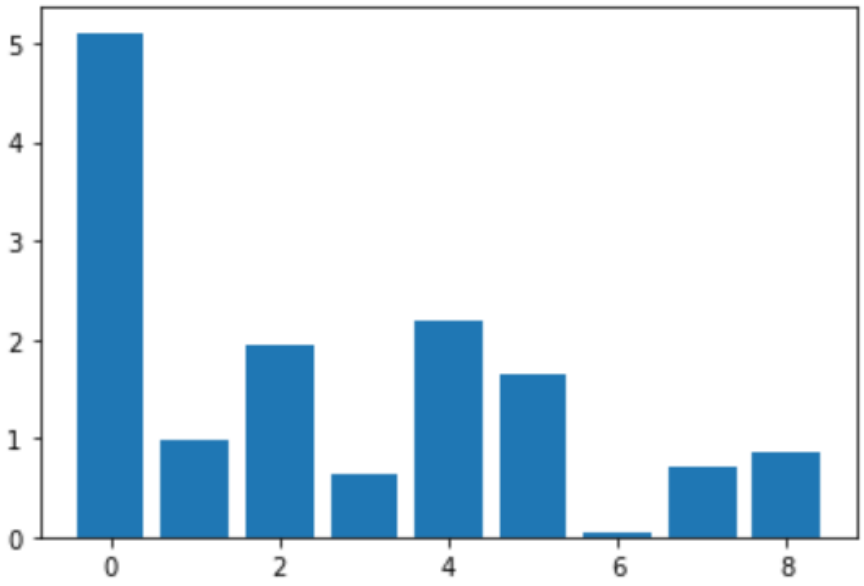


Figure 5.9: Features selected through Mutual Information

Mutual Information has considered almost all features as critical in the evaluation. It can result from the joint probability distribution that the MI method uses for calculation. MI may have also considered the effect of input variables on each other during this evaluation, making all of them significant for calculations.

5.3.4 Random Forest Feature Importance method: Embedded

In the case of embedded methods, feature selection is made using Random forest. The result obtained in Figure 5.10 is close to the ones by RFE using Random forest but not the same. In this case, the selected features are also the same. Depth, Downhole WOB, Downhole torque, and Standpipe pressure are most important.

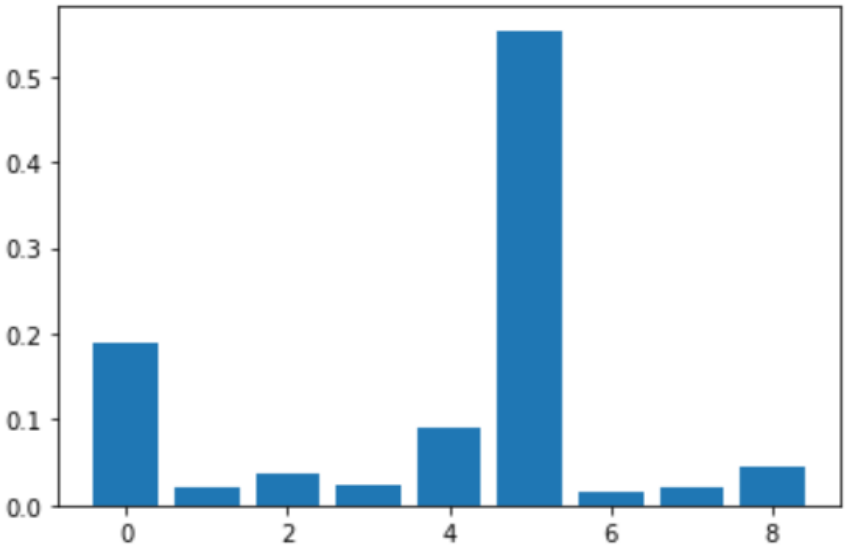


Figure 5.10: Features selected through RF Feature importance

5.3.5 Permutation method: Embedded

For the last method, permutation was implemented because this method helps overcome the error of impurity in Random forest feature importance. Because Permutation can work with all regressors, Random forest and KNN models are selected. The first plot, Figure 5.11, shows the results from Random forest, and the second plot in Figure 5.12 shows results from KNN.

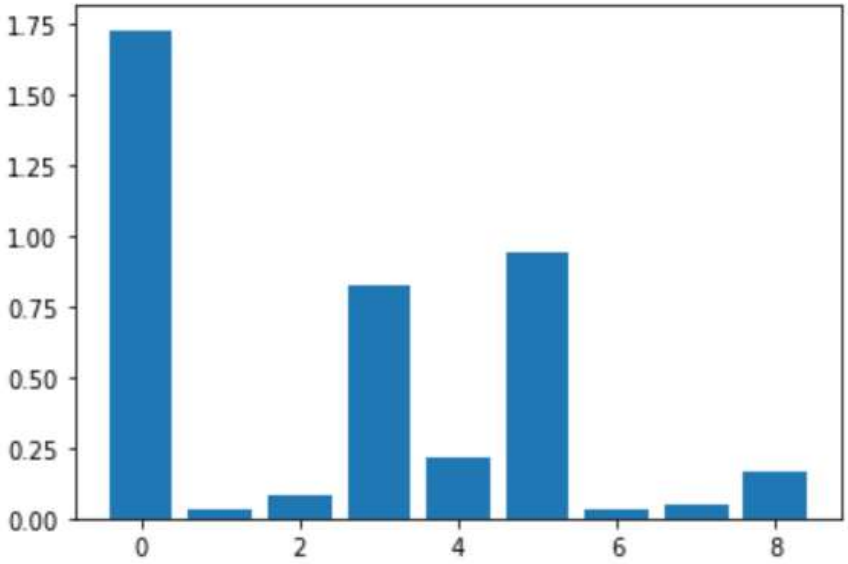


Figure 5.11: Features selected through Permutation of Random Forest

Through Random forest, the features selected are Surface Torque, Depth, and Downhole Torque. The KNN model only chose the first and last columns in the process. The features chosen by KNN are Depth and Standpipe Pressure.

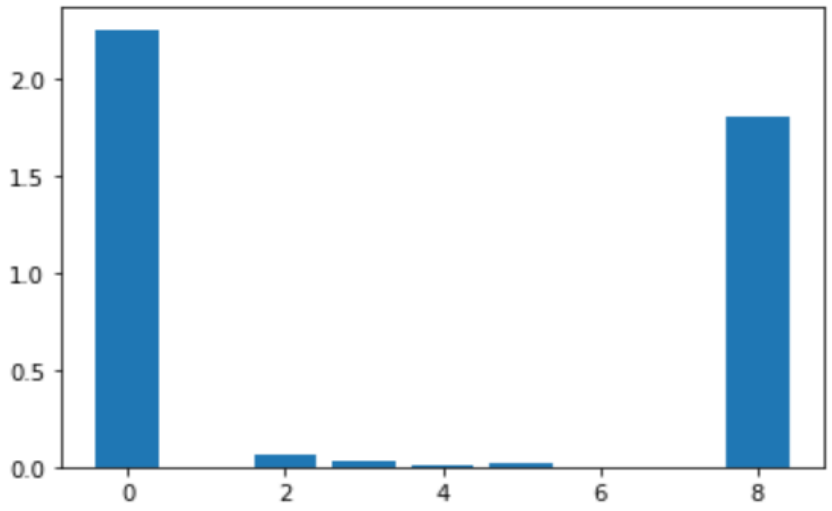


Figure 5.12: Features selected through Permutation for KNN

6. SALib- Sensitivity Analysis Library

SALib is an open-source library written in Python for performing sensitivity analysis (Herman and Usher, n.d.). Its robust toolbox offers various SA methods for calculating sensitivity indices. Although it provides several processes like Sobol Sensitivity Analysis (Sobol′, 2001a),(Saltelli, 2002), Morris method (Morris, 1991b), and Fourier Amplitude Sensitivity Test (Cukier et al., 1973), the main focus will be on this work's Sobol and Morris method.

SALib library provides a wide range of SA analysis methods that can be selected according to the desired output. It does not provide aid directly with implementing computational or mathematical models. The input samples can be generated using the "sample" function and used for the input for the model to be tested. The model can be implemented, and the output result is

saved. After that, the model's output can be used with the "analyze" function for sensitivity analysis.

. It is a great toolbox to understand the effect of each input, which can help design the model accurately. The main four steps that this library follows are:

1. Identifying model inputs
2. Generating input samples by using the sample function
3. In this step, it evaluates the model of the user and generates outputs
4. Calculates sensitivity indices by using the value of outputs

There are two main reasons for using the SALib toolbox during this work. Firstly, identify how the sample size of input parameters will affect the prediction of sensitivity indexes. It will help in future investigations using ML models and will help with using the correct sampling size. Secondly, a comparison will be drawn between two SA models and their predictions. Only Sobol and Morris methods are chosen for the analysis due to the limited time frame for completing this thesis work. Also, all the other SA methods follow similar concepts used; therefore, these two were selected for analysis.

As the SALib toolbox provides a visualization of the effect of each input on the output, it also helps eliminate inputs and irrelevant inputs. It makes the process of decision-making and modeling more accurate.

Three mathematical functions and four ML models are considered to study the working of this library. In each case, further investigation of sampling size, input boundary range, and use of a particular SA model will be made.

6.2 Sobol Analysis:

In the Sobol method following outputs can be obtained:

1. First-order sensitivity index: Effect of single input on a target output, represented as S1
2. Second-order sensitivity index: Effect on output by the interaction of two input parameters, represented as S2
3. Total sensitivity index: Combination of a first and second-order index in case of each input, represented as ST

It calculates the impact of each input on output, the interaction of two inputs, and the total effect of the inputs on output.

6.2.1 Case 1: Use of Ishighami function

SALib provides a built-in test function that can is used to understand this library's step-by-step working before implementing one's own mathematical or computational models. The test function in Equation (6.1) is the Ishigami function (Ishigami and Homma, 1990).

$$f(x) = \sin(x_1) + \text{asin}^2(x_2) + bx_3^4 \sin(x_1) \quad (6.1)$$

Ishigami function has three independent variables and follows a uniform distribution. It is a test case for sensitivity analysis because of its non-linear and nonmonotonic behavior. It can be seen from the Equation that parameters x1,x2, and x3 affect the output individually, and interaction between them also affects the output y. In this case, the initial value for a was set equal to 7 and 0.1 for b by default.

First, the Sobol method was used to analyze the sensitivity indices. Following are the primary four steps performed in order:

1. Defining models input parameter: In this case, x1, x2, and x3 were defined in a uniform distribution, with a range of -π to +π set for all three parameters.
2. Generating samples using Saltelli sampler in the case of the Sobol method. The satellite sampler uses the formula N*(2D + 2), where N is the number of sample sizes we can select, and D is the number of input parameters; in this case, it is 3.
3. Next, we run the test model.
4. In the end, sensitivity indices are calculated using the "analyze" function.

Figure 6.1 showed the values for sensitivity indices when N was 1024 initially; according to the user manual of the SALib library, N needs to be set as a number equal to 2^n. It also states that setting a value greater than 1000 ensures better results.

```
           ST   ST_conf
x1   0.555860  0.091326
x2   0.441898  0.045446
x3   0.244675  0.029285
           S1   S1_conf
x1   0.316832  0.062538
x2   0.443763  0.049478
x3   0.012203  0.057422
                 S2   S2_conf
(x1, x2)  0.009254  0.079117
(x1, x3)  0.238172  0.093764
(x2, x3) -0.004888  0.061966
```

Figure 6.1: Sensitivity indices for Ishigami function using Sobol method

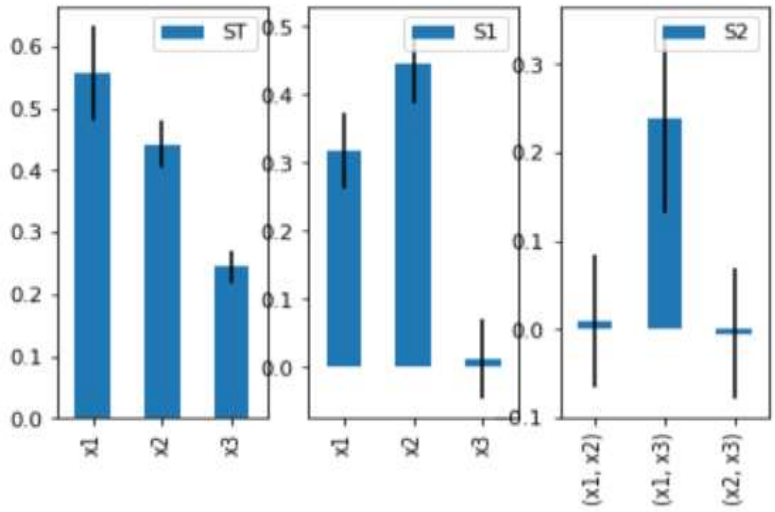


Figure 6.2: Bar plot showing results for sensitivity indices for Ishigami function

"S1" shows first-order indices, "S2" represents second-order indices, and "ST" offers total sensitivity indices. In comparison, "ST," "ST1", and "ST2" are used to store confidence intervals with a confidence level of 95%.

From Figures 6.1 and 6.2, it was observed that x2 has the highest first-order sensitivity index. It means x2 is the most sensitive input. Since x2 is multiplied by the term a, set to 0.7, this multiplication factor also makes x2 influential. Nevertheless, x1 has the most significant total sensitivity index; x1 has the highest effect on output y. The first order index only represents the individual effect of input parameters, but the total sensitivity index represents both individual and combined effect of input parameters on output. By combined parameter, it means interaction with other input parameters as well.

If we look back at the Equation of Ishigami Equation 6.1, it can be seen that x1 is being multiplied by x3; therefore, it is present twice in the Equation. Once x1 is present individually, and in the other half of the Equation, it is multiplied by x3, making it the most influential input parameter. However, it is hard to predict the sensitivity of x2 and x3 just by looking at the

Equation. It shows that the behavior of input parameters can be unpredictable, and sensitivity analysis can give a good insight into the importance of input parameters.

Another thing worth noticing in Figure 6.1 is that the value for the second-order sensitivity index for x2-x3 interaction is negative; this error can occur if the sample size is small. The sample size was increased five times to see the effect on results. The following results were observed.

```
          ST   ST_conf
x1  0.556875  0.043422
x2  0.441304  0.019915
x3  0.242614  0.012947
          S1   S1_conf
x1  0.315609  0.025653
x2  0.442477  0.023594
x3  0.001502  0.022103
                S2   S2_conf
(x1, x2)  0.000681  0.031936
(x1, x3)  0.240978  0.039996
(x2, x3) -0.001224  0.027333
[0.31560883 0.44247699 0.00150237]
```

Figure 6.3: Table with values for sensitivity indices when the sample size is increased to 5 times

It can be seen that increasing the sample size five times than the original size does not remove this error; this is because the sample size is still tiny to give accurate results. Saltelli et al. (2004) describe in the book that the Sobol method is computationally expensive as it requires to be iterated over an ample sample space. The sample size is increased by 10 to observe new results in Figure 6.4.

```
           ST   ST_conf
x1  0.558332  0.026057
x2  0.442808  0.012573
x3  0.243287  0.008307
           S1   S1_conf
x1  0.317753  0.019271
x2  0.442066  0.015024
x3  0.002669  0.017687
                 S2   S2_conf
(x1, x2)  0.000324  0.023453
(x1, x3)  0.240069  0.032153
(x2, x3)  0.000625  0.021387
[0.31775266 0.44206595 0.00266892]
```

Figure 6.4: Values for sensitivity indices with a sample size ten times greater than original

Increasing the sample size ten times greater than the original removes the error as the value has changed to a positive number. However, this also increases the computational cost and decreases sensitivity index confidence values.

6.2.2 Case 2: Testing sin wave function

For the second case, a compound sin function was implemented.

$$y = \frac{1}{2}\pi sin\frac{x_1}{Y} + \frac{1}{4}\pi\, sin\frac{x_2}{Y} + \frac{3}{5}\pi\, sin\frac{x_3}{Y} \cdot sin\frac{x_1}{Y} \tag{6.2}$$

For Equation (6.2), Sobol SA was performed. The sample size was 1024, and output was obtained in the first-order, second-order, and total sensitivity indexes. The limits for each variable were set uniformly from -1 to 1 because the range of the sin Equation is between [-1,1]. The following Figure 6.5 shows the results.

```
          ST   ST_conf
x1  0.805472  0.058591
x2  0.195279  0.017258
x3  0.023808  0.002749
          S1   S1_conf
x1  0.781206  0.062317
x2  0.195194  0.034050
x3  0.000193  0.013245
                S2   S2_conf
(x1, x2)  0.000074  0.086408
(x1, x3)  0.023464  0.084111
(x2, x3) -0.000080  0.050844
[7.81205758e-01 1.95193959e-01 1.92997388e-04]
```

Figure 6.5: Output results for sin Equation

From the results, it can be observed that x1 is the most significant input variable. Equation (6.2) can verify it because x1 is present as an individual parameter and interacts with the variable x3. However, the results for the First-order index are pretty interesting. By looking at the Equation, it is hard to interpret the individual effect of each input variable on output. Nevertheless, it is pretty easy to visualize the results through such analysis. The following Figure 6.6 shows the results:

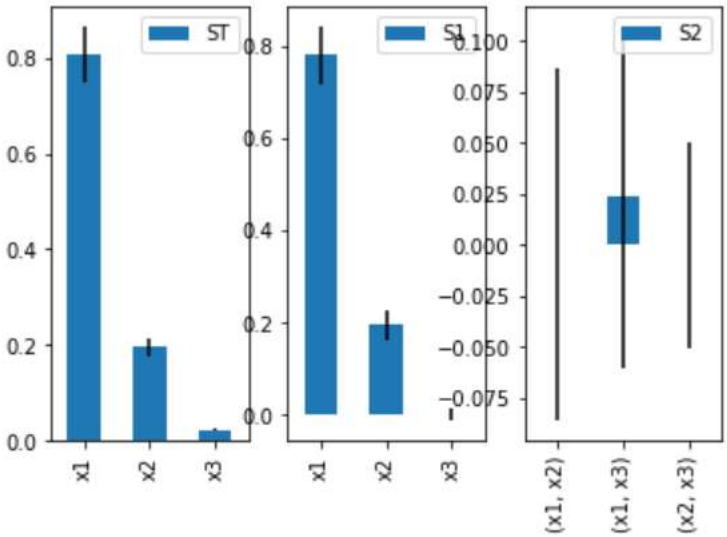


Figure 6.6: Graphical results for case study 2

If the results from Figure 6.6 are considered, it is visible that there is the exact error encountered in case 1. There is a negative value for the second-order sensitivity index in the case of x2 and x3 interaction. Again, it is proved from the last example that changing the sample size can significantly impact outputs. However, if the graphs are observed closely, there is no relationship between x1 and x2 and x2 and x3 for the second-order index. Thus graphical representation provides better information about the sensitivity indices.

6.2.3 Case 3: Exponential function on model coefficients

For the third problem, the exponential function was considered. In this case, not all variables in Equation 6.3 contribute to sensitivity analysis.

$$y = a.e^{x^2} + b \tag{6.3}$$

In Equation (6.3), a and b affect the output, and x is initially varied over the range [-1,1]. As expected, a and b are the only parameters affecting the output sensitivity. They are varied over different ranges to study the depth of sensitivity analysis for exponential functions. Firstly, the range of both a and b is set for [-1,1], and x is varied between -1 and 1. Figure 6.7 shows the result for a and b. It is seen that initially, the value for b is more significant than a. However, as time passes, there is an exponential decay in values of b, whereas values of a grow exponentially for the values of x after 0. The values for the sensitivity index are shown in Figure 6.8.

