Chair of Petroleum and Geothermal Energy Recovery

Master's Thesis

Hydrodynamic prediction of geothermal water injection using machine learning

Redha Hamel, BSc

August 2023

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

AFFIDAVIT

I declare on oath that I wrote this thesis independently, did not use other than the specified sources and aids, and did not otherwise use any unauthorized aids.

I declare that I have read, understood, and complied with the guidelines of the senate of the Montanuniversität Leoben for "Good Scientific Practice".

Furthermore, I declare that the electronic and printed version of the submitted thesis are identical, both, formally and with regard to content.

Date 30.08.2023

Signature Author
Redha Hamel

Redha Hamel
Master Thesis 2023
Geoenergy Engineering

Hydrodynamic prediction of geothermal water injection using machine learning

Supervisor: Univ.-Prof. Ph.D. Keita Yoshioka

Advisors: Dr. Francesco Parisio
Fredrick Titoes Libert

Chair of Petroleum and Geothermal Energy Recovery

dpe

MONTAN UNIVERSITÄT LEOBEN

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor, Prof. Keita Yoshioka, for his guidance during the completion of this Thesis. His knowledge and mentorship have been instrumental in shaping my research and enhancing my understanding of the subject matter.

I am also deeply thankful to Dr. Francesco Parisio, whose expertise in machine learning has been helpful for the analytical framework of this thesis. And Frederick Titoes Libert for his assistance in understanding the field data.

Additionally, I wish to acknowledge the continued encouragement, support, and attention provided by my parents, sister, and girlfriend.

Abstract

To ensure efficient and sustainable reinjection of water into geothermal reservoirs, a proper prediction of the hydrodynamics of the geothermal system is necessary. However, a deep understanding of the reservoir's properties is typically required. This study investigates the use of machine learning to forecast the wellhead pressure of geothermal injection wells using only historical data of wellhead pressure, fluid injection rates, and temperatures. These datasets will be used to construct several machine learning (ML) algorithms, including Multiple Linear Regression (MLR), Random Forest (RF), eXtreme Gradient Boosting (XGBoost), Support Vector Machine (SVM), as well as Artificial Neural Network (ANN) such as Multilayer Perceptron (MLP) and Long Short-Term Memory (LSTM). Water injection data from four injection wells underwent rigorous data cleaning and outlier removal to ensure the fidelity of the results. The performance of the six algorithms was then validated and examined using metrics like the coefficient of determination (R squared), the root mean squared error (RMSE), and the symmetric mean absolute percentage error (sMAPE).

Results show that LSTM outperforms other tested algorithms with a sMAPE of 8.02%, followed closely by XGBoost with 8.28% and MLP with 8.55%. RF and SVM achieved a sMAPE of 10.18% and 10.48%, respectively, while the MLR, used in this thesis as a way to better gauge the upper limit of the models, got a sMAPE of 20.56%. It's essential to note that these results are subjective on the specific dataset used in this study, as well as the particular preprocessing, feature engineering, and hyperparameter tuning.

Future studies might consider implementing other ML techniques or hybrid learning models, offering valuable insights for geothermal optimization. As ML becomes increasingly important in the geothermal energy sector, such exploration is of significant interest.

Zusammenfassung

Um eine effiziente und nachhaltige Rückinjektion von Wasser in geothermische Reservoire zu gewährleisten, ist eine angemessene Vorhersage der Hydrodynamik des geothermischen Systems notwendig. Allerdings wird in der Regel ein tiefgehendes Verständnis der Eigenschaften des Reservoirs benötigt. Diese Studie untersucht den Einsatz von machine learning zur Vorhersage des Brunnenkopfdrucks von geothermischen Injektionsbohrungen, wobei nur historische Aufzeichnungen von Brunnenkopfdruck, Flüssigkeitsinjektionsraten und Temperaturen verwendet werden. Diese Datensätze werden durch verschiedene Algorithmen des machine learning (ML) genutzt, einschließlich Multiple Linearer Regression (MLR), Random Forest (RF), eXtreme Gradient Boosting (XGBoost), Support Vector Machine (SVM) sowie Artificial Neural Network (ANN) wie Multilayer Perceptron (MLP) und Long Short-Term Memory (LSTM). Wasserinjektionsdaten von vier Injektionsbohrungen wurden einer gründlichen Datenreinigung und Entfernung von Ausreißern unterzogen, um die Genauigkeit der Ergebnisse zu gewährleisten. Anschließend wurden die sechs Algorithmen hinsichtlich ihrer Leistung anhand von Metriken wie dem Bestimmtheitsmaß (R-Quadrat), dem quadratischen Mittelwertfehler (RMSE) und dem Symmetrischen durchschnittlichen prozentualen absoluten Fehler (sMAPE) validiert und getestet.

Die Ergebnisse zeigen, dass LSTM die anderen getesteten Algorithmen mit einem sMAPE von 8,02% übertrifft, dicht gefolgt von XGBoost mit 8,28% und MLP mit 8,55%. RF und SVM erreichten einen sMAPE von 10,18% bzw. 10,48%, während die MLR, die in dieser Arbeit als Maßstab für die obere Grenze der Modelle verwendet wurde, einen sMAPE von 20,56% erzielte. Es ist wichtig zu beachten, dass diese Ergebnisse abhängig von dem spezifischen Datensatz sind, der in dieser Studie verwendet wurde, sowie von der speziellen Vorverarbeitung, Merkmalsentwicklung und Hyperparameterabstimmung.

Zukünftige Studien könnten in Erwägung ziehen, andere ML-Techniken oder hybride Lernmodelle einzusetzen, die wertvolle Erkenntnisse für die geothermische Optimierung bieten. Da ML im geothermischen Energiesektor immer wichtiger wird, ist eine solche Erkundung von erheblichem Interesse.

Table of Contents

Chapter 1

Introduction

1.1 Background and context

Geothermal energy is a mature technology. However, despite its vast potential, geothermal energy accounts in 2021 for only a small fraction, approximately 0.3%, of the world's energy mix, as reported by the International Renewable Energy Agency (Taylor et al., 2021). Nevertheless, the International Energy Agency reports that geothermal energy has a promising outlook, with continued growth anticipated by 2026 (IEA, 2021). Despite its favorable outlook, several challenges impede its development, such as the availability and quality of heat reservoirs, economic viability, and the disposal of large volumes of low-temperature water condensate. If not disposed properly, residual geothermal water can have adverse environmental impacts, in particular on soil, vegetation, wildlife, and groundwater resources. Thus, it is crucial to implement sustainable disposal methods that minimize these impacts and ensure the long-term sustainability of geothermal energy development (Bundschuh & Tomaszewska, 2017; Sharmin et al., 2023).

A valuable solution for a sustainable geothermal energy extraction process is to continuously reinject part, if not all, of the produced wastewater back into the geothermal reservoir to lower the amount of residual water that needs to be disposed of (Zarrouk & McLean, 2019). However, this needs to be closely managed not to lower the reservoir temperature and thus decrease the efficiency of the geothermal system. The amount of fluid that can be injected into the reservoir through an injection well depends on its injectivity (Grant & Bixley, 2011). As a result, precise predictions of injectivity changes over time can aid in optimising water injection, adjusting the volume of fluid in the

reservoir, and minimising the amount of water and other fluids released into the environment.

Several studies have been published on injection performance change using different well-test analyses (Nolte, 1988; Yoshioka et al., 2008, 2019). However, these evaluations take time and require a deep understanding of the geological formation and reservoir properties.

1.2 Scope and objectives

This thesis investigates the application of machine learning (ML) to the modeling of the hydrodynamic of geothermal wells. The goal of the modeling is to dynamically adapt the volumes of water reinjected into the reservoir and predict the future wellhead pressures.

Historical injection and pressure data from a geothermal field will be analysed using ML algorithms to identify patterns and trends that can aid in predicting future pressure changes. The process of developing effective ML models involves a series of steps. Firstly, the raw data must be carefully cleaned and organized to ensure that it is of sufficient quality to be used in ML. This process involves finding and handling any missing data, fixing any formatting issues, and normalising the data to get rid of any biases or inconsistencies that might compromise the models' accuracy. Once the data has been pre-processed, it is partitioned into training, test, and validation sets. These sets are used to train, evaluate, and fine-tune the ML models. During this phase, ML algorithms, including Multiple Linear Regression (MLR), Random Forest (RF), Support Vector Machine (SVM), eXtreme Gradient Boosting (XGBoost), Multilayer Perceptron (MLP), and Long Short-Term Memory (LSTM)., are applied and compared to identify the most effective one. Choosing the best algorithm for the given task is a crucial step and requires extensive analysis of the performance metrics. Once the most effective ML algorithm has been identified, it is used to develop a model that can accurately predict pressures based on the injection data. The best machine learning algorithm is then used to create a model that can precisely predict well head pressures based on injection data.

Chapter 2

Literature Review

2.1 Geothermal injection management

The optimal recovery of geothermal resources while minimizing the environmental impacts requires the effective management of geothermal production processes. One critical process is the reinjection of fluids, which serves several purposes, such as resource recovery optimization, residual water disposal, pressure support, and increased thermal recovery (Bundschuh & Tomaszewska, 2017). Achieving these goals and ensuring effective field management requires careful design of the injection system. Despite the advantages of injection, it can be difficult to maintain reinjection capacity, particularly when dealing with a limited reservoir capacity, scaling in surface pipelines, cooling of production wells, and reservoir clogging (Axelsson, 2010). To mitigate these adverse effects, continuous monitoring is necessary. This involves measuring parameters such as wellhead pressure, injection rate, and temperature to be able to detect any changes in the injection system and take appropriate action to maintain optimal performance (Axelsson, 2012).

The wellhead pressure in a water injection well is influenced by several factors(Grant & Bixley, 2011). Firstly, there is the hydrostatic pressure $P_{hydrostatic}$ exerted by the column of water present in the tubing given by equation (1).

$$P_{hydrostatic} = \rho g h \tag{1}$$

Where, ρ is the density of the water, g the acceleration due to gravity, and h the vertical height of the water column.

Then the frictional losses in the tubing are given by equation (2), which depends on the flow rate, the rheological properties of the water, the diameter and roughness of the tubing, and the path of the flow.

$$\Delta P = f * (\frac{L}{\mathrm{D}}) * \frac{1}{2}\rho v^2 \quad (2)$$

With ΔP as the pressure drop, f the Darcy friction factor, L the length of the tubing, D the diameter of the tubing, and v the velocity of the water.

And finally, the formation pressure $P_{reservoir}$ will resist the injection. Therefore, the wellhead pressure will need to overcome this formation pressure to inject water into the reservoir.

The final required wellhead pressure for injection is given by equation (3):

$$P_{wh} = - P_{hydrostatic} + \Delta P + P_{reservoir} \quad (3)$$

Another important parameter that can be estimated is the injectivity index, which provides valuable information about the performance of the injection system and is a measure of the well's ability to accept injected fluid at a given wellhead pressure (Zarrouk & McLean, 2019). It is calculated by dividing the fluid injection rate by the wellhead pressure change as shown in equation (4):

$$II = \frac{\Delta \dot{m}}{\Delta P} \quad (4)$$

where II is the injectivity index in kg/s/kPa, $\Delta \dot{m}$ is the injection rate in kg/s and ΔP is the change in well head pressure in kPa.

Another diagnostic tool to assess the injection performance of a geothermal well is using Hall plot. Hall plot is a graphical representation proposed by R. E. Hall in 1963 (Hall, 1963). It displays how the total amount of water injected and the so-called Hall integral relate to one another. The Hall integral is calculated using the time integral of the wellhead pressure p_{wh} and the average reservoir pressure p_{avg} as shown in equation (5) (James T. Smith & William M. Cobb, 1997).

$$\int_0^t (p_{wh} - p_{avg})dt \quad (5)$$

This graphical representation helps to determine changes in injection conditions for a particular geothermal system. For instance, when there are reservoir obstructions or wellbore plugging, there is a gradual increase in the skin factor, leading to a decrease in the injectivity and an increase in the slope of the Hall plot. On the other hand, if fracture

growth results in an increase in injection rate and a decrease in the slope of the Hall plot, the skin factor will increase as a result of injecting pressure exceeding fracture pressure. Figure 1 illustrates the different conditions of injection on the Hall plot (James T. Smith & William M. Cobb, 1997).

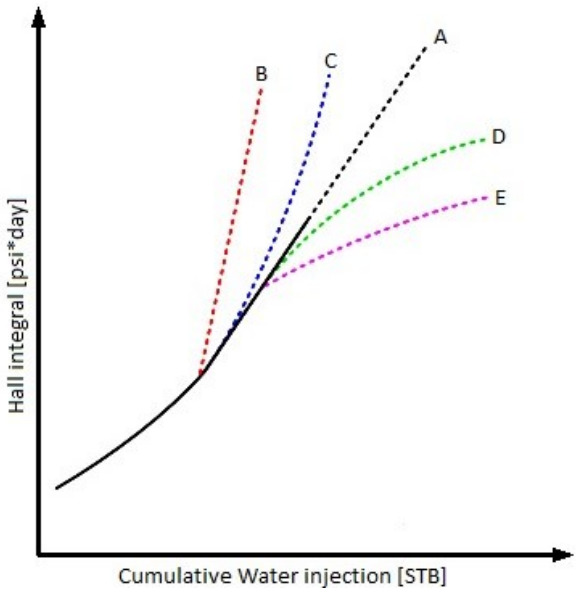

Figure 1 Hall plot for different injection conditions. (A) normal injection with no change, (B) abrupt skin increase due to well plugging, (C) Gradual skin increase, (D) gradual stimulation, skin decrease and enhanced injectivity, (E) Abrupt skin decrease, fracture near the well. Figure adapted from (Boumi Mfoubat & Zaky, 2020).

By monitoring and collecting data related to well injection, operators can effectively use tools like injectivity index estimation and Hall plot interpretation to identify any changes in the well's performance (Bundschuh & Tomaszewska, 2017). With this information, they can take appropriate corrective actions to ensure the well is functioning optimally. However, analysing and monitoring injection well problems using numerical models can be time-consuming and challenging, especially when dealing with large amounts of injection and pressure data. In such cases, ML can be a useful tool for automating the analysis and monitoring process (Harry et al., n.d.). ML algorithms trained to detect patterns and anomalies in the data can be used by operators to quickly identify potential issues with the injection system and take corrective action to prevent them from affecting the performance of the geothermal system.

2.2 Concept and theoretical framework of machine learning

Artificial intelligence (AI) is the ability of machines to carry out tasks that have historically required cognitive skills similar to those of humans, such as understanding natural language, spotting patterns, having the ability to generalise, or remembering past experiences. (Ryszard S. Michalski et al., 1983). As shown in Figure 2, ML is a subclass of AI that uses probabilistic algorithms and statistical models to enable computers to learn from data and make predictions or decisions that improve over time.

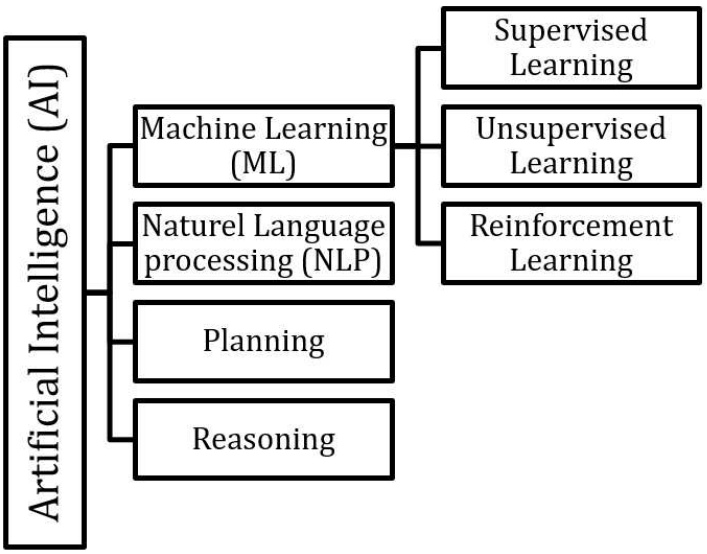

Figure 2 Diagram of the different branches of AI and ML. Figure adapted from (Understand the Basics of Artificial Intelligence, 2021).

