BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering and

Communication

MASTER'S THESIS

Brno, 2024 Ing. Islam Elrefaei, BA

BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

TIME SERIES FORECASTING USING MACHINE LEARNING

TIME SERIES FORECASTING USING MACHINE LEARNING

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — Ing. Islam Elrefaei, BA
AUTOR PRÁCE

SUPERVISOR — doc. Ing. Jiří Hošek, Ph.D.
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study program **Communications and Networking**

Department of Telecommunications

Student: Ing. Islam Elrefaei, BA ***ID:*** 233552

Year of study: 2 ***Academic year:*** 2023/24

TITLE OF THESIS:

Time Series Forecasting Using Machine Learning

INSTRUCTION:

Time series forecasting has been widely used in various fields such as engineering, medicine, and economics. The goal of this diploma thesis is to develop and apply multiple machine learning models on time-series datasets to predict the future, which can be related to economics, networking, etc. For example, the delay in a network can be predicted based on past datasets. The developed algorithms will be verified through numerical simulations as well as implementation in an experimental environment. The preferred programming language for this work is Python.

The first step will be to review the state-of-the-art machine learning models used for time series forecasting. Then, the student will prepare the dataset required for the machine learning models using Python. Next, one selected machine learning model will be applied to the dataset. Once the selected ML model is verified, the student will focus on applying multiple ML models on various time series datasets to predict the future. Based on the analysis of the achieved results, the optimal models will be identified and compared with other models available on the market.

RECOMMENDED LITERATURE:

[1] Shai Shalev-Shwartz, Shai Ben-David, Understanding Machine Learning: From Theory to Algorithms, ISBN:1107057132, Pages: 397, Year: 2014.

[2] Aileen Nielsen, Practical Time Series Analysis : Prediction with Statistics and Machine Learning, ISBN:1492041653, Pages: 400, Year: 2019.

Date of project specification: 5.2.2024 ***Deadline for submission:*** 21.5.2024

Supervisor: doc. Ing. Jiří Hošek, Ph.D.

doc. Ing. Jiří Hošek, Ph.D.
Chair of study program board

WARNING:

Abstract

The aim of this thesis is to explore the application of various artificial intelligence (AI) techniques for the prediction of time series data, which is prevalent in fields such as finance, economics, and engineering. Accurate time series prediction is essential for effective decision-making and planning. This thesis reviews several traditional and state-of-the-art AI techniques used for time series prediction, including linear regression, ARIMA, support vector regression, random forests, and deep learning.

These techniques are applied to different time series datasets, encompassing both univariate and multivariate data. The performance of the predictive models is evaluated using various scalar metrics. The performance of the models was different depending on the type of the dataset. Additionally, this thesis includes the development of a user interface application that allows users to change parameters and forecast new results based on their entries. Furthermore, the thesis discusses the challenges and limitations of using AI techniques for time series prediction and provides suggestions for future research directions.

Keywords

Time Series, Forecasting, Python, Machine Learning

ELREFAEI, Islam. *Time Series Forecasting Using Machine Learning* [online]. Brno, 2024 [cit. 2024-05-21]. Available from: https://www.vut.cz/studenti/zav-prace/detail/153602. Master's Thesis. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Department of Telecommunications. Supervisor Jiří Hošek.

Author's Declaration

Author:	*Islam Elrefaei*
Author's ID:	*233552*
Paper type:	*Master's Thesis*
Academic year:	*2023/24*
Topic:	*Time Series Forecasting Using Machine Learning*

I declare that I have written this paper independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the project and listed in the comprehensive bibliography at the end of the project.

As the author, I furthermore declare that, with respect to the creation of this paper, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation S 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

.

Brno, May 21, 2024 ----------

author's signature

ACKNOWLEDGEMENT

I want to thank my supervisor doc. Ing. Jiří Hošek, Ph.D. and consultant Ing. Nabhan Khatib, Ph.D. from AT&T Global Network Services Czech Republic s.r.o. for providing valuable knowledge, guidance, patience and helpful suggestions. I would like to extend my sincere thanks to my parents for moral and financial support throughout the studies.

Contents

INTRODUCTION

INTRODUCTION

Time series forecasting can aid in comprehending how past data impacts the future, making it a valuable tool. By examining previous data, identifying patterns, and generating short or long-term predictions, this process is accomplished. Time series is considered a special type of data set in which one or more variables are measured over time. This measurement could be daily, monthly, quarterly, or yearly [1].

Artificial intelligence (AI) has become a powerful tool for time series prediction, allowing for the analysis and forecasting of patterns and trends in data. The use of AI for time series prediction has the potential to greatly improve our ability to analyse and forecast patterns and trends. With continued advances in AI and deep learning, we can expect to see even more powerful and accurate models for time series prediction in the future. However, time series prediction using AI is not without its challenges. One major challenge is dealing with missing or incomplete data, as well as handling the large amounts of data that are typically present in time series datasets. Another challenge is dealing with non-stationary data, where the statistical properties of the data change over time. By using preprocessing techniques to adjust the datasets and feature engineering to improve them, we can develop very powerful models for time series forecasting [2].

Mainly using ML (Machine Learning), which is a subset of artificial intelligence, enables computers to learn without being explicitly programmed with predefined rules. One of the biggest features of machine learning algorithms is their ability to improve over time. As larger quantities of data are processed, ML technology can enhance its efficiency and precision. This enables the algorithm to gain more experience, leading to better decision-making and predictions. With the advent of machine learning methods, it becomes more robust and more convenient to deal with the aforementioned difficulties [3].

In this thesis, multiple machine learning models are developed and applied to time-series datasets to predict the future. The datasets can be related to fields such as economics and networking. Several machine learning methods will be utilized, including supervised and unsupervised learning. Moreover, deep learning algorithms will also be employed. Furthermore, the thesis illustrates which groups of algorithms are the most suitable and have the best performance.

1. TIME SERIES

Time series is a special type of dataset in which one or more variables are measured over time. In time series, observations are measured over time, with each data point corresponding to a specific point in time. This means that there is a relationship between different data points in your dataset. This has important implications for the types of machine learning algorithms that you can choose and apply to the time series dataset [4].

There are already many readily available datasets on the internet, and in this master's thesis, multiple datasets will be used to try different models and compare the performance of each model's algorithms, selecting the most suitable one.

1.1 Time Series Main Purposes and Usages

There are many uses of time series analysis, and some of the main applications are:

Forecasting: Time series analysis can be used in fields like finance, where forecasting stock prices and market patterns is crucial for making investment decisions. It estimates future values of a variable based on past observations.
Trend analysis: Time series analysis can identify patterns in data over time. This is a useful tool for identifying shifts in consumer behaviour, market demand, and other elements that affect business performance.
Seasonality analysis: Time series analysis can help identify seasonal patterns in data. This is useful for many industries, such as retail, where sales tend to rise at particular times of the year.
Anomaly detection: Time series analysis can detect anomalies or outliers in data. This is helpful for various purposes, including identifying manufacturing equipment failure and financial transaction fraud.
Control charting: Time series analysis can be used to create control charts that monitor processes over time. This is useful in manufacturing, where monitoring the quality of production processes is critical for ensuring product quality.

In general, time series analysis is a valuable method for comprehending and analysing time-dependent data in a range of fields and applications [5].

1.2 Time Series Forecasting In Machine Learning

Machine learning, like the human brain, acquires knowledge and comprehension through input. The process of machine learning commences with the collection of observations or data, including examples, direct experience, or instruction. It seeks patterns in the data to facilitate future inferences based on the given examples. The primary aim of ML is to allow computers to learn autonomously without human intervention or assistance and adjust actions accordingly. Machine learning forecasting has proven to be the most efficient method for capturing patterns in sequences of both structured and unstructured data for further time series analysis and forecasting.

ML has proven to be valuable because it can solve problems at a speed and scale unattainable by humans. With massive amounts of computational ability behind a single task or multiple specific tasks, machines can be trained to identify patterns and relationships in input data and automate routine processes [6].

2. TIME SERIES ANALYSIS AND PREDICTION

Time series analysis is a statistical method used to analyse time-dependent data collected at regular intervals, enabling observation of past and present data to forecast future trends.

The process of time series analysis encompasses several steps:

- Data collection: Time series data is collected at regular intervals, such as hourly, daily, weekly, or monthly.
- Data preparation: The data is pre-processed to ensure suitability for analysis, including verification for missing values, outliers, and other errors.
- Data analysis: The data is analysed and visualized to identify patterns, trends, and other characteristics.
- Model selection: A statistical model that best fits the data and enables prediction is chosen.
- Model estimation: Using the provided data, the parameters of the selected model are calculated.
- Model evaluation: The performance of the model is assessed using various metrics, including mean squared error, mean absolute error, and others.
- Forecasting: The model is utilized to forecast future values of the time series [7].

Below in the Figure 2.1 it shows the steps of the process from the start to the end.

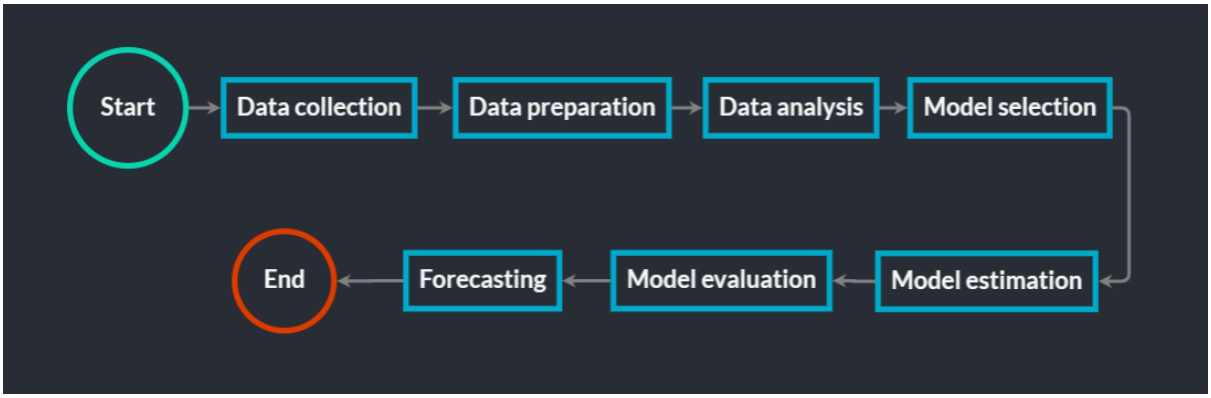

Fig 2.1 : Flowchart of the Time Series analysis

2.1 Time series stationary data

Time series data that is stationary has statistical features like mean, variance, and autocorrelation that don't change over time. The time series' statistical characteristics, in other words, do not change over time.
Because almost every time series models require stationarity, stationary data is crucial in time series analysis. Making predictions based on historical data is made simpler by stationarity, which also streamlines the modelling process.