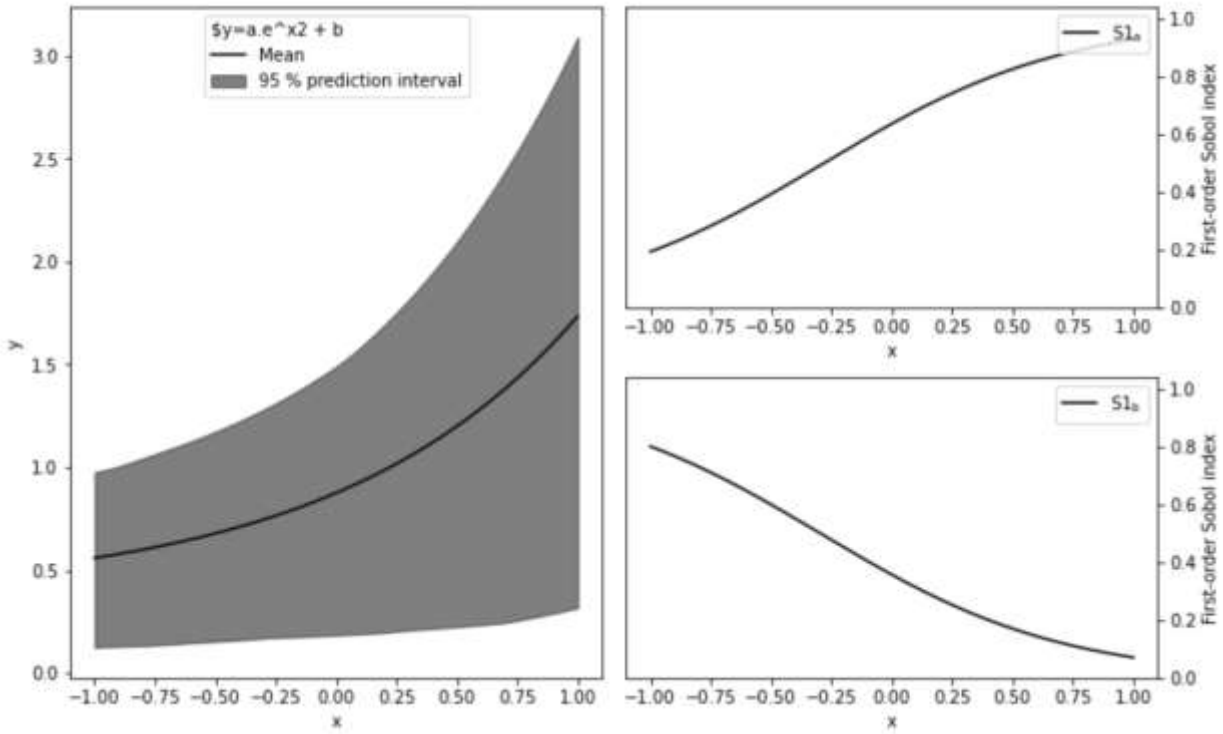


Figure 6.7: First-order sensitivity indices graph for an exponential function

```
[[0.11883889 0.87682153]
 [0.11925795 0.87639734]
 [0.11967829 0.87597187]
 ...
 [0.87725629 0.11950306]
 [0.87768238 0.11908334]
 [0.87810719 0.11866489]]
```

Figure 6.8: First-order sensitivity indices values for an exponential function

Because the sensitive variables are set to be a and b, x is not considered in the sampling process. Therefore, all three variables were varied to study the effect on output in the following parts. A range was set between [-1,1], and b has a range of [0,1], whereas the boundary for x remains the same. The following graph in Figure 6.9 is obtained, which is different from the default graph. In this case, the mean is constant, unlike the default function where the mean was a rising exponential function as expected.

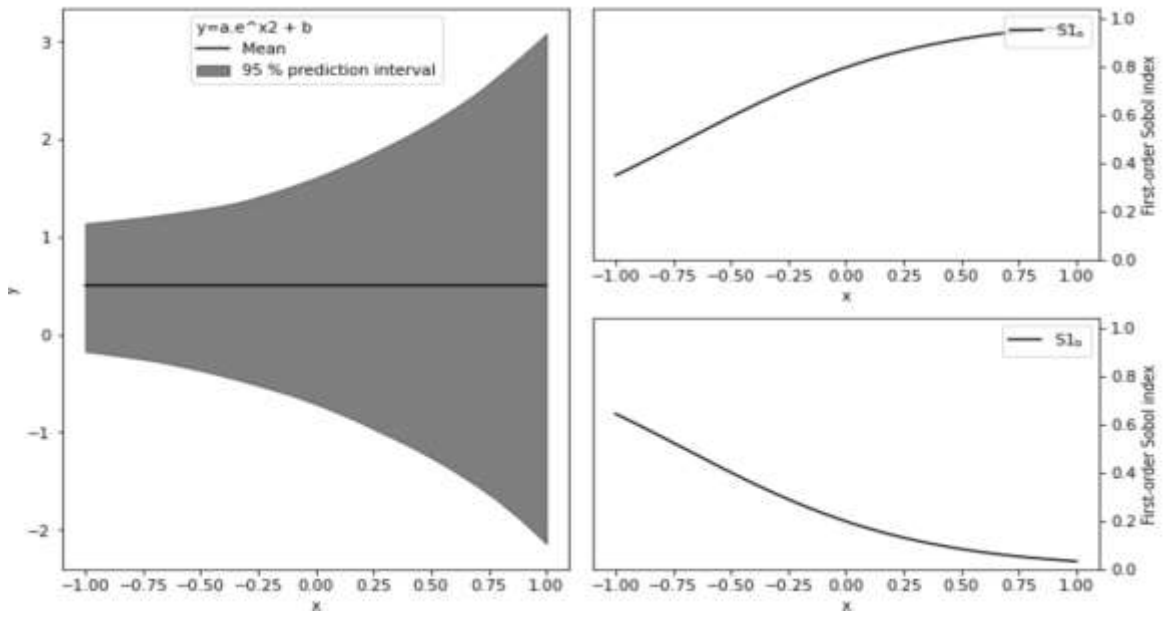


Figure 6.9: First-order sensitivity indices graphs for an exponential function; with a range of a between [-1,1] and b in the range of [0,1]

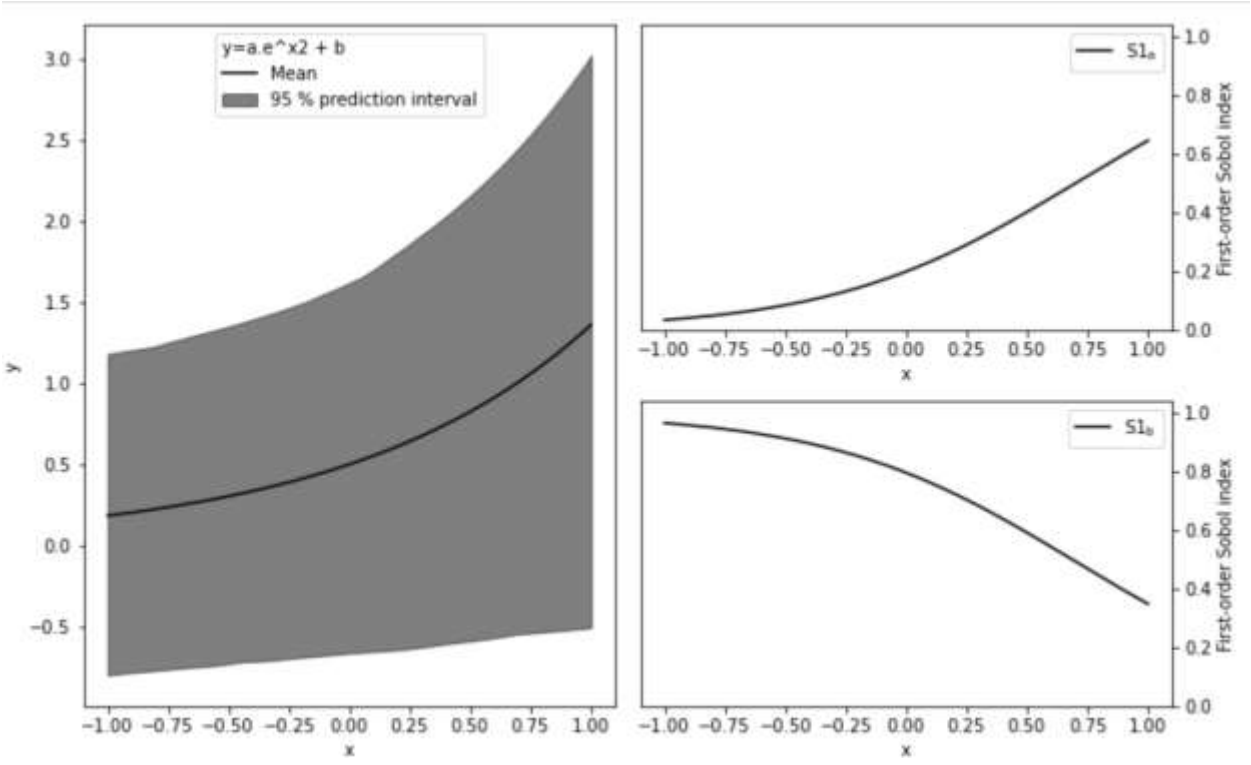
For the sensitivity index for a and b, a still shows exponential growth, and b shows exponential decay. However, the initial values of both variables have changed significantly, as seen in Figure 6.10. The initial value of a has increased while b decreases.

```
[[0.34951601 0.64470303]
 [0.35042334 0.64379374]
 [0.35133175 0.64288339]
 ...
 [0.9657059  0.032888  ]
 [0.96583696 0.03276102]
 [0.96596753 0.03263452]]
```

Figure 6.10: First-order sensitivity indices values for an exponential function; with a range of a between [-1,1] and b in the range of [0,1]

For the second scenario, the range for b was set to [-1,1] and [0,1] for both a and x. In this case, the mean of the function shows an exponential rise again, as seen in Figure 6.11. Both a and b show the same exponential trends, but now b has greater starting values than a. It is

understandable because as we increase the range of one variable, it now has more values to accommodate and thus has greater starting values.




```
[[0.03268237 0.96455323]
 [0.03280906 0.96442285]
 [0.03293622 0.96429198]
 ...
 [0.64382604 0.35081735]
 [0.64473772 0.34991027]
 [0.64564834 0.34900427]]
```

Figure 6.11:First-order sensitivity indices graphs for an exponential function; with a range of b between [-1,1] and a in the range of [0,1]

For the last case, both a and b were fixed, and the range of x was shifted to see the changes in function. As seen in Figure 6.12, it means that the function is constant rather than an exponential rising. Also, the function overall shows both exponential rise and decay function. From Figure 6.13, the starting points of both a and b are the same.

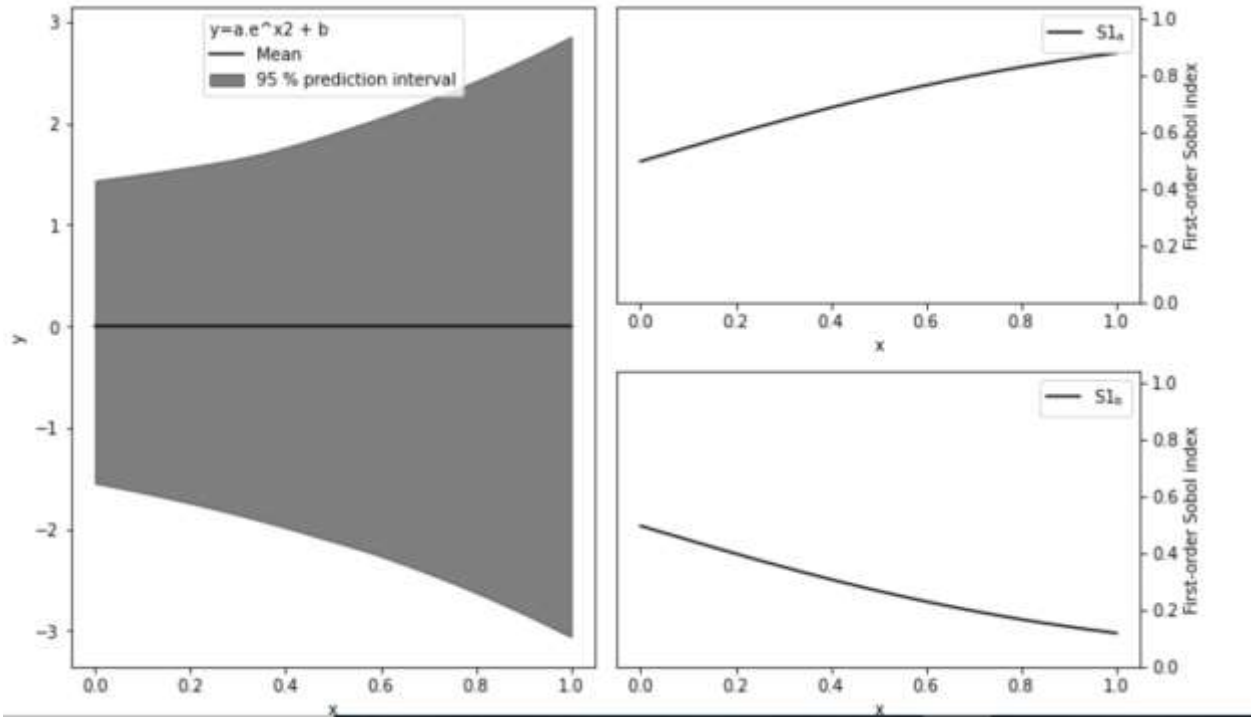


Figure 6.13: First-order sensitivity indices graphs for an exponential function; by altering the value of x

```
[[0.49745084 0.49672251]
 [0.49794879 0.49622529]
 [0.49844674 0.49572807]
 ...
 [0.87768238 0.11908334]
 [0.87789495 0.11887396]
 [0.87810719 0.11866489]]
```

Figure 6.14: First-order sensitivity indices values for an exponential function; by altering the value of x

Therefore, the SALib tool helps in a detailed study of the effect of each parameter. Moreover, one can learn about the effects of different scenarios by making small changes to the model.

6.3 Morris Method:

Morris method in SALib library return results of sensitivity analysis in the form of mean and standard deviation. The outputs in this method are:

1. mu: mean elementary effect
2. mu_star: absolute of mean elementary effect
3. sigma: standard deviation of elementary effect

Unlike the Sobol method, the Morris method calculates global sensitivity using the OAT method and randomly varying inputs. Identification and ranking of the essential variables are based on the difference computed between a pair of model simulations (Franczyk, 2019).

One drawback of the mean elementary effect is that it does not incorporate negative values and cancels negative with a positive value. Therefore, in the revised Morris method, the absolute mean value is considered, which considers both negative and positive values while looking at the importance of each input parameter. Sigma identifies factors that interact with one another or have a non-linear influence (Saltelli, 2004).

6.3.1 Case 1: Ishigami Function

When the Ishigami function Equation 6.1 is simulated using the Morris method for a 1000 input sample size. x1 has the highest mean elementary effect, and the value for the absolute mean is the same as the mean elementary effect. The absolute mean elementary effect is highest for x2,

with the most negative values associated with the mean. X3 has the highest standard deviation value, which shows the highest interaction with another input variable.

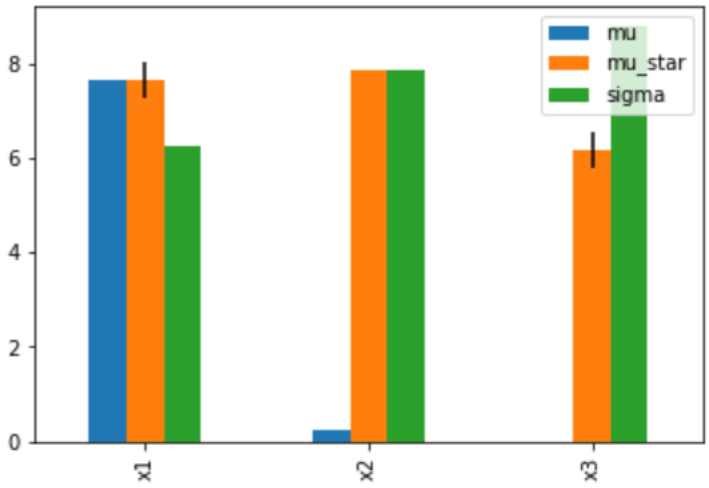


Figure 6.15: Results of Morris method for Ishigami function

For the same Ishigami function, the input sample size was decreased by ten times and increased by ten times. Figure 6.16 shows results when the sample size was decreased, and Figure 6.17 shows results from an increased sample size.

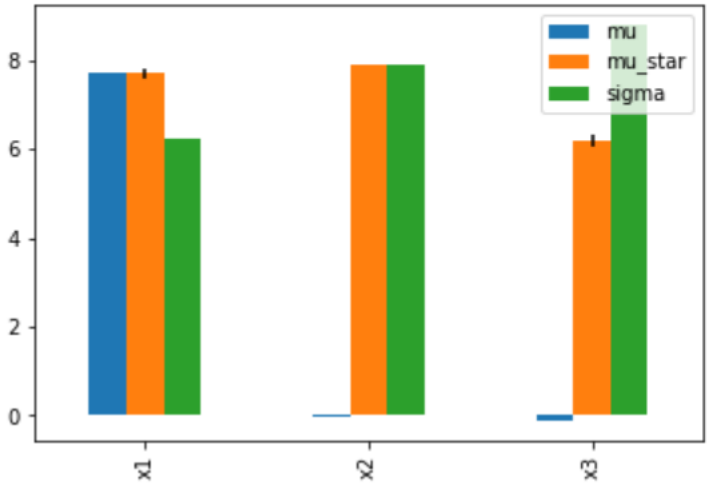


Figure 6.16: Results of Morris method for Ishigami function when the input sample size is decreased 10x times

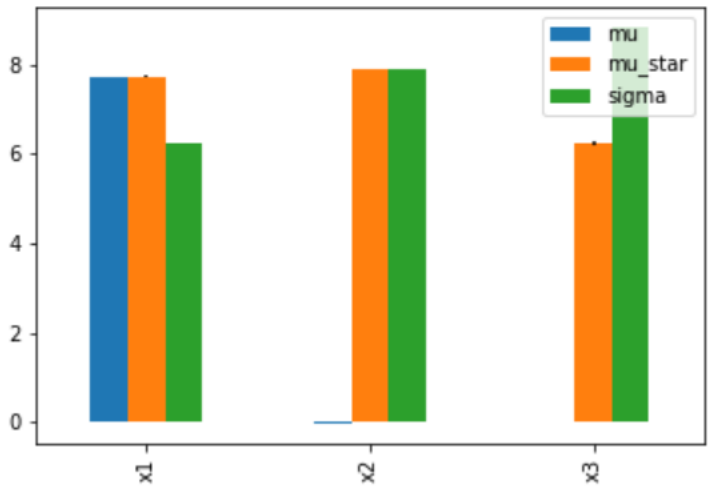


Figure 6.17: Results of Morris method for Ishigami function when the input sample size is increased 10x times

Figure 6.16 shows that when the input sample size is decreased, the mean values for x3 appear and are negative. However, all other values remain the same. On the other hand, when the input sample size is increased, the negative values are decreased, but overall values for each parameter remain the same. It can be concluded that sample size does not affect the results obtained from the Morris method. It is possible because the absolute mean considers all negative and positive values associated with the final results, so changing the sample size does not affect it.