Arthur Samuel, a pioneer in artificial intelligence and a computer scientist, described machine learning as the "field of study that gives computers the ability to learn without being explicitly programmed" (Samuel, 1959). Since its inception, ML has grown in popularity across many industries thanks to the emergence of big data and advances in computing power (Attaran & Deb, 2018). Some of the earliest applications of ML include handwriting recognition in the 1970s, speech recognition in the 1980s and email spam filtering in the 1990s. Nowadays, ML has the potential to greatly enhance productivity in many industries and can provide businesses with a competitive advantage by enabling them to make data-driven decisions and improve their operations (Sanil et al., 2022). As technology continues to advance, the potential applications of ML are likely to expand as well. According to a 2021 study by McKinsey, AI and ML are estimated to create 13 trillion US dollars of value annually by the year 2030 (Michael Chu et al., 2021).

According to Mjolsness and DeCoste, (2001), the main steps to create an ML model could be summarized as follows:

1. Data collection: Collect and acquire the relevant data that is needed to solve the problem.
2. Data preparation: Clean and preprocess the data by eliminating noise, addressing missing values, and formatting it appropriately for analysis. Separate the data into a training, test and validation set.
3. Feature selection: Determine and choose the model's most important features.
4. Model selection: Select the appropriate ML algorithm that can be used to solve the problem.
5. Model training: Train the ML model on the training data using the selected method.
6. Performance evaluation: Evaluate the cost function of the trained model by using a suitable metric.
7. Model optimization: Fine-tune the model parameters to optimize its performance using the test data.
8. Model deployment: Deploy the trained model to make predictions on new data or integrate it into a larger system to solve the problem.

Note that the ML process is iterative, and steps 4 to 7 may need to be repeated multiple times until the desired outcome is achieved. The best model can then be chosen by comparing the various generated models to one another using the validation data. As previously stated, there are many types of ML (Figure 2). However, this thesis will only focus on supervised learning.

A model is trained on a labelled dataset using supervised learning, where the right output is provided for each input (Caruana & Niculescu-Mizil, 2006).

Supervised learning is a type of ML, where a model is trained on a labelled dataset, where the right output is provided for each input (Caruana & Niculescu-Mizil, 2006). Figure 3 illustrate the iterations of supervised ML where the overarching goal is to fine-tune the model through such that the difference between the labelled output y and the predicted output $\hat{y}$ diminishes, leading to enhanced prediction accuracy.

Supervised learning is frequently used in classification and regression problems. In all classification problem, the target variables are discrete (Anderson et al., 1983). This indicates that the objective is to classify a data point from the input into one of many predefined categories. This can be (true/false) for binary classification problems or

several possible values for multi-class classification problems. For example, handwriting recognition models can be trained to accurately identify handwritten letters and classify them into their correct equivalent of the alphabet. On the other hand, in all regression problem, the target variables are continuous. In other words, the models look for the best-fit line that can represent the overall trend in data with continuous numerical values or attempt to predict a target value based on a set of input features. This thesis will utilize supervised regression ML models.

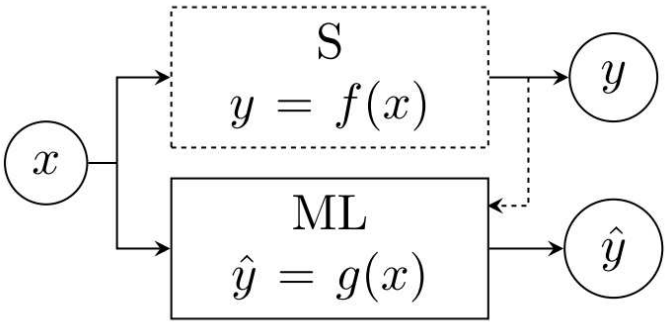

Figure 3 Flow chart of a supervised ML iterations. S is the regular system, x the input, y the labeled output, and ŷ the predicted output.

2.3 Regression metrics

In regression-based ML models, the aim is to minimize the discrepancy between the predicted values and the actual values. This is accomplished by calculating the error between these values (6),

$$Residual\ error\ =\ y\ -\ \hat{y} \tag{6}$$

with $\hat{y}$ as the predicted value and y as the actual values. The error is then calculated as the difference between these values (Borchgrave, 2019). An error of zero, which would indicate perfect predictions for all values, would be the ideal situation but is unlikely to happen.

To determine how well the model can make these predictions, regression metrics, also referred to as loss functions are used. They typically follow the same three-step: first, calculating the distance between predicted and expected target values for each data point; second, normalizing the distance using a chosen method; and finally, combining the normalized distances to obtain an overall measure of model performance. The goal is to find the best parameters which have the smallest error (Botchkarev, 2019).

The four main categories of metrics are primary metrics, extended metrics, composite metrics, and hybrid sets of metrics (Botchkarev, 2019). The choice of regression metrics can depend on the specific problems and the goals of the analysis.

Due to the fact that they offer a straightforward and understandable measure of error, primary metrics are frequently used as a starting point for assessing regression models. Mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE) are the three that are most frequently used.

By including more details about the distribution of the target variable, extended metrics can provide more granular insights into the performance of a model. This is accomplished by adding unique normalisation techniques to the fundamental metrics. For instance, the Root Mean Square Error is normalised by the standard deviation to produce the Normalised Root Mean Square Error (NRMSE). The difference between the data's maximum and minimum values can also be used for normalisation.

Composite metrics can be useful for comparing the performance of different models, as they provide a single summary measure that incorporates multiple performance metrics.

Hybrid metrics can be particularly useful when multiple aspects of model performance need to be evaluated, such as bias and accuracy.

To get a comprehensive understanding of a model's performance, various regression metrics should be used to evaluate the error. To evaluate the effectiveness of the proposed model, this thesis utilizes the coefficient of determination (R squared or $R2$) the root mean squared error (RMSE) and the mean absolute percentage error (sMAPE) as evaluation metrics (Ethem Alpaydin, 2014).

Coefficient of determination (R squared),

$$R^2 = \frac{1}{n}\frac{\sum_{i=1}^{D}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{D}(\hat{y}_i - \bar{y})^2} \quad (7)$$

Root mean square error (RMSE),

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{D}(\hat{y}_i - y_i)^2} \quad (8)$$

Symmetric mean absolute percentage error (sMAPE),

$$sMAPE = \frac{100}{n}\sum_{i=1}^{D}\frac{|\hat{y}_i - y_i|}{|\hat{y}_i| + |y_i|} \quad (9)$$

$\hat{y}$ and y are D dimensional vectors representing the predicted value $\hat{y}_i$ and the actual value y_i . The index i indicates the value on the ith dimension of $\hat{y}$ and y. n is the number of observations and $\bar{y}$ the mean of the actual values y_i.

R2 is the proportion of variance in the dependent variable that is predictable from the independent variables (Ethem Alpaydin, 2014). The value of R2 ranges between 0 and 1. Poor predictions are indicated by an R2 close to 0, while a model that fits the data perfectly is indicated by an R2 close to 1.

The Mean Square Error (MSE) calculate the average square of the errors. Even if the residual error is negative, a positive value is calculated (Ethem Alpaydin, 2014). MSE is less biased towards higher values and does not penalize large errors disproportionately, meaning that MSE is less likely to be affected by outliers present in the dataset. The RMSE and mean squared error share many similarities as RMSE is essentially the square root of MSE (Ethem Alpaydin, 2014). Since each error has an impact on RMSE that is directly proportional to the squared error, outliers can cause results to be exaggerated or magnified. However, RMSE is better at reflecting performance when dealing with large error values, as in Figure 4, where a random noise was added to a sinusoidal curve to simulate a difference between actual and predicted values..

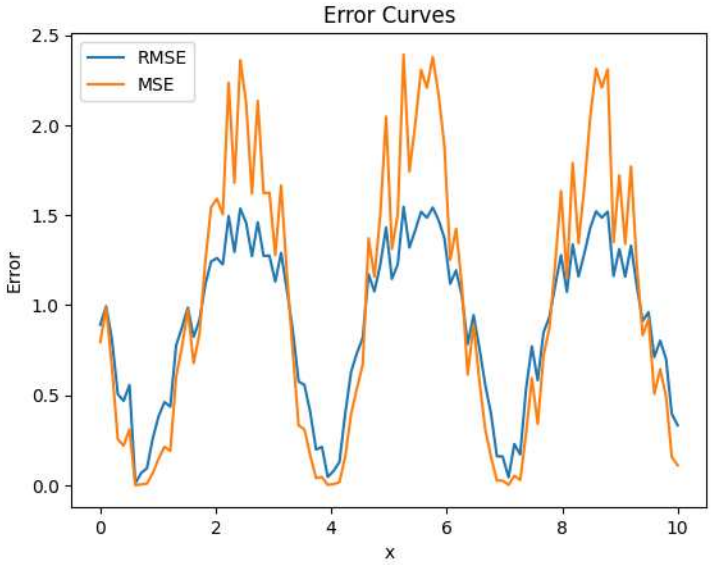

Figure 4 Error metric plot of MSE and RMSE using a noisy sinusoidal signal with.

sMAPE provides the average percentage error between forecasted and actual values. For instance, an sMAPE of 5% indicates that, on average, forecasts deviate by 5% from the actual values. This metric offers a straightforward method to interpret and compare across different scales. Its symmetry avoids the bias that can emerge in using the basic

mean absolute percentage error when actual values approach zero because it treats over-forecasts and under-forecasts equally. A high sMAPE value signifies less accurate forecasts, whereas a low value indicates greater accuracy (Ethem Alpaydin, 2014).

The next section will dive deeper into the architecture of supervised regression ML models, more specifically MLR, RF, SVM and ANNs.

2.4 Regression

Regression is a statistical technique used to assess the relationship between variables. Linear regression is one of the simplest and most widely used statistical algorithms in predictive modelling. According to Fahrmeir et al. (2007), it seeks to identify a linear relationship between a dependent variable and one or more independent variables. When there is a single input variable, the regression is called Simple Linear Regression (SLR). When there are more than one input variables, it's called Multiple Linear Regression (MLR). By solving a linear equation(10), linear regression seeks to determine the "line of best fit" for SLR (Figure 5) and the ideal hyperplane for MLR (Draper & Smith, 1998).

$$y = a_0 + a_1x_1 + a_2x_2 + \cdots + a_nx_n + \epsilon \qquad (10)$$

with y is the dependent variable, x_i the ith independent variables, the intercept a_0 represent the value of y when all other parameters are set to 0, a_n represent the nth regression coefficient of the nth independent variable and ϵ the error term indicating the variability of the dependent variable that is not described by the independent variables.

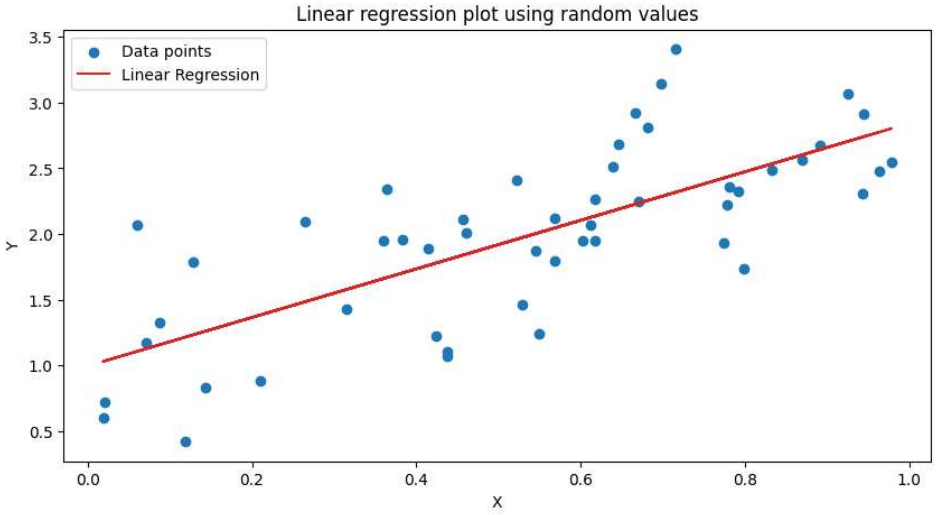

Figure 5 representation of a SLR using random values.

2.5 Decision trees

Decision trees are versatile statistical models that find applications across numerous domains. The concept of decision trees originated from the research conducted by Morgan and Sonquist in 1963, during their research into the factors influencing social circumstances (Clark & Deurloo, 2005). Since then, decision trees have evolved and gained widespread recognition for their effectiveness in data analysis by providing an intuitive approach for solving classification and regression problems by modelling decisions and their consequences in a tree-like structure. As shown in Figure 6Figure 6 decision trees consist of nodes, branches, splits, and leaf nodes. Nodes represent features or attributes, branches represent possible values or decisions, and splits divide the dataset based on these decisions. The leaf nodes are the final predictions or outcomes (Tretter, 2003). The path from the root node to a leaf node forms a sequence of decisions, allowing users to trace the decision-making process.

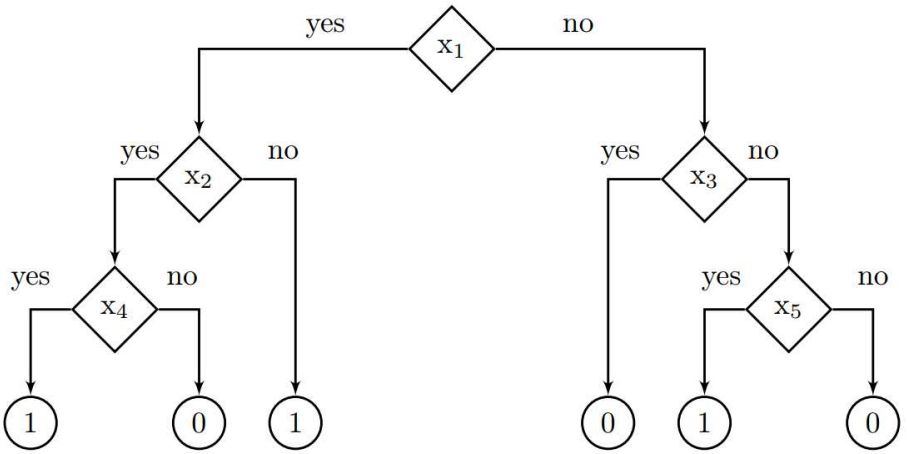

Figure 6 example of a decision tree architecture with x_1, x_2, x_3, x_4 and x_5 as features nodes with yes or no binary splits and 0 or 1 as the leaf nodes.

However, as stated before, decision trees are not only limited to classification tasks but are also applicable to regression problems. The Classification and Regression Trees algorithm, extends decision trees to handle continuous target variables (Breiman, 2001).

The creation of a decision tree involves the consideration of three key parameters. The first parameter is feature selection, which aims to identify the most informative features that effectively discriminate between different classes(Steinberg Dan, 2009).

Once the feature selection process is completed, the next step involves identifying the conditions for dividing the data. The nature of the attribute determines the specific conditions required for splitting:

- Categorical Attributes: In the case of categorical attributes, each possible attribute value gives rise to a separate branch in the decision tree. The data is divided according to the value of the attribute, producing distinct branches that represent the various attribute values.
- Continuous Attributes: Continuous attributes need the creation of split points or thresholds. Optimal split points are determined using various algorithms that evaluate different thresholds, such as binary or multiway splits, based on statistical measures like variance reduction or information gain.

Finally, to ensure effective generalization and prevent overfitting, it is crucial to establish stopping criteria. Overfitting occurs when the model's complexity escalates to a point where it begins learning the noise embedded in the data along with its intrinsic patterns. On the other hand, underfitting occurs when the model's simplicity prevents it from accurately capturing the complexity of the dataset. Using a stopping criterion which determine when to cease expanding the decision tree and convert a node into a leaf node can help to mitigate overfitting. One such criterion is the maximum depth, which places a predetermined limit on the decision tree's depth. When this limit is reached, further splitting is stopped. The minimum samples needed in a node before starting a split is another stopping criterion. By setting a threshold, overfitting can be avoided, and each split can be based on a reliable amount of data. (Steinberg Dan, 2009).