Stationarity has two types:

Strict stationarity: A time series is strictly stationary if the joint distribution of any collection of time points is not affected by time translation. This indicates that regardless of when the data is obtained, the distribution of the data remains constant. Mathematically, if X_t represents the time series, then for any $t_1, t_2, \dots, t_k$, and any time shift h, the joint distribution of $X_{t1}, X_{t2}, \dots, X_{tk}$, is the same as the joint distribution of $X_{t1+h}, X_{t2+h}, \dots, X_{tk+h}$ [8].

Weak stationarity: If a time series' mean, variance, and autocorrelation remain constant throughout its course, it is considered weakly stationary. This indicates that while the distribution of the data may change over time, the statistical characteristics of the data remain constant.
The main distinction between strict and weak stationarity is that strict stationarity demands that the statistical characteristics of the time series remain constant regardless of the measurement time, whereas weak stationarity permits some change in these characteristics across time.

Non-stationary time series are those that are not constant throughout time. Non-stationary data may indicate trends, seasonality, and other recurring patterns. It may be essential to alter the data in order to remove the trend or seasonality before performing an analysis on non-stationary data [9] [8].

2.2 Time series decomposition

Time Series Decomposition is a technique used to extract multiple types of variation from your dataset.
Seasonality is a recurring movement that is present in your time series variable.
Trend can be a long-term upward or downward pattern and describe increasing or decreasing behaviour of the time series frequently presented in linear modes.
Noise refers to the non-systematic aspect of a time series that deviates from the common model values and cannot be explained by seasonality or trend.
A created dataset could be used as an example to understand how to decompose a time series in Python. The code to import the data is as follows: (Dataset to data frame using Pandas). After that, the decomposition can be done using the Statsmodels' seasonal_decompose function to generate a plot that will split the time series into trend, seasonality, and noise. An example of how it should look like is provided. This decomposition data shows an upward trend and strong seasonality [10].

2.3 Time series models univariate vs multivariate

There are two types of time series models univariate vs multivariate
The **Univariate** time series models are forecasting models that use only one variable (the target variable) and its temporal variation to forecast the future. Univariate models are specific to time series.
(Time series with a one time-dependent variable and a single independent variable) .

If there is other data that will influence the predictions. In this case, **Multivariate** time series models can be used. Multivariate time series models are similar to univariate time series models but are adapted to integrate external variables.
(Time series with one time-dependent variable and more than one independent variable) [11].

In the Table 2.1 briefly describes the differences between The Univariate Model and the Multivariate Model.

Univariate Model	Multivariate Model
Uses only one independent variable	Uses more than one independent variable
Used when the model depends on only a single variable.	Used when other data will influence the predictions.
The model assumes that there is no relationship between the target variable and other variables.	It takes into account the relationships between multiple variables.
Simple and requires less computation and resources compared to multivariate models	More complex and computationally intensive. However, it can offer a deeper understanding of the modelled system

Tab. 2.1: Comparison of Univariate and Multivariate Models

3. TIME SERIES FORECASTING MODELS

There are several classical and modern machine learning methods that differ in many things like accuracy, speed, errors and losses.
The properties of the time series data, the forecasting horizon, the presence of trends and seasonality, and the available processing power all influence model selection.
Several different forms of time series analysis would be discussed below.

3.1 Classical time series models

Typically, these models are only suitable for time series analysis and not applicable to other types of machine learning. They rely heavily on temporal variations within a time series and work well with univariate time series. Some advanced options exist to add external variables into the models as well.
Although they may have prioritized linear relationships, these models are still advanced and exhibit strong performance on a diverse range of problems, assuming that your data is suitably prepared and the method is well-configured.

3.1.1 ARIMA Family

The ARIMA family of models is a set of smaller models that can be combined. Each part of the ARMIA model can be used as a stand-alone component, or the different building blocks can be combined. When all of the individual components are put together, the SARIMAX model is obtained [12].

1. **Autoregression (AR)** Represents the model of the following step in the sequence as a linear equation involving the observations from previous time steps. The notation for the model involves specifying the order of the model p as a parameter to the AR function, The simplest model is the AR(1) model: It utilizes solely the preceding timestep's value to forecast the current value. The maximum number of values that can be used is the total length of the time

series. The method is suitable for univariate time series without trend and seasonal components.

The mathematics equation of an autoregressive model of order p (AR(p)) can be expressed as:

$$y_t = c + \Sigma(\alpha_i * y_{\{t-i\}}) + \varepsilon_t. \quad (3.1)$$

where y_t stands for the dependent variable at time t, c is a constant term, α_i is the dependent variable's lag coefficients up to order p, and ε_t is the error term at time t, which is assumed to have a normal distribution with a mean of zero and a constant variance. Based on y_t 's previous values up to order p, weighted by the associated coefficients, the autoregressive model forecasts its value.

2. **Moving average (MA)** The Moving Average is the second building block of the larger SARIMAX model. It operates similarly to the AR model: it uses past values to predict the current value of the variable. A moving average model is different from calculating the moving average of the time series. The MA model can use multiple steps back in time as well. This is represented in the order parameter called q. For example, an MA(1) model has an order of 1 and uses only one time step back. The method is suitable for univariate time series without trend and seasonal components.

   The mathematics equation of a moving average model of order q (MA(q)) can be expressed as:

$$y_t = \mu + \varepsilon_t + \Sigma(\beta_j * \varepsilon_{\{t-j\}}). \quad (3.2)$$

   where y_t represents the dependent variable at time t, μ is the mean of the series, ε_t is the error term at time t, and β_j are the coefficients of the lagged error terms up to order q. The moving average model predicts the value of y_t based on the past error terms up to order q, weighted by the corresponding coefficients [13].

3. **Autoregressive moving average (ARMA)** it combines the two previous building blocks into one model. ARMA can employ both the value and the forecast errors from the previous instances. ARMA can have different values for the lag of the AR and MA processes. For example an ARMA(1, 0) model has an AR order of 1 (p = 1) and an MA order of 0 (q=0). This is actually just an AR(1) model. The MA(1) model is the same as the ARMA(0, 1) model. Other combinations are possible: ARMA(3, 1) for example has an AR order of 3 lagged values and uses 1 lagged value for the MA.
   The method is suitable for univariate time series without trend and seasonal components.

   The mathematics equation of an Autoregressive Moving Average model of order p and q (ARMA(p,q)) can be expressed as:

$$y_t = c + \Sigma(\alpha_i * y_{\{t-i\}}) + \Sigma(\beta_j * \varepsilon_{\{t-j\}}) + \varepsilon_t. \quad (3.3)$$

   where y_t represents the dependent variable at time t, c is a constant term, α_i are the coefficients of the lags of the dependent variable up to order p, β_j are the coefficients of the lagged error terms up to order q, ε_t is the error term at time t, which is assumed to be normally distributed with mean zero and constant variance. The ARMA model predicts the value of y_t based on its past values up to order p and the past error terms up to order q, weighted by the corresponding coefficients.

4. **Autoregressive integrated moving average (ARIMA)** It represents the subsequent step in the sequence as a linear equation involving the differenced observations and residual errors from previous time steps. It combines both Autoregression (AR) and Moving Average (MA) models as well as a differencing pre-processing step of the sequence to make the sequence stationary, called integration (I). For example, an ARMA(1,1) that needs to be differenced one time would result in the following notation: ARIMA(1, 1, 1).

The first 1 is for the AR order, the second one is for the differencing, and the third 1 is for the MA order. ARIMA(1, 0, 1) would be the same as ARMA(1, 1). This model needs a stationary time series, stationarity means that a time series remains stable. The method is suitable for univariate time series with trends and without seasonal components.

The mathematics equation of an Autoregressive Integrated Moving Average model of order p, d, q (ARIMA(p,d,q)) can be expressed as:

$$\Delta^d y_t = c + \Sigma\left(\alpha_i * \Delta^d y_{\{t-i\}}\right) + \Sigma\left(\beta_j * \varepsilon_{\{t-j\}}\right) + \varepsilon_t. \quad (3.4)$$

where Δ^d represents the differencing operator applied d times to the series, y_t represents the dependent variable at time t, c is a constant term, α_i are the coefficients of the lags of the differenced dependent variable up to order p, β_j are the coefficients of the lagged error terms up to order q, ε_t is the error term at time t, which is assumed to be normally distributed with mean zero and constant variance. The ARIMA model predicts the value of the differenced series at time t based on its past values up to order p, the past error terms up to order q, and the degree of differencing d. The original series can be obtained by reversing the differencing operation [14].

5. **Seasonal autoregressive integrated moving-average (SARIMA)** SARIMA notation is quite a bit more complex than ARIMA, as each component incorporates a seasonal parameter in addition to the standard parameter. The model represents the subsequent step in the sequence as a linear equation involving the differenced observations, errors, differenced seasonal observations, and seasonal errors from previous time steps. It combines the ARIMA model with the ability to perform the same autoregression, differencing, and moving average modelling at the seasonal level. For example, let's consider the ARIMA(p, d, q) as seen before. In SARIMA notation, this becomes SARIMA(p, d, q)(P, D, Q)m.

6. **Seasonal autoregressive integrated moving-average with exogenous regressors (SARIMAX)** It's the most complex variant. It regroups Autoregressive AR, Moving Average MA, differencing, and seasonal effects. And it adds the X: external variables.
   So it is considered as a significant advancement in time series forecasting because of its capacity to adapt to both external forces and seasonal patterns. SARIMAX captures complicated temporal dynamics better than ARIMA since it includes seasonal components and exogenous variables. Because of its adaptability, SARIMAX can forecast time series data that change seasonally and are impacted by outside variables with greater accuracy and dependability. SARIMAX is the best option in the ARIMA group for tackling real-world forecasting difficulties since it gives analysts and researchers an effective tool for modelling and predicting a broad range of time series phenomena.

3.1.2 Vector autoregression (VAR) and its derivatives VARMA and VARMAX

Vector Autoregression, or **VAR** as a multivariate alternative to Arima. So instead of predicting **one** dependent variable, it can predict **multiple** time series at the same time. This can be especially useful when there are strong relationships between your different time series [15].