6.3.2 Case 2: Sine Function

For the sine function from Equation (6.2), x1 has the highest mean and absolute mean elementary effect, which means it is the most crucial input variable. Figure 6.18 shows Morris method parameters for Equation (6.2).

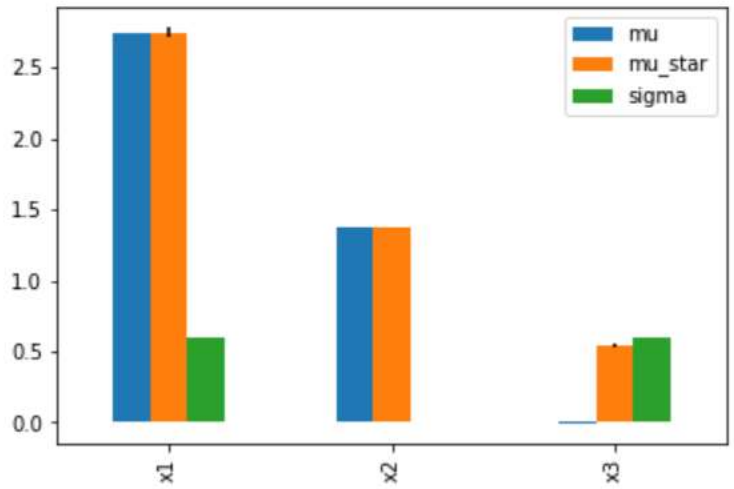


Figure 6.18: Results of Morris method for the sine function

x1 and x3 have the same values for standard deviation, representing interaction with each other. x2 has zero standard deviation, meaning it does not interact with any other parameter. X1 has the highest mean and absolute mean value because it is multiplied by 0.5 in Equation (6.2).

6.3.3 Case 3: Exponential Functions of model coefficients

To implement an exponential function in Equation (6.3), a,b, and x are considered sensitive variables. It can be seen from Figure 6.19 that “a” is the most influential input variable, and “a” and “x” have values for standard deviation showing relation with each other. The standard deviation value for “x” is slightly higher than “a.” It is possible because x also interacts with “b.”

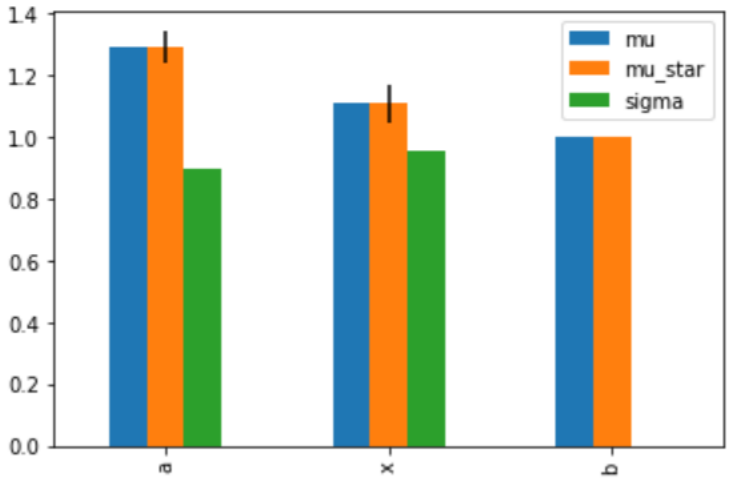


Figure 6.19: Results of Morris method for the sine function

6.4 Sensitivity Methods using Machine Learning models

After observing the working of the Sobol and Morris method in Sections 6.2 and 6.3, it was possible to integrate Machine Learning models with the Sensitivity Analysis method. Both SA methods are used with Random Forest, Gradient Booster, K-Nearest Neighbour, and Local Cascade Ensemble regressors.

The steps needed to implement Sensitivity methods using SALib with ML models are following.

- Clean the raw data by removing any outliers or noise
- Normalize the dataset using Min Max scaler
- Split the dataset into training and testing data
- Run the ML model for train and test the dataset
- Create a dictionary of input parameters and define boundary limits
- Use sample function from the SALib library to generate input samples
- Normalize the input samples using the MinMax scaler
- Use predict function of ML model upon normalized input parameters

- Import analyze function for corresponding SA method from SALib
- Run the analyze function over the ML model and input sample space
- Visualize the results

7. Emukit

In the previous two chapters, several methods were discussed which can be selected for sensitivity analysis. SALib tool is powerful but does not provide a wrapper class to facilitate the calculations. Integrating machine learning data with the SALib library becomes a new modeling problem. It is easy to implement a less complex computational and mathematical model with SALib. However, it is hard to integrate without a wrapper class when the dataset becomes more extensive and complicated. For the following part, feature selection was tested for sensitivity analysis. Although feature selection proved helpful, there were still two main problems. First, all methods gave different results, which is still a useful pre-process method. After reviewing a lot of literature and research, another toolbox was discovered, which addresses both of these issues.

Emukit is a python decision-making toolkit created by Paleyes et al. (2021). It contains all the tools required to measure uncertainty in a single toolbox. Emukit is versatile and allows users to integrate Python-based models. Emukit acts as a surrogate model or emulator in uncertainty calculation, where a study of variable inputs is required for fixed output. An emulator or surrogate model is known as the model of the model built on the top of the original model. Surrogate models statistically relate input data to output data acquired by the complex system simulation (Davis et al., 2017).

7.1 Features of Emukit

Emukit provides the following features:

1. Multi-fidelity emulation
2. Bayesian optimization
3. Experimental design/Active learning

4. Sensitivity analysis
5. Bayesian quadrature

7.1.1 Multi-fidelity emulation

Multi-fidelity emulation is required when both high fidelity and low fidelity data are present, and it is essential to combine both to get a high fidelity result. High fidelity data is one of high accuracy but is computationally expensive and not readily available. Low fidelity data usually comes from simulations or approximations (Chen et al., 2022).

There are few systems where it is impossible to get high-accuracy data like climate and environmental scenarios. Therefore, simulations or physics-based models are used to et low fidelity data. Low-fidelity model (e.g., a simplified physics approximation, a reduced model, a data-fit surrogate) approximates the same output quantity as the high-fidelity model (Peherstorfer et al., 2018). Later this limited high fidelity data is used with low fidelity data to get more accurate results.

Emukit provides an emulator environment where multi-fidelity data can be processed using the Gaussian process (Rasmussen, 2004).

7.1.2 Bayesian optimization

Shahriari et al. (2016) and Frazier (2018) have described the working of Bayesian optimization. Bayesian optimization is a sequential decision-making method for determining the best value for objective functions that are difficult to evaluate. Bayesian optimization can optimize any black-box function (Agnihotri and Batra, 2020). Bayesian optimization is used when hyperparameters are present in huge quantities, and it is essential to choose the best-performing ones.

Emukit provides a built-in function for Bayesian optimization, which can be used with the Gaussian process (Rasmussen and Williams, 2006).

7.1.3 Experimental Design

Experimental design or active learning is a type of uncertainty reduction method. It is used when sizeable unlabelled data is required, and labeling it is an expensive job. This method proposes labeling the point whose model uncertainty is the highest (Agnihotri and Batra, 2020).

In experimental design, it is decided that a model must be evaluated for higher accuracy at which input space. The best input points are selected using variance, which is the uncertainty.

7.1.4 Bayesian Quadrature

O'Hagan (1991) proposed that the Bayesian quadrature is an active learning method. Bayesian quadrature returns a posterior distribution of the integral value to be evaluated, and it can be used in successive iterations to facilitate decision-making. Bayesian quadrature comes in handy when evaluating an integrand is computationally expensive. For further information on the topic, O'Hagan (1991), Bretthorst (1990), and van den Bos et al. (2020) can be consulted.

7.1.5 Sensitivity Analysis

The methods discussed from Section 7.1.1 to 7.1.4 are out of the scope of this work, and the built-in function of interest is sensitivity analysis. The literature review of sensitivity analysis is discussed in detail in chapter 2. Emukit allows global sensitivity analysis using the variance-based method (Saltelli et al., 2010). The outputs obtained are "main effects," first-order sensitivity indices, and "total effect," which is the total sensitivity index.

Emukit performs sensitivity analysis using Monte Carlo with Saltelli estimators (Sobol′, 2001b). Implementing sensitivity analysis through Emukit has been explained visually in Figure 7.1 using a flowchart.

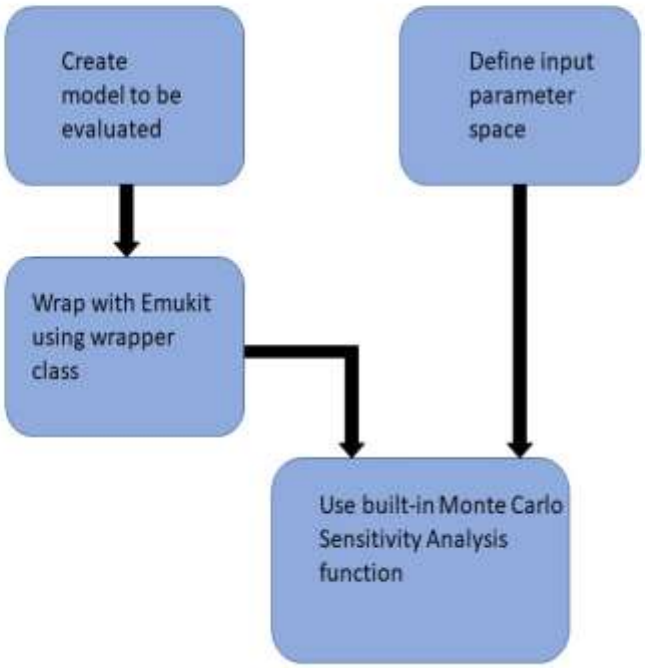


Fig 7.1: Workflow of implementing Sensitivity Analysis using ML models with Emukit

Emukit allows a simplified way of implementing a much harder implementation of sensitivity analysis.

- The first step is to define a model that is to be tested. This model can be mathematical or computational.
- Once the test model is ready, it can be wrapped with Emukit using a built-in wrapper class.

- To implement Monte Carlo with the Saltelli estimator, it is vital to define the input space. In this step, the domain for each variable is defined.
- After these two steps are completed, the model and input space can be plugged into the "MonteCarloSensitivity" function. This function calculates first-order and total sensitivity indices using Monte Carlo steps combined with Saltelli estimation.

Two models are integrated with Emukit to perform sensitivity analysis. The first model used is a mathematical model, the Ishighami function Equation (), as used in Chapter (salib). Moreover, the second model is a computational model which uses the Volve dataset to evaluate the sensitivity index, with ROP being the targeted output variable.

Ishigami function gives us insight and comparison with the SALib library. It helps to evaluate the results obtained from both toolboxes.

7.2 Case study1: Implementing Ishigami function

Ishigami function is implemented in Emukit through

- Sobol Method
- Monte Carlo estimation
- The Gaussian process

The aim of performing these different methods is to compare the performances of the traditional method, Monte Carlo estimation, and the Gaussian process. Figure 7.2 shows the First-order indices comparing the Sobol method and Monte Carlo estimation, and Figure 7.3 shows the First-order indices using the three methods.

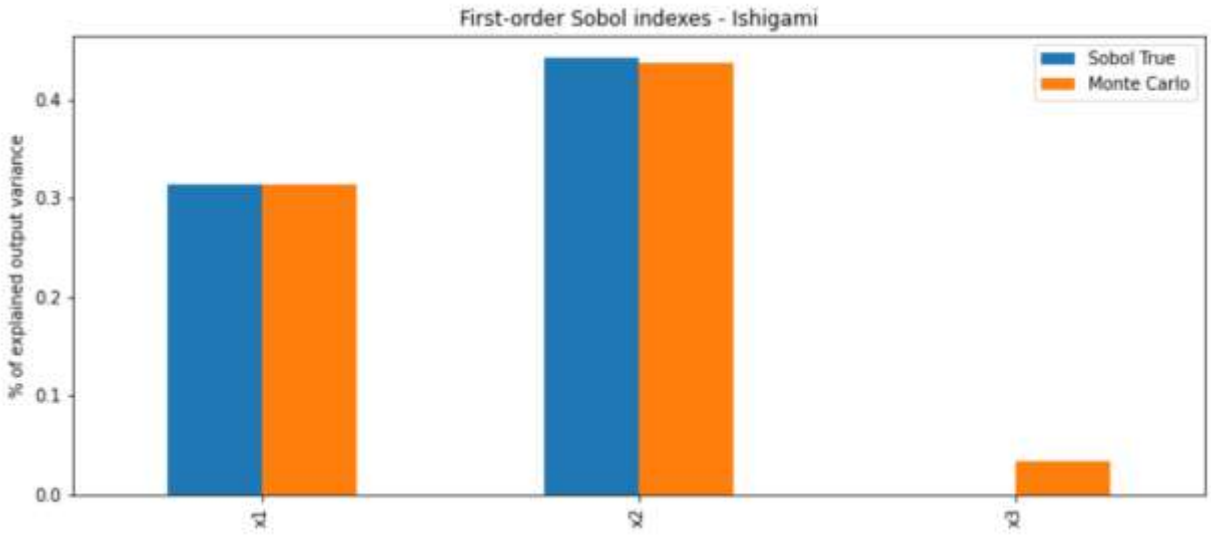


Figure 7.2: Comparison of First-order indices of Ishigami function using Sobol method

Figure 7.3: Comparison of First-order indices of Ishigami function using Sobol method, Gaussian Process method, and Monte Carlo simulation

Like results obtained for the Ishigami function from the SALib tool, x2 is the most significant input variable when calculating the first-order indices. Now we shall have a look at the total sensitivity index.

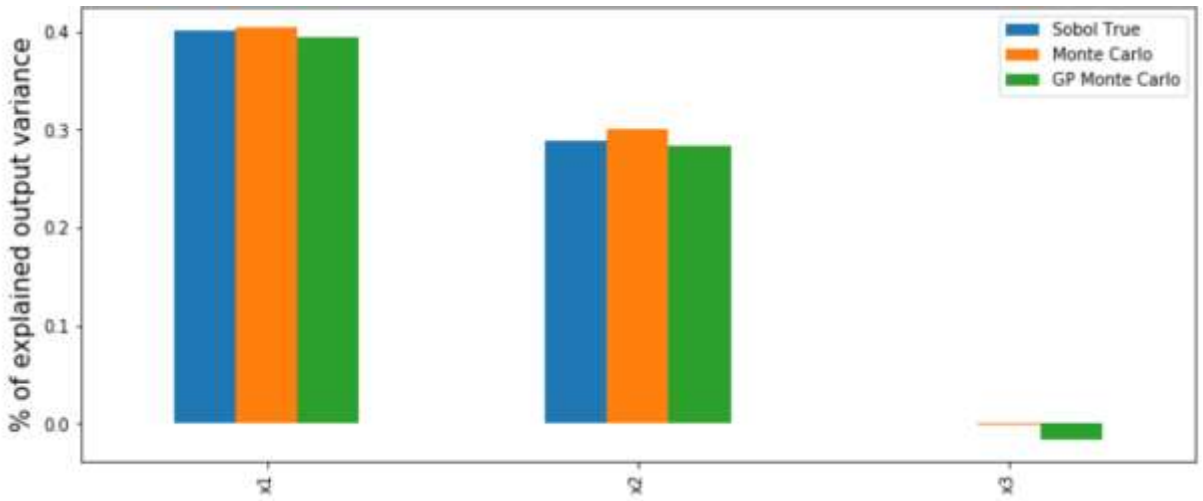


Figure 7.4: Comparison of First-order indices of Ishigami function using Sobol method, Gaussian Process method, and Monte Carlo simulation

In the total sensitivity index, x1 is the most influential input variable. It is because it also interacts with other input variables, increasing its significance.

There is a little difference in the result obtained from the Gaussian process. It is because only 500 samples were considered, whereas 10000 steps were considered for Monte Carlo. It also shows that the Gaussian process can give high accuracy at small sample size, but the step size was kept low due to computational cost.

7.3 Case study 2: Implementing Sensitivity Analysis for Volve Dataset

Emukit provides all sufficient tools to implement sensitivity analysis. When the number of input increase, it becomes hard to implement traditional sensitivity analysis. So it was essential to develop a sensitivity analysis method that can be implemented when there are a large number of input variables, and it can work with ML models. Emukit provides all the necessary functions, but integrating it with different datasets is the main task.

The following steps were used to implement a sensitivity analysis model using Emukit.

- Split the dataset into test and train dataset

- Separate target output variable and input variables
- Train the dataset using GP Regressor
- Wrap it with Emukit using the wrapper function
- Define a sample space for each input variable
- Combine the input sample space and regression model through a decision loop
- Set sample size for Monte Carlo and feed the decision loop to its function
- Results are obtained as a first-order index and total sensitivity index

The results are discussed in Chapter 8.

8. Results & Discussion

In this section, we will observe the input parameters that are influential on the Rate of Penetration modeling. Results from three SA methods will be compared with each other and with the feature selection methods.

8.1 Sobol Method

To implement the Sobol method, the sample size for input parameter sampling was set to 1024. It is a computationally expensive method, therefore increasing the sample size could increase the processing time. The confidence level was set to 95%. Results for first-order, second-order, and total sensitivity indices were observed. For selection criteria, the input parameters having a percentage equal to or greater than 5% are considered important parameters.

8.1.1 Random Forest Regressor

Figure 8.1 shows the input parameter selected by the Random Forest regressor. The first-order sensitivity indices represent the effect of each input parameter with considering correlation. Therefore, it is possible that the number of parameters ranked as important is less than the parameters selected through the total sensitivity index. The input parameters selected based on input criteria mentioned in Section 8.1 are Measured Depth and Standpipe Pressure. It shows that both Measured Depth and Standpipe pressure are the most important parameters in the test dataset.

Figure 8.2 represents the graph with total sensitivity index values. In this case, Standpipe Pressure, Measured Depth, and Weight on the Bit are the most influential parameters. As

compared to the results from first-order sensitivity indices, WOB is the new addition as a selected parameter. This is because in the total sensitivity index measurements interaction with other parameters is also considered.