2.5.1 Random forest

Decision trees can be prone to overfitting. This can make them less effective at predicting new, unseen data. In order to strengthen and enhance the performance of decision trees, the RF algorithm was developed (Breiman, 2001). It functions by creating a diverse group of decision trees, like shown in Figure 7, thereby mitigating the impact of the training data on a single tree.

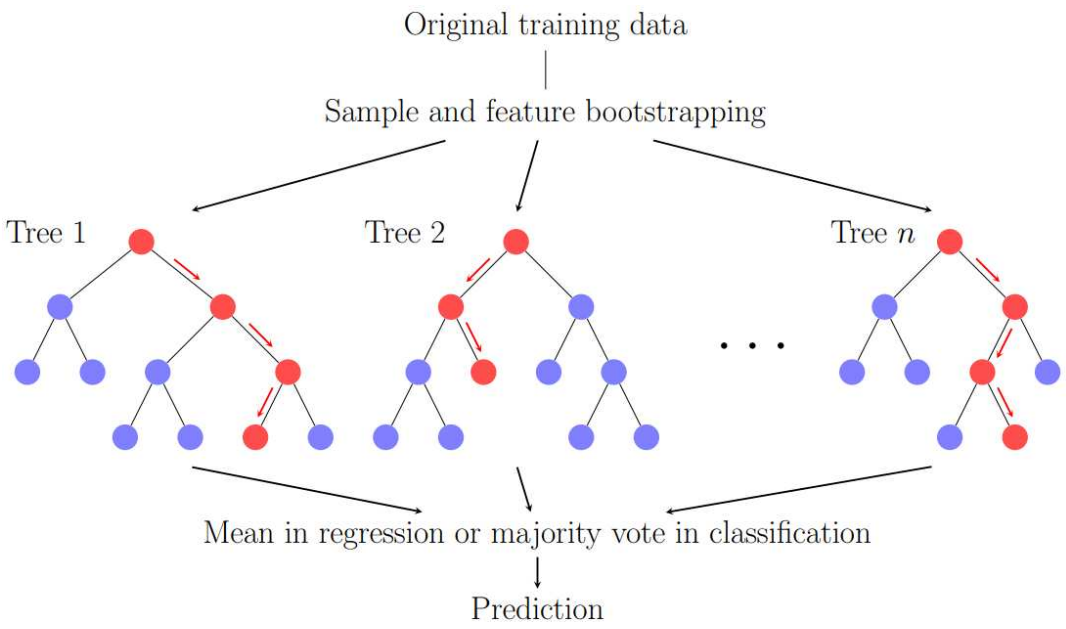

Figure 7 example of the architecture of a Random Forest algorithm with n trees.

The first stage in the RF algorithm involves creating new datasets from the original data. This is done using random sampling with replacement, ensuring the same dimensionality of the dataset. This procedure is known as bootstrapping or bagging (Hastie et al., 2009). An important note is that by using bootstrapping, individual data points may recur multiple times in a single dataset. The following stage involves training decision trees independently from other dataset. However, unlike the conventional decision tree, not all features are used in training. Instead, each tree is trained using a randomly chosen subset of features, enhancing the forest's diversity. The final stage is the prediction. Here, the model uses an aggregation process, in which the data point is successively passed through each tree. Each unique tree prediction is recorded, and then combined to arrive at the final output. The mean is calculated for regression problems and the majority vote is used for classification problems (Steinberg Dan, 2009). By guaranteeing a wider, more varied sampling of the data, this approach lowers the possibility of overfitting and improves the model's capacity for generalisation.

2.5.2 XGBoost

In this thesis an advanced version of decision trees called eXtreme Gradient Boosting (XGBoost) has also been utilized.

Making a prediction, which by default is set to the mean of the target variable, is the first step of the XGBoost algorithm. This default value is then used to compute the residuals, which represent the differences between the observed and predicted values. Following

this, XGBoost fits a regression tree. In contrast to RF, which builds trees one at a time, XGBoost builds trees sequentially, with each new tree attempting to fix the mistakes of the previous one (*Xgboost 2.0.0-Dev Documentation*, n.d.). The quality of the prediction can be quantified using the loss function for regression, as given by equation *(11)* (Christophe Pere, 2020)

$$\sum_{i=1}^{n} L(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 \qquad (11)$$

with y_i the observed values and $\hat{y}_i$ the predicted values.

XGBoost utilizes this loss functions to construct trees by minimizing equation *(*12*)*:

$$\sum_{i=1}^{n} L(y_i, \hat{y}_i) + \gamma T + \frac{1}{2}\lambda O_{value}^2 \qquad (12)$$

In this equation, L stands for the loss function, T for the number of leaves, and γ for a user-definable penalty that encourages tree pruning. Pruning aims to shorten the tree, making it less susceptible to overfitting. Finally, $\frac{1}{2}\lambda O_{value}^2$ is a regularization term where λ a regularization parameter used to control overfitting and O_{value} is the similarity score also called optimal value. The objective of XGBoost is to find splits that maximize the difference in similarity scores between the child leaves. A large difference indicates that the split has effectively separated the data into a distinct group and improving the model performance. (Chen & Guestrin, 2016).

Upon creating the leaves, the gain of splitting the residuals into two groups is evaluated. This gain is equivalent to the sum of the similarity scores of both leaves, minus the root similarity score. This gain value is also used as a way to stop splitting the nodes once it becomes negative(Christophe Pere, 2020). In XGBoost, each tree iteration is built in a greedy way, which means selecting the best split among all potential splits at each node to maximise the gain without taking the global optimal into account.

2.6 Support vector machine

Support Vector Machines (SVM) were originally invented in the 1990s primarily for binary classification tasks (Cortes et al., 1995). Nevertheless, over time, their utility has extended to cover diverse classification and regression problems.

2.6.1 Building a support vector machine

SVM is based on the idea of a decision boundary separating the data points. Considering the simplest case where data is linearly separable, SVM finds the optimal hyperplane that maximizes the margin between two classes as shown in Figure 8.

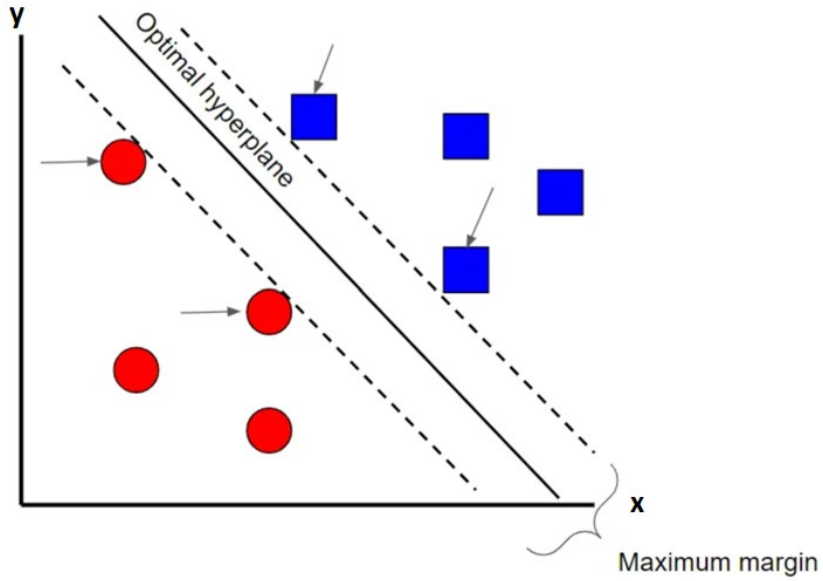

Figure 8 2D binary classification scenario with a hyperplane represented as a straight line with arrows pointing at the support vectors. The data points belonging to one group are in red, while the other group are in blue. Figure adapted from (Terence Shin, 2021).

The so-called support vectors refer to the data points that reside along the margin and help in determining the position of the hyperplane. In the simple linear case, this function takes the form of a plane (Terence Shin, 2021). For non-linearly separable data, SVM employs a technique called the kernel trick, which transforms the input space into a higher-dimensional feature space where the classes can be separated linearly (Schölkopf & Smola, 2018).

To solve regression problems, the same idea behind SVM can be used. This is called Support vector regression (SVR). As shown in Figure 9, SVR algorithm determines a boundary that includes as many data points as possible within that limit. Points outside this threshold are called slacks. SVR can also use the kernel trick to carry out regression in the transformed space, allowing for non-linear regression (Beny Maulana Achsan, 2019)

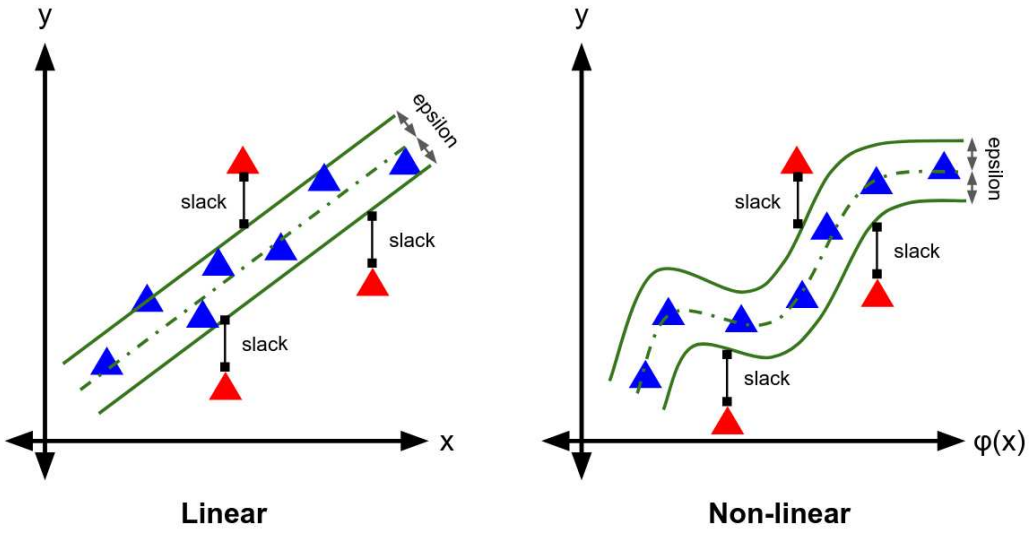

Figure 9 Linear regression (depicted on the left) and non-linear regression (shown on the right), each illustrated with an epsilon-insensitive band and slack points marked in red. Figure adapted from (Beny Maulana Achsan, 2019).

2.6.2 Optimizing a support vector machine

In most linear regression models, the primary goal is to minimize the errors. However, SVR focus mainly on maintaining the error within an acceptable boundary ε like defined in equation (13) (Terence Shin, 2021).

$$|y_i - w_i x_i| \leq \varepsilon \tag{13}$$

The optimization of SVR start by defining an acceptable error threshold ε within the model and efficiently determines an optimal hyperplane, to align with the data. However, this algorithm might not provide suitable results for all data points. It attempts to optimize the objective function, but occasionally, some points might fall outside the established boundary. It is thus crucial to consider the probability of errors exceeding the established ε limit. This requires the use of slack variables, labelled as ξ. These variables capture deviations of any data point that strays beyond the ε limit. These deviations become an added component in the optimization the function (14) (Terence Shin, 2021).

$$|y_i - w_i x_i| \leq \varepsilon + \xi_i \tag{14}$$

The level of error tolerance will vary depending on the specific problem and the level of accuracy required for the solution. For conditions involving higher error tolerance, the epsilon error margin can be increased. Or it can be decreased for models with a lower acceptance of errors.

2.7 Artificial neural network

The original motivation behind the invention of neural networks in the early 1940s by McCulloch and Pitts was to create software that could replicate the learning processes of the biological brain (McCulloch & Pitts, 1943). The human brain is considered to be the epitome of intelligence and surpasses anything that has been developed so far. This is why neural networks were initially created with the aim of imitating the brain. Although, due to our limiting understanding of how the brain actually works, today's neural networks have evolved to become quite distinct from how we perceive the brain to function. However, some of the original biological inspirations still remain in the way we conceptualize ANNs.

2.7.1 Perceptron

Multiple interconnected processing nodes, also known as neurons, make up ANNs, which can be trained to recognise patterns in different types of data. A neuron processes input from other neurons and then generates an output. Each neuron is equipped with n input units x_i along with a bias b_i and a corresponding weight w_i (Ethem Alpaydin, 2014).

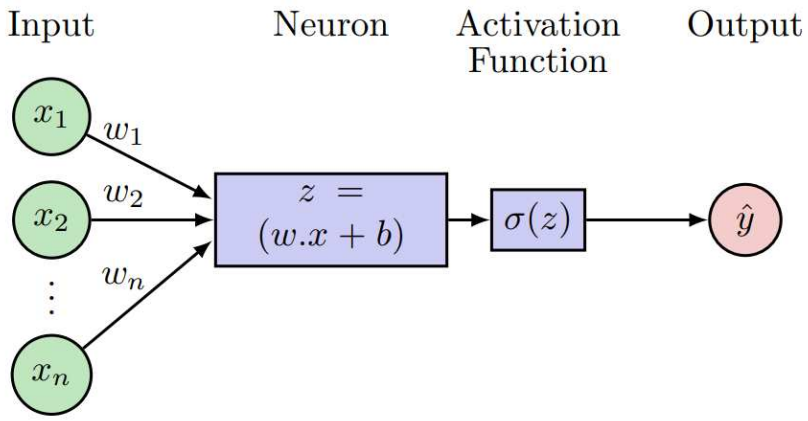

Figure 10 Representation of the inside of a perceptron with input x_i, associated weight w_i, bias b_i, activation function $\sigma(z)$ and output $\hat{y}$.

ANNs consist of input, hidden, and output layers, as shown in Figure 10. The input layer is the first layer of the neural network and receives the raw data. Each unit in the input layer represents one feature of the data and thus the number of units in the input layer is equal to the number of input features. The final output of the neural network is produced by the output layer. The input data is transformed by applying mathematical functions, called activation functions, to the weighted sum of the input values and biases at each neuron (Anderson et al., 1983).

The number of neurons in the output layer depends on the task being performed. For instance, in a binary classification task, the output layer would contain a single neuron that produces a value between 0 and 1 that represents the probability of belonging to the class. On the other hand the output layer of a multi-class classification task would contain numerous neurons, each of which would stand in for a different class. The hidden layers perform the intermediate computations required to map the input to the output.

A perceptron is a basic form of an ANN, composed of a single hidden layer of neurons as shown in Figure 11. This linear binary classifier can be employed to model the relationship between inputs and outputs in a supervised learning model. In a neural network, the procedure of passing input data sequentially through its layers is known as forward propagation. Up until the output layer is reached, each neuron's output from a layer serves as the input for the following layer. The output layer ultimately generates the result, which is based on the information the network has collected.

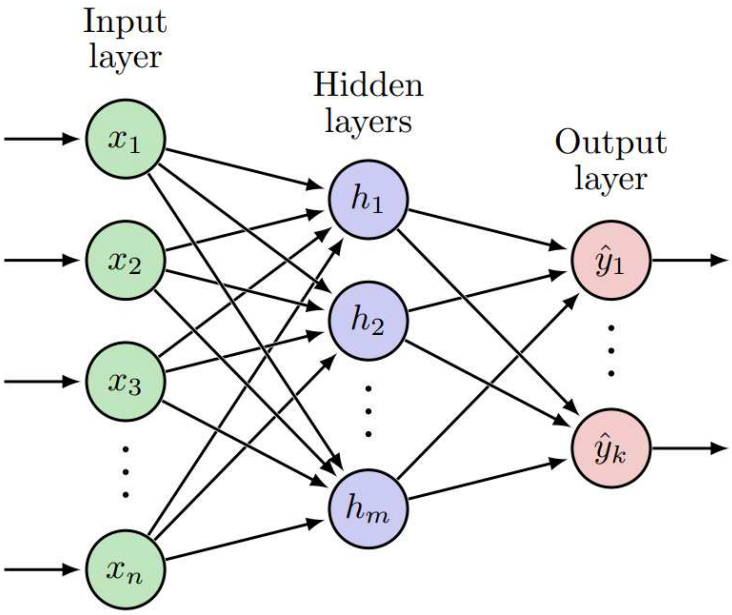

Figure 11 Representation of a feed forward ANN model. With n input values (green), m neurons in the hidden layer (blue) and k output values (red).