1. The **VARMA** model is the multivariate equivalent of the ARMA model. VARMA is to ARMA what VAR is to AR: it adds a Moving Average component to the model.
2. **VARMAX** The X represents external (exogenous) variables. Exogenous variables are variables that can help your model to make better forecasts, but that do not need to be forecasted themselves.
3. More advanced versions, such as seasonal **VARMAX (SVARMAX)**, do exist, but they can become quite complex and specialized. It can be challenging to find efficient and user-friendly implementations for these models. When models become overly complex, it may become difficult to understand their inner workings, and it is often better to explore other, more familiar models.

3.2 Supervised models

Supervised models are a family of models that are used for many machine-learning tasks. A supervised machine learning model employs well-defined input variables and one or multiple output (target) variables. The main difference between Classic models and Supervised models is that they consider that variables are either dependent variables or independent variables. Dependent variables, or target variables, are the variables that you want to predict. Independent variables are the variables that help you to predict. And supervised learning models can be divided into two groups **Regression** and **Classification** [16].

3.2.1 Linear regression

It's the simplest supervised machine learning model. Linear Regression estimates linear relationships: Each independent variable possesses a coefficient that reflects its impact on the target variable.

3.2.2 Simple Linear Regression (SLR)

is a Linear Regression in which there is only one independent variable. An example of a Simple Linear Regression model in non-time series data could be the following: hot chocolate sales that depend on the outside temperature
The model should relate between two variables, the independent variable (often denoted as x) and the dependent variable (often denoted as y) [17].

The mathematical equation for a simple linear regression model is:

$$y = \beta_0 + \beta_1 x + \varepsilon. \quad (3.5)$$

where:
y is the dependent variable (or response variable)
x is the independent variable (or predictor variable)
β_0 is the intercept (or constant)

β_1 is the slope (or regression coefficient)

ε is the error term (or residual)

The relationship between the independent variable x and the dependent variable y is represented by a straight line in the equation. The value of y, when x is equal to zero, is represented by the intercept, while the slope indicates the change in y for each unit change in x, and we can use the regression equation to make predictions about the dependent variable based on new values of the independent variable [18].

3.2.3 Multiple Linear Regression (MLR)

Rather than using only one independent variable, multiple independent variables are employed. It's like converting a 2D graph into a 3D graph, where the third axis represents the variable Price. In this case, a linear model that explains sales using temperature and price is built. As many variables as needed could be added.

However, in this example, the dataset is not a time series. Therefore, it should be slightly modified to employ this technique for time series data, incorporating variables such as year, month, day of the week, etc. [19].

In multiple linear regression, where there are more than one independent variable, the equation is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon. \quad (3.6)$$

Where $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, ..., $\boldsymbol{x_n}$ are the n independent variables and $\boldsymbol{\beta_1}$, $\boldsymbol{\beta_2}$, ..., βn are their corresponding regression coefficients [18].

3.2.4 Decision Tree (DT)

A decision tree is a simple algorithm with a tree-like structure used for both classification and regression tasks. It has a hierarchical tree structure consisting of a root node, branches, internal nodes, and leaf nodes. The decision tree begins with a root node that does not have any incoming branches. Outgoing branches from the root node

feed into internal nodes, also known as decision nodes. Leaf nodes represent all possible outcomes within the dataset, and the root node represents the entire dataset.

The decision tree model is not represented by a single mathematical equation. Instead, it is a hierarchical structure consisting of nodes and branches representing decision rules based on the values of input features.

In the decision tree model, nodes and branches represent decision rules based on input features. Each internal node corresponds to a feature or attribute, and each branch represents a decision rule based on the value of that feature. The decision tree recursively partitions the data into smaller subsets based on the values of the input features until a stopping criterion is met.

Once the decision tree reaches a leaf node, it provides a prediction for the corresponding input features. The prediction can be a single value, such as in regression trees, or a class label, such as in classification trees. Therefore, the decision tree model is not expressed as a mathematical equation, but rather as a set of decision rules represented by the tree structure [20].
The concept of information gain is one of the main features determining the best splitting ways of the data for each node to achieve the best performance.
Another concept is entropy, which measures the impurity in the dataset and quantifies the randomness in the data. A node with low entropy is considered pure, while high entropy indicates mixed data.
Information gain, therefore, aims to reduce the entropy of the data and create more homogeneous subsets, resulting in a purer dataset.
This is the method used to select the best split at each node - utilizing information gain - to achieve a more effective partitioning of the data [21].
In the below Figure 3.1 shows the decomposition of the decision tree.

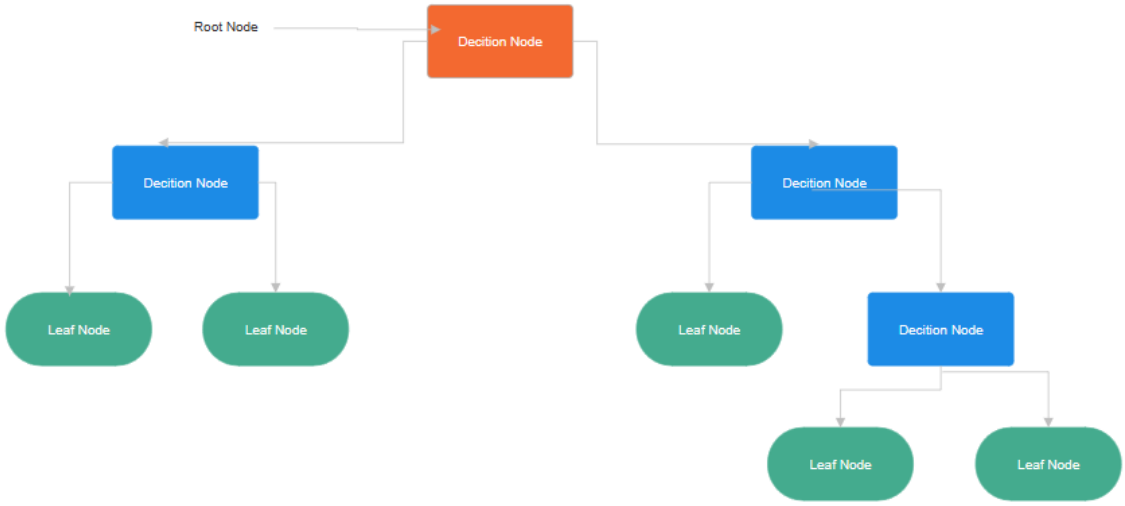

Fig 3.1 : Graphs of the Decision tree decomposition

Overfitting is one of the most common issues in the Decision tree, as they often produce intricate models that capture noise in training data rather than underlying patterns.

To overcome this issue, several strategies can be employed:

- Pruning: Cutting off parts of the tree that don't significantly contribute to making predictions. This simplifies the tree and prevents it from overfitting to noise.
- Limiting the Depth: Restricting the depth of the tree or making decisions based on a minimum number of examples. This prevents the tree from learning too much from small details.
- Minimum Samples per Group: Ensuring that each group in the tree has a sufficient number of examples. This prevents the tree from making decisions based on too few examples.
- Cross-Validation: Assessing how well the tree performs on new data that it hasn't seen before. This helps in selecting the best settings for the tree and prevents it from overfitting to the training data.

Ensemble Methods: Using many trees together to make predictions. This helps to smooth out the mistakes that individual trees might make and generate more reliable predictions [22].

3.2.5 Random Forest (RF)

It is a multiple **DT** algorithms running at once, It is an ensemble of decision trees. These many trees are constructed in a certain “random” way from a Random Forest. Each of the trees makes its own individual prediction and these predictions are then calculated as average to obtain a single result.
The averaging process makes a random forest more effective than a single decision tree, improving its accuracy and reducing overfitting. There is no single mathematical equation for the random forest model as a whole; rather, each individual decision tree in the forest is represented by a set of decision rules and corresponding weights that define the importance of each input feature in the tree [20].
Figure 3.2 below describes how the concept of an RF algorithm goes through, dataset is randomly split into multiple subsets with each subset assigned to a different decision tree. Forecasts are then gathered from each decision tree, and the optimal path is chosen.

By choosing the most effective splitting point at each node according to factors like variance reduction for regression or information gain for classification, the tree grows recursively.

After training each decision tree, predictions are generated by averaging the forecasts of each individual tree. This process is referred to as voting (for classification) or averaging (for regression). In classification tasks, the final prediction is determined by computing the mode, or the most frequent class label, among the predictions. For regression tasks, the average of the predictions is utilized.

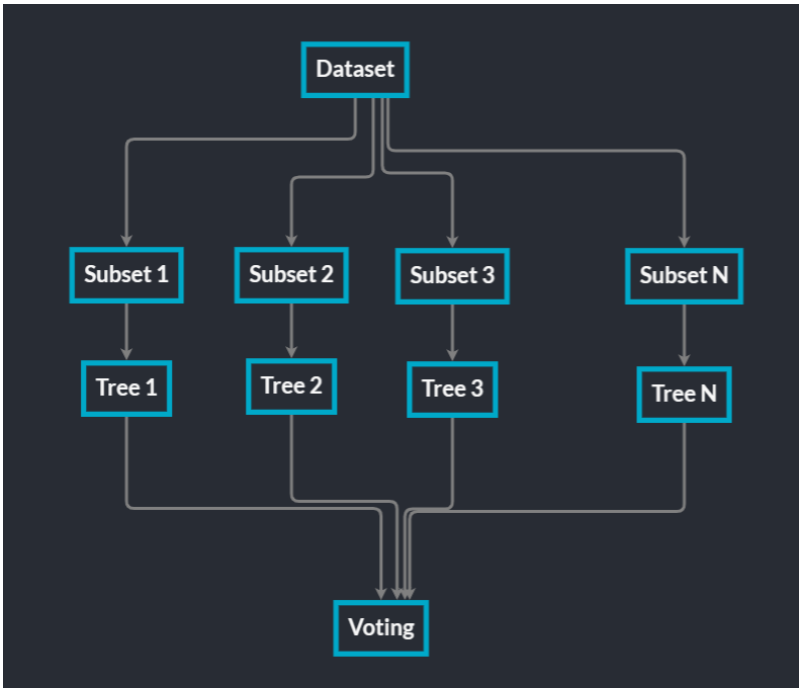

Fig 3.2 : Flowchart of the selection in Random Forest

3.2.6 XGBoost

It is an implementation of gradient-boosting decision trees and it is designed for speed, ease of use, and performance on large datasets. It stands out for its speed, as it does not require parameter optimization or tuning, allowing immediate use after installation without further configuration. Despite its speed, XGBoost maintains high accuracy. As an ensemble learning technique, XGBoost creates decision trees sequentially, with each tree correcting the mistakes of its predecessors.