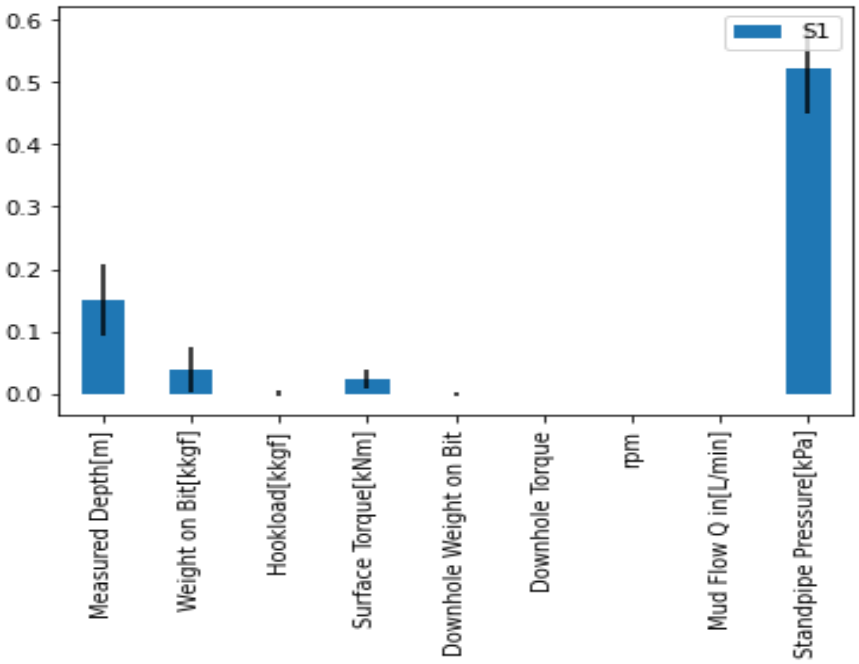


Figure 8.1: First-order indices obtained from Random Forest regressor

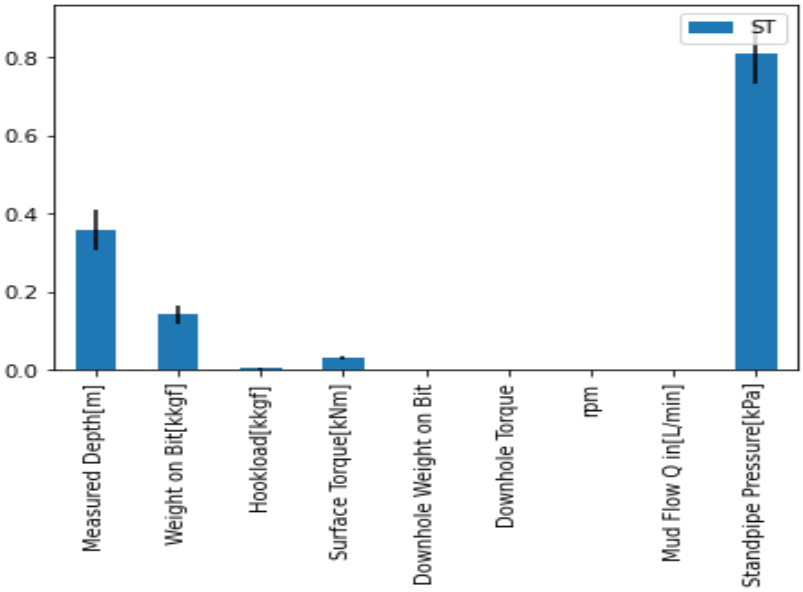


Figure 8.2: Total Sensitivity Index obtained from Random Forest regressor

Figure 8.3 shows the graph for interaction between all parameters with each other. The most significant relation is between Weight on Bit and Standpipe Pressure. The values from simulations show that there is no interaction between Speed and Downhole Torque and Speed and Standpipe Pressure.

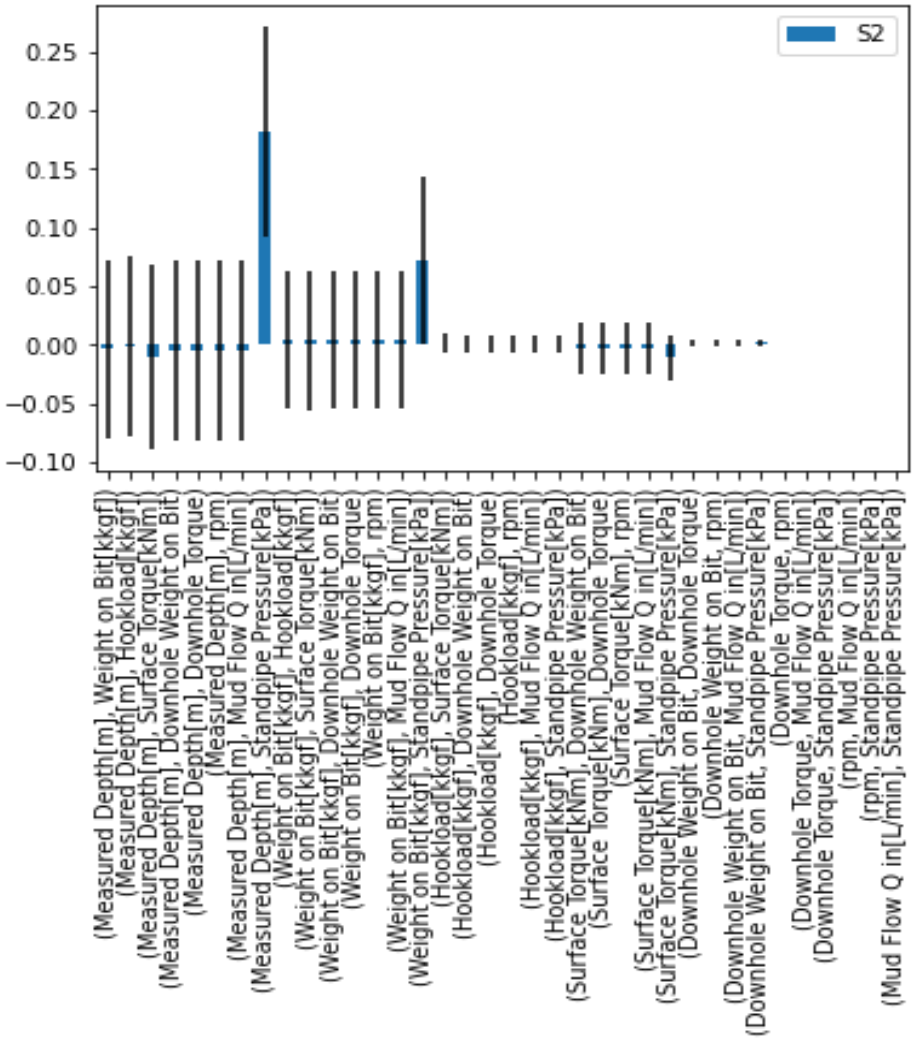


Figure 8.3: Second-order Sensitivity Index obtained from Random Forest regressor

8.1.2 Gradient Booster Regressor

In contrast to the Random Forest regressor, GBR predicted more input parameters to be important. The parameters selected were Measured Depth, Weight on the bit, Mud Flow, and Standpipe pressure.

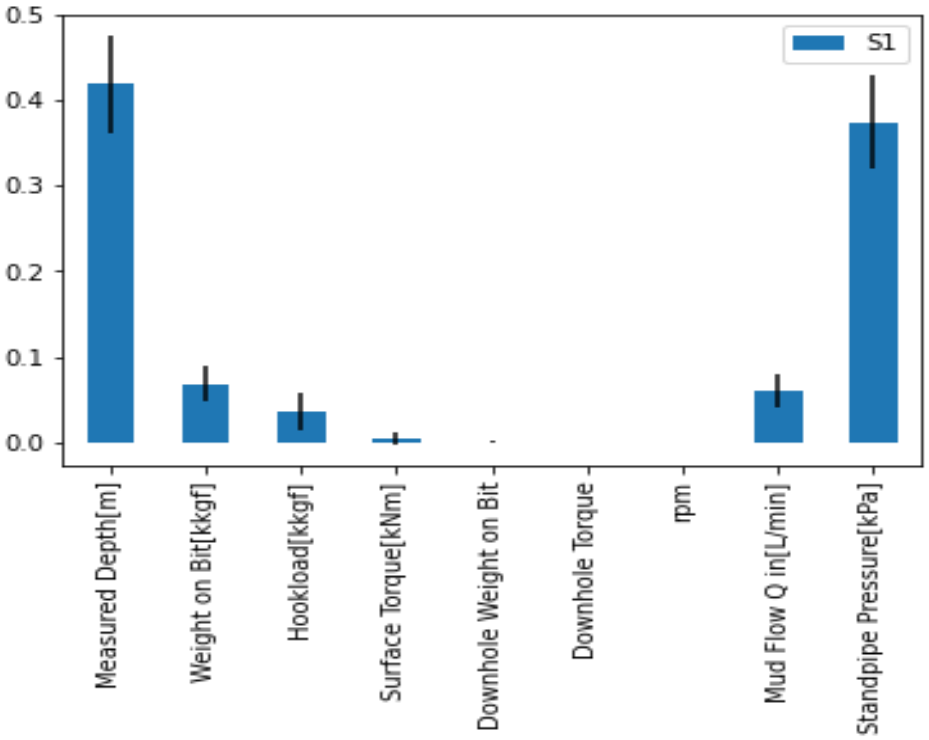


Figure 8.4: First-order indices obtained from Gradient Booster regressor

In the case of total sensitivity values, Measured Depth, Weight on the bit, Hookload, Mud Flow, and Standpipe pressure were chosen to be the most important. The graph with the values for the total sensitivity index is shown in Figure 8.5.

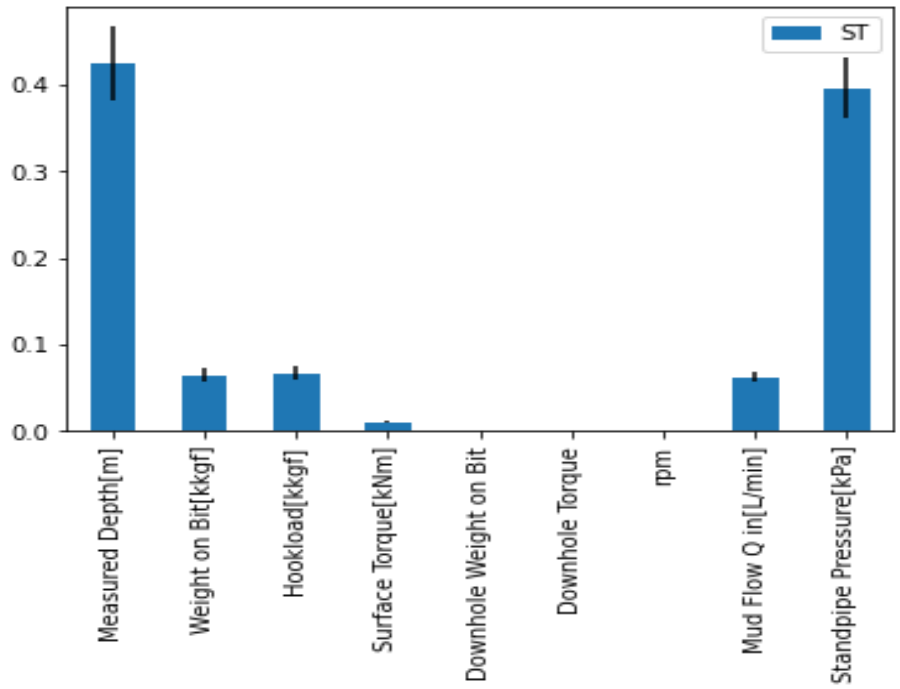


Figure 8.5: Total Sensitivity index values obtained from Gradient Booster regressor

When the values for second-order sensitivity indices were evaluated, no relationship between two input parameters was observed to be significant enough. As it can be seen from Figure 8.6

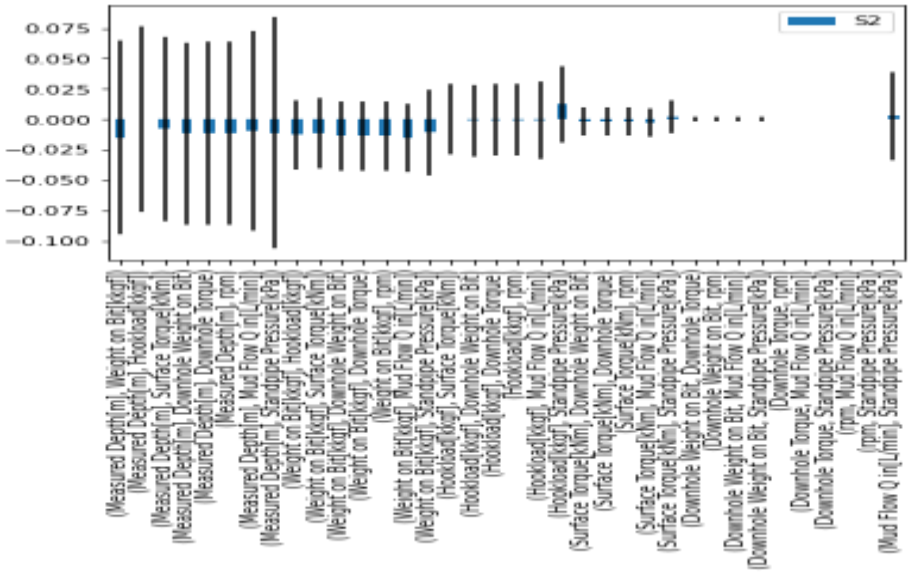


Figure 8.6: Second-order Sensitivity index values obtained from Gradient Booster regressor

8.1.3 K-Nearest Neighbour

KNN showed a different behavior than other ML models used in this work. This is since the other three ML models fall in the category of ensemble methods. For the first-order sensitivity index, Speed (rpm) had the highest value for variance nearing 62%. Interestingly, as seen in Figure 8.7 no other parameter was considered important.

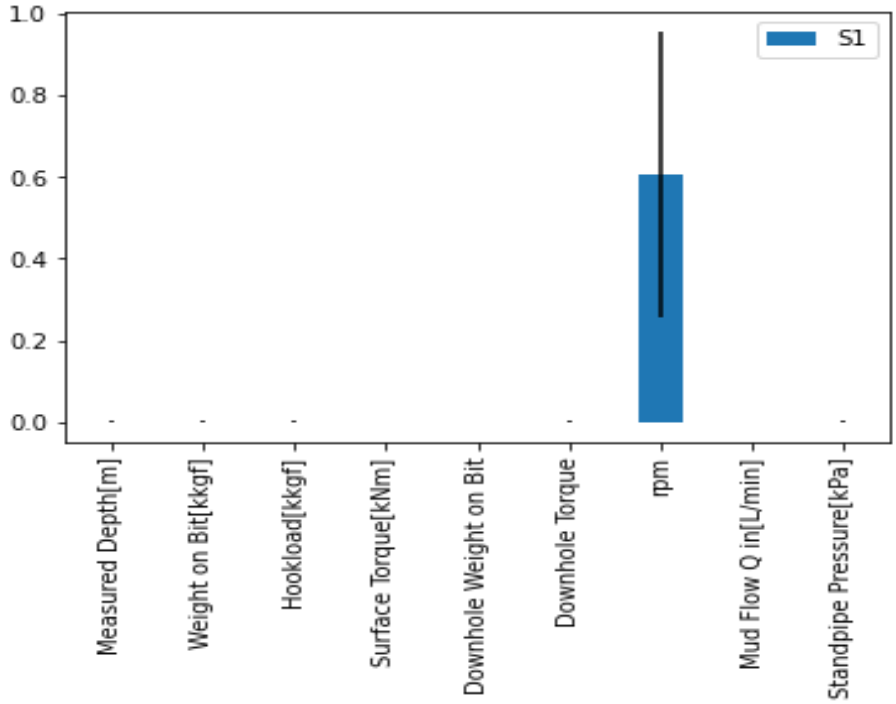


Figure 8.7: First-order indices obtained from KNN regressor

From Figure 8.8, the parameters selected based on total sensitivity index values are Speed (rpm), Weight on Bit, and Downhole Torque. The second-order sensitivity indices values are observed in the graph in Figure 8.9. The significant interactions were between Speed & Standpipe Pressure, Speed & Mudflow, Speed & Downhole Torque, Speed & Weight on Bit, and Speed & Measured Depth. The point worth noting here is that all the interactions have speed involved. Because of this, the total sensitivity index value of Speed is highest.

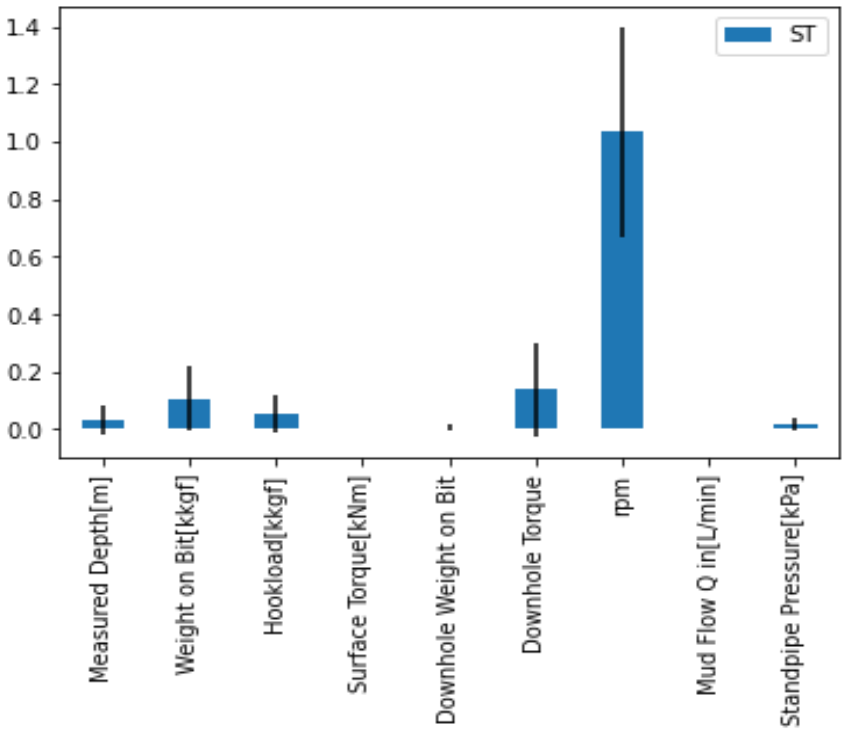


Figure 8.8: Total sensitivity index obtained from KNN regressor

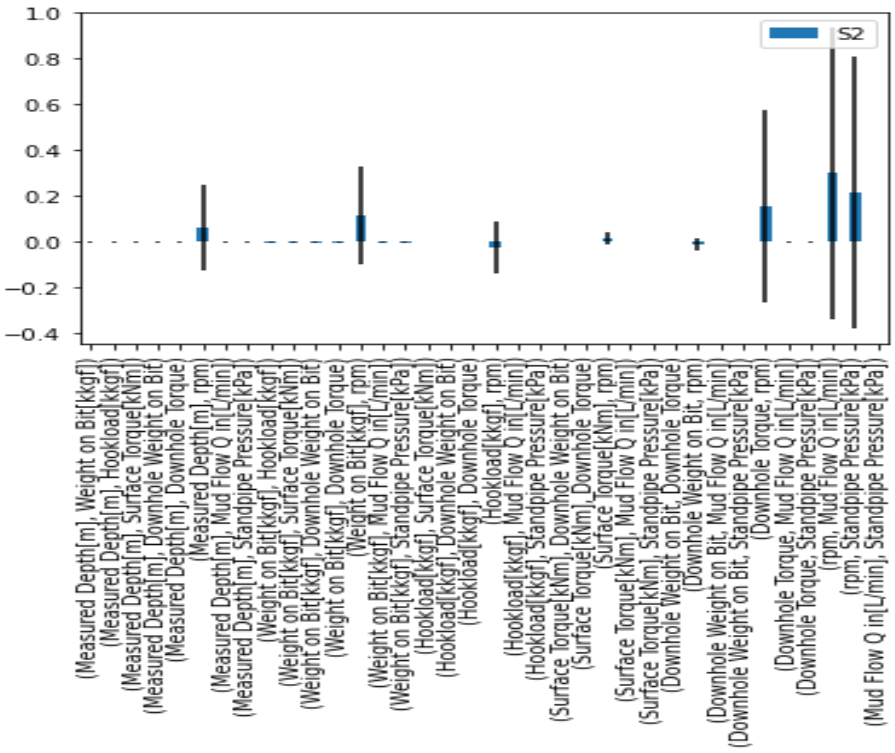


Figure 8.9: Total sensitivity index obtained from KNN regressor

8.1.4 Local Cascade Ensemble Regressor

The results obtained from the LCA regressor are a combination of Random Forest and Gradient Booster regressor. This validates the algorithm of Local Cascade Ensamble which works on the principle of RF and GB regressors.