The connections between the neurons in different layers are represented by weights. The weights determine the degree to which a given input will affect the neuron's output. If weight w_1 is greater than weight w_2, the input x_1 will apply a greater influence on the output than input x_2. The input value x_i is multiplied by the corresponding weight w_i and added to all the resulting products to produce the output of a neuron(Ethem Alpaydin, 2014).

$$\sum_{i=1}^{n}(w_i.x_i) = wx \quad (15)$$

Then, the bias term b Is added to the sum of multiplied values, the resulting value is represented by z. The bias is needed to shift the activation function to produce the desired output (Ethem Alpaydin, 2014).

$$z(x) = wx + b \quad (16)$$

Finally, to add nonlinearity to the network and enable it to model complex relationships in the data, activation functions are applied to the weighted sum of the input values and biases at each neuron in the network. For example, if a simple binary step function (17) is chosen as the activation function, the output of the function is 1 if the input belongs to class 1, or 2 if it belongs to class 2 (Ethem Alpaydin, 2014).

$$\sigma(z) = \begin{cases} 1 & if \quad z \geq 0 \\ 0 & otherwise \end{cases} \quad (17)$$

Common activation functions are sigmoid (18) and ReLU (rectified linear unit) (19), plotted in Figure 12.

Sigmoid activation function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (18)$$

ReLU activation function:

$$\sigma(z) = max(0, z) \quad (19)$$

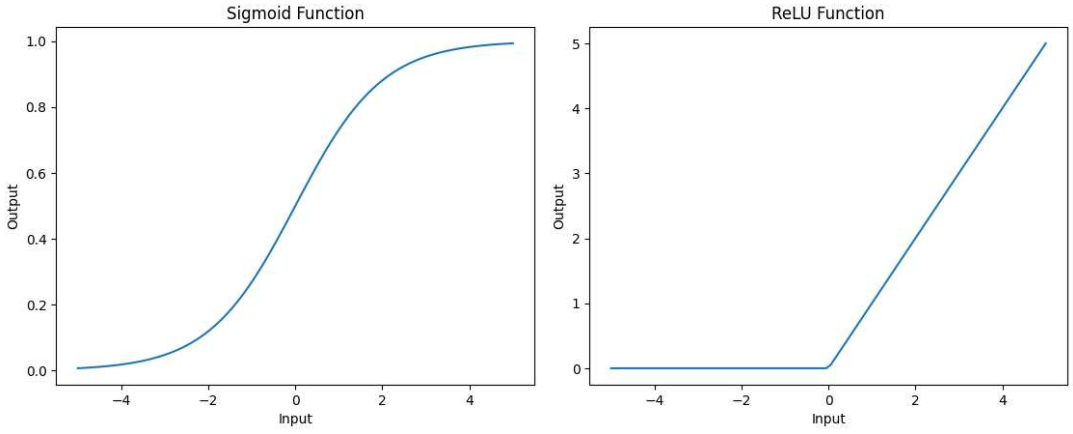

Figure 12 Plot of the sigmoid and ReLU activation Functions

Perceptrons can only learn linearly separable functions, which limits their use compared to more complex ANN models like multilayer perceptrons and other deep learning architectures (Akshay L Chandra, 2018).

2.7.2 Multilayer perceptron

A neural network with more than one hidden layer is called a deep learning model. A deep neural network can learn hierarchical representations of the input data due to the increased number of hidden layers, which can result in forecasts that are more precise. On the negative side, deep learning models frequently need a lot of data and processing power to train well (Ian Goodfellow et al., 2016).

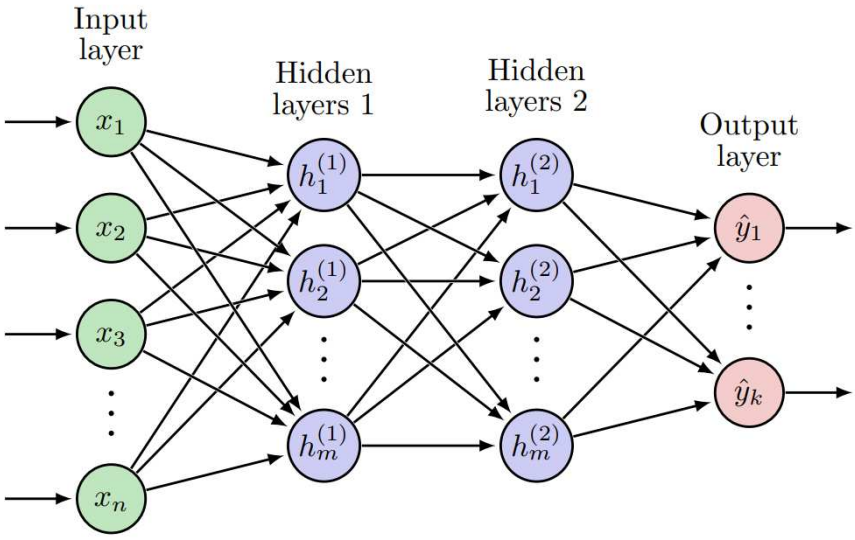

Figure 13 Representation of a multilayer perceptron model using 2 hidden layers. With n input values (green), m neurons in the hidden layer 1 and 2 (blue) and k output values (red).

Am MLP is a type of deep learning model composed of multiple layers of interconnected nodes as shown in Figure 13. Because it is a feedforward neural network, data flows from the input layer through the hidden layers and finally to the output layer in a single direction. Each node in the MLP thus receives input from the nodes in the preceding layer, processes the weighted sum of those inputs using an activation function, and then passes the result to the nodes in the subsequent layer (Anderson et al., 1983).

2.7.2.1 Training a feedforward neural network.

Backpropagation is a widely used algorithm for training feedforward neural networks. This algorithm uses a stochastic gradient descent optimisation algorithm to iteratively adjust the weights and biases of the network after a random initialization (Ethem

Alpaydin, 2014). During each iteration, the algorithm computes the error between the predicted output and the actual output for a given input and propagates this error backwards through the network to update the weights and biases.

The backpropagation algorithm works by computing the gradient of the cost function with respect to the network's weights and biases. According to Them Alpaydin (2014), the cost function calculates the difference between the expected and actual output. For example, using the mean squared error as the loss function to calculate the cost function C across the entire dataset, resulting in:

$$C = MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \quad (20)$$

Then to find the optimal weights and bias the gradients of the cost function is utilized. But as the cost function is not directly dependent on w_i, the chain rule will be used for this purpose:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial w_i} \quad (21)$$

It is possible to simplify the gradient of the cost function C with respect to the predicted value $\hat{y}$ as follows:

$$\frac{\partial C}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{2}{n}\sum(\hat{y} - y) \quad (22)$$

Assuming the use of the Sigmoid activation function σ. the gradient of the predicted value with respect to the z can be broken in to:

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial}{\partial z}\frac{1}{n}\left(\frac{1}{1+e^{-z}}\right) = \sigma(z) * \big(1 - \sigma(z)\big) \quad (23)$$

Finally, the gradient of z with respect to the weight w_i is:

$$\frac{\partial z}{\partial w_i} = \frac{\partial}{\partial wi}\sum_{i=1}^{n}(x_i * w_i + b) = x_i \quad (24)$$

Thus, the outcome is:

$$\frac{\partial C}{\partial w_i} = \frac{2}{\mathrm{n}} * \sum(\hat{y} - y) * \sigma(z) * \big(1 - \sigma(z)\big) * x_i \quad (25)$$

To find the most optimized ML model available, an optimization algorithm needs to be used. The most used optimization algorithm is gradient descent (Ethem Alpaydin, 2014; Ian Goodfellow et al., 2016). Backpropagation is used to iteratively adjust the weights and biases based on the cost function's negative gradient's direction. This process is repeated until the cost function reaches a local minimum or a predetermined number of iterations have been reached.

$$w_{n+1} = w_n - \alpha \frac{\partial C(w_n, b)}{\partial w_n} \quad (26)$$

$$b_{n+1} = b_n - \alpha \frac{\partial C(w, b)}{\partial b_n} \quad (27)$$

w_n and b_n are the nth weight and bias, w_{n+1} and b_{n+1} the new improved weight and bias. The learning rate α is a hyperparameter controlling the range in which the weights and biases are adjusted (Ian Goodfellow et al., 2016). If the learning rate is too high, the algorithm may fail to converge, while a too low learning rate may result in slow convergence.

2.7.3 Recurrent neural network

The recurrent neural network (RNN) is a frequently used neural network for processing sequential data, such as time series or natural language text. As shown in Figure 14, RNNs have recurrent connections that enable them to transfer information from one time step to the next in addition to the feedforward connections that are typical of multilayer perceptrons. This allows the RNN to map not just from an input vector to an output vector, but from the entire history of previous inputs to the output (Ian Goodfellow et al., 2016).

During the forward pass of an RNN, the input vector x with n input units is fed into the network, consisting of m hidden neurons and k output units. However, unlike an MLP, the hidden neurons in an RNN receive additional inputs from hidden layer of the previous time step activation (Ian Goodfellow et al., 2016). The value of the input i at time t is denoted as x_i^t, while a_j^t represent the inputs of node j at time t.

$$a_h^t = \sum_{i=1}^{n} \left(w_{ih} . x_i^t\right) + \sum_{j=1}^{m} \left(w_{j_h} . a_j^{t-1}\right) \quad (28)$$

Once the weighted sum is computed in a neuron, the activation function is applied:

$$h_h^t = \sigma_h(a_h^t) \quad (29)$$

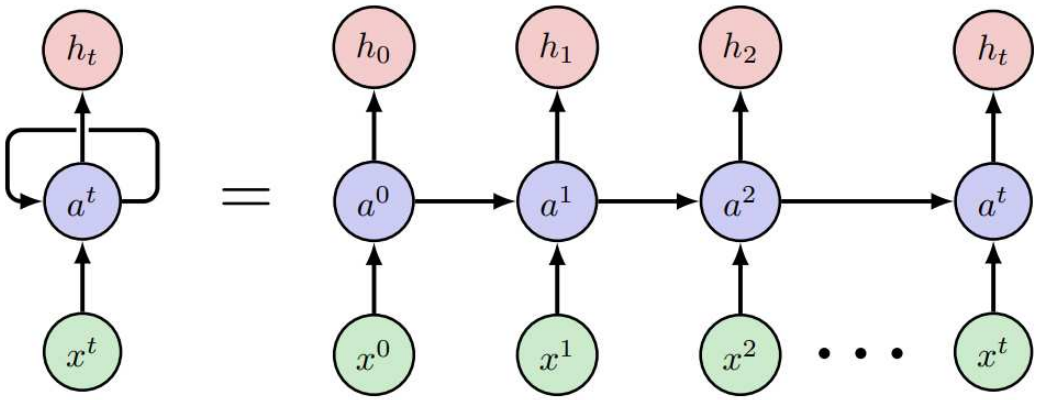

Figure 14 Representation of a Recurrent Neural Network (RNN) neuron with 1 input and output at t iteration.

The RNN requires initial values, which correspond to the network's state before it receives any input. There are various approaches to choosing these initial values. One possibility is to set all of them to zero. The output layer has no recurrent inputs since it is the last layer in the network (Ian Goodfellow et al., 2016).

2.7.3.1 Training recurrent neural network

Just like the multilayer perceptron, RNN measures the difference between the predicted output and the actual output using partial derivatives of the loss function. However, due to its recurrent architecture and dependence on earlier time steps, traditional backpropagation cannot be directly applied (Ian Goodfellow et al., 2016). The output of each neuron in a conventional feed-forward neural network only depends on the input and weights at the same layer, allowing the use of the conventional backpropagation algorithm for training. However, in an RNN, the output of each neuron depends not only on the input and weights but also on the previous hidden state, which introduces a temporal dependency between the layers. As a result, we need to use specialized algorithms such as backpropagation through time (BPTT) to train RNNs (Ian Goodfellow et al., 2016).

BPTT unfold the RNN over time and then applies the usual backpropagation method. However, during this process, the gradients can either shrink too much (vanishing) or grow excessively (exploding), making it hard for the network to learn effectively. (Graves, 2012). To address this issue, more advanced ANN structures have been developed, such as the use of gated recurrent units (GRUs) and long short-term memory (LSTM) algorithms, which can mitigate the vanishing gradient problem and better capture long-term dependencies in sequential data (Graves, 2012).

2.8 Time series modelling

A time series is a group of data points, each of which was recorded at a particular time t. When a time series comprises solely of records for one variable, it is called a univariate time series. On the other hand, if it consists of measurements for multiple variables, it is known as a multivariate time series (Brockwell & Davis, 2016).

Trend, seasonality, cyclicality, and irregularity are the four main categories of time series data. The seasonal component represents a cyclical pattern that happens every year, while the trend component reflects the overall long-term movement of the time series. While irregular components represent random variations, cyclical components are medium-term fluctuations (Adhikari & Agrawal, 2013).

Outlier detection refers to the process of identifying rare or unlikely events. The goal of this process is to decide whether or not the data measured at each time step is an outlier (Boris Iglewicz & David C. Hoaglin, 1993; Cousineau & Chartier, 2010). Typically, an anomaly score is used to measure the dissimilarities between the data and the non-anomalous data. The sample is deemed an outlier if this score exceeds a predetermined threshold (GörnitzNico et al., 2013; Lin et al., 2020). It has been discovered that autoencoders are a useful tool for detecting outliers. Time-series data that needs to be analysed is often high-dimensional, so an autoencoder is used to perform dimensionality reduction and extract the most important features from it (Lopez Pinaya et al., 2020a). An autoencoder is typically unable to reconstruct abnormal data, resulting in a high reconstruction error, which can be used to identify outliers (Lopez Pinaya et al., 2020b).

Chapter 3

Exploratory Data Analysis & Modelling

In data science and statistical analysis, the first step towards understanding the complicated patterns and correlations within a dataset is often the most crucial. This step known as exploratory data analysis (EDA) was first introduced by John Tukey in his seminal work in 1977 (Tukey, 1977). It represents a method that facilitates observing what the data can uncover beyond what modeling or hypothesis testing tasks can give. It involves exploring and understanding the properties of the data through an open-ended process. It forms a critical initial step in any data analysis, providing the context necessary to develop appropriate modeling strategies and helping to highlight potential challenges that might be encountered, such as missing data, outliers or the need for data transformation (Komorowski et al., 2016).

The ultimate goal of EDA is to visualize the data, understand the underlying dynamics of the chosen variables, to extract meaningful information, and to guide the selection of appropriate tools for more detailed analysis. EDA is a critical step to make informed decisions in predictive modeling and ML.

3.1 Data cleaning and pre-processing

The dataset needed to be cleaned and pre-processed in order for the data to be consistent enough for further analysis. Initially, the dataset was imported from an Excel file containing four extensive tables corresponding to the four injection wells, with more that 8000 distinct data points. These well data were represented in various formats and units. The data included the date, which was split into three separate columns as day, month, and year; wellhead pressure (WHP) in psi; injection rate of brine in kilo-pound per hour; injection rate of condensate in kilo-pound per hour; and injection rate of sump water in

kilo-pound per hour. To improve clarity and organization, the tables were first placed into four separate sheets. These sheets were read using the pandas library in Python, and the data from all four sheets were put into four different pandas DataFrames. All further manipulations were done on all four DataFrames at the same time.

The date was reformatted into a standardized 'Day-Month-Year' format to ensure consistency throughout the entire dataset. At the same time, condensate injection rate and swam injection rate columns were combined as they have the same temperature when injected. Extraneous columns were removed, and field units were converted to SI metric units, streamlining the dataset and guaranteeing uniformity. Furthermore, new columns were added to the dataset to indicate the temperature of the injected brine and condensate. The temperature was set to 40°C when condensate was injected and 175°C when brine was injected. Additionally, the injected volume of water was calculated by summing the brine injection rate and condensate injection rate, and the cumulative sum of the injected volumes was computed and incorporated as a new column in the DataFrame.