The boosting technique, on which XGBoost is based, aggregates the predictions of multiple weak learners, typically decision trees. Unlike bagging techniques such as Random Forest, which constructs trees independently, boosting techniques build trees sequentially, with each new tree learning from the mistakes of the previous ones.

XGBoost uses many regularisation strategies to mitigate overfitting and enhance overall generalisation.

These strategies include shrinkage, depth constraint, and minimum child weight.
And one of the main of its efficient handling of missing values, which allows it to handle real-world data with missing values without requiring significant pre-processing. Additionally, XGBoost has built-in support for parallel processing taking advantage of multicore processors and distributed computing frameworks like Apache Spark to speed up training on large datasets [23].

The objective function in the XGBoost combines a loss function L with a regularization term Ω.

$$Objective = L + \Omega. \tag{3.7}$$

Loss Function: It measures the difference between the predicted values and the actual labels.
Regularization Term: It condemns complex models to prevent overfitting. It typically consists of two parts:

- Tree Complexity Term: Measures the complexity of individual trees.
- Number of Leaves Term.

3.2.7 Support vector machines (SVM)

It is a learning model that is used for classification or regression. This approach works well with high-dimensional spaces and can be used with small data sets effectively. When the algorithm is trained on a dataset, it can easily classify new observations efficiently. It achieves this by creating one or multiple hyperplanes that can separate the dataset into two classes. Hyperplanes serve as boundaries separating different classes, and their dimensionality depends on the number of input variables [24].
SVM can take many different forms, including Linear SVM which is the simplest form which tries to find a linear decision boundary and it works better when the data is linearly separable.

There are several other forms of SVM that are used to handle non-linearly separable data, those alternative forms use different kernel functions and methods aiming to modify the inputs by swapping them to be linearly separable again.
The choice of the specific kernel function would be depending on the type or the nature of the input data.
Polynomial SVM: this form uses from its name a polynomial kernel function which is aiming to map the input data to become a higher-dimensional space.

$$K(x^t, x') = (x^t x' + c)^d. \quad (3.8)$$

where d is the degree of the polynomial and c is a constant.
if the degree of the d is increased, the ability of the SVM to capture more complex decision boundaries gets better.
Radial Basis Function (RBF) SVM: Also from its name, it uses RBF kernel function.

$$K(x, x') = \exp(-\gamma||x - x'||^2). \quad (3.9)$$

here γ represents the kernel's bandwidth.
This form is widely used with non-linear decision boundaries as it's more fixable.
Also, there are many other forms, even there are Custom Kernels which would be tailored to be matched with the captured data and designed after that [25].

3.2.8 Naive Bayesian Model

It's also one of the most popular Supervised models and it works greatly with very small data sets. but even with this simplification, The algorithm could be successfully applied to complex problems. It is not a single algorithm but a family of algorithms where all of them share a common principle. This model draws on common data assumptions, such as each attribute is independent [26].
The mathematical equation for the Naive Bayes model can be expressed as:

$$P(y \mid x_1, x_2, \ldots, x_n) = P(y) * P(x_1 \mid y) * P(x_2 \mid y) * \ldots * P(x_n \mid y) / P(x_1, x_2, \ldots, x_n). \quad (3.10)$$

y is the class variable.

x_1, x_2, ..., x_n are the feature variables.

P (y | x_1, x_2, ..., x_n) is the posterior probability of y given the values of x_1, x_2, ..., x_n.

P(y) is the prior probability of y.

P(x_1 | y), P(x_2 | y), ..., P(x_n | y) are the conditional probabilities of the values of x_1, x_2, ..., x_n given y.

The joint probability model would be expressed as:

$$P(y|x) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}. \quad (3.11)$$

After a few adjustments, the equation used by the classifier can be expressed like this:

$$\hat{y} = argmax\, P(y) \prod_{i=1}^{n} P(x_i|y). \quad (3.12)$$

This equation can be used to compute the posterior probability of each class y for a given input x_1, x_2, ..., x_n and the class with the highest probability can be chosen as the predicted class for the input [27].

3.3 Deep learning-based time series models

After the classical models and the supervised models have been discussed, each of them with their specific ways of fitting the models.

Classical time series models primarily focus on the relationship between past and present data.

Supervised machine learning models concentrate on identifying cause and effect relationships.

Now, delving deeper into deep learning-based time series models, which is an advanced subfield of ML that employs algorithms inspired by the structure and function of

Artificial Neural Networks. These models improve on their own by analysing computer algorithms. There are various types of algorithms used in Deep Learning, some of which will be discussed below [28].

3.3.1 Recurrent Neural Network (RNN)

I t's basically a neural network with memory that can be used for predicting time-dependent targets. RRNs are capable of retaining the previously recorded input state, enabling them to make decisions for future time steps. This allows the network to learn from sequential data, making it useful for various applications, including understanding time series [29].

The mathematical equation for the Recurrent Neural Network model can be expressed as:

$$h^t = f(h^{t-1}; x). \tag{3.13}$$

When x is inputted into this network, state h, also referred to as a hidden state that is sent forward, incorporates it. A single time step is delayed as indicated by the black square.

It simply allows the information which persists over time with recurrent connections to move from the previous time step to a new output, allowing the network to capture the dependencies in the data [30].

During the whole-time steps, the same shared set of parameters remains unchanged saving some processing and efficiently training the RNN more.

And RNNs could be involved in many different applications or tasks because of their good ability to capture temporal dependencies making them perfect to suit any task whatever the data looks like.

Below in the Figure 3.3 describes a recurrent neural network with no output which represents the equation.

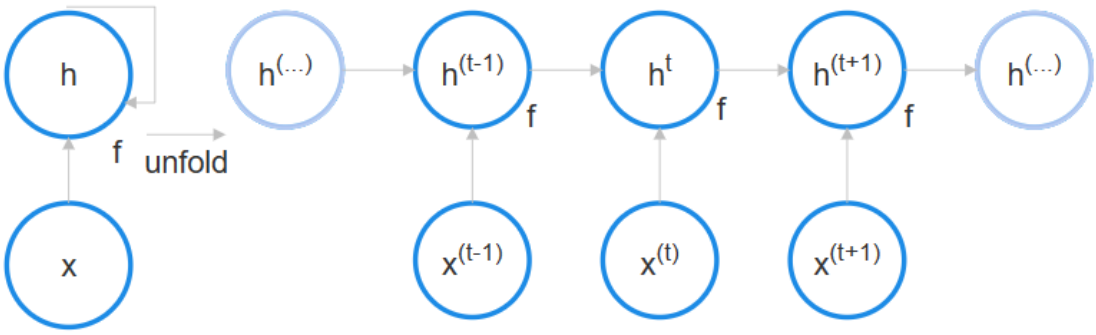

Fig 3.3 : A recurrent neural network with no output which represents the equation

3.3.2 LSTM (Long Short-Term Memory)

LSTMs are a specific type of RNNs. They have proven useful for time series forecasting on multiple occasions. By retaining the previous input state, It is empowered to make decisions for future time steps, facilitating the network's learning from sequential data. And they were created mainly to find a solution to the gradient problem by providing the model with several gates to choose from. These gates let the model decide what information to identify as meaningful and what information to ignore [31].

LSTM mainly came for handling and vanishing gradients, it processes the sequence data and introduces a more sophisticated memory mechanism which is some cells repeating one after another to control the flow of information.
Those cells consist of many components such as:
- **Forget Gate f_t** which decides if the information is needed or it could be discarded from the cell state.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f). \quad (3.14)$$

This gate (Forget Gate) decides which parts of the previous cell state c_{t-1} are to be forgotten.
σ is the sigmoid function, which outputs a value between 0 and 1.
This value multiplies the previous cell state c_{t-1} , and then decides the extent to which each component of the cell state is remembered or forgotten. A value close to 0 means "forget it", while a value close to 1 means "retain it".

- W_f is the weight matrix for the forget gate.
- h_{t-1} is the output from the previous time step.
- x_t is the current input.
- b_f is the bias term for the forget gate.

- Cell State c_t which holds the important information and gets many time steps with many different states.

$$C_t = f_t * C_{t-1} + i_t * C_t. \quad (3.15)$$

The previous cell state C_{t-1} multiplied by the forget gate output f_t determines how much the old state can retain.
The candidate cell state C_t multiplied by the input gate output i_t determines how much of the new state to add.

- Input Gate i_t and Output Gate o_t, those gates are for the determination of the state of the cell and if it should be stored or output from the cell state. And the output gate o_t determines which parts of the cell state should be output as the hidden state h_t.
The output gate o_t regulates the information flow from the cell state c_t to the hidden state h_t The hidden state is passed to the next time step and can also be used as the final output of the LSTM.

For the Input Gate:

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i). \quad (3.16)$$

For the Output Gate:

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o). \quad (3.17)$$

And for the hidden state h_t:

$$h_t = o_t * \tanh(C_t). \quad (3.18)$$

When this tanh is applied to C_t scales the cell state values are between -1 and 1.
All of this is described in the below Figure 3.4 as the LSTM architecture.

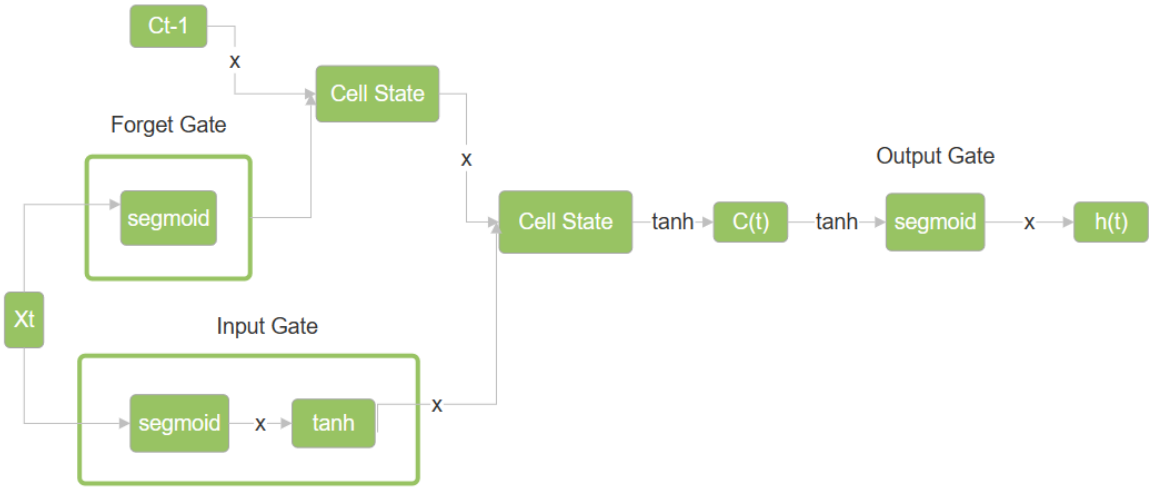

Fig 3.4 : LSTM architecture

3.3.3 Prophet and DeepAR

They are 2 time series libraries that were open-sourced by Facebook and Amazon they are considered as black-box models. The idea is to have a Python library that does all the heavy lifting for you. Forecasts can be generated with minimal user input as they require little user specification. This can be an advantage, as it's possible to automatically generate forecasting models without much knowledge or effort. But on the other hand, there is also a potential risk that if not carefully monitored, the automated model-building tool may produce a model that appears to be effective but does not actually perform well in reality [32] [33].