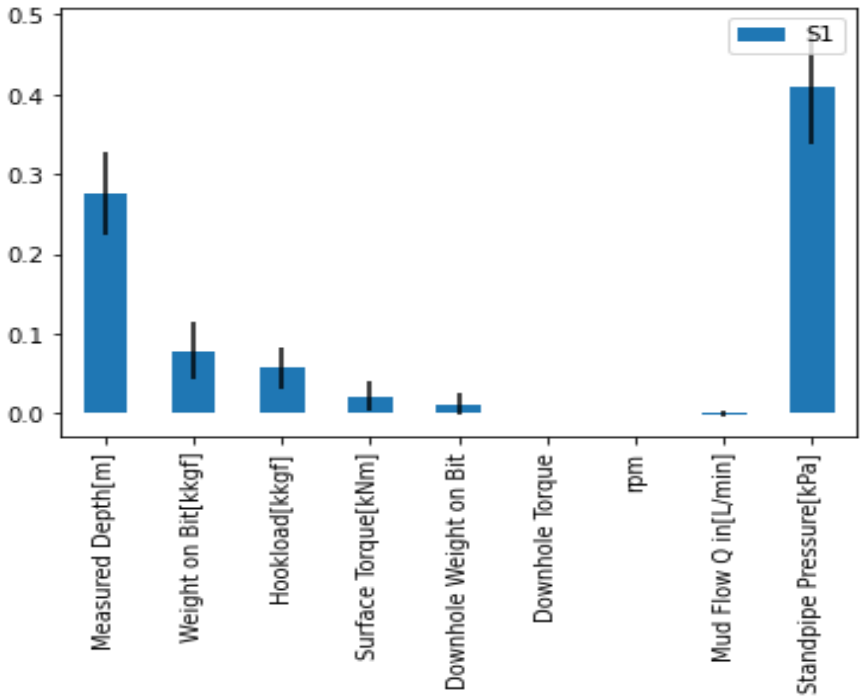


Figure 8.10: First-order indices obtained from LCE regressor

From Figure 8.10, the sensitive parameters based on first-order sensitivity indices are Measured Depth, Weight on Bit, Hookload, and Standpipe pressure.

The sensitive parameters based on the total sensitivity index are Measured Depth, Weight on Bit, Hookload, Surface Torque, and Standpipe Pressure. The results can be observed in Figure 8.11.

Fig 8.12, shows interactions between different parameters, no interaction was significant enough to be considered of great importance.

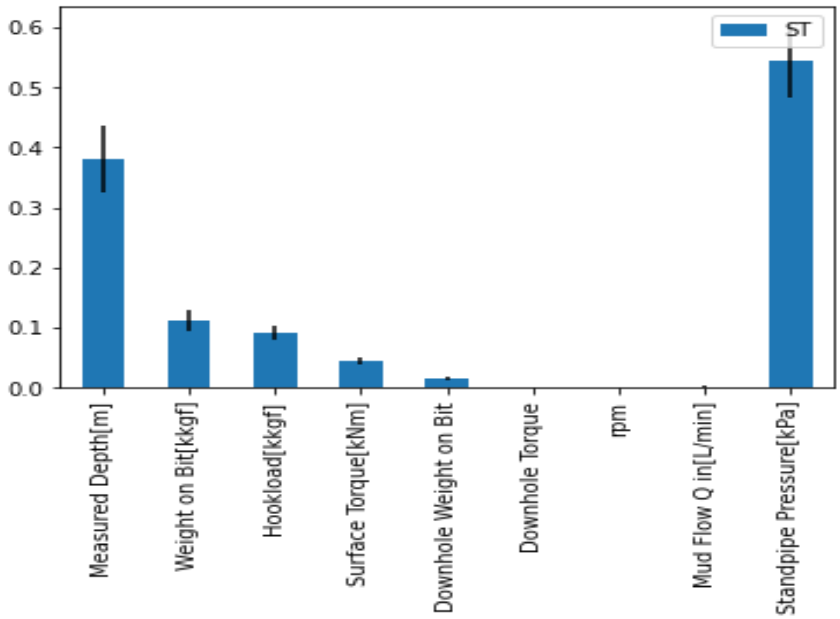


Figure 8.11: Total Sensitivity index obtained from LCE regressor

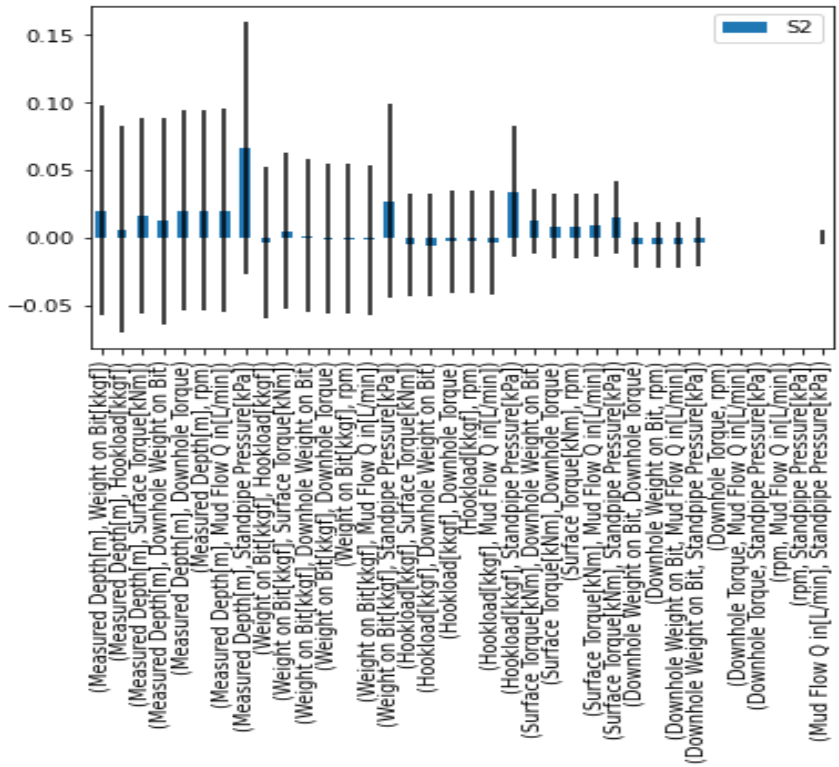


Figure 8.12: Second-order indices obtained from LCE regressor

8.2 Morris Method

The results of the Morris method are obtained in the form of mean, absolut mean, and standard deviation. To select the sensitive input parameter only the absolute mean value is considered. Any parameter having a value of absolute mean equal to or greater than 2 is considered important. Because at this value of absolute mean the selected parameters match with the one selected by the Sobol method.

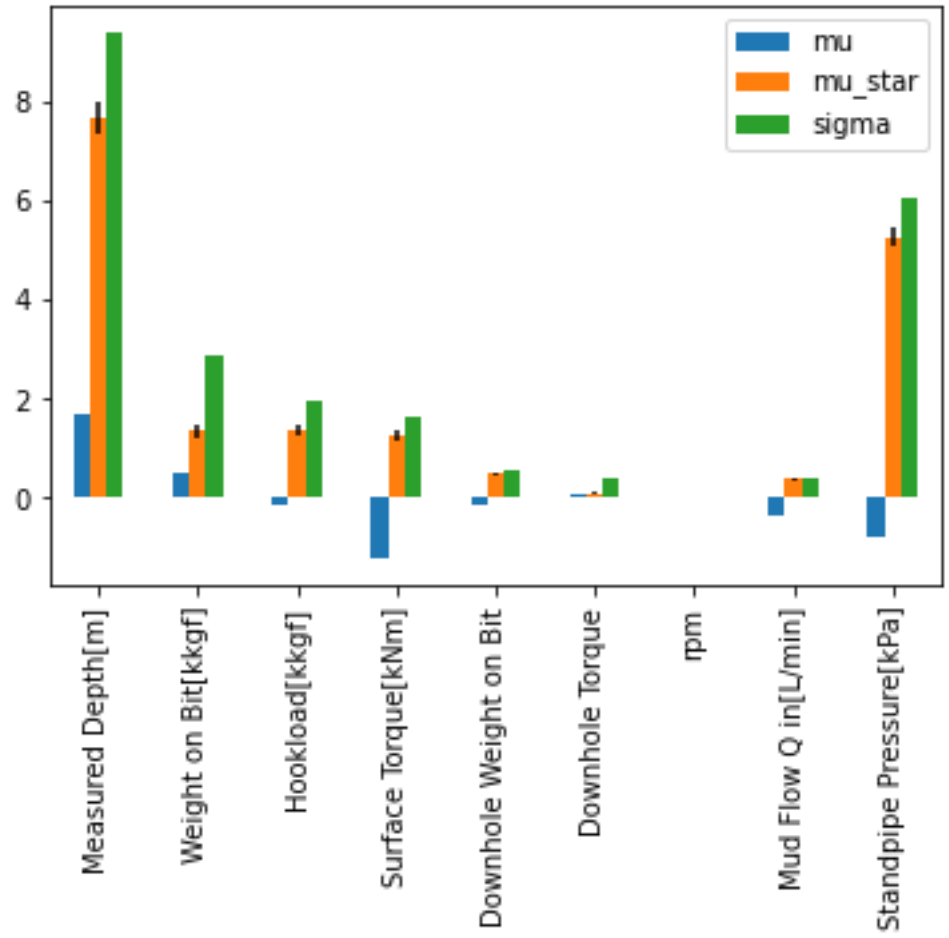


Figure 8.13: Results for Morris method obtained from Random Forest regressor

Figure 8.13 shows the results from running the Morris method for the Random Forest regressor, the parameters that are sensitive based on set criteria are Measured Depth and Standpipe

pressure. This result for RF is the same as parameters selected when only first-order sensitivity indices were considered.

Based upon the selection criteria, Measured Depth, Standpipe pressure, Hookload, and Mudflow is the most sensitive parameters in the case of Gradient Booster regressor. Figure 8.14 shows the results.

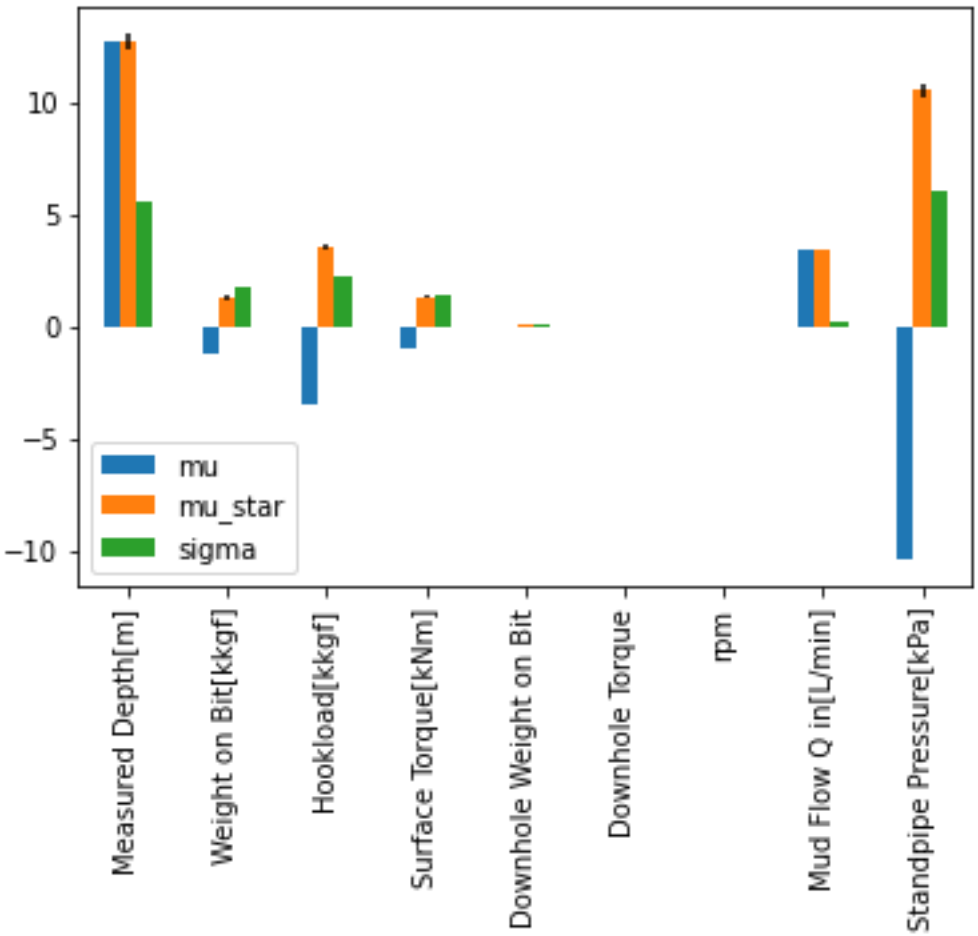


Figure 8.14: Results for Morris method obtained from Gradient Booster regressor

In the case of KNN, the input parameters selected through the Morris method are Measured Depth, Hookload, Downhole Torque, Speed, and Standpipe pressure as in Figure 8.15. In the

case of the Morris method, the KNN regressor has ranked more input parameters are sensitive as compared with the Sobol method.

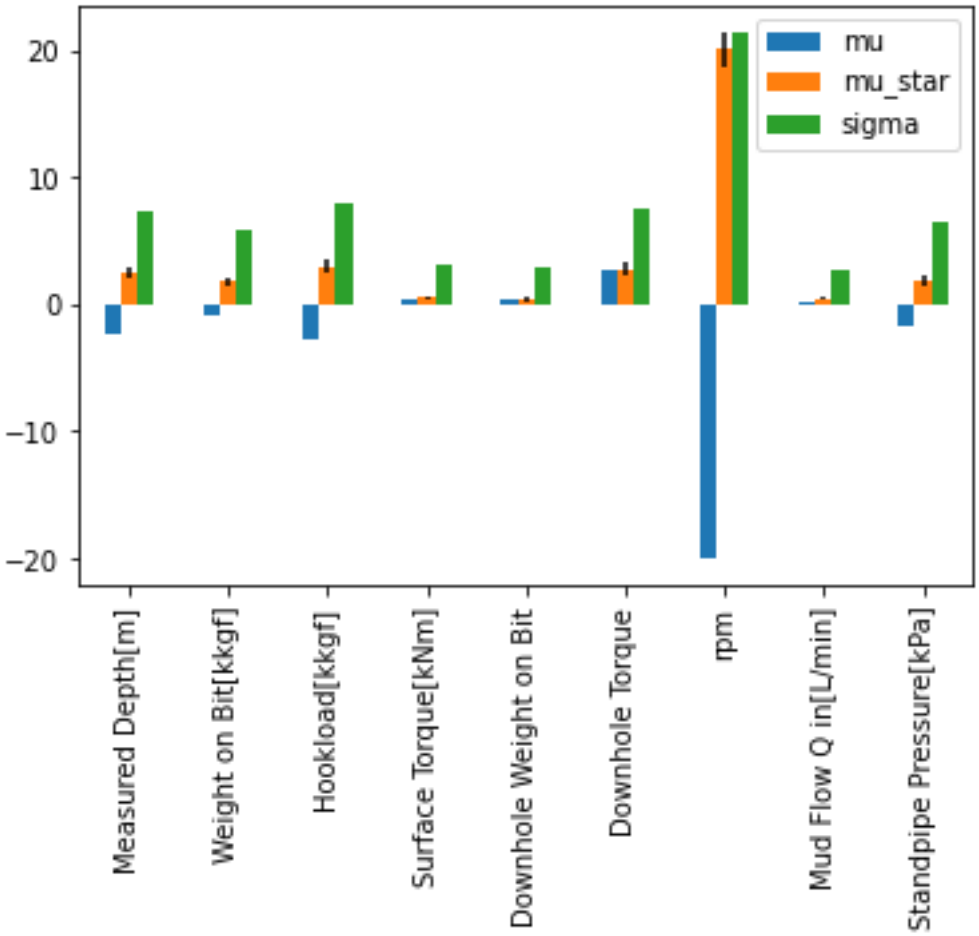


Figure 8.15: Results for Morris method obtained from KNN regressor

LCE results in the same parameters as Random Forest in this case. From Figure 8.16 it can be seen that the most sensitive input parameters are Measured Depth and Standpipe pressure.

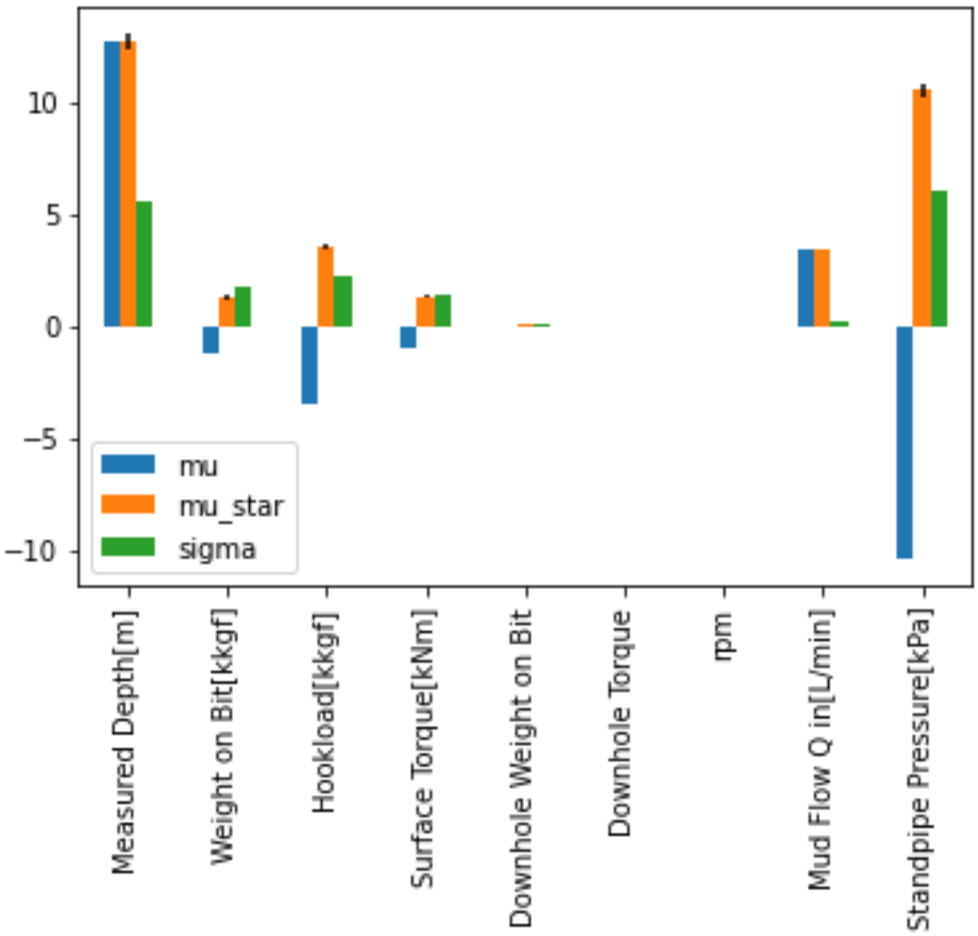


Figure 8.16: Results for Morris method obtained from LCE regressor

8.3 Monte Carlo Simulation

In the case of Monte Carlo simulations, only the Gaussian process regressor is considered. The Gaussian process itself is a computationally expensive method. The same selection criteria were set as that of the Sobol method. For first-order sensitivity indices only Mud Flow and Standpipe pressure as in Figure 8.17.