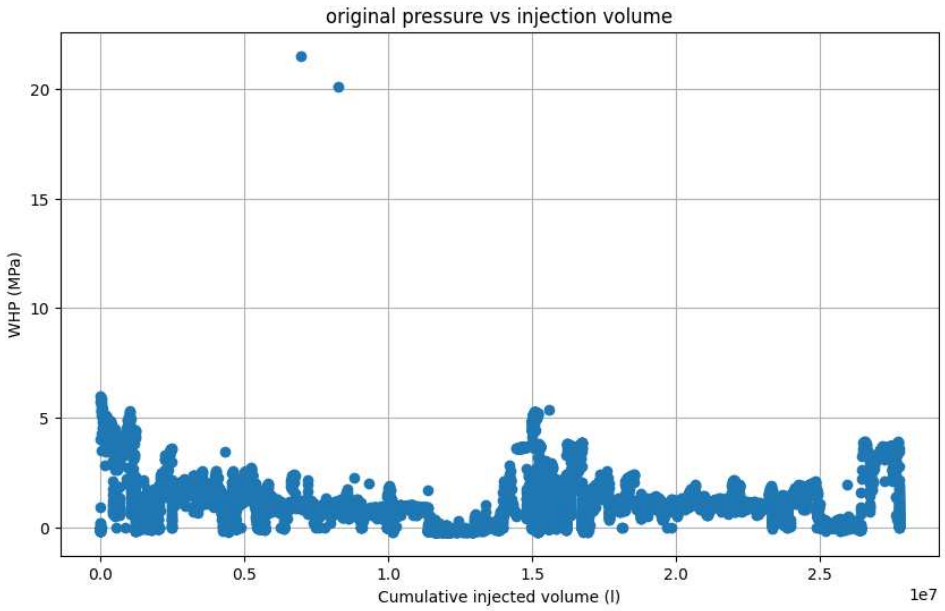

Figure 15 pressure vs cumulative injection volume plot of the four injection wells.

The next step in the data cleaning process is to identify and remove outliers from the dataset, which helps reduce noise and improve overall data quality. Figure 15 illustrate the pressure vs cumulative injection volume plot of the initial data. Outliers can be found and eliminated using a variety of techniques, including the Standard Deviation Method, Z-score method, and Interquartile Range Method. (Lin et al., 2020).

In this thesis, an enhanced approach for outlier detection is proposed by combining the Modified Z-Score Method with an autoencoder, a specific kind of neural network, in order to increase the efficacy and accuracy of outlier detection. The Modified Z-Score Method, a variation of the standard Z-Score Method, offers a robust statistical measure for identifying outliers based on their deviation from the mean. Meanwhile, the autoencoder utilizes its powerful neural network architecture to capture complex data patterns to effectively identify outliers that conventional statistical methods may overlook (Cousineau & Chartier, 2010). Finally, a straightforward and effective method to identify outliers is by utilizing real-world potential values. Understanding that in these specific injection wells, the injection pumps have a maximum rate of 10 MPa, making it possible to flag all pressures greater than 10 MPa as outliers.

3.1.1 Statistical outlier detection

The process of identifying data points in a dataset that significantly deviate from expected or normal behaviour is known as statistical outlier detection (Boris Iglewicz & David C. Hoaglin, 1993). One commonly used method for outlier detection is the Z-score, also known as the standard score. The Z-score Z_i quantifies how far away each data point x_i is from the mean μ of the dataset in terms of standard deviations σ of the dataset (Cousineau & Chartier, 2010; Vishal jain, 2020).

$$Z_i = (x_\mathrm{i} - \mu)/\sigma \tag{30}$$

By calculating the Z-score for each data point, it is possible to determine whether a specific data point is an outlier by comparing its Z-score to a threshold. If the Z-score of a particular data point i exceeds this threshold ($|Z_i| > threshold$), it is considered an outlier. This indicates that the data point is drastically deviated from the mean compared to the rest of the dataset (Boris Iglewicz & David C. Hoaglin, 1993)

The standard Z-Score uses the mean and standard deviation, which are both greatly influenced by outliers. In other words, a single extreme value can significantly alter the mean and standard deviation, and subsequently, the Z-Score. That's why a more reliable method for detecting outliers involves utilizing a modified Z-score. The modified Z-score is computed as follows (Vishal jain, 2020):

$$Z_i = 0.6745 * \left(x_i - \tilde{X}\right)/\mathrm{MAD} \tag{31}$$

$$MAD = \left(\widetilde{|x_\iota - \tilde{X}|}\right) \tag{32}$$

The Modified Z-Score uses the median value of the dataset $\tilde{X}$ and median absolute deviation (MAD) which is the median of the absolute difference between each data point x_i and the median of the dataset. Finally, by multiplying the expression by 0.6745, the Modified Z-Score is scaled to have a similar magnitude as the Z-Score when the data follows a normal distribution. In a normally distributed dataset, approximately 50% of the data points fall within one standard deviation of the mean. The value 0.6745 corresponds to the distance from the mean to the point that separates the middle 50% of the data, called the first quartile. This allows for a more direct comparison between the Modified Z-Score and the Z-Score (Vishal jain, 2020). These calculation differences make the Modified Z-Score more robust to outliers. This means that a single extreme value will have less influence on the Modified Z-Score compared to the traditional Z-Score. Moreover, while the Z-Score assumes that the data follows a normal distribution, the Modified Z-Score is more flexible in this regard. Although it performs optimally on normally distributed data, it can still be utilized for datasets that exhibit some deviation from normality. (Misra et al., 2020).

Similar to the conventional Z-Score approach, the identification of outliers using the Modified Z-Score requires the selection of an appropriate threshold. According to Iglewicz and Hoaglin, a recommended threshold for Modified Z-Scores is ± 3.5 meaning ($|Z_i| > 3.5$) (Boris Iglewicz & David C. Hoaglin, 1993). Any values exceeding this threshold was flagged as potential outliers.

3.1.2 Autoencoder

An autoencoder is an unsupervised learning model that learns to reconstruct its input data. It consists of two parts: an encoder and a decoder (Lopez Pinaya et al., 2020a). The encoder reduces the input data's dimensionality, creating a lower-dimensional representation called the latent space. The decoder then reconstructs the input. In this thesis, a simple feedforward autoencoder is built using TensorFlow and Keras. The model consists of an input layer, an encoding layer, and a decoding layer that reconstructs the input data. The activation functions used are Rectified Linear Unit (ReLU) for the encoding layer and sigmoid for the decoding layer. The model is compiled using the Adam optimizer and MSE as the loss function. These are the most common hyperparameters employed in autoencoders that often yield suitable outcomes (Santiago L. Valdarrama, 2021).

The autoencoder is trained on the normalized dataset for 5 epochs. The number of epochs represents the number of times the autoencoder iterates over the entire dataset during training. Depending on the dataset and the complexity of the autoencoder architecture,

the ideal number of epochs can change (Lopez Pinaya et al., 2020a). The data is shuffled before each epoch to ensure better generalization during training. After training, the autoencoder tries to reconstruct the input data from the lower-dimensional representation it has learned. The reconstruction error (MSE) is computed for each data point by comparing the original normalized data with the reconstructed data from the autoencoder. The idea is that the autoencoder will have a higher reconstruction error for outliers, as these data points are harder to reconstruct accurately (Ian Goodfellow et al., 2016). To establish which data points are deemed outliers, a threshold for the reconstruction error need be specified. The threshold is set based on 98.5% of the reconstruction error, meaning that 1.5% of the data points with the highest reconstruction errors will be considered outliers. This threshold was chosen after several iterations. If data points, such as pressures exceeding the maximum pump capacity of 10 MPa, still appear in the data, it becomes evident that the filtering method is not sufficient in capturing all anomalies. Data points with reconstruction errors above the threshold are identified as outliers. The outliers are then removed from the original data, resulting in a filtered dataset Figure 16 .

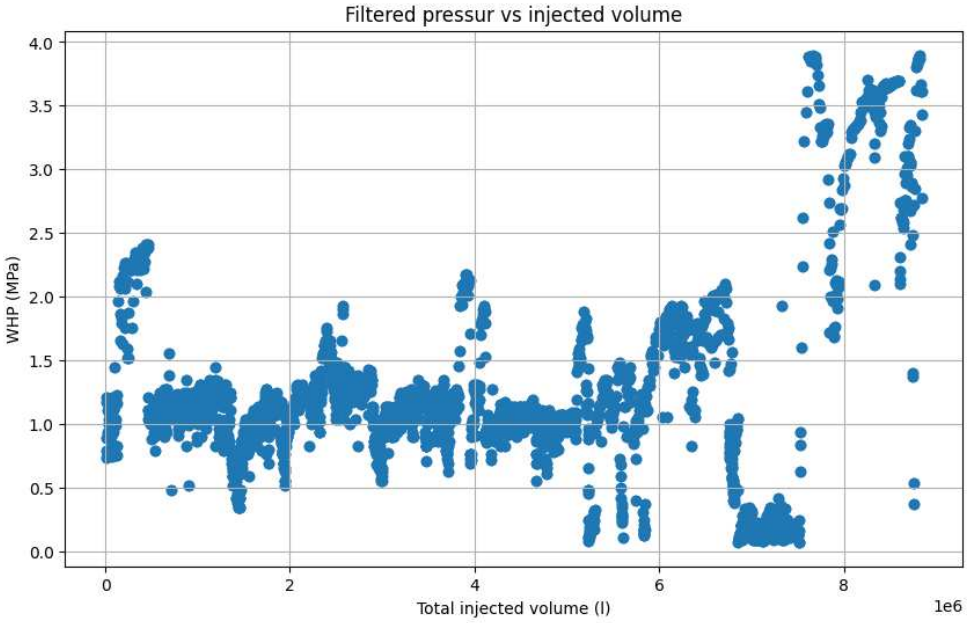

Figure 16 Filtered pressure vs total injection volume plot of the four injection wells after the outliers were removed.

3.2 Feature relationship

Examining the correlation matrices before and after outlier removal provides insight into the relationships between flow rate, well head pressure, and temperature. With 1

denoting perfect positive correlation, -1 perfect negative correlation, and 0 denoting no correlation at all. Outliers, which are data points that differ significantly from others, can skew these relationships. Outliers, being data points radically different from others, can distort these relationships. Once the outliers are eliminated, there is a clear increase in correlation between these variables. The comparison in Table 1 shows the degree to which outliers were influencing the data.

Table 1 correlation matrices between flow rate, well head pressure and temperature before and after outlier detection.

Befor filters	Flow Rate (kg/h)	WHP (MPa)	Temperature (C)
Flow Rate (kg/h)	1	0.18	0.07
WHP (MPa)	0.18	1	0.08
Temperature (C)	0.07	0.08	1
After filters	**Flow Rate (kg/h)**	**WHP (MPa)**	**Temperature (C)**
Flow Rate (kg/h)	1	0.58	0.12
WHP (MPa)	0.54	1	0.17
Temperature (C)	0.12	0.17	1

Finally, the pairplot in Figure 17 and Figure 18, which includes a histogram and scatter plot for Flow Rate and WHP, provides as a visual aid in analysing the distribution of both variables. It clearly illustrates a significant improvement in the normal distribution of the data after the cleaning process, indicating more reliable and representative data. In addition, there's a stronger linear correlation between flow rate and WHP, pointing to a more predictable relationship making accurate predictions more likely.

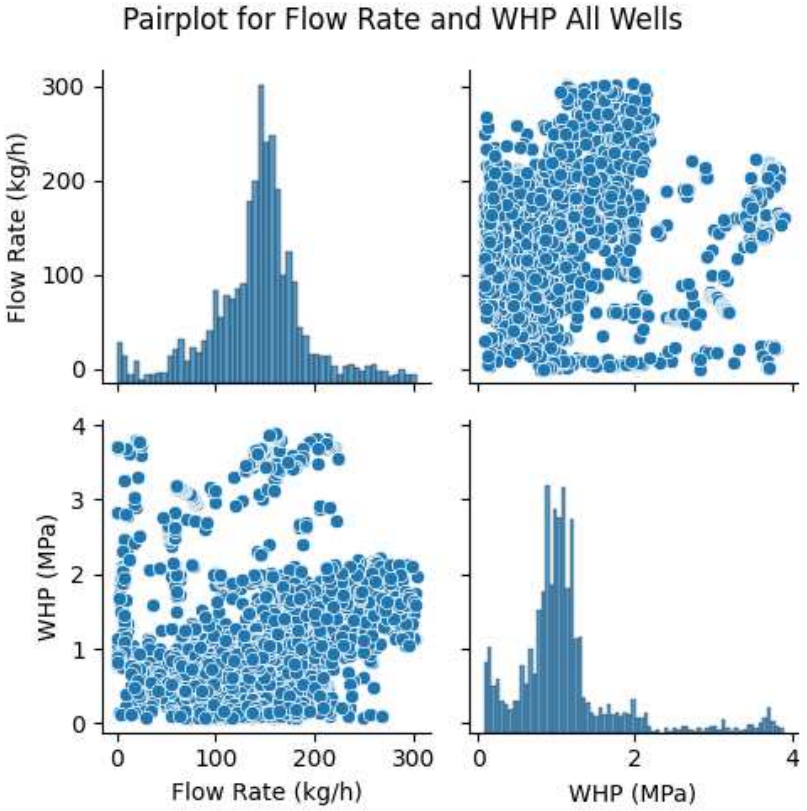

Figure 17 Pair plot Analysis of flow rate and wellhead pressure of the original dataset.

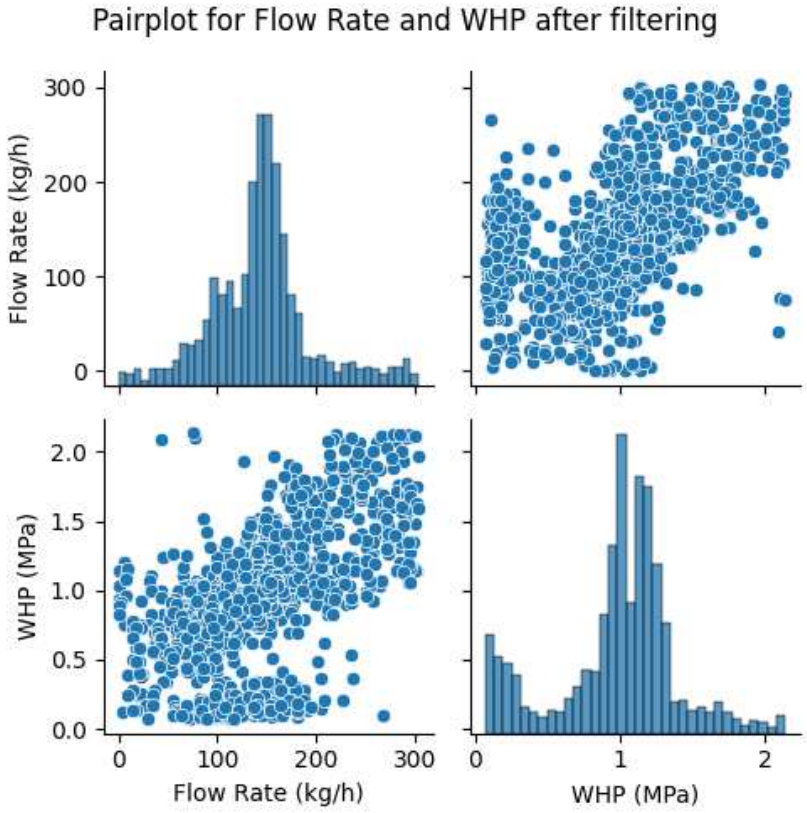

Figure 18 Pair plot Analysis of flow rate and wellhead pressure of the cleaned dataset.

3.3 Machine learning modelling

This chapter will examine each ML model's implementation as well as the model training and hyperparameter tuning processes. The construction of these models was an iterative process, which required a continuous examination between theoretical assumptions, empirical observations, and model performance.