The good thing about Prophet is that it is specifically designed for time series forecasting tasks with daily observations that exhibit trends, seasonality, and holiday effects, which is the main topic of this study.
And what could be known about the model is that it decomposes the time series into three main components: trend, seasonality, and holidays.

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t. \quad (3.19)$$

g(t) represents the trend component.
s(t) represents the seasonal component.
h(t) represents the holiday component.
ε_t is the error term, representing random noise in the data.

And Prophet usually detects Seasonality automatically using the Fourier series even for the complex effects making it more flexible and give the option to specify and define the holiday component all of those options with easy and simple interface for the user without any complexity like in the RNN-LSTM for example all of this with a good handling and fitting for any large datasets [33].

Also the same for the DeepAR from Amazon was developed specifically for forecasting time series data and it should be using RNNs mixed with some autoregressive approach where previous observations provide the basis for the forecasts developed at each time step. and the RNNs capture dependencies over time.
One of the advantages of this model is that it can parallelize training across multiple GPUs and instances, making it faster to train any big datasets and easier to customise requirements like hyperparameters [34].

4. TIME SERIES FORECASTING LIBRARIES

There are a wide variety of tools or techniques that could be used for time series forecasting as discussed in the first part of the thesis from classical statistical methods, the Supervised way, or advanced approaches like deep learning.
There are many different choices of libraries or frameworks that can be used for time series forecasting. Each of the libraries has different methods for dealing with the various time series learning tasks regression, classification, or forecasting.
Here are many of the main and most used ones with forecasting.

4.1 Statsmodels

Statsmodels is a Python library for statistical modelling and econometric analysis also it has Python packages that provide a complement to SciPy for statistical computations including a range of tools for classical statistical tests and models, including linear regression, time series analysis, and generalized linear models also advanced statistical methods such as panel data analysis, survival analysis, and Bayesian statistics. is designed to work with NumPy arrays and Pandas data frames. An extensive list of result statistics is available for each estimator. The results are tested against existing statistical packages to ensure that they are correct.
It also provides tools for visualizing time series data, including line plots, scatter plots, autocorrelation plots, and partial autocorrelation plots. This helps to explore and diagnose a lot of factors and even potential issues over the data.
Statsmodels is widely used in academia, finance, and industry for data analysis and modelling. It can be used of course Time Series Analysis with different modelling frameworks primarily focused on classical statistical modelling such as ARIMA, ARIMAX, VARMA, and VARMAX models as it includes classes for estimating those models and it allows the user to modify or specify the parameters of the model so it obtains the future forecast [35].

There are many other functions included in the library that could be useful for time series forecasting The ‘seasonal_decompose’ function which decomposes a time series into its trend, seasonal, and residual components and this is very useful to analyse the patterns of the data.

4.2 Sktime

A unified framework for machine learning with time series. It takes inspiration from scikit-learn with a similar API and it provides an easy-to-use, flexible, and modular open-source framework for a wide range of time series machine learning tasks and provides a wide range of tools and algorithms for time series forecasting, classification, clustering, and regression.
Sktime supports both univariate and multivariate time series and includes various pre-processing, feature extraction, and evaluation methods specific to time series data. It also provides an interface for integrating external time series libraries and datasets. And the main technical specifications are: -In-memory computation of a single machine, no distributed computing. It is designed to be user-friendly and scalable. And is used in diverse applications such as finance, healthcare, and transportation.

- Medium-sized data in pandas and NumPy.
- Modular, principled, and object-oriented API.

It’s mostly used and supports in focused on machine learning with time series data, including forecasting, classification, clustering, and regression like Supervised Time Series algorithms and methods, including implementation of interval-based classifiers, such as the supervised time series forest, as well as ROCKET [36].

4.3 Keras

It is also one of the most used deep learning frameworks that support LSTMs and RNNs models. It’s Simple, Flexible, and Powerful The primary reason to use Keras is its guiding principle of being user-friendly, which translates to its ease of use for both

learning and building models, Keras offers the advantages of broad adoption and support for a wide range of production deployment options.
It could be running on top of multiple backend libraries and be compatible with them like TensorFlow, Theano, or Microsoft Cognitive Toolkit. Switching between those different engines without any changes to the code.
There are two main types of models available in Keras, the Sequential model, and the Functional one. As the sequential one is simpler and made of layers one is on the top of another like a stack each layer has one input and one output, and the data should be flowing sequentially through the layers, from input to output.
On the other hand, The Functional API is more powerful and flexible, as it allows to modification of the whole architecture defines the model with multiple inputs and outputs, and then connects them using functional API layers. Making it perfect for more complex tasks with its advanced features.
Keras also supplies many of the common deep-learning sample datasets via the Keras. Datasets class, for example, cifar10 and cifar100 small colour images, IMDB movie reviews [37].

4.4 Prophet

Prophet is open-source software released by Facebook and it is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It is most effective when working with time series data that exhibit prominent seasonal effects and have a substantial amount of historical data spanning multiple seasons. Prophet is robust to missing data and shifts in the trend and typically handles outliers well [32].

And finally, below in Table 4.1 a comparison between the Time Series Forecasting Libraries.

Statsmodels	Sktime	Keras	Prophet
Python library for statistical modelling	Python library for machine learning	Python library for deep learning	Python library for forecasting with seasonality
It’s widely used in the academic community, finance, and industry for data analysis and modelling	It’s mostly used and focused on machine learning with time series but it’s not widely used in the academic community	It offers a variety of deep learning models for time series forecasting, like LSTM and GRU.	It is widely used in industry but it may not be suitable for complex time series forecasting tasks.
It may not be as user-friendly as other libraries. it provides many features but it may require some programming skills to use	It's easy to use and it offers a unified API for time series forecasting, making it simple to compare and evaluate various models.	It is easy to use, and it provides a high-level API for building deep learning models but it may require more computational resources compared to other libraries	It is easy to use, and it provides a simple API for forecasting besides having unique features like automatic trend detection and seasonal decomposition
It’s primarily focused on classical statistical modelling	It’s mainly focused on ML Supervised Time Series algorithms and methods	Mainly focused on deep learning Modelling	Mainly focused on simple time series forecasting tasks

Tab. 4.1: Comparison of Time Series Forecasting Libraries

5. TIME SERIES DATA PRE-PROCESSING

The online datasets or the collected ones can have various formats and contain various data. Data pre-processing involves converting raw data into a useful and comprehensible format. Real-world or raw data usually has inconsistent formatting, and human errors, and can also be incomplete. So data pre-processing is a critical step that improves the completeness and efficiency of datasets for performing data analysis It's a crucial process that can affect the success of data mining and machine learning projects. It may affect the performance of machine learning models [38].

And this procedure includes many main steps like:

1. Data Cleaning: The data can have many irrelevant and missing parts so to fix that

- Missing Data can be handled in various ways like Ignore the tuples or fill the Missing values.

Also, the missing data or values can be handled by many different approaches like replacing the missing values with the most recent observations by the forward-filling approach. Or replacing the missing values with the next observed value with the Backward-filling approach. Or just remove those time points if they could be neglected and will not affect the estimation or the forecasting

- Noisy data refers to data that lacks meaning and cannot be interpreted by machines. This type of data is often generated as a result of faulty data collection or data entry errors. etc. It can be handled in the following ways Binning Method, Regression, and Clustering.

2. Data Transformation: This step is taken in order to transform the data into appropriate forms suitable for the mining process. This involves the following ways Aggregation, Discretization, and Normalization.

---

https://colab.research.google.com/drive/1UmTOoX332sYs0CT4n9vVXZCFdvcQRTzU?usp=sharing

There are many common methods to bring the data to a common scale and to promote the development of the model including Min-max scaling and in that method, the data would be scaled with some certain range according to its value.

Or other ways like Log transformation and Z-score normalization.

3. Data Reduction: Since data mining is a technique that is used to handle huge amounts of data. While working with a huge volume of data, analysis became harder in such cases. In order to get rid of this, we use a data reduction technique. It aims to increase the storage efficiency and reduce data storage and analysis costs.

And also many other ways to clean and prepare the data to be processed.

By performing these pre-processing steps, Time series data can be cleaned, transformed, and made ready for analysis and modelling, which will increase the accuracy as well as reliability of any projections or insights derived from the data [39].

5.1 Dataset pre-processing for the models

In the project, there will be 3 different datasets with the kind of univariate used and for each dataset, there will be 3 models applied on each and the datasets will be pre-processed in a different way for each model to be ready for the processes and the forecasting eventually.

Then there would be 2 another different Multivariate datasets which will be complex because there will be more than one column, also there will be two different models applied to them.