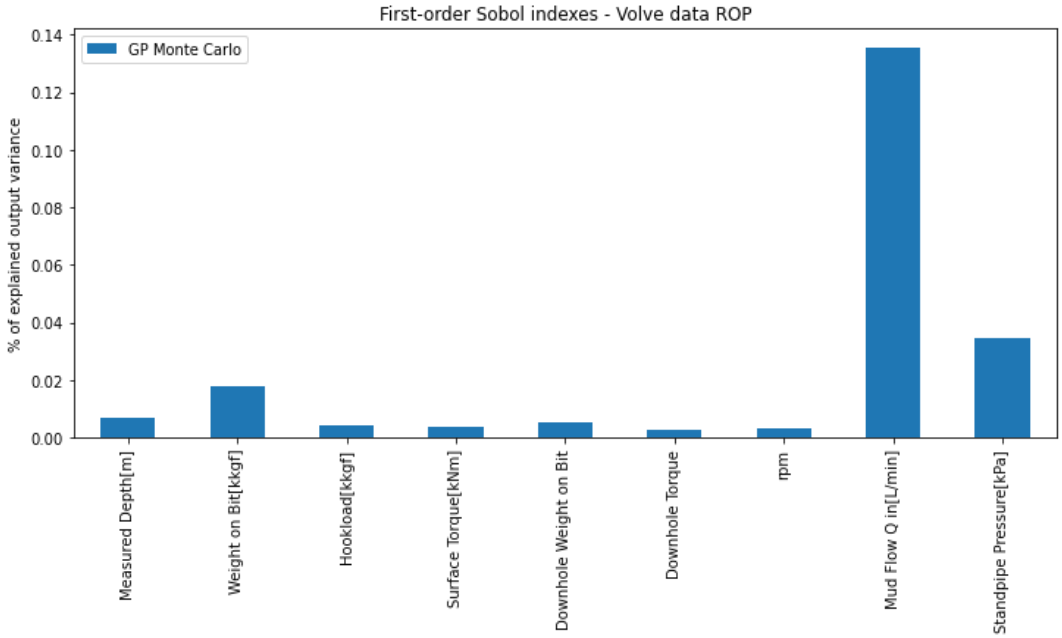


Figure 8.17: Results of Monte Carlo Simulation for first-order sensitivity indices

In the case of total sensitivity index, Measured Depth, Weight on Bit, Hookload, Downhole torque, and Speed were considered the most sensitive parameters. But speed is the most influential parameter of all.

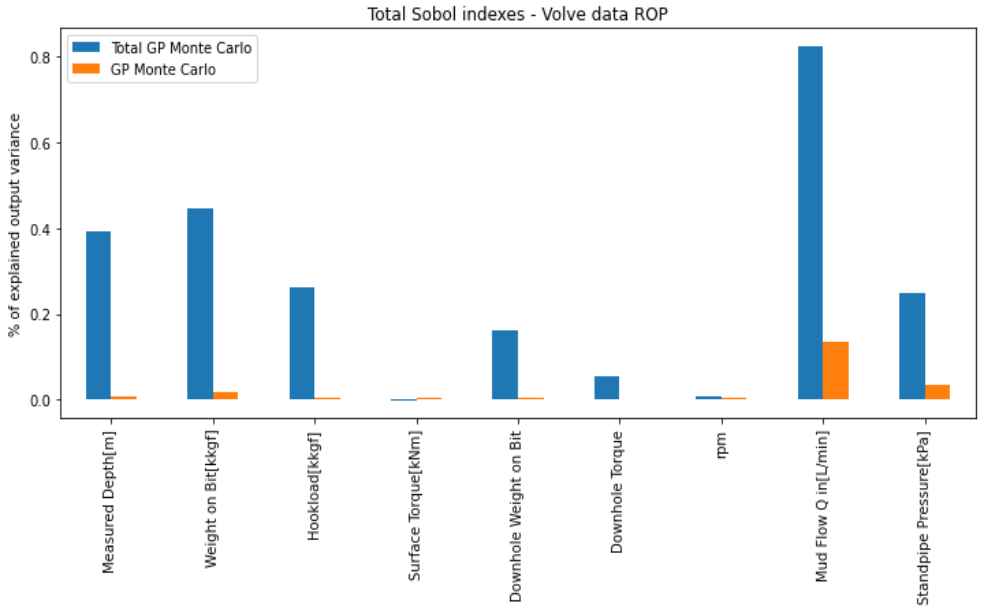


Figure 8.18: Results of Monte Carlo Simulation for first-order sensitivity indices

To compare the results from Monte Carlo Simulation with the Sobol and Morris methods, the Gaussian Process regressor was integrated with both methods. The Gaussian method only ranked Speed as an important parameter with the Sobol method. But in the case of the Morris method, as in Figure 8.19, it gave results closer to Monte Carlo Simulation.

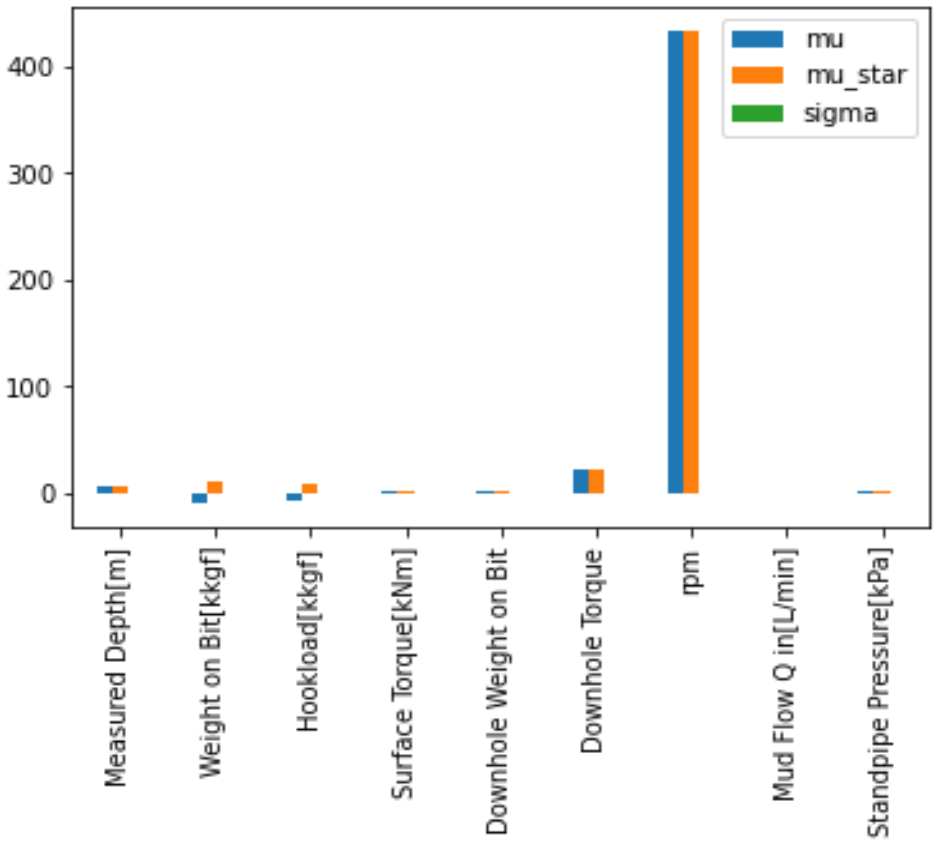


Figure 8.19: Results of Morris method for the Gaussian process

The important features are Measured Depth, Weight on Bit, Hookload, Downhole Torque, and Speed. These parameters are the same as the one predicted for Monte Carlo Simulation considering the total sensitivity index.

It can be concluded that the parameters selected based on the value of the absolute mean in the Morris method gives results similar to the total sensitivity index in the Sobol method and Monte Carlo simulation.

8.4 Discussion

After observing the sensitive parameters selected by each method, in this section, a comparison will be drawn between all and observe which parameters were selected most of the time. The input parameters selected by Feature Selection are discussed in Chapter 5. Table 8.1 gives an overview of features selected in Chapter 5.

	Recursive Method			**Pearson's Correlation**	**Mutual Information**	**Permutation**	
	RF	**GB**	**Decision Tree**			**RF**	**KNN**
Weight on Bit	True	True	True	False	True	False	False
Hookload	False	False	True	False	True	False	False
Surface Torque	False	False	False	True	True	True	False
DWOB	True	True	True	True	True	False	False
Downhole Torque	True	True	True	True	True	True	False
Mudflow	False	False	False	False	True	False	False
Standpipe Pressure	True	False	False	False	True	False	True
Measured Depth	True	True	True	False	True	True	True
Speed	False	False	False	False	False	False	False

Table 8.1: Comparison of parameters selected by using different Feature Selection Techniques

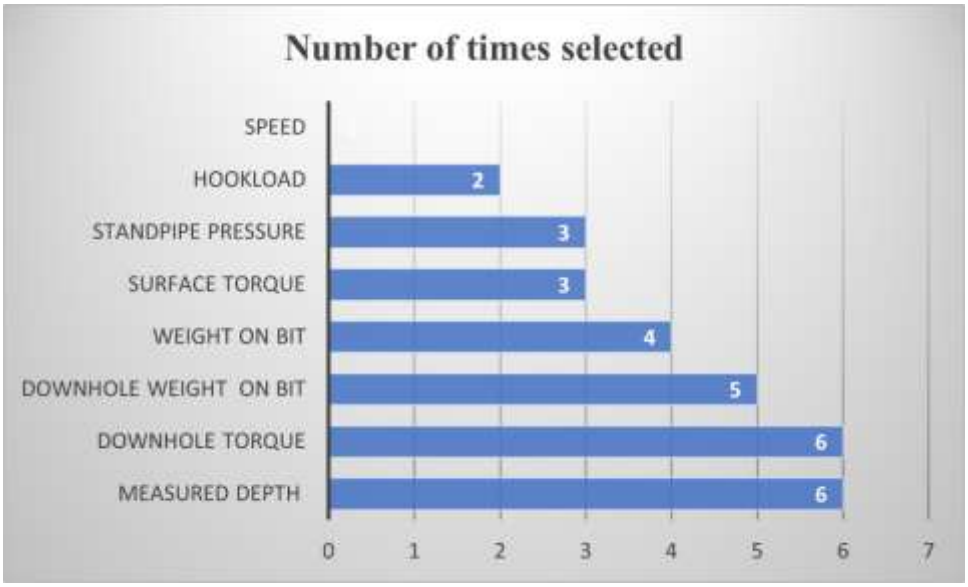


Figure 8.20: Ranking of Parameters selected through Feature Selection

In the case of Feature Selection techniques, Measured Depth and Downhole Torque were selected by six out of 8 methods. Speed was not selected by any technique.

In Figure 8.22, a ranking of input parameters selected through Sensitivity Analysis is presented. In the case of the Sobol method and Monte Carlo simulations parameters selected during the evaluation of the total sensitivity index are considered. In the case of the Morris method, the parameters selected through the value of absolute mean are considered. Four ML methods are used with both Sobol and Morris methods and one method is used with Monte Carlo making a total of 9 methods for Sensitivity Analysis of input parameters. Measured Depth and Standpipe Pressure are selected 8 out of 9 times making them the most influential.

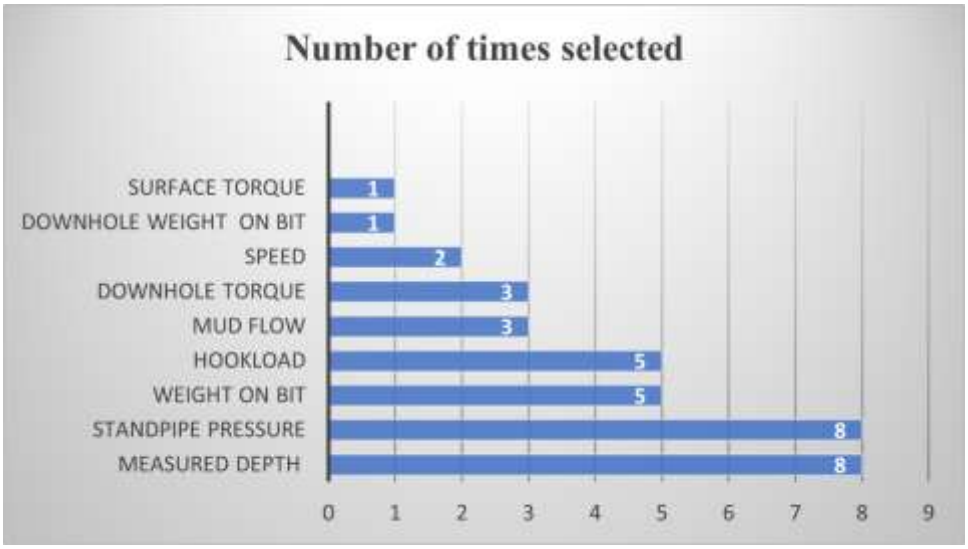


Figure 8.21: Ranking of Parameters selected through the Sensitivity Analysis method

If the comparison is drawn between Feature Selection Techniques and SA methods, Measured Depth remains at the top in both cases. But other parameters do not follow the same ranking order. In the last step bar graph for both techniques is plotted together to observe which input parameters stay at the top if the results from 17 methods, eight FS techniques, and nine SA methods are combined.

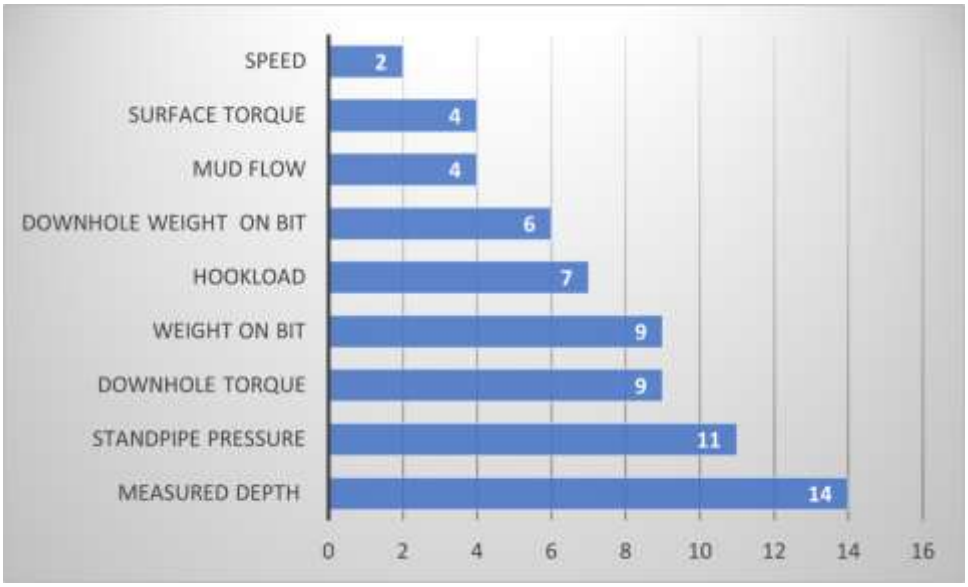


Figure 8.22: Ranking of Parameters selected through combining the results of the Feature Selection & Sensitivity Analysis method

If we choose the parameters that are chosen at least by half of the methods then Measured Depth, Standpipe pressure, Downhole Torque, and Weight on Bit. It means that Measured Depth, Standpipe pressure, Downhole Torque, and Weight on Bit are the parameters that have a great effect on the Rate of penetration modeling. By applying SA methods we have shrunk the size of the dataset to half. Originally the dataset had nine input parameters and among them, four turned out to be the most sensitive.

9. Conclusions

Sensitivity Analysis can help in reducing uncertainty in various models. The performance of any model can be improved by implementing an accurate Sensitivity Analysis method. Sensitivity Analysis methods can help uncover the black box behavior of Machine Learning models by selecting the most influential input parameters. Unwanted input parameters can be deleted by sorting them from most important to least important, increasing efficiency and lowering computational cost.

Sensitivity Analysis, unlike Principal Component Analysis, determines the importance of each input parameter rather than just lowering the dimension of the dataset. In this work, we have used three different concepts to study Sensitivity Analysis for a drilling dataset. The three concepts used are Sensitivity Analysis through Feature Selection, SALib library, and Emukit toolbox. Feature Selection is usually considered a pre-processing step, but it proved to be an easy and efficient way of predicting influential input parameters. The drawback of Feature Selection methods is that the results obtained did not consider the correlation with other input parameters. The Mutual Information Feature Selection method was the only method that ranked all input parameters as somewhat important, by considering the correlation as well. All other methods predicted four input parameters, with Measured Depth being selected every time.

Machine Learning models are integrated with Sensitivity Analysis methods by using the concept of emulation. The Gaussian process proved to be expensive computationally. Random forest, Gradient Booster, and K-Nearest Neighbours showed high performance in predicting ROP. The parameter that was selected most of the time was Measured Depth, and the parameter selected the least time were Surface Torque and Downhole Weight on Bit. The Feature Selection

technique alone cannot work for Sensitivity Analysis. It is a good practice to use it during pre-processing steps. It is also observed that the Sobol, Morris, and Monte Carlo methods give the same results for a particular Machine Learning model. Among all these methods Morris method is a computationally inexpensive method.

9.1 Future Work

In this work, only a handful of Sensitivity Analysis methods are used. There are a lot of Sensitivity methods present that were not used in this study because of limited time. Before considering possibilities for future work, the following points must be considered:

- To select an appropriate Sensitivity method, pre-define the requirements to identify the relation between input and output
- Take help from theoretical concepts to verify results from Sensitivity Analysis methods

Future work can include the following things:

- In this thesis, the results are mainly discussed based on feature analysis and SA without considering the physics and real-time operations. The hybrid-mode analysis combining physics-based info and data-based info for model development and evaluation is of great interest in future work.
- Add more Machine Learning models for evaluating Sensitivity Analysis methods
- Introduce Artificial intelligence and deep learning methods
- Use other built-in libraries for Sensitivity methods in the SALib tool
- Integrate Machine Learning models with Emukit to evaluate the performance of Monte Carlo Simulations

- Combine computational and theoretical knowledge to test the accuracy of ranked input parameters

References

Aggarwal, C. C., 2017, An Introduction to Outlier Analysis, *in* C. C. Aggarwal, ed., Outlier Analysis: Cham, Springer International Publishing, p. 1–34.

Agnihotri, A., and N. Batra, 2020, Exploring Bayesian Optimization: Distill, v. 5, no. 5, p. e26.

Ali, Peshawa Jamal Muhammad, Rezhna Hassan Faraj, E. Koya, Peshawa J. Muhammad Ali, and Rezhna H. Faraj, 2014, Data normalization and standardization: a technical report: Mach Learn Tech Rep, v. 1, no. 1, p. 1–6.