3.3.1 Training the algorithm for machine learning modelling

For MLR, the model was constructed using the `LinearRegression` function available in the Scikit-learn library. After preprocessing the data, the input and output variables are indicated, and the model is fitted using the `fit` function.

The RF model is built using the `RandomForestRegressor` function in Scikit-learn. Similar to MLR, the model is trained using the `fit` function. The number of trees in the forest (`n_estimators`) and the maximum depth of the trees (`max_depth`) are among the important hyperparameters that need to be tuned to optimize the performance of the model(Scikit-Learn 1.3.0 Documentation, n.d.). A larger number of trees generally improves the model's performance but also increases computational complexity. This means more time and resources to train the model. Also, after a certain number of trees the performance stop improving but keeps on increasing the computational cost. The distance from a tree's root to its largest leaf is considered the tree's maximum depth. A higher depth often allows the model to fit more complex patterns, which can lead to overfitting if the model becomes too specific to the training data and will then performs poorly on unseen data.

For the XGBoost model, it was built using the `XGBRegressor` function available in the XGBoost library. Like the RF algorithm, XGBoost also involves some similar hyperparameters that need to be fine-tuned to maximize the performance of the model. These include `n_estimators` (number of trees), `max_depth` (maximum tree depth), `learning_rate` the step size that decides how fast the model learns, and `gamma` the minimum loss required to partition a leaf node. Increasing gamma making it more difficult for the model to overfit the training set, but accuracy suffers as a result. (*Xgboost 2.0.0-Dev Documentation*, n.d.).

The SVM model was built using the `SVR` function, again from Scikit-learn. The `kernel` type and the penalty parameter `C` are hyperparameters that need to be adjusted to enhance the model's predictive performance. The primary role of the kernel is to transform the input data into a higher dimension to make it possible to solve complex

problems that are nonlinear. As explained previously in section 2.6.1 in nonlinear problems the separation of the regions is not limited by a line but rather by a hyperplane. This can be computed by selecting specific kernels like linear, polynomial, or radial basis functions. For this specific problem the radial basis function was used as it is a popular choice for non-linear data and has the ability to create complex regions within the feature space. The penalty parameter C is a parameter that impact the size of the margin. A larger value of C creates a narrower margin and fewer errors, while a smaller value of C creates a wider margin and allow more errors while improving the model's overall consistency.

The ANN model was constructed using the TensorFlow library. Two hidden layers, as well as an output layer, make up the model's three layers. Due to its computational effectiveness and its ability to help with the vanishing gradient problem during backpropagation, the ReLU activation function is utilised (TensorFlow Core: Multilayer Perceptrons, 2023). The output layer has a single neuron corresponding to the predicted WHP. For model optimization, the `Adam` optimizer with a learning rate of 0.001 is utilized. As a method of stochastic gradient descent, `Adam` is easy to configurate and deliver good performance. The model's performance is evaluated using the MSE.

An epoch is an iteration of the entire dataset. An early stopping callback is implemented to prevent overfitting and conserve computational resources. This function keeps track of the validation loss and stop training if no improvement is seen after a predetermined number of epochs, in this case 5. Finally, the `fit` method is used to train the model over a fixed number of epochs. This function generates a `history` object that retains a record of loss and metric values throughout training. This record is used to plot the training and validation loss curves like shown in Figure 19. If the training loss doesn't converge, this indicates that model isn't fitting the training data well, which is a sign of underfitting. This means the model may not be capturing the underlying patterns in the data, leading to unreliable training outcomes. On the other hand, if the validation loss doesn't converge, it indicates that the model may not perform well on new, unseen data, witch is a sign of overfitting. This is because the model might have difficulty generalizing beyond the training data.

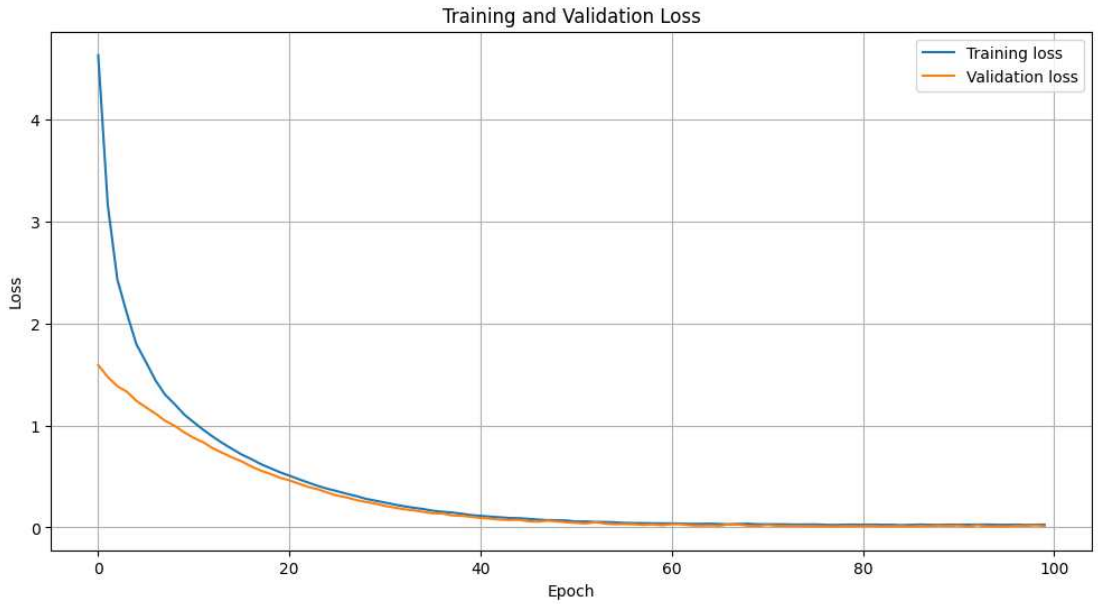

Figure 19 MLR's validation and training loss curves.

Scikit-learn provides the `train_test_split` function to split the dataset into training and testing sets. The training dataset is used to fit all the used models and validate them using k-fold cross validation, and the holdout set is then used as the test dataset for the final evaluation of their performance.

For hyperparameter tuning a `GridSearchCV` function has been used on RF and XGBoost. `GridSearchCV` is a library function from Scikit-learn that helps to loop through predefined hyperparameters and fit the model in the training set to output the different results for each hyperparameter change and be able to select the best hyperparameters values. However the grid search method was computationally expensive, particularly when searching over a large hyperparameter list. A solution was to use for SVM, MLP and LSTM `RandomizedSearchCV`. In randomize search not all hyperparameter values undergo testing. Instead, a fixed number of parameter settings is sampled from the specified distributions, leading to a reduction in computational cost but at the detriment of better hyperparameter tuning optimisation.

3.3.2 Model validation and selection

Model validation is a crucial step in assessing the reliability and performance of ML models. K-fold cross-validation is a widely used method applicable to almost any supervised machine learning model.

K-fold cross-validation offers significant advantages in model validation. Unlike other methods that require a separate validation set k-fold cross validation allows for the full

utilization of the dataset for both training and validation by partitioning the data differently in each iteration. It is however often recommended to have a separate test set outside of k-fold cross-validation to assess the final performance of the model (Matthew Terribile, 2017). The training data is divided into k equal-sized subsets or folds for as shown in Figure 20.

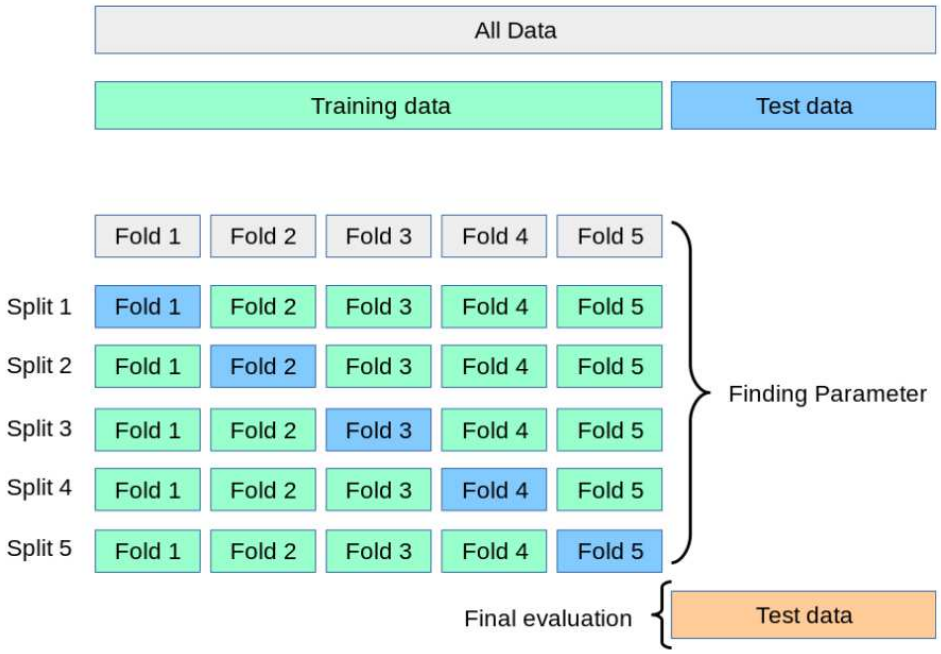

Figure 20 : Example of K-Fold cross validation with 5 iterations. Figure from (Scikit-Learn — 3.1. Cross-Validation, n.d.)

One fold is used as the validation set for each iteration, and the remaining $k - 1$ folds are used for training. This process is repeated k times, with each fold serving as the validation set once. This approach guarantees that the model is exposed to various subsets of the data, providing training for each fold of the training data and a thorough assessment of the model's performance. Another benefit is by averaging evaluation metrics over multiple iterations, it minimizes the effect of random variations present in the dataset (Matthew Terribile, 2017). Typically, the value of k in k-fold cross-validation is set at either 5 or 10. The k value chosen for this study is 5 (Scikit-Learn — 3.1. Cross-Validation, n.d.).

Once the model validation is complete, the next step is model selection by evaluating and comparing the different regression metrics from each model using the test data.

Chapter 4

Results & Discussion

After the construction and training of the different algorithms, the best models were selected and compared against each other to evaluate their performance and reliability.

4.1 Results

Table 2 provides a summary of the 5-split k-fold cross-validation results for all the models using RMSE and R2 as metric. XGBOOST consistently performs the best, with the lowest over all RMSE for each split as well as the highest R2. Followed by LSTM and then MLP, RF and SVP within the same range of RMSE and R2 values across the 5 splits.

As stated in section 2.3 Regression metrics an R2 of 1, an RMSE of 0 and an sMAPE of 0% represent a perfect fit of the model.

Table 2 k-fold cross validation results

	LSTM		MLP		XGBOOST		RF		SVM		MLR	
	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE
Split 1	0.82	0.18	0.96	0.10	0.99	0.02	0.96	0.10	0.96	0.09	0.84	0.18
Split 2	0.99	0.04	0.91	0.08	0.99	0.03	0.96	0.09	0.97	0.08	0.84	0.19
Split 3	0.98	0.06	0.95	0.10	0.99	0.03	0.97	0.09	0.97	0.09	0.80	0.20
Split 4	0.98	0.06	0.95	0.11	0.99	0.03	0.96	0.09	0.97	0.09	0.84	0.20
Split 5	0.99	0.05	0.97	0.07	0.99	0.04	0.96	0.09	0.96	0.09	0.83	0.19

Table 3 shows the RMSE, sMAPE, and R2 result for the test data set. As previously explained the k-fold split was performed in conjunction with a hold-out set before even doing the 5-fold. After model validation, this hold-out set is used as the test data set to assess the model, as shown in Figure 20. This will serve as the model's final assessment and produce a more accurate score for the model, ensuring that it is less prone to overfitting and more accurately depicts its performance.

Table 3 final evaluation using the regression metrics of the test set data.

Regression Metrics	LSTM	MLP	XGBoost	RF	SVM	MLR
R-squared	0.98	0.97	0.97	0.95	0.95	0.83
RMSE	0.06	0.08	0.07	0.10	0.10	0.20
sMAPE	8.02	8.55	8.28	10.18	10.48	20.56

Figure 21 - Figure 26 illustrate the observed versus predicted plot of the test data. This plot is also known as an actual versus predicted plot, which is a representation of the results obtained from the ML models compared to the real values. This plot is a visual overview of how well a model's predictions align with actual values.

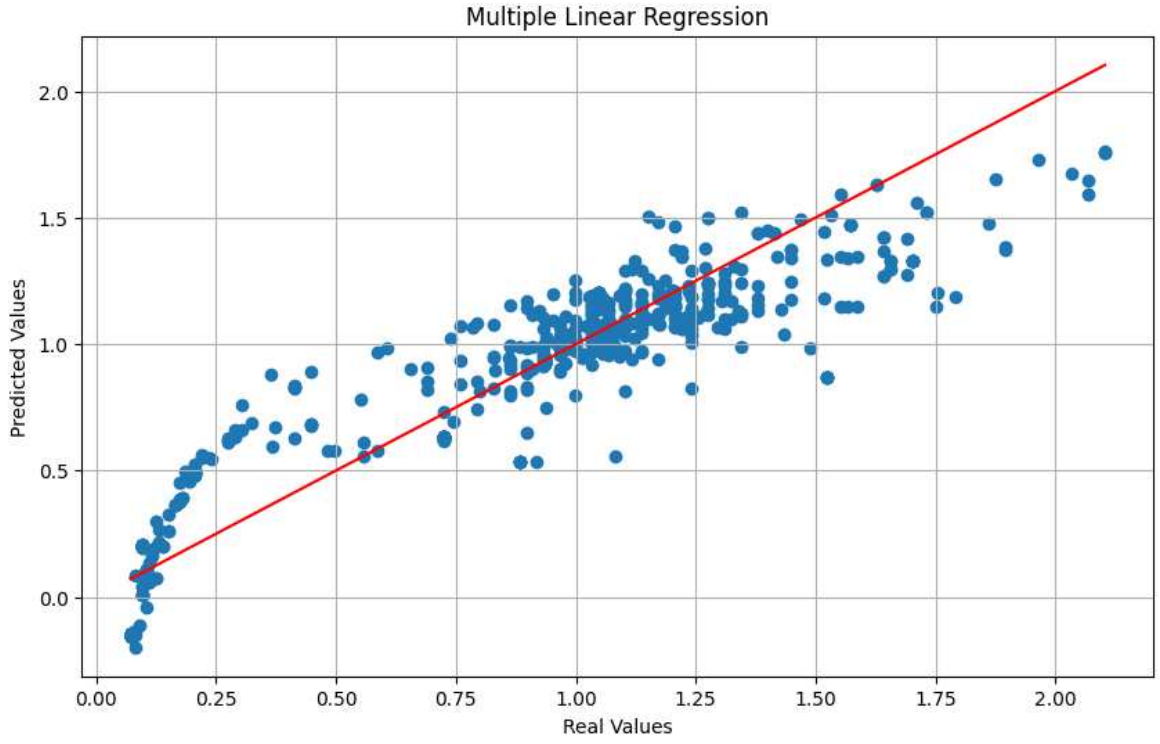

Figure 21 Actual versus predicted plot of the multivariate linear regression model with a sMAPE of 20.56% on the test data