And below in the figures are the snippets of the datasets which would be used:

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.496293	493729600
2010-01-05	7.664286	7.699643	7.616071	7.656429	6.507525	601904800
2010-01-06	7.656429	7.686786	7.526786	7.534643	6.404015	552160000
2010-01-07	7.562500	7.571429	7.466071	7.520714	6.392176	477131200
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.434674	447610800
...	...	...	...	...	...	...
2022-12-23	130.919998	132.419998	129.639999	131.860001	131.477127	63814900
2022-12-27	131.380005	131.410004	128.720001	130.029999	129.652435	69007800
2022-12-28	129.669998	131.029999	125.870003	126.040001	125.674026	85438400
2022-12-29	127.989998	130.479996	127.730003	129.610001	129.233658	75703700
2022-12-30	128.410004	129.949997	127.430000	129.929993	129.552719	77034200

3272 rows × 6 columns

Fig. 5.1: The generated dataset from Yahoo Finance

1	VALID_FROM|CNB_DISCOUNT_RATE_IN_%
2	19900101|4.00
3	19900401|5.00
4	19901001|7.00
5	19901111|8.50
6	19910101|10.00
7	19910908|9.50
8	19920325|9.00
9	19920826|8.00
10	19921230|9.50
11	19930610|8.00
12	19941024|8.50
13	19950626|9.50
14	19960621|10.50
15	19970527|13.00
16	19980814|11.50
17	19981027|10.00

3	Chart time period 1/1/2006 to 3/31/2023	
4		
5	Date	number of accidents
6	01/01/2006	499
7	02/01/2006	492
8	03/01/2006	571
9	04/01/2006	535
10	05/01/2006	518
11	06/01/2006	555
12	07/01/2006	337
13	08/01/2006	243
14	09/01/2006	656
15	10/01/2006	630
16	11/01/2006	633
17	12/01/2006	647
18	13/01/2006	704
19	14/01/2006	382
20	15/01/2006	251
21	16/01/2006	607
22	17/01/2006	649
23	18/01/2006	1076
24	19/01/2006	677

Fig. 5.2: Dataset 1 - Discount Rate & Dataset 2 - Number of accidents

The first dataset we have it will be about the discount rate changed over time here in the Czech Republic according to CNB (Czech National Bank) it's a kind of simple dataset

that won't need a lot of pre-processing. But regarding the model that will be applied to the dataset, there should be some kind of modifications that should be applied first to the dataset to be able to be worked on, So for the first model which will be applied (SARIMAX) it should be converted to this valid format first as it's shown in Listing 5.1.

```
# Convert 'VALID_FROM' column to datetime format
df['VALID_FROM'] = pd.to_datetime(df['VALID_FROM'],
format='%Y%m%d')

# Generate some random data
x = df['VALID_FROM']
y = df["CNB_DISCOUNT_RATE_IN_%"]

# Set 'VALID_FROM' column as the index
df = df.set_index('VALID_FROM')

# Sort the dataframe by index (date)
df = df.sort_index()
```

Listing 5.1: Dataset pre-processing for SARIMAX Model

And now then the dataset should be pre-processed and it can be continued with the next steps for the model.

For the same dataset also there will be another model which will be applied on – below in Listing 5.2 - and it will need to be modified a little to be prepared and ready for this model. The other model would be (Prophet) it's a time series forecasting model developed by Facebook. In addition to the previous pre-processing to the dataset there should be extra steps done to the dataset so it would be ready for the model. For this specific model the columns must be named as "ds" and "y" so the model can work on the dataset.

```
# Load the dataset
df = pd.read_csv(path+"/data.csv", sep="|")
```

```
# Convert 'VALID_FROM' column to datetime format
df['VALID_FROM'] = pd.to_datetime(df['VALID_FROM'],
format='%Y%m%d')

# Set 'VALID_FROM' column as the index
df = df.set_index('VALID_FROM')

# Sort the dataframe by index (date)
df = df.sort_index()


# Prepare the data for Prophet
train_data_prophet = train_data.reset_index()
train_data_prophet.columns = ['ds', 'y']
```

Listing 5.2: Dataset pre-processing for Prophet Model

Finally for the third model (LSTM) which is one of the most famous Deep learning models for Time series Forecasting it needs a lot of more data pre-processing to be able to work on the dataset effectively. So here also it will be like the previous codes it reads the data from a CSV file located at the specified path. The columns are renamed to 'date' and 'discount_rate', and the 'date' column is converted to a datetime object with the format '%Y%m%d'. Finally, the 'date' column is set as the index for the DataFrame.

But after that there will be extra steps done starting from Scales the 'discount_rate' values in the DataFrame using the MinMaxScaler from the scikit-learn library. The values are scaled between 0 and 1. Scaling the data: Scaling the data to a common range, such as between 0 and 1, is often necessary to ensure that the model can learn from the data effectively. In this code, the MinMaxScaler from the scikit-learn library is used to scale the 'discount_rate' values between 0 and 1.

There also will be splitting the data which it will be mentioned in the next pages so it could be skipped for now but there will be an important step also regarding the data pre-processing which is called reshaping by restructuring the training and testing sets into a time series analysis-friendly manner, where each training example consists of a single input value (X_train) and an output value (Y_train), which is next in the sequence.

Afterward, two arrays are created, X_train and Y_train, each having the input and output values for the training set, and X_test and Y_test, each containing the input and output values for the testing set. In time series analysis, the data must frequently be reshaped into a certain format that is appropriate for the model being employed. Each training example in this code consists of a single input value and its associated output value. The training and testing data are reshaped into a format appropriate for time series analysis.

```
# Read the data
df = pd.read_csv(path+'/data.csv', sep='|')
df.columns = ['date', 'discount_rate']
df['date'] = pd.to_datetime(df['date'], format='%Y%m%d')
df.set_index('date', inplace=True)

# Scale the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_values = scaler.fit_transform(df.values.reshape(-1, 1))

# Split into train/test
train_size = int(len(scaled_values) * 0.8)
train, test = scaled_values[0:train_size,:],
scaled_values[train_size:len(scaled_values),:]

# Reshape into X=t and Y=t+1
X_train, Y_train = [], []
for i in range(len(train)-1):
    X_train.append(train[i])
    Y_train.append(train[i+1])
X_train, Y_train = np.array(X_train), np.array(Y_train)

X_test, Y_test = [], []
for i in range(len(test)-1):
    X_test.append(test[i])
    Y_test.append(test[i+1])
X_test, Y_test = np.array(X_test), np.array(Y_test)
```

Listing 5.3: Dataset pre-processing for LSTM Model

All of this pre-processing which took place previously on the first dataset almost would be the same on the second dataset. Which is nearly close to the first dataset, but it’s

more complex. And it’s about the number of car accidents in the Czech Republic since 2006 on a daily basis. But there will be of course some few changes needs to be done on the dataset to be prepared.

For the third dataset, it will be not used a ready dataset from the internet like in the first two datasets. But this time the dataset would be created using API from Yahoo Finance to generate the third dataset. The ticker symbol "AAPL" is commonly used to represent Apple Inc. in financial markets, including stock exchanges. Ticker symbols are unique identifiers assigned to publicly traded companies to facilitate trading and tracking their stock prices.

First, it’s needed to import Yahoo Finance and then fetch historical stock data for Apple Inc. from Yahoo Finance. And then generate the dataset and save it in df3 . Symbol = "AAPL" sets the variable symbol to "AAPL", which is the stock ticker symbol for Apple Inc. As shown in the Listing 5.4 below, the start-date and the end-date were provided to generate the desired dataset.

```
import yfinance as yf

# Fetching data from Yahoo Finance
yf.pdr_override()
symbol = "AAPL"
start_date = "2010-01-01"
end_date = "2022-12-31"
df3 = pdr.get_data_yahoo(symbol, start=start_date, end=end_date)
```

Listing 5.4: Dataset generated using API

And here it will be the generated dataset looks like as in the below Figure 5.1

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.496293	493729600
2010-01-05	7.664286	7.699643	7.616071	7.656429	6.507525	601904800
2010-01-06	7.656429	7.686786	7.526786	7.534643	6.404015	552160000
2010-01-07	7.562500	7.571429	7.466071	7.520714	6.392176	477131200
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.434674	447610800
...	...	...	...	...	...	...
2022-12-23	130.919998	132.419998	129.639999	131.860001	131.477127	63814900
2022-12-27	131.380005	131.410004	128.720001	130.029999	129.652435	69007800
2022-12-28	129.669998	131.029999	125.870003	126.040001	125.674026	85438400
2022-12-29	127.989998	130.479996	127.730003	129.610001	129.233658	75703700
2022-12-30	128.410004	129.949997	127.430000	129.929993	129.552719	77034200

3272 rows × 6 columns

Fig. 5.1: The generated dataset

Hypothesis Test

A hypothesis test is used in time series forecasting to determine if a pattern or trend found in previous data is statistically significant and likely to continue in the future. A hypothesis on the time series' behaviour, such as whether it follows a specific pattern or trend, provides the basis of the study. After that, information from the time series is gathered, and statistical tests are run on it to determine how strong the evidence is in behalf of the hypothesis. With a certain degree of confidence, future values of the time series can be predicted using the outcomes of the hypothesis test.

The code of the Hypothesis Test is described in the below Listing 5.5.

```
from scipy import stats

stat, p = stats.normaltest(df.Global_active_power)
print('Statistics=%.3f, p=%.3f' % (stat, p))
```

```
alpha = 0.05
if p > alpha:
    print('Data looks Gaussian (fail to reject H0)')
else:
    print('Data does not look Gaussian (reject H0)')
```

Listing 5.5: Hypothesis Test

So this code was to check whether the data follows a normal distribution or not, based on the calculated test statistic and p-value.
And based on that it would show that or determine if a pattern or trend will be found in the data is statistically significant and likely to continue in the future.

Dickey-Fuller test

The Dickey-Fuller test is a statistical test used to determine whether a time series is stationary or not.

By using this test, analysts can ascertain whether there are any trends or seasonality in a time series that should be taken into consideration before using forecasting models.

- Null Hypothesis (H0): It suggests the time series has a unit root, meaning it is non-stationary. It has some time-dependent structure.
- Alternate Hypothesis (H1): It suggests the time series does not have a unit root, meaning it is stationary. It does not have a time-dependent structure.
- p-value > 0.05: Accept the null hypothesis (H0), the data has a unit root and is non-stationary.
- p-value <= 0.05: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.

```
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import pacf

#df2=df1.resample('D', how=np.mean)
df2=df1.resample('D').agg(np.mean)
```

```
def test_stationarity(timeseries):
    rolmean = timeseries.rolling(window=30).mean()
    rolstd = timeseries.rolling(window=30).std()

    plt.figure(figsize=(14,5))
    sns.despine(left=True)
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')

    plt.legend(loc='best'); plt.title('Rolling Mean & Standard
Deviation')
    plt.show()

    print ('')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4],
                         index=['Test Statistic','p-value','#Lags
Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
test_stationarity(df2.Global_active_power.dropna())
```

Listing 5.6: The Dickey-Fuller Test

6. TIME SERIES DATA SPLITTING

Splitting a dataset refers to dividing a given dataset into two or more separate subsets, The main objective of splitting a dataset is to train and evaluate machine learning models effectively. This entails randomly separating the data into subsets, one of which will be used to train the model and the other for evaluating how well it performs. Machine learning frequently uses dataset splitting to evaluate the model's accuracy and generalizability.

Splitting time series data is a crucial step in creating and testing prediction models. We can make sure that our models appropriately depict the patterns and trends in the data by maintaining the temporal order of the data. The size and complexity of the dataset, as well as the particular research topic being addressed, will determine which splitting strategy is used [40].