Almuallim, H., and T. G. Dietterich, 1994, Learning Boolean concepts in the presence of many irrelevant features: Artificial Intelligence, v. 69, no. 1, p. 279–305.

Barraza, N., S. Moro, M. Ferreyra, and A. de la Peña, 2019, Mutual information and sensitivity analysis for feature selection in customer targeting: A comparative study: Journal of Information Science, v. 45, no. 1, p. 53–67.

Bollen, K. A., 1984, Multiple indicators: Internal consistency or no necessary relationship? Quality and Quantity, v. 18, no. 4, p. 377–385

van den Bos, L. M. M., B. Sanderse, and W. A. A. M. Bierbooms, 2020, Adaptive sampling-based quadrature rules for efficient Bayesian prediction: Journal of Computational Physics, v. 417, p. 109537, doi:10.1016/j.jcp.2020.109537.

Breiman, L., 2001, Random Forests: Machine Learning, v. 45, no. 1, p. 5–32, doi:10.1023/A:1010933404324.

Bretthorst, G. L., 1990, Bayesian analysis. I. Parameter estimation using quadrature NMR models: Journal of Magnetic Resonance (1969), v. 88, no. 3, p. 533–551, doi:10.1016/0022-2364(90)90287-J.

Campolongo, F., A. Saltelli, and J. Cariboni, 2011, From screening to quantitative sensitivity analysis. A unified approach: Computer Physics Communications, v. 182, no. 4, p. 978–988.

Chandrashekar, G., and F. Sahin, 2014, A survey on feature selection methods: Computers & Electrical Engineering, v. 40, no. 1, p. 16–28.

Chen, J., Y. Gao, and Y. Liu, 2022, Multi-fidelity Data Aggregation using Convolutional Neural Networks: Computer Methods in Applied Mechanics and Engineering, v. 391, p. 114490.

Cheng, X., G. Li, R. Skulstad, P. Major, S. Chen, H. P. Hildre, and H. Zhang, 2019, Data-driven uncertainty and sensitivity analysis for ship motion modeling in offshore operations: Ocean Engineering, v. 179, p. 261–272.

Chu, X., I. F. Ilyas, S. Krishnan, and J. Wang, 2016, Data Cleaning: Overview and Emerging Challenges, *in* Proceedings of the 2016 International Conference on Management of

Data, New York, NY, USA: Association for Computing Machinery, SIGMOD '16, p. 2201–2206.

Cukier, R. I., C. M. Fortuin, K. E. Shuler, A. G. Petschek, and J. H. Schaibly, 1973, Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. I Theory: The Journal of Chemical Physics, v. 59, no. 8, p. 3873–3878.

Davis, S. E., S. Cremaschi, and M. R. Eden, 2017, Efficient Surrogate Model Development: Optimum Model Form Based on Input Function Characteristics, *in* A. Espuña, M. Graells, and L. Puigjaner, eds., Computer Aided Chemical Engineering: Elsevier, 27 European Symposium on Computer Aided Process Engineering, p. 457–462

Di Mauro, M., G. Galatro, G. Fortino, and A. Liotta, 2021, Supervised feature selection techniques in network intrusion detection: A critical review: Engineering Applications of Artificial Intelligence, v. 101, p. 104216.

Duncan, T. E., 2006, On the Calculation of Mutual Information: SIAM Journal on Applied Mathematics.

El Aboudi, N., and L. Benhlima, 2016, Review on wrapper feature selection approaches, *in* 2016 International Conference on Engineering MIS (ICEMIS), 2016 International Conference on Engineering MIS (ICEMIS): p. 1–5.

El Naqa, I., and M. J. Murphy, 2015, What is machine learning?, *in* machine learning in radiation oncology: Springer, p. 3–11.

Ester, M., H.-P. Kriegel, J. Sander, and X. Xu, 1996, A density-based algorithm for discovering clusters in large spatial databases with noise., *in* kdd: p. 226–231.

Fauvel, K., É. Fromont, V. Masson, P. Faverdin, and A. Termier, 2022, XEM: An explainable-by-design ensemble method for multivariate time series classification: Data Mining and Knowledge Discovery, v. 36, no. 3, p. 917–957.

Fidalgo, J. N., 2001, Feature selection based on ann sensitivity analysis-a practical study.

Franczyk, A., 2019, Using the Morris sensitivity analysis method to assess the importance of input variables on time-reversal imaging of seismic sources: Acta Geophysica, v. 67, no. 6, p. 1525–1533.

Frazier, P. I., 2018, A Tutorial on Bayesian Optimization, arXiv:1807.02811: arXiv.

Friedman, J. H., 2001, Greedy function approximation: a gradient boosting machine: Annals of statistics, p. 1189–1232.

Geffray, C., A. Gerschenfeld, P. Kudinov, I. Mickus, M. Jeltsov, K. Kööp, D. Grishchenko, and D. Pointer, 2019, 8 - Verification and validation and uncertainty quantification, *in* F. Roelofs, ed., Thermal Hydraulics Aspects of Liquid Metal Cooled Nuclear Reactors: Woodhead Publishing, p. 383–405.

Gupta, N., S. Mujumdar, H. Patel, S. Masuda, N. Panwar, S. Bandyopadhyay, S. Mehta, S. Guttula, S. Afzal, and R. Sharma Mittal, 2021, Data quality for machine learning tasks, *in* Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining: p. 4040–4041.

Guyon, I., and A. Elisseeff, 2003, An introduction to variable and feature selection: Journal of Machine Learning Research, v. 3, no. 7–8, p. 1157–1182.

Guyon, I., J. Weston, S. Barnhill, and V. Vapnik, 2002, Gene Selection for Cancer Classification using Support Vector Machines: Machine Learning, v. 46, no. 1, p. 389–422.

Hallajian, B., H. Motameni, and E. Akbari, 2022, Ensemble feature selection using distance-based supervised and unsupervised methods in binary classification: Expert Systems with Applications, v. 200, p. 116794, doi:10.1016/j.eswa.2022.116794.

Hamby, D. M., 1994, A review of techniques for parameter sensitivity analysis of environmental models: Environmental monitoring and assessment, v. 32, no. 2, p. 135–154.

Helton, J. C., 1993, Uncertainty and sensitivity analysis techniques for use in performance assessment for radioactive waste disposal: Reliability Engineering & System Safety, v. 42, no. 2, p. 327–367.

Herman, J., and W. Usher, n.d., SALib Documentation: p. 64.

Homma, T., and A. Saltelli, 1996, Importance measures in global sensitivity analysis of nonlinear models: Reliability Engineering & System Safety, v. 52, no. 1, p. 1–17.

Huang, N., G. Lu, and D. Xu, 2016, A permutation importance-based feature selection method for short-term electricity load forecasting using random forest: Energies, v. 9, no. 10, p. 767.

Ikonen, T., 2016, Comparison of global sensitivity analysis methods – Application to fuel behavior modeling: Nuclear Engineering and Design, v. 297, p. 72–80.

Ilyas, I. F., and X. Chu, 2019, Data Cleaning: Morgan & Claypool, 284 p.

Ishigami, T., and T. Homma, 1990, An importance quantification technique in uncertainty analysis for computer models, *in* [1990] Proceedings. First International Symposium on Uncertainty Modeling and Analysis, [1990] Proceedings. First International Symposium on Uncertainty Modeling and Analysis: p. 398–403.

Jaxa-Rozen, M., and J. Kwakkel, 2018, Tree-based ensemble methods for sensitivity analysis of environmental models: A performance comparison with Sobol and Morris techniques: Environmental Modelling & Software, v. 107, p. 245–266.

Jeusfeld, M. A., 2009, Metamodel, *in* L. LIU, and M. T. ÖZSU, eds., Encyclopedia of Database Systems: Boston, MA, Springer US, p. 1727–1730.

Kalakech, M., P. Biela, L. Macaire, and D. Hamad, 2011, Constraint scores for semi-supervised feature selection: A comparative study: Pattern Recognition Letters, v. 32, no. 5, p. 656–665,.

Kamalov, F., 2018, Sensitivity analysis for feature selection, *in* 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA): IEEE, p. 1466–1470.

Khan, K., S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady, 2014, DBSCAN: Past, present and future, *in* The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014): IEEE, p. 232–238.

Kiparissides, A., S. S. Kucherenko, A. Mantalaris, and E. N. Pistikopoulos, 2009, Global sensitivity analysis challenges in biological systems modeling: Industrial & Engineering Chemistry Research, v. 48, no. 15, p. 7168–7180.

Kohavi, R., and G. H. John, 1997, Wrappers for feature subset selection: Artificial Intelligence, v. 97, no. 1, p. 273–324.

Lal, T. N., O. Chapelle, J. Weston, and A. Elisseeff, 2006, Embedded Methods, *in* I. Guyon, M. Nikravesh, S. Gunn, and L. A. Zadeh, eds., Feature Extraction: Foundations and Applications: Berlin, Heidelberg, Springer, Studies in Fuzziness and Soft Computing, p. 137–165.

Le Gratiet, L., S. Marelli, and B. Sudret, 2017, Metamodel-Based Sensitivity Analysis: Polynomial Chaos Expansions and Gaussian Processes, *in* R. Ghanem, D. Higdon, and H. Owhadi, eds., Handbook of Uncertainty Quantification: Cham, Springer International Publishing, p. 1289–1325.

Learned-Miller, E. G., 2013, Entropy and mutual information: Department of Computer Science, University of Massachusetts, Amherst.

Li, J., K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, 2017, Feature Selection: A Data Perspective: ACM Computing Surveys, v. 50, no. 6, p. 94:1-94:45.

Liu, H., H. Motoda, R. Setiono, and Z. Zhao, 2010, Feature Selection: An Ever Evolving Frontier in Data Mining, *in* Proceedings of the Fourth International Workshop on Feature Selection in Data Mining, Feature Selection in Data Mining: PMLR, p. 4–13.

Liu, H., and R. Setiono, n.d., A Probabilistic Approach to Feature Selection - A Filter Solution: Morgan Kaufmann, p. 319–327.

Liu, Q., Z. Zhao, Y.-X. Li, and Y. Li, 2012, Feature selection based on sensitivity analysis of fuzzy ISODATA: Neurocomputing, v. 85, p. 29–37.

Loucks, D. P., E. van Beek, J. R. Stedinger, and E. van Beek, 2005, Water resources systems planning and management: an introduction to methods, models and applications: Paris, UNESCO, Studies and reports in hydrology, 680 p.

Mann, P. S., 2010, Introductory Statistics: John Wiley & Sons, 749 p.

Metropolis, N., and S. Ulam, 1949, The monte carlo method: Journal of the American statistical association, v. 44, no. 247, p. 335–341.

Miyazaki, C., N. Katsumasa, K. C. Huang, and Y. F. Liu, 2021, Evaluation of economic burden with biologic treatments in Crohn's disease patients: A mirror image study using an insurance database in Japan: PloS one, v. 16, no. 7, p. e0254807.

Morio, J., 2011, Global and local sensitivity analysis methods for a physical system: European Journal of Physics, v. 32, p. 1577, doi:10.1088/0143-0807/32/6/011.

Morris, M. D., 1991a, Factorial sampling plans for preliminary computational experiments: Technometrics, v. 33, no. 2, p. 161–174.

Morris, M. D., 1991b, Factorial Sampling Plans for Preliminary Computational Experiments: Technometrics, v. 33, no. 2, p. 161–174.

Müller, A. C., and S. Guido, 2016, Introduction to Machine Learning with Python: A Guide for Data Scientists: O'Reilly Media, Inc., 400 p.

Nasteski, V., 2017, An overview of the supervised machine learning methods: HORIZONS.B, v. 4, p. 51–62.

Nettleton, D., 2014, Commercial Data Mining: Processing, Analysis and Modeling for Predictive Analytics Projects: Elsevier, 361 p.

O'Hagan, A., 1991, Bayes–Hermite quadrature: Journal of Statistical Planning and Inference, v. 29, no. 3, p. 245–260.

Paleyes, A., M. Pullin, M. Mahsereci, C. McCollum, N. D. Lawrence, and J. Gonzalez, 2021, Emulation of physical processes with Emukit, arXiv:2110.13293: arXiv.

Park, C., C. C. Took, and J.-K. Seong, 2018, Machine learning in biomedical engineering: Biomedical Engineering Letters, v. 8, no. 1, p. 1–3.

Peherstorfer, B., K. Willcox, and M. Gunzburger, 2018, Survey of Multifidelity Methods in Uncertainty Propagation, Inference, and Optimization: SIAM Review, v. 60, no. 3, p. 550–591.

Pelfrene, G., H. Al-Ajmi, J. Bashir, F. Al-Otaibi, A. Al-Nuaimi, H. Al-Jiran, A. Baroun, A. Al-Kandari, T. Huseen, and B. Mikhail, 2019, Statistical Optimization of PDC Bit Designs Based on 3D Simulations Applied to Demanding Directional Applications, *in* Abu Dhabi International Petroleum Exhibition & Conference: OnePetro.

Pianosi, F., K. Beven, J. Freer, J. W. Hall, J. Rougier, D. B. Stephenson, and T. Wagener, 2016, Sensitivity analysis of environmental models: A systematic review with practical workflow: Environmental Modelling & Software, v. 79, p. 214–232.

Plúa, C., M.-N. Vu, D. M. Seyedi, and G. Armand, 2021, Effects of inherent spatial variability of rock properties on the thermo-hydro-mechanical responses of a high-level radioactive

waste repository: International Journal of Rock Mechanics and Mining Sciences, v. 145, p. 104682.

Qi, C., Z. Meng, X. Liu, Q. Jin, and R. Su, 2018, Decision Variants for the Automatic Determination of Optimal Feature Subset in RF-RFE: Genes, v. 9, p. 301.

Rasmussen, C. E., 2004, Gaussian Processes in Machine Learning, *in* O. Bousquet, U. von Luxburg, and G. Rätsch, eds., Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures: Berlin, Heidelberg, Springer, Lecture Notes in Computer Science, p. 63–71.

Rasmussen, C. E., and C. K. I. Williams, 2006, Gaussian processes for machine learning: Cambridge, Mass, MIT Press, Adaptive computation and machine learning, 248 p.

Ratto, M., A. Castelletti, and A. Pagano, 2012, Emulation techniques for the reduction and sensitivity analysis of complex environmental models: Elsevier, p. 1–4.

Sainis, N., D. Srivastava, and D. R. Singh, 2018, Feature Classification and Outlier Detection to Increased Accuracy in Intrusion Detection System: v. 13, no. 10, p. 7.

Saltelli, A., 2002, Making best use of model evaluations to compute sensitivity indices: Computer Physics Communications, v. 145, no. 2, p. 280–297.

Saltelli, A. (Andrea), 2004, Sensitivity analysis in practice : a guide to assessing scientific models: Chichester ; Hoboken, NJ : Wiley, 232 p.

Saltelli, A., P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola, 2010, Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index: Computer Physics Communications, v. 181, no. 2, p. 259–270.

Saltelli, A., K. Chan, and E. M. Scott, 2000, Wiley series in probability and statistics: Sensitivity analysis.

Saltelli, A., S. Tarantola, F. Campolongo, and M. Ratto, 2004, Sensitivity analysis in practice: a guide to assessing scientific models: Wiley Online Library.

Saltelli, A., S. Tarantola, and K. Chan, 2012, A Quantitative Model-Independent Method for Global Sensitivity Analysis of Model Output: Technometrics, v. 41.

Saranya, G., and A. Pravin, 2021, An Efficient Feature Selection Approach using Sensitivity Analysis for Machine Learning based Heart Desease Classification, *in* 2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT), 2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT): p. 539–542.

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, 2016, Taking the Human Out of the Loop: A Review of Bayesian Optimization: Proceedings of the IEEE, v. 104, no. 1, p. 148–175.

Singh, D., and B. Singh, 2020, Investigating the impact of data normalization on classification performance: Applied Soft Computing, v. 97, p. 105524.

Singh, A., N. Thakur, and A. Sharma, 2016, A review of supervised machine learning algorithms, *in* 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom): p. 1310–1315.

Sobol, I. M., 2001, Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates: Mathematics and computers in simulation, v. 55, no. 1–3, p. 271–280.

Sobol′, I. M., 2001a, Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates: Mathematics and Computers in Simulation, v. 55, no. 1, p. 271–280.

Sobol′, I. M., 2001b, Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates: Mathematics and Computers in Simulation, v. 55, no. 1, p. 271–280.

Sobol, I. M., 1993, Sensitivity analysis for non-linear mathematical models: Mathematical modelling and computational experiment, v. 1, p. 407–414.

Steffensen, J. F., 2006, Interpolation: Courier Corporation.

Tavakoli, S., A. Mousavi, and S. Poslad, 2013, Input variable selection in time-critical knowledge integration applications: A review, analysis, and recommendation paper: Advanced Engineering Informatics, v. 27, no. 4, p. 519–536.

Taylor, R., 1990, Interpretation of the Correlation Coefficient: A Basic Review: Journal of Diagnostic Medical Sonography, v. 6, no. 1, p. 35–39.

Teshale, E. Z., K. Hoegh, S. Dai, R. Giessel, and C. Turgeon, 2020, Ground penetrating radar sensitivity to marginal changes in asphalt mixture composition: Journal of Testing and Evaluation, v. 48, no. 3, p. 2295–2310.

Tunkiel, A. T., D. Sui, and T. Wiktorski, 2020, Data-driven sensitivity analysis of complex machine learning models: A case study of directional drilling: Journal of Petroleum Science and Engineering, v. 195, p. 107630.

Uddin, Md. T., and Md. A. Uddiny, 2015, A guided random forest based feature selection approach for activity recognition, *in* 2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT), 2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT): p. 1–6.

Wang, S.-Y., 2010, Huan Liu and Hiroshi Motoda (eds): Computational methods of feature selection: Pattern Analysis and Applications, v. 13, no. 2, p. 247–249.

Wang, J., Z. Zhao, G. Liu, and H. Xu, 2022, A robust optimization approach of well placement for doublet in heterogeneous geothermal reservoirs using random forest technique and genetic algorithm: Energy, p. 124427.

Whaley III, D. L., 2005, The interquartile range: Theory and estimation: East Tennessee State University.

Zhang, P., 2019, A novel feature selection method based on global sensitivity analysis with application in machine learning-based prediction model: Applied Soft Computing, v. 85, p. 105859.

Zhang, X.-Y., M. N. Trame, L. J. Lesko, and S. Schmidt, 2015, Sobol Sensitivity Analysis: A Tool to Guide the Development and Evaluation of Systems Pharmacology Models: CPT: pharmacometrics & systems pharmacology, v. 4, no. 2, p. 69–79.

Zhang, M.-L., and Z.-H. Zhou, 2007, ML-KNN: A lazy learning approach to multi-label learning: Pattern Recognition, v. 40, no. 7, p. 2038–2048.

Zhao, M., Z. Zhang, and T. W. S. Chow, 2012, Trace ratio criterion based generalized discriminative learning for semi-supervised dimensionality reduction: Pattern Recognition, v. 45, no. 4, p. 1482–1499.