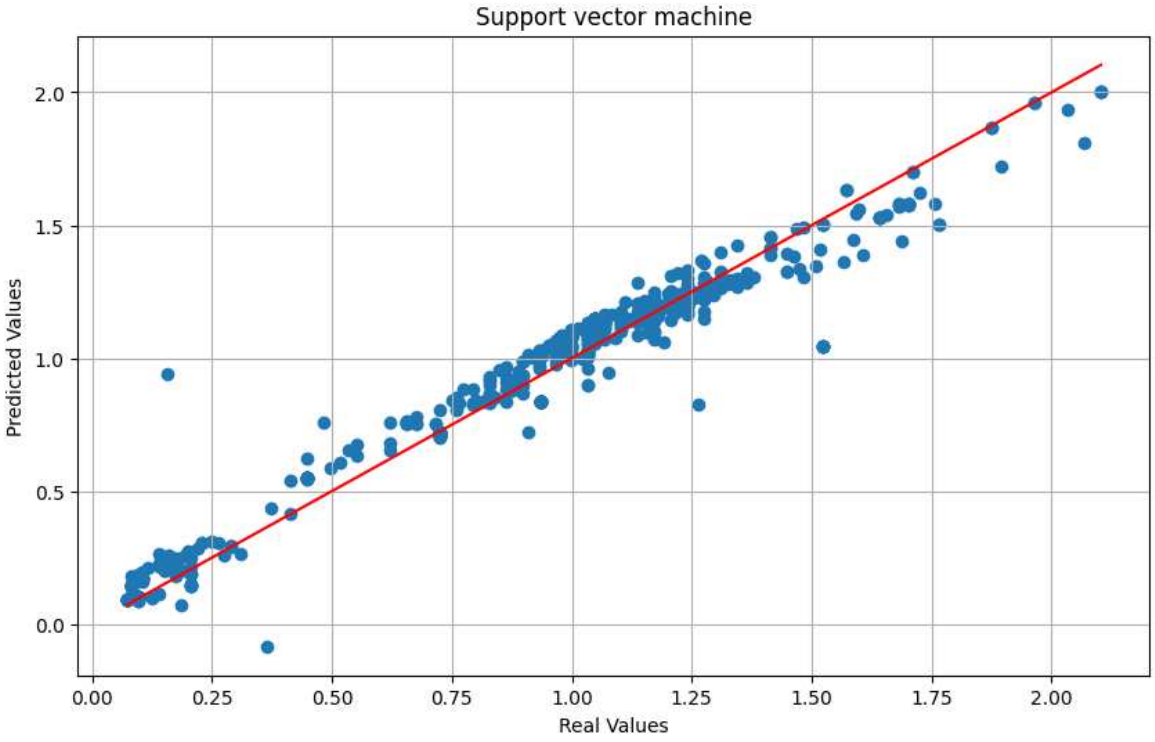

Figure 22 Actual versus predicted plot of the support vector machine model with a sMAPE of 10.48% on the test data

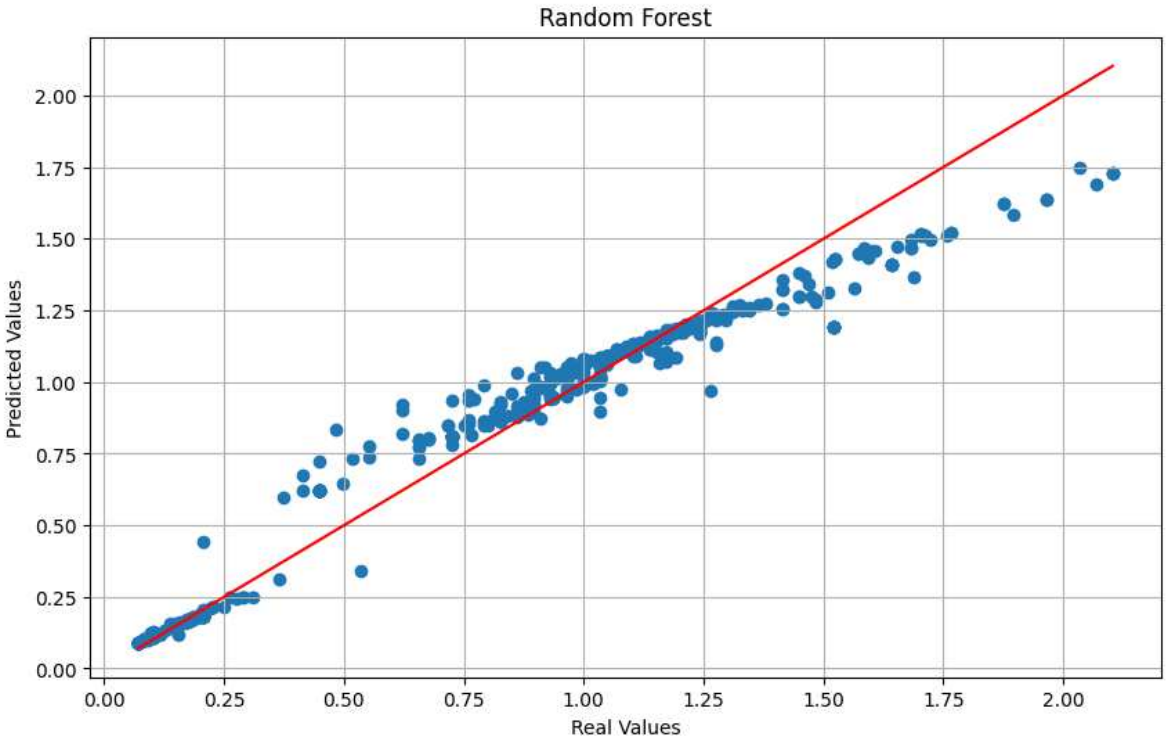

Figure 23 Actual versus predicted plot of the random forest model with a sMAPE of 10.18% on the test data

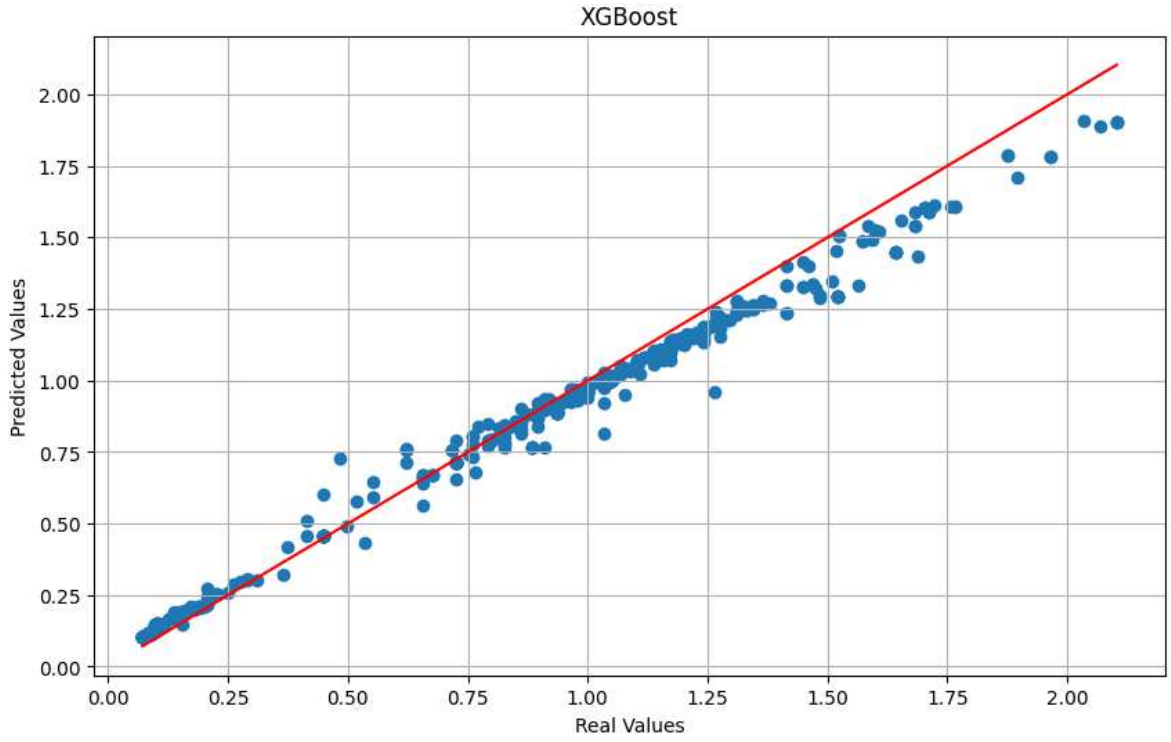

Figure 24 Actual versus predicted plot of the XGBoost model with a sMAPE of 8.28% on the test data

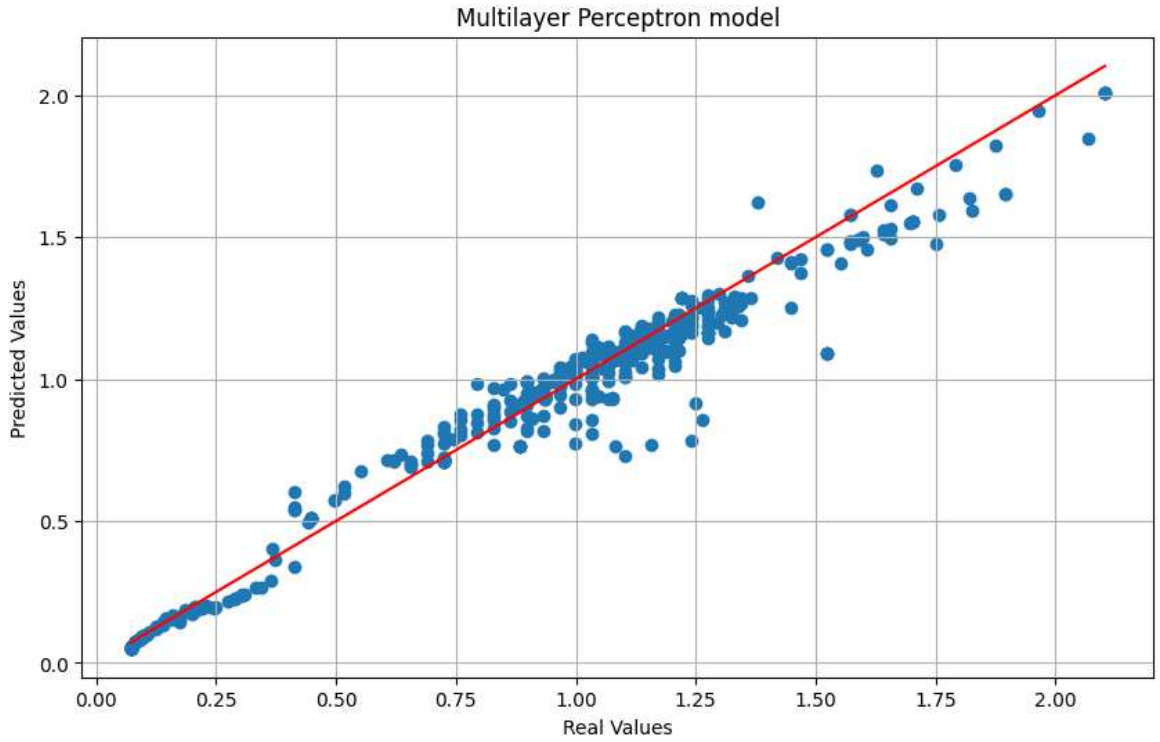

Figure 25 Actual versus predicted plot of the multilayer perceptron model with a sMAPE of 8.55% on the test data

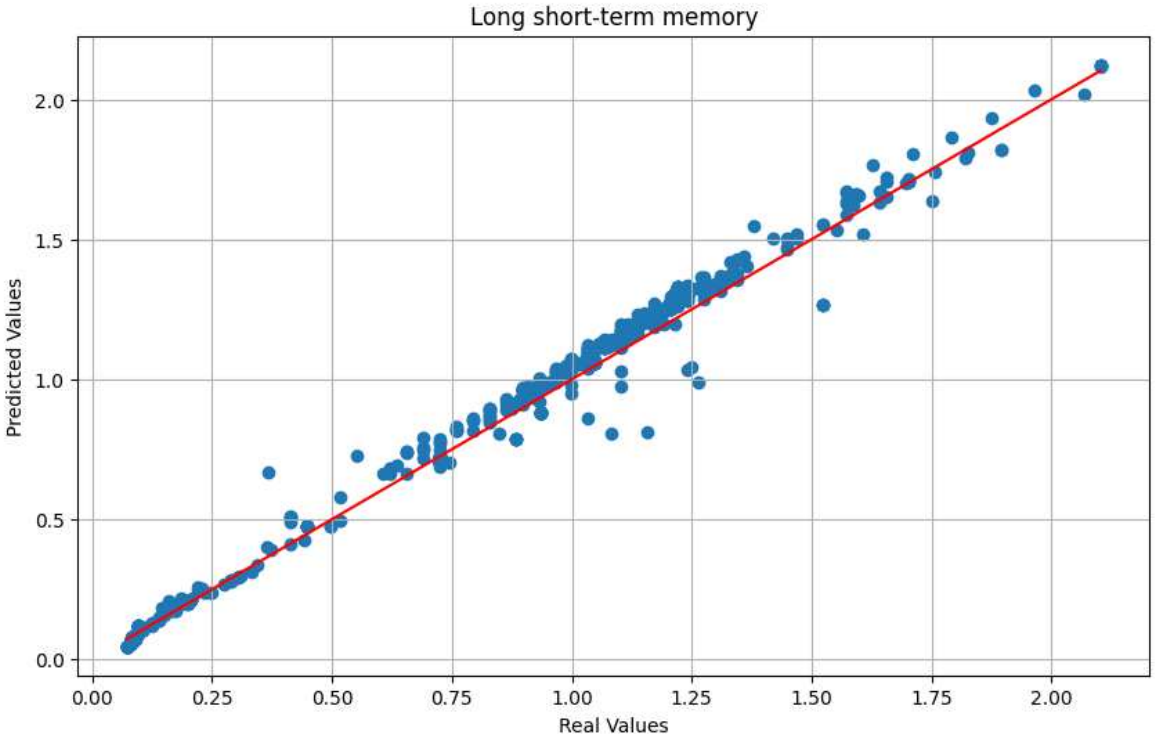

Figure 26 Actual versus predicted plot of the multilayer perceptron model with a sMAPE of 8.02% on the test data

4.2 Discussion

From previous results, all models outperform MLR with its sMAPE of 20,56% in the test data. The MLR model was used mainly as a baseline to compare with other models. As fitting a hyperplane to this data will never be able to produce a low error prediction.

RF and SVM exhibit similar performance, with an sMAPE of approximately 10% in the test data. Tuning SVMs for regression problems can be complicated. Unlike methods that aim to minimize error directly, SVR keep the error within a specified boundary, which is determined by its hyperparameters. This characteristic is evident when comparing the actual versus predicted plots of RF and SVM in Figure 22 and Figure 23. Although both models have comparable error rates, the distribution of the predicted points for SVM is more tightly clustered along the red line than for RF.

The best performing ML models are XGBoost, MLP, and LSTM. Among these, XGBoost consistently outperformed the others during k-fold cross-validation, achieving a consistent R2 of 0.99 and an RMSE of around 0.03. However, its performance on the test dataset was slightly lower, with an R2 of 0.97, an RMSE of 0.07 and an sMAPE of 8.28%.

This discrepancy can be due to over-tuning of the training data and overfitting, so the model is performing exceptionally well during cross-validation but underperforming on the unseen test data. This shows the importance of separating the data into training and testing. While k-fold cross-validation gives a good indication of the model's overall performance on different splits of the training data, it does not guarantee the model's performance on completely unseen data. In contrast to that, the MLP model was performing wors than XGBoost in the validation splits but was only slightly outperformed it in the test data, with an sMAPE of 8.55%, indicating a better generalisation of the model compered to the validation.

LSTM had the lowest sMAPE of 8.02% in the test data. This reinforces the efficacy of LSTM in handling complex data. LSTM stand out in this task because of its intrinsic ability to remember past information. This ability is decisive for accurate forecasting, as it allows the model to make use of past observations to predict future values. However, a challenge associated with optimizing LSTM is hyperparameter tuning. Tuning the hyperparameters of LSTM was difficult and took a lot of computing power. The `GridSearchCV` algorithm was initially employed to attempt to determine the model's ideal hyperparameters, but the computational cost was significant and the program's execution time was too long. So, instead, the `RandomizedSearchCV` was used to reduce these computational costs, making the optimization more time efficient but also less precise. Comparing this process with the implementation of the XGBoost algorithm, which was relatively straightforward and yielded almost similar accuracy levels. For XGBoost, `GridSearchCV` was employed successfully for hyperparameter tuning. Even though LSTM performed slightly better regarding the test data, it can be pretty demanding in terms of computer power. A simpler method like XGBoost might be a good choice since it can deliver almost the same results without using as much processing power. This is of course not an all-encompassing conclusion for every forecasting problem. It is subject to the specific dataset used in this study, as well as the particular pre-processing, feature engineering, and hyperparameter tuning applied.