Time series data can be split using a variety of methods, but we'll probably go with the Fixed Split: In a fixed split, the dataset is simply divided into two parts, one of which is used for training and the other for testing. The first 80% of the data, for instance, can be used for training, and the remaining 20% is suitable for testing as shown in the below Figure 6.1. Although this methodology is simple to use, but it’s sometimes not the optimal one when dealing with huge datasets or time series with complex patterns [41].

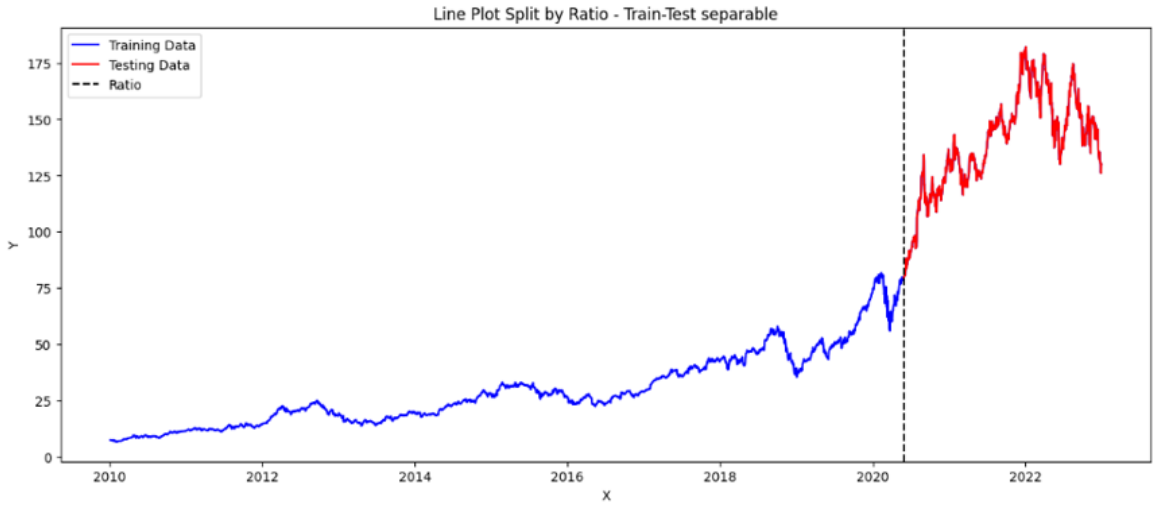

Fig. 6.1: Splitting the dataset into training and test, As X= Date and Y= Stock Price’s Close

6.1 Train / Test Split

In this thesis as mentioned before in the previous section the data pre-processing. Three models would be applied to 3 datasets, and all of those datasets would be handled in this section almost with the same way with the fixed split which mentioned before in the last paragraph.

So for the first dataset it would be split in the ratio 80% for the training and the rest 20% for the test which mentioned in the below Listing 6.1 .

```
# Define the ratio point (where to split the plot)
ratio = 0.8

# Split the data into training and testing sets
train_size = int(len(df) * 0.8)  # 80% for training, 20% for
testing
train_data = df[:train_size]
test_data = df[train_size:]
```

Listing 6.1: Splitting data into training and test

And here it will be the result as in the Figure 6.2 below

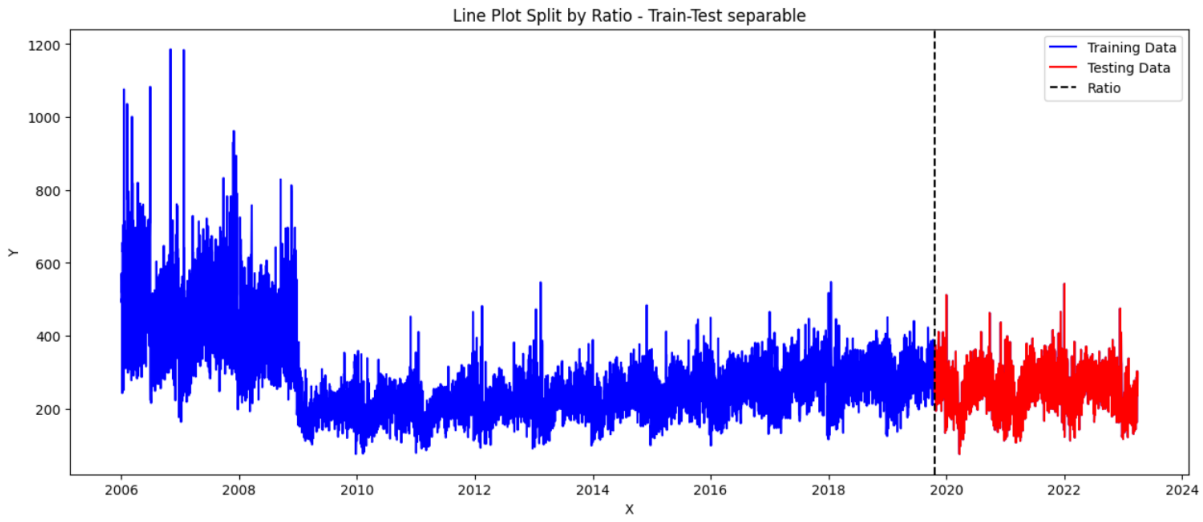

Fig. 6.2: Splitting data into training and test As X= Date and Y= Number of Accidents

This splitting would be applied in the first and third models (SARIMAX) and (LSTM) but for the second model (Prophet) it won't be applied because it's a kind of black box model Because the method is built to handle time-series data with well-known seasonality patterns, the Prophet model does not normally divide the dataset into training and testing sets. The model analyses the previous data and uses it to forecast the future while considering seasonal changes and other data patterns.

In other words, the Prophet model is a forecasting model which predicts future values by studying historical trends and patterns. It can manage seasonality and non-linear trends in the data since it models trend changes and seasonal patterns using a Bayesian framework. And this may lead to a good performance sometimes and a very bad performance other times depending on the dataset and its pattern which will be discussed in the next parts of the thesis.

7. TIME SERIES FORECASTING MODELS

In this thesis, three different kinds of Models will be applied to the datasets to compare their performances and accuracy those 3 models will be from the 3 different categories so the first model would be (SARIMAX) it's from the classic statistical time series model used for forecasting, The second model would be (Prophet) it's kind of black-box model which developed by Facebook and it is considered a type of machine learning model that falls under the category of additive models. The third model will be (LSTM) Long Short-Term Memory and it is a type of Recurrent Neural Network (RNN) which is a category of Deep Learning models used for time series forecasting. And here there will be the details of these models when they were applied to the different datasets and the differences between their implementations.

7.1 SARIMAX

The SARIMAX model is from Statsmodels. And Statsmodels is a Python library where a large selection of statistical models and tools are provided for data analysis. It contains functionality for statistical modelling, estimation, hypothesis testing, and more and is developed on top of NumPy, SciPy, and Pandas. One of the key components of Statsmodels is the tsa module, which focuses on time series analysis.

Within the tsa module, Statsmodels provides the SARIMAX class, which stands for Seasonal Autoregressive Integrated Moving Average with Exogenous regressors. SARIMAX is an extension of the popular ARIMA (Autoregressive Integrated Moving Average) model, capable of managing seasonal trends in time series data.

SARIMAX allows you to model and forecast time series by incorporating autoregressive (AR), differencing (I), moving average (MA), and seasonal (S) components. The model parameters, denoted as (p, d, q) × (P, D, Q, s), represent the orders of the AR, I, MA, and seasonal components, respectively.

Here's a breakdown of the parameters in SARIMAX:

- p: The order of the autoregressive component, representing the number of lagged observations to include in the model.
- d: The order of differencing, indicating the number of times the series needs to be differenced to achieve stationarity.
- q: The order of the moving average component, representing the number of lagged forecast errors to include in the model.
- P: The order of the seasonal autoregressive component.
- D: The order of seasonal differencing.
- Q: The order of the seasonal moving average component.
- s: The length of the seasonal cycle (e.g., 12 for monthly data with yearly seasonality).

And starting with code the first step would be importing the model from the library in Listing 7.1.

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Listing 7.1: Importing the model from the Statsmodels library

this of course would be after the general imports within Listing 7.2.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Listing 7.2: General imports

and then the next steps before applying the model on the dataset there should be some other steps which discussed before in the previous parts of the thesis from importing or generating the dataset itself, then making the data pre-processing and splitting it into training and testing. After that, the models themselves would be applied. Then the last step is the performance evaluation to determine which model is the best with the dataset.

7.1.1 Dataset 1 - Discount Rate

First SARIMAX() function creates a SARIMA model instance.

- The first argument (train_data) is the training data used to train the model.
- The (order) parameter specifies the order of the autoregressive, integration, and moving average components of the model, respectively. In this case, it is set to (1,1,1), which means the model has one autoregressive term, one differencing term, and one moving average term.
- The (seasonal_order) parameter specifies the order of the seasonal autoregressive, integration, and moving average components of the model, as well as the number of time steps in the seasonal pattern. In this case, it is set to (1,1,1,12), which means the model has one seasonal autoregressive term, one seasonal differencing term, one seasonal moving average term, and the seasonal pattern repeats every 12 time-steps (months, in this case).
- The fit() function is used to train the SARIMA model using the specified training data. And then the trained model is stored in the (result) variable.
- After that, the trained SARIMA model will be ready to predictions on new data in Listing 7.3 below.

```
# Create and train the SARIMA model
model = SARIMAX(train_data, order=(1, 1, 1), seasonal_order=(1, 1,
1, 12))
result = model.fit()

# Perform predictions on the test set
predictions = result.predict(start=len(train_data), end=len(df)-1)
```

Listing 7.3: Creating the train and test model

After that it should be the Visualize the actual values vs. predicted values to compare the graphs and see the behaviour and the accuracy of the forecasting.

The next step which would be discussed in the next sections would be to the evaluation of the performance and the accuracy of the predictions and forecasting using some

metrics. But at least you would be able to see that difference in the graphs below (Figure 7.1).

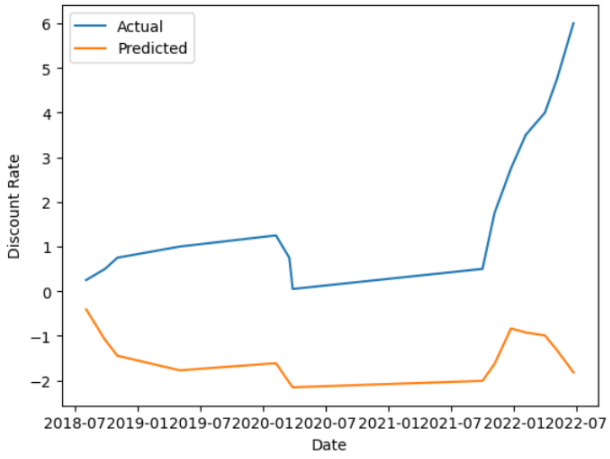

Fig. 7.1: Difference between Actual and predicted data

As shown in the graph the behaviour of the actual data and the predicted ones are not following the same numbers or the same pattern and the accuracy is not that much good of this model on this dataset but this will be discussed later.