ZHENCHAO, Q., S. YIMING, Y. Liu, and X. PINGPING, 2019, AN APPROACH OF SENSITIVITY ANALYSIS FOR FINITE ELEMENT SIMULATION OF CFRP DRILLING PROCESS BASED ON SOBOL METHOD: UPB SCIENTIFIC BULLETIN, SERIES D: MECHANICAL ENGINEERING, v. 81, no. 1, p. 155–164.

Appendix

Feature Selection

```
#Feature Selection Through Decision Tree
from sklearn.tree import DecisionTreeRegressor
# define the model
model = DecisionTreeRegressor()
# fit the model
model.fit(x_train, y_train.ravel())
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()

#Feature Selection Through Random Forest
from sklearn.ensemble import RandomForestRegressor

# define the model
model = RandomForestRegressor()
# fit the model
model.fit(x_train, y_train.ravel())
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Listing A.1: Code Snippet for Feature Selection through Feature Importance

```
#Recrusive Feature Selection by decision tree
# create pipeline
rfe = RFE(estimator=DecisionTreeRegressor())
model = DecisionTreeRegressor()
pipeline = Pipeline(steps=[('s',rfe),('m',model)])
# fit RFE
dt.fit(x_train, y_train)
# summarize all features
for i in range(x_train.shape[1]):
    print('Column: %d, Selected %s, Rank: %.3f' % (i, rfe.support_[i],
rfe.ranking_[i]))

#RFE using Gradient Booster
```

```
# create pipeline
rfe = RFE(estimator=GradientBoostingRegressor())
model = GradientBoostingRegressor()
pipeline = Pipeline(steps=[('s',rfe),('m',model)])
# fit RFE
rfe.fit(x_train, y_train)
# summarize all features
for i in range(x_train.shape[1]):
    print('Column: %d, Selected %s, Rank: %.3f' % (i, rfe.support_[i],
rfe.ranking_[i]))

#RFE using Random Forest
# create pipeline
rfe = RFE(estimator= RandomForestRegressor())
model =  RandomForestRegressor()
pipeline = Pipeline(steps=[('s',rfe),('m',model)])
# fit RFE
rfe.fit(x_train, y_train)
# summarize all features
for i in range(x_train.shape[1]):
    print('Column: %d, Selected %s, Rank: %.3f' % (i, rfe.support_[i],
rfe.ranking_[i]))
```

Listing A.2: Code Snippet for Feature Selection through Recursive Feature Selection

```
#Feature Selection Based on pearson correlation
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from matplotlib import pyplot

# feature selection
def select_features(X_train, y_train, X_test):
    # configure to select all features
    fs = SelectKBest(score_func=f_regression, k='all')
    # learn relationship from training data
    fs.fit(X_train, y_train)
    # transform train input data
    X_train_fs = fs.transform(X_train)
    # transform test input data
    X_test_fs = fs.transform(X_test)
    return X_train_fs, X_test_fs, fs

# load the dataset
X= df[['Measured Depth[m]','Weight on Bit[kkgf]','Hookload[kkgf]','Surface
Torque[kNm]','Downhole Weight on Bit','Downhole Torque','rpm','Mud Flow Q
in[L/min]','Standpipe Pressure[kPa]']]
y= df["ROP[m/h]"]
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
train_size=0.8, random_state=43)
# feature selection
X_train_fs, X_test_fs, fs = select_features(X_train, y_train, X_test)
```

```
# what are scores for the features
for i in range(len(fs.scores_)):
    print('Feature %d: %f' % (i, fs.scores_[i]))
# plot the scores
pyplot.bar([i for i in range(len(fs.scores_))], fs.scores_)
pyplot.show()
```

Listing A.3: Code Snippet for Feature Selection through Pearson's correlation

```
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from matplotlib import pyplot

# feature selection
def select_features(X_train, y_train, X_test):
    # configure to select all features
    fs = SelectKBest(score_func=mutual_info_regression, k='all')
    # learn relationship from training data
    fs.fit(X_train, y_train)
    # transform train input data
    X_train_fs = fs.transform(X_train)
    # transform test input data
    X_test_fs = fs.transform(X_test)
    return X_train_fs, X_test_fs, fs

# load the dataset
X= df[['Measured Depth[m]','Weight on Bit[kkgf]','Hookload[kkgf]','Surface
   Torque[kNm]','Downhole Weight on Bit','Downhole Torque','rpm','Mud Flow Q
   in[L/min]','Standpipe Pressure[kPa]']]
y= df["ROP[m/h]"]
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
   train_size=0.8, random_state=43)
# feature selection
X_train_fs, X_test_fs, fs = select_features(X_train, y_train, X_test)
# what are scores for the features
for i in range(len(fs.scores_)):
    print('Feature %d: %f' % (i, fs.scores_[i]))
# plot the scores
pyplot.bar([i for i in range(len(fs.scores_))], fs.scores_)
pyplot.show()
```

Listing A.4: Code Snippet for Feature Selection through Mutual Information

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.inspection import permutation_importance
perm_importance = permutation_importance(model, x_test, y_test)

# get importance
importance = perm_importance .importances_mean
# summarize feature importance
```

```
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()

# define the model
model = KNeighborsRegressor()
# fit the model
model.fit(x_train, y_train)
# perform permutation importance
results = permutation_importance(model, x_test, y_test)
# get importance
importance = results.importances_mean
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Listing A.5: Code Snippet for Feature Selection through Permutation

Sensitivity Analysis Using SALib

```
#install library
pip install SALib

#Case study 1 Ishigami function
#Sobol method
from SALib.sample import saltelli
from SALib.analyze import sobol
from SALib.test_functions import Ishigami
import numpy as np
import matplotlib.pyplot as plt

# Define the model inputs
problem = {
'num_vars': 3,
'names': ['x1', 'x2', 'x3'],
'bounds': [[-3.14159265359, 3.14159265359],
[-3.14159265359, 3.14159265359],
[-3.14159265359, 3.14159265359]]
}
# Generate samples
param_values = saltelli.sample(problem, 1024)
# Run model (example)
Y = Ishigami.evaluate(param_values)
# Perform analysis
Si = sobol.analyze(problem, Y, print_to_console=True)

```

```
#Morris method
from SALib.sample.morris import sample
from SALib.analyze import morris

# Generate samples
X = sample(problem, 100000, num_levels=4)
# Run model (example)
Y = Ishigami.evaluate(X)
# Perform analysis
Si = morris.analyze(problem,X, Y, conf_level=0.95, print_to_console=True,
num_levels=4)
```

Listing A.6: Code Snippet for Sensitivity Analysis using SALib for Ishigami function

```
#define function
import numpy as np


def Function1(X: np.ndarray) -> np.ndarray:

    D = np.zeros(X.shape[0])
    D = 0.5*3.14*np.sin(X[:, 0]) + 0.25*3.14*np.sin(X[:, 1]) +
0.167*3.14*np.sin(X[:, 2]) * np.sin(X[:, 0])

    return D

#define boundaries for input parameter
limits = {
'num_vars': 3,
'names': ['x1', 'x2', 'x3'],
'bounds': [[-1, 1],
[-1, 1],
[-1, 1]]}

#Generate input samples
param_values = saltelli.sample(limits, 1024)
#evaluate
D = Function1(param_values)
# analyse
Test = sobol.analyze(limits, D, print_to_console=True)

#Morris method
# sample
param_values = sample(limits, 100, num_levels=4)
#evaluate
D = Function1(param_values)
# analyse
Test = morris.analyze(limits,param_values, D,conf_level=0.95,
print_to_console=True)
```

Listing A.7: Code Snippet for Sensitivity Analysis using SALib for sine function

```
#Exponential function of coefficents
```

```
#define function
def Expo(x, a, b):
    """Return y = a*e**x + b*x"""
    return a*np.exp(x) + b

#define boundaries for input parameter
prob = {
'num_vars': 2,
'names': ['a','b'],
'bounds': [[-1, 1],
[-1, 1]]
}

#Generate input samples
param_values = saltelli.sample(prob, 2**6)
# evaluate
x = np.linspace(0, 1, 1000)
y = np.array([Expo(x, *params) for params in param_values])
# analyse
sobol_indices = [sobol.analyze(prob, Y) for Y in y.T]

#Morris method
# sample
param_values = sample(prob,1000)
#evaluate
y = Expo(param_values)
# analyse
final = morris.analyze(prob, param_values, y, conf_level=0.95,
print_to_console=True)
```

Listing A.8: Code Snippet for Sensitivity Analysis using SALib for exponential function

```
#SA using Random Forest Sobol method

#implement RF regressor
rf =RandomForestRegressor(n_estimators=70,random_state=0,max_depth=30)
# fit the regressor with x and y data
rf.fit(X_train, y_train.ravel())
#Prediction
y_pred_rf=rf.predict(X_test)
R2_rf = r2_score(y_test, y_pred_rf)
      print ("R2 Score for random forest:",R2_rf)

      #define boundaries for input parameters
      problem = {'num_vars': 9,
                 'names': ['Measured Depth[m]',
                           'Weight on Bit[kkgf]',
                           'Hookload[kkgf]',
                           'Surface Torque[kNm]',
                           'Downhole Weight on Bit',
```

```
                            'Downhole Torque','rpm',
                            'Mud Flow Q in[L/min]',
                           'Standpipe Pressure[kPa]'],
                'bounds': [[1900.677364, 2399.617067],
                            [-3.886818,3.066797],
                            [128.031828, 141.763380],
                             [15.692000, 16.919667],
                             [1.142146, 1.506834],
                             [-(2.4)25558,1.342260],
                            [-150.000334,149.369995],
                             [3515.573242,3515.969970],
                            [18594.666544,20633.999632 ]]
                }

    #Generate input parameters sample
    param_values = saltelli.sample(problem, 1024)
    param_values = scaler.transform(param_values)
    #analyze the model
    Y_rf = rf.predict(param_values)
    Si = sobol.analyze(problem, Y_rf, print_to_console=True)

    #Morris method
    #Generate input parameters sample
    X_rf_morris = sample(problem, 1000, num_levels=4)
    X_rf_morris= scaler.transform(X_rf_morris)
    #analyze the model
    Y_rf = rf.predict(X_rf_morris)
    Si_rf_morris = morris.analyze(problem,X_rf_morris, Y_rf, conf_level=0.95, print_to_co
```

Listing A.9: Code Snippet for Sensitivity Analysis using SALib for Random Forest

```
#SA using Gradient Booster Sobol method

#implement GB regressor
gbr =GradientBoostingRegressor(n_estimators= 1000, max_depth=3,
random_state=100, learning_rate = 0.05)
# fit the regressor with x and y data
gbr.fit(X_train, y_train.ravel())
#Prediction
y_pred_gbr=gbr.predict(X_test)
R2_gbr = r2_score(y_test, y_pred_gbr)
print ("R2 Score for gbr :",R2_gbr)

#analyze the model
Y_gbr = gbr.predict(param_values)
Si_gbr = sobol.analyze(problem, Y_gbr, print_to_console=True)

#Morris method
#Generate input parameters sample
```

```
X_gbr_morris = sample(problem, 1000, num_levels=4)
X_gbr_morris= scaler.transform(X_gbr_morris)
#analyze the model
Y_gbr = gbr.predict(X_gbr_morris)
Si_gbr_morris = morris.analyze(problem,X_gbr_morris, Y_gbr, conf_level=0.95,
print_to_console=True, num_levels=4)
```

Listing A.10: Code Snippet for Sensitivity Analysis using SALib for Gradient Booster

```
#SA using knn Sobol method

#implement knn regressor
from sklearn.neighbors import KNeighborsRegressor

# define the model
knn = KNeighborsRegressor(algorithm='brute',leaf_size = 30, metric =
'minkowski',
                          n_neighbors=3, weights = 'distance')
# fit the model
knn.fit(X_train, y_train.ravel())
#Prediction
y_pred_knn=knn.predict(X_test)
R2_knn = r2_score(y_test, y_pred_knn)
print ("R2 Score for knn :",R2_knn)

#generate input samples
param_values_knn = saltelli.sample(problem, 16)
param_values_knn = scaler.transform(param_values)

#analyze the model
Y_knn = knn.predict(param_values)
Si_knn = sobol.analyze(problem, Y_knn, print_to_console=True)

#Morris method
#Generate input parameters sample
X_knn_morris = sample(problem, 1000, num_levels=4)
X_knn_morris= scaler.transform(X_knn_morris)
#analyze the model
Y_knn = knn.predict(X_knn_morris)
Si_knn_morris = morris.analyze(problem,X_knn_morris, Y_knn, conf_level=0.95,
print_to_console=True, num_levels=4)
```

Listing A.11: Code Snippet for Sensitivity Analysis using SALib for KNN

```
#SA using LCE Sobol method

#Install library
pip install lcensemble --user
#implement lce regressor
from lce import LCERegressor
# Train LCERegressor with default parameters
reg = LCERegressor(n_jobs=-1, random_state=100, max_depth=3)
```

```
# fit the model
reg.fit(X_train, y_train)
#Prediction
y_pred_lce=reg.predict(X_test)
R2_lce = r2_score(y_test, y_pred_lce)
print ("R2 Score for lce :",R2_lce)

#analyze the model
Y_lce = reg.predict(param_values)
Si_lce = sobol.analyze(problem, Y_lce, print_to_console=True)

#Morris method
#Generate input parameters sample
X_lce_morris = sample(problem, 1000, num_levels=4)
X_lce_morris= scaler.transform(X_lce_morris)
#analyze the model
Y_lce = reg.predict(X_lce_morris)
Si_lce_morris = morris.analyze(problem,X_lce_morris, Y_lce, conf_level=0.95,
print_to_console=True, num_levels=4)
```

Listing A.12: Code Snippet for Sensitivity Analysis using SALib for Local Cascade Ensemble

Sensitivity Analysis using Monte Carlo Simulation- Emukit

```
#SA using Emukit
#Testing Ishigami function

#Install library
pip install emukit
from emukit.test_functions.sensitivity import Ishigami ### change this one for
the one in the library

###Load the Ishigami function
ishigami = Ishigami(a=7, b=0.1)
target_simulator = ishigami.fidelity1

###Define the input space in which the simulator is defined
variable_domain = (-np.pi,np.pi)
x_grid = np.linspace(*variable_domain,1024)

#define boundaries for input parameters
from emukit.core import ContinuousParameter, ParameterSpace

target_simulator = ishigami.fidelity1
variable_domain = (-np.pi,np.pi)

space = ParameterSpace([ContinuousParameter('x1', variable_domain[0],
variable_domain[1]),
                        ContinuousParameter('x2', variable_domain[0],
variable_domain[1]),
```

```
                        ContinuousParameter('x3', variable_domain[0],
    variable_domain[1])])
#generate Monte Carlo Samples
from emukit.sensitivity.monte_carlo import ModelFreeMonteCarloSensitivity
np.random.seed(10)
num_mc = 1024# Number of MC samples
senstivity_ishigami = ModelFreeMonteCarloSensitivity(target_simulator, space)

#evaluate

main_effects, total_effects, _ =
    senstivity_ishigami.compute_effects(num_monte_carlo_points = num_mc)
```

Listing A.13: Code Snippet for Sensitivity Analysis using Emukit for Ishigami function

```
#SA using Emukit
#Testing Volve dataset using Gaussian Process
#Install libraries
pip install emukit
pip install notutils
pip install gpy
pip install pyDOE

from GPy.models import GPRegression
    from emukit.model_wrappers import GPyModelWrapper
    from emukit.sensitivity.monte_carlo import MonteCarloSensitivity

    #Implement Gaussian process regressor
    model_gpy = GPRegression(X,Y)
    #use a wrapper to wrap with Emukit
    model_emukit = GPyModelWrapper(model_gpy)
    model_emukit.optimize()



    ###Define the input space in which the simulator is defined
    variable_domain1 = (0,1)
    variable_domain2 = (-0.508121,0.528973)
    variable_domain3 = (0.128143,0.991169)
    variable_domain4 = (0.127880,0.333602)
    variable_domain5 = (-0.185727,0.184713)
    variable_domain6 = (-0.094495,0.062246)
    variable_domain7 = (0.312254,0.528439)
    variable_domain8 = (-0.820515,0.227995)
    variable_domain9 = (0.397440,0.913323)

    space = ParameterSpace(
              [ContinuousParameter('Measured Depth[m]', *variable_domain1),
```

```
                ContinuousParameter('Weight on Bit[kkgf]', *variable_domain2),
                ContinuousParameter('Hookload[kkgf]', *variable_domain3),
                ContinuousParameter('Surface Torque[kNm]', *variable_domain4),
                ContinuousParameter('Downhole Weight on Bit', *variable_domain5),
                ContinuousParameter('Downhole Torque', *variable_domain6),
                ContinuousParameter('rpm', *variable_domain7),
                ContinuousParameter('Mud Flow Q in[L/min]', *variable_domain8),
                ContinuousParameter('Standpipe Pressure[kPa]', *variable_domain9)])


    #generate Monte Carlo Samples
    from emukit.sensitivity.monte_carlo import ModelFreeMonteCarloSensitivity
    num_mc = 10000
    senstivity_gpbased = MonteCarloSensitivity(model = model_emukit,input_domain = space)

    #evaluate

    main_effects, total_effects, _ = senstivity_ishigami.compute_effects(num_monte_carlo_
    main_effects_gp, total_effects_gp, _ = senstivity_gpbased.compute_effects(num_monte_c
```

Listing A.14: Code Snippet for Sensitivity Analysis using Emukit for Ishigami function

Outlier Removal

```
 #OUTLIER REMOVAL THROUGH IQR METOD
#removing outlier in ROP
for x in ['ROP[m/h]']:
    q75,q25 = np.percentile(df.loc[:,x],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)

    df.loc[df[x] < min,x] = np.nan
    df.loc[df[x] > max,x] = np.nan

#Same steps are followed for each input parameter

#Outlier removal using DBSCAN
from sklearn.cluster import DBSCAN
def remove_outliers_DBSCAN(df,eps,min_samples):
    outlier_detection = DBSCAN(eps = eps, min_samples = min_samples)
    clusters = outlier_detection.fit_predict(new_df.values.reshape(-1,1))
    data = pd.DataFrame()
    data['cluster'] = clusters
    return data['cluster']
clusters=remove_outliers_DBSCAN((df['Weight on Bit[kkgf]']),0.5,5)
clusters.value_counts().sort_values(ascending=False)
clusters=remove_outliers_DBSCAN((df['Hookload[kkgf]']),0.5,5)
```

```
clusters.value_counts().sort_values(ascending=False)
clusters=remove_outliers_DBSCAN((df['Surface Torque[kNm]']),0.5,5)
clusters.value_counts().sort_values(ascending=False)
clusters=remove_outliers_DBSCAN((df['Downhole Weight on Bit']),0.5,5)
clusters.value_counts().sort_values(ascending=False)
clusters=remove_outliers_DBSCAN((df['Downhole Torque']),0.5,5)
clusters.value_counts().sort_values(ascending=False)
clusters=remove_outliers_DBSCAN((df['rpm']),0.5,5)
clusters.value_counts().sort_values(ascending=False)
clusters=remove_outliers_DBSCAN((df['Mud Flow Q in[L/min]']),0.5,5)
clusters.value_counts().sort_values(ascending=False)
clusters=remove_outliers_DBSCAN((df['Standpipe Pressure[kPa]']),0.5,5)
clusters.value_counts().sort_values(ascending=False)
clusters=remove_outliers_DBSCAN((df['ROP[m/h]']),0.5,5)
clusters.value_counts().sort_values(ascending=False)
```

Listing A.15: Code Snippet for Outlier removal