Ultimately, these results reveal the high efficiency of ML models in WHP prediction. The production data, crucial for the day-to-day operations of the geothermal field, is readily available and was employed to construct these ML prediction models. With meticulous cleaning of this data, its use in ML becomes straightforward. These models serve as a faster, more effective alternative to traditional pressure predictions. In their paper "*The prediction of wellhead pressure for multiphase flow of vertical wells using artificial neural networks*" (2021) Gomaa et al. built an ANN to predict WHP for an oil production well.

They then compared their ANN model with five different numerical correlations used in the oil industry to predict WHP. They found that their ML model vastly outperformed any of the numerical methods shown in Figure 27. The shortcomings of those correlations come from the fact that they were all created in the past. The most recent was in 1985, using small data sets for specific conditions and assumptions that do not apply to all oil fields. ML models can be trained on vast, diverse datasets, allowing them to recognize complex patterns and nuances which traditional models might miss. This gives them a considerable edge.

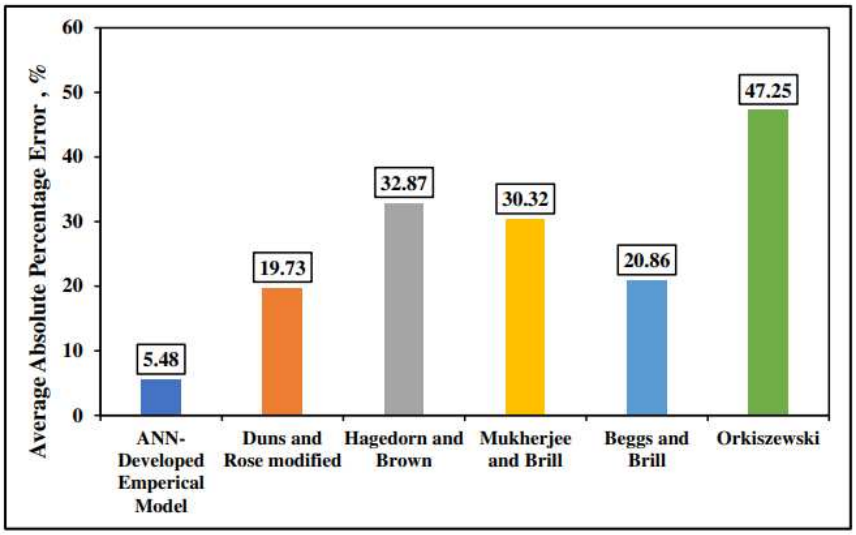

Figure 27 Comparison of the Average Absolute Percentage Error (MAPE) with an ANN model with five other correlations utilized in the oil industry for WHP predictions. Figure from (Gomaa et al., 2021)

Chapter 5

Conclusion

5.1 Evaluation

This thesis highlights the potential of ML algorithms in simulating the hydrodynamics of geothermal injection wells. Utilizing water injection data, comprised of wellhead pressure, flowrate, and temperature, a rigorous data cleaning and outlier removal process was employed. This meticulous preprocessing was necessary to ensure that the ensuing analysis was rooted in high-quality and dependable data. Building upon this, various ML algorithms were tested. The ML model comparison provides a thorough analysis of the benefits and drawbacks of each algorithm. This comparative analysis show that the LSTM model had the highest prediction accuracy with an sMAPE of 8.02%. However, the less computationally heavy model, XGBoost, is a close second with a sMAPE of 8.28%.

It's crucial to emphasise that the production data used in these ML prediction models is openly accessible and necessary for the geothermal water injection operations. Utilizing this dataset for ML is straightforward, provided that the data is rigorously cleaned. Such models offer a swifter, more efficient alternative to conventional numerical models.

5.2 Future work

The efficiency of ML models can often be improved by feature engineering. Feature engineering is the ability of pinpointing the most significant features in a dataset and create them. A more refined feature selection might strengthen the predictive capabilities of all reviewed models.

Another crucial step in model refinement is hyperparameter tuning, which has its own set of difficulties, particularly the cost of computing these optimisations. These Hyperparameters dictate the learning trajectory and significantly influence model efficacy. In particular ANNs, have a lot of tuneable hyperparameters. Tracking an effective

approach to hyperparameter optimisation in future research could potentially lead to significant improvements in these models' predictive accuracy.

This thesis emphasizes the substantial potential of machine learning in the geo-energy industry. Predicting well head pressure in geothermal wells with machine learning models has delivered encouraging results. Future studies would benefit from examining diverse machine learning algorithms and examining the use of hybrid learning models.

References

Adhikari, R., & Agrawal, R. K. (2013). *An Introductory Study on Time Series Modeling and Forecasting*. https://arxiv.org/abs/1302.6613v1

Akshay L Chandra. (2018). *Perceptron Learning Algorithm: A Graphical Explanation Of Why It Works | by Akshay L Chandra | Towards Data Science*. Towards Data Science. https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975

Anderson, J. R. (John R., Michalski, R. S. (Ryszard S., Carbonell, J. G. (Jaime G., & Mitchell, T. M. (Tom M. (1983). *Machine learning : an artificial intelligence approach*. M. Kaufmann. http://www.sciencedirect.com:5070/book/9780080510545/machine-learning

Axelsson, G. (2012). Role and Management of Geothermal Reinjection. *UNU-GTP and La-Geo*. https://rafhladan.is/bitstream/handle/10802/8536/UNU-GTP-SC-14-29.pdf?sequence=1

Beny Maulana Achsan. (2019, December 10). *Support Vector Machine: Regression* . IT Paragon. https://medium.com/it-paragon/support-vector-machine-regression-cf65348b6345

Boris Iglewicz, & David C. Hoaglin. (1993). *How to Detect and Handle Outliers* (Vol. 16). ASQ Quality Press.

Botchkarev, A. (2019). A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, *14*, 045–076. https://doi.org/10.28945/4184

Boumi Mfoubat, H. R. N., & Zaky, E. I. (2020). Optimization of waterflooding performance by using finite volume-based flow diagnostics simulation. *Journal of Petroleum*

Exploration and Production Technology, *10*(3), 943–957. https://doi.org/10.1007/S13202-019-00803-5/FIGURES/17

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324/METRICS

Brockwell, P. J., & Davis, R. A. (2016). *Introduction to Time Series and Forecasting*. https://doi.org/10.1007/978-3-319-29854-2

Bundschuh, J., & Tomaszewska, B. (2017). *Geothermal water management* (1st Edition). CRC Press. https://doi.org/https://doi.org/10.1201/9781315734972

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. *ACM International Conference Proceeding Series*, *148*, 161–168. https://doi.org/10.1145/1143844.1143865

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/2939672

Christophe Pere. (2020). *What is XGBoost? And how to optimize it?* . Towards Data Science. https://towardsdatascience.com/what-is-xgboost-and-how-to-optimize-it-d3c24e0e41b4

Clark, W. A. V., & Deurloo, M. C. (2005). Categorical Modeling/Automatic Interaction Detection. *Encyclopedia of Social Measurement*, 251–258. https://doi.org/10.1016/B0-12-369398-5/00359-5

Cousineau, D., & Chartier, S. (2010). Outliers detection and treatment: a review. *International Journal of Psychological Research*, *3*(1), 58–67. https://doi.org/10.21500/20112084.844

Ethem Alpaydin. (2014). *Introduction to Machine Learning*. MIT Press.

Gomaa, I., Gowida, A., Elkatatny, S., & Abdulraheem, A. (2021). The prediction of wellhead pressure for multiphase flow of vertical wells using artificial neural networks. *Arabian Journal of Geosciences*, *14*(9). https://doi.org/10.1007/S12517-021-07099-Y

GörnitzNico, KloftMarius, RieckKonrad, & BrefeldUlf. (2013). Toward supervised anomaly detection. *Journal of Artificial Intelligence Research*. https://doi.org/10.5555/2512538.2512545

Grant, M. A., & Bixley, P. F. (2011). Geothermal Reservoir Engineering. *Geothermal Reservoir Engineering*. https://doi.org/10.1016/C2010-0-64792-4

Graves, A. (2012). *Long Short-Term Memory*. 37–45. https://doi.org/10.1007/978-3-642-24797-2_4

Hall, H. N. (1963). How to Analyze Waterflood Injection Well Performance. *World Oil*, 128–130.

Harry, M., Aditya Wahyudi, M. F., Midat Al Islam, M. P., Sabrina, N. A., & Situmorang, J. (n.d.). Comparative Study of Decline Curve Prediction in Geothermal Injection Well Using Machine Learning and Wellbore Simulator. *PROCEEDINGS*, *46*.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning* (Second Edition). Springer New York. https://doi.org/10.1007/978-0-387-84858-7

Ian Goodfellow, Yoshua Bengio, & Aaron Courville. (2016). Deep Learning. In *MIT Press*. https://www.deeplearningbook.org/

IEA. (2021). *International Energy Agency Renewables 2021 - Analysis and forecast to 2026*. www.iea.org/t&c/

James T. Smith, & William M. Cobb. (1997). *Waterflooding*. Midwest Office of the Petroleum Technology Transfer COuncil. https://books.google.co.uk/books/about/Waterflooding.html?id=cqx5HAAACAAJ&redir_esc=y

Komorowski, M., Marshall, D. C., Salciccioli, J. D., & Crutain, Y. (2016). Exploratory data analysis. In *Secondary Analysis of Electronic Health Records*. Springer International Publishing. https://doi.org/10.1007/978-3-319-43742-2_15

Lin, S., Clark, R., Birke, R., Schonborn, S., Trigoni, N., & Roberts, S. (2020). Anomaly Detection for Time Series Using VAE-LSTM Hybrid Model. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, *2020-May*, 4322–4326. https://doi.org/10.1109/ICASSP40776.2020.9053558

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020a). Autoencoders. *Machine Learning: Methods and Applications to Brain Disorders*, 193–208. https://doi.org/10.1016/B978-0-12-815739-8.00011-0

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020b). Autoencoders. *Machine Learning: Methods and Applications to Brain Disorders*, 193–208. https://doi.org/10.1016/B978-0-12-815739-8.00011-0

Matthew Terribile. (2017, July 29). *Understanding Cross Validation's purpose* . Medium. https://medium.com/@mtterribile/understanding-cross-validations-purpose-53490faf6a86

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. https://doi.org/10.1007/BF02478259/METRICS

Michael Chu, Bryce Hall, Alex Singla, & Alex Sukharevsky. (2021). *The state of AI in 2021*. https://www.mckinsey.com/capabilities/quantumblack/our-insights/global-survey-the-state-of-ai-in-2021

Misra, S., Osogba, O., & Powers, M. (2020). Unsupervised outlier detection techniques for well logs and geophysical data. *Machine Learning for Subsurface Characterization*, 1–37. https://doi.org/10.1016/B978-0-12-817736-5.00001-6

Mjolsness, E., & DeCoste, D. (2001). Machine Learning for Science: State of the Art and Future Prospects. *Science*, *293*(5537), 2051–2055. https://doi.org/10.1126/SCIENCE.293.5537.2051

Nolte, K. G. (1988). Principles for Fracture Design Based on Pressure Analysis. *SPE Production Engineering*, *3*(01), 22–30. https://doi.org/10.2118/10911-PA

Ryszard S. Michalski, Jaime G. Carbonell, & Tom M. Mitchell. (1983). *Machine Learning, An Artificial Intelligence Approach* (R. S. Michalski, J. G. Carbonell, & T. M. Mitchell, Eds.). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-12405-5

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *1–2*. https://doi.org/10.1147/RD.441.0206

Sanil, H. S., Singh, D., Raj, K. B., Choubey, S., Bhasin, N. K. K., Yadav, R., & Gulati, K. (2022). Role of machine learning in changing social and business eco-system – a qualitative study to explore the factors contributing to competitive advantage during COVID pandemic. *World Journal of Engineering*, *19*(2), 238–243. https://doi.org/10.1108/WJE-06-2021-0357/FULL/PDF

Santiago L. Valdarrama. (2021). *Keras documentation: Convolutional autoencoder for image denoising*. https://keras.io/examples/vision/autoencoder/

Schölkopf, B., & Smola, A. J. (2018). Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. *The MIT Press*. https://doi.org/10.7551/MITPRESS/4175.001.0001

scikit-learn developers — 3.1. Cross-validation: evaluating estimator performance. (n.d.). Retrieved August 26, 2023, from https://scikit-learn.org/stable/modules/cross_validation.html

Sharmin, T., Khan, N. R., Akram, M. S., & Ehsan, M. M. (2023). A State-of-the-Art Review on Geothermal Energy Extraction, Utilization, and Improvement Strategies: Conventional, Hybridized, and Enhanced Geothermal Systems. *International Journal of Thermofluids*, *18*. https://doi.org/10.1016/J.IJFT.2023.100323

sklearn.ensemble.RandomForestRegressor — scikit-learn 1.3.0 documentation. (n.d.). Retrieved August 13, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Steinberg Dan. (2009). *Chapter 10 CART: Classification and Regression Trees*. https://www.researchgate.net/publication/265031802_Chapter_10_CART_Classification_and_Regression_Trees

Taylor, M., Ralon, P., Al-Zoghoul, S., Epp, B., & Jochum, M. (2021). *Renewable Power Generation Costs 2020*. 1–180. www.irena.org

TensorFlow Core | Multilayer perceptrons . (2023). https://www.tensorflow.org/guide/core/mlp_core

Terence Shin. (2021). *A Mathematical Explanation of Support Vector Machines* . Towards Data Science. https://towardsdatascience.com/a-mathematical-explanation-of-support-vector-machines-e433ffe04362

Tretter, M. J. (2003). Data Mining. *Encyclopedia of Information Systems*, 477–488. https://doi.org/10.1016/B0-12-227240-4/00033-2

Tukey, J. W. (1977). *Exploratory data analysis* (Vol. 2). Addison-Wesley.

Understand the basics of Artificial Intelligence . (2021, April 27). https://balayokeshmani.blogspot.com/2021/04/understand-basics-of-artificial.html

Vishal jain. (2020, February 25). *Anomaly detection by Z-Score and Modified Z-Score*. Analytics Vidhya. https://medium.com/analytics-vidhya/anomaly-detection-by-modified-z-score-f8ad6be62bac

xgboost 2.0.0-dev documentation. (n.d.). Retrieved August 13, 2023, from https://xgboost.readthedocs.io/en/latest/python/python_api.html

Yoshioka, K., Izgec, B., & Pasikki, R. (2008). Optimization of Geothermal Well Stimulation Design Using a Geomechanical Reservoir Simulator. *Presented at 33rd Workshop on Geothermal Reservoir Engineering, Stanford, CA*, 28–39.

Yoshioka, K., Pasikki, R., & Stimac, J. (2019). A long term hydraulic stimulation study conducted at the Salak geothermal field. *Geothermics*, *82*, 168–181. https://doi.org/10.1016/J.GEOTHERMICS.2019.06.005

Zarrouk, S. J., & McLean, K. (2019). Geothermal Well Test Analysis: Fundamentals, Applications and Advanced Techniques. *Geothermal Well Test Analysis: Fundamentals, Applications and Advanced Techniques*, 1–349. https://doi.org/10.1016/C2017-0-02723-4

List of Figures

List of Tables

Abbreviations

AI	Artificial intelligence
ANN	Artificial neural network
BPTT	Backpropagation through time
EDA	Exploratory data analysis
GRU	Gated recurrent units
II	Injectivity index
LSTM	Long short-term memory
MAE	Mean absolute error
MAD	Median absolute deviation
ML	Machine learning
MLP	Multilayer perceptron
MLR	Multiple linear regression
MSE	Mean squared error
ReLU	Rectified Linear Unit
RF	Random forest
RNN	Recurrent neural network
RMSE	Root mean squared error
MAPE	Mean absolute percentage error
sMAPE	Symmetric mean absolute percentage error
SVM	Support vector machine
SVR	Support vector regression
WHP	Well head pressure
XGBoost	eXtreme Gradient Boosting