7.1.2 Dataset 2 - Number of accidents

As the previous dataset this dataset also will be pretty much the same but with some little differences.

So here instead of training on a subset of the data, it is using the entire dataset “df2” to see if the accuracy will be better or if the training would be longer or be done on a bigger amount of data.

And then with the (order) and (seasonal_order) which are hyperparameters that specify the characteristics of the model discussed before

- The (order) parameter is a tuple that specifies the order of the non-seasonal part of the ARIMA model. The (1,0,1) tuple means that the model is an ARIMA(1,0,1), where p=1 is the order of the autoregressive term, d=0 is the degree of differencing (which means that the time series is not differenced), and q=1 is the order of the moving average term.

- The (seasonal_order) parameter is a tuple that specifies the order of the seasonal part of the ARIMA model. The (1,0,1,12) tuple means that the model is a SARIMA(1,0,1)(1,0,1,12), where the first tuple (1,0,1) is the same as the non-seasonal order, and the second tuple (1,0,1,12) specifies the order of the seasonal component of the model. The s=12 means that the model has a seasonal period of 12 (monthly data)

Compared to the previous dataset the degree of differencing (which means that the time series is not differenced here is zero instead of one in the previous dataset and differencing refers to the process of computing the differences between consecutive observations of a time series. The time series does not need to be differed to make it stationary when the order of differencing (d) is zero. For many time series models, a stationary time series which has consistent statistical features across time, such as constant mean and variance is optimum.

This technique could make a difference in the accuracy of the prediction. By removing trend and seasonality from a time series, which simplifies modelling and forecasting. The model might not be able to recognize and take into account any underlying trends or seasonality in the data if the time series are not differencing, which could result in less precise forecasts. However, whether or not differencing improves the accuracy of the prediction depends on the specific time series and the pattern of the data. In some cases, differencing may not be necessary or may even lead to worse predictions. It is important to evaluate the model's performance with and without differencing to determine the best approach for the specific problem.

Finally, the model is used to make predictions for a specified time period from 2020-04-01 to 2025-05-31 in the Listing 7.4.

```
model = SARIMAX(df2, order=(1, 0, 1), seasonal_order=(1, 0, 1, 12))
model_fit = model.fit()
predictions = model_fit.predict(start=pd.to_datetime('2020-04-01'),
end=pd.to_datetime('2024-05-31'))
```

Listing 7.4: Model Fitting

And the final result after these modifications that the accuracy of the predictions is higher than the previous dataset with the same model as it could be shown in the Figure 7.2 below.

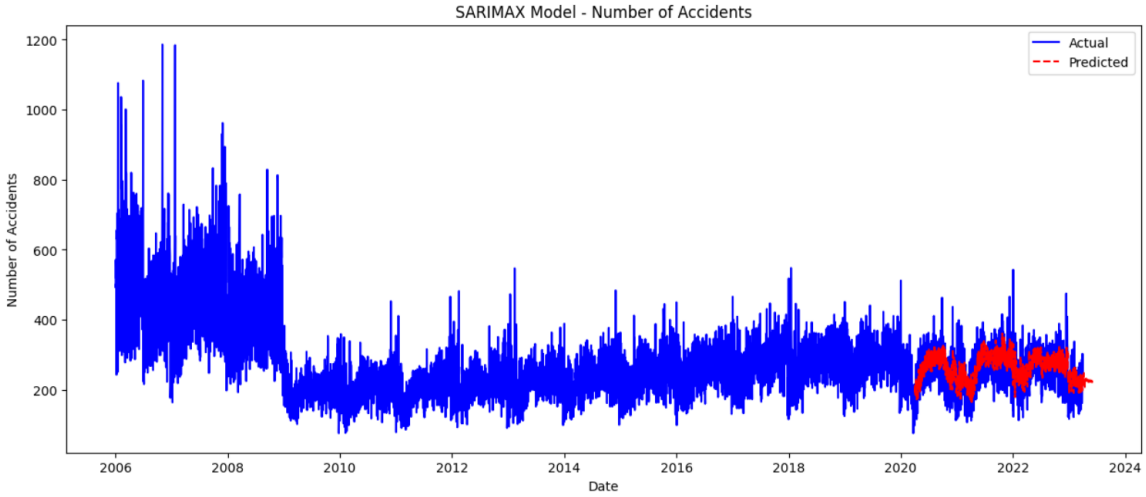

Fig. 7.2: Difference between Actual and predicted data

7.1.3 Dataset 3 - Yahoo Finance

In the third dataset the properties of the model would be almost identical to the previous dataset and also the prediction accuracy to the test data is very high and almost follow the pattern as shown in the below Figure 7.3.

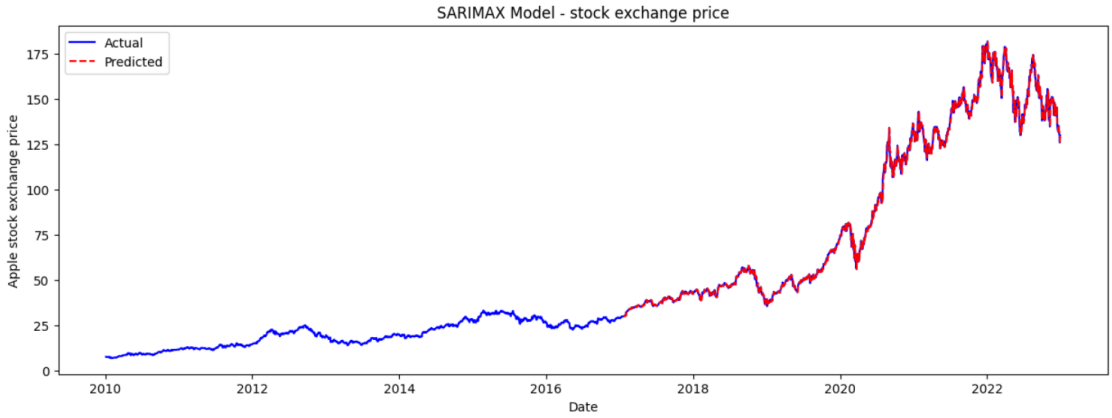

Fig. 7.3: Difference between Actual and predicted data

7.2 Prophet

The second model which will be applied to the 3 datasets would be the Prophet.

The Facebook Core Data Science team developed Prophet, a time series forecasting model. It is made to handle a variety of time series forecasting tasks, including those that involve erratic patterns, trends, seasonality, and holiday effects. Prophet creates flexible additive decomposition models with the strength of Bayesian inference to produce forecasts that are precise and easy to understand. It's kind of Black-box model that automatically detects patterns and relationships in the data, making it easy to use even for non-experts. It's not a must to know everything about the statistics of the dataset or to split the dataset to train and test the model is simpler but at the same time if the data is complex or the pattern does not have trends or seasonality the accuracy and the performance of the model won't be the best.

7.2.1 Dataset 1 - Discount Rate

The first step is to prepare the training data for the Prophet model. The training data is in the form of a Pandas DataFrame train_data with two columns, one containing the dates and the other containing the values of the time series. The code creates a new DataFrame train_data_prophet which is a copy of the train_data DataFrame, but with the column names renamed to "ds" and "y" to be compatible with the Prophet model mentioned before.

The next step is to create and fit the Prophet model using the training data. The Prophet() function is called to create an instance of the Prophet model and then the fit() method is called with the train_data_prophet DataFrame as the argument to train the model.

Once the model is trained, the code creates a future DataFrame containing the dates for which the model will make predictions. The make_future_dataframe() method is called with the argument periods=len(test_data) to create a DataFrame with the same frequency as the training data and with a length equal to the length of the test data.

The predict() method is then called on the model object with the future DataFrame as the argument to obtain the forecasted values for the time series.

Finally, the code (Listing 7.5) extracts the predicted values from the forecast DataFrame by selecting the 'yhat' column and filtering out the training data. The predicted values are stored in the predictions variable for later use.

```
# Prepare the data for Prophet
train_data_prophet = train_data.reset_index()
train_data_prophet.columns = ['ds', 'y']

# Create and fit the Prophet model
model = Prophet()
model.fit(train_data_prophet)

# Forecast on the test set
future = model.make_future_dataframe(periods=len(test_data))
forecast = model.predict(future)

# Extract the predicted values
predictions = forecast['yhat'][train_size:]
```

Listing 7.5: Preparing the data for the model

The default parameters of the Prophet model are used in this script due to their general effectiveness across various datasets.

And here's the Figure 7.4 while training the dataset

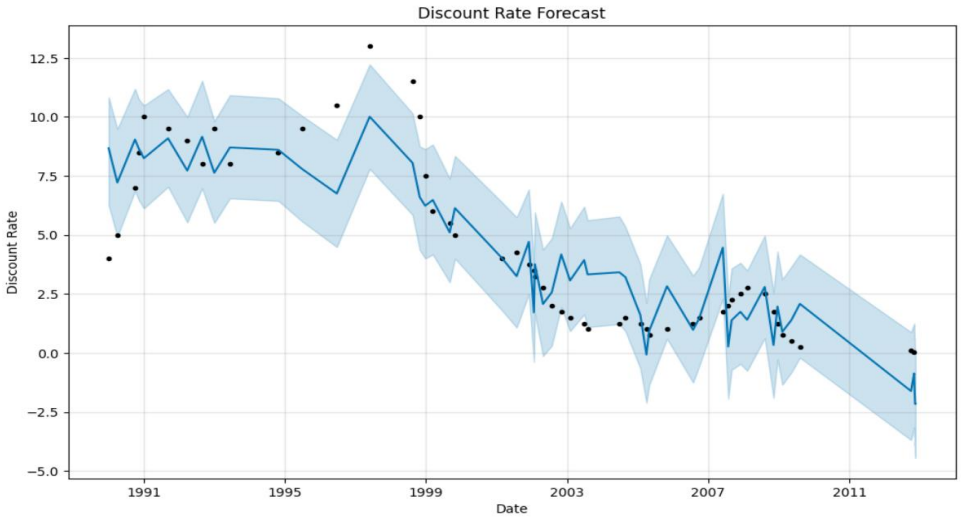

Fig. 7.4: Training the model

And the predicted data compared to the actual data below in Figure 7.5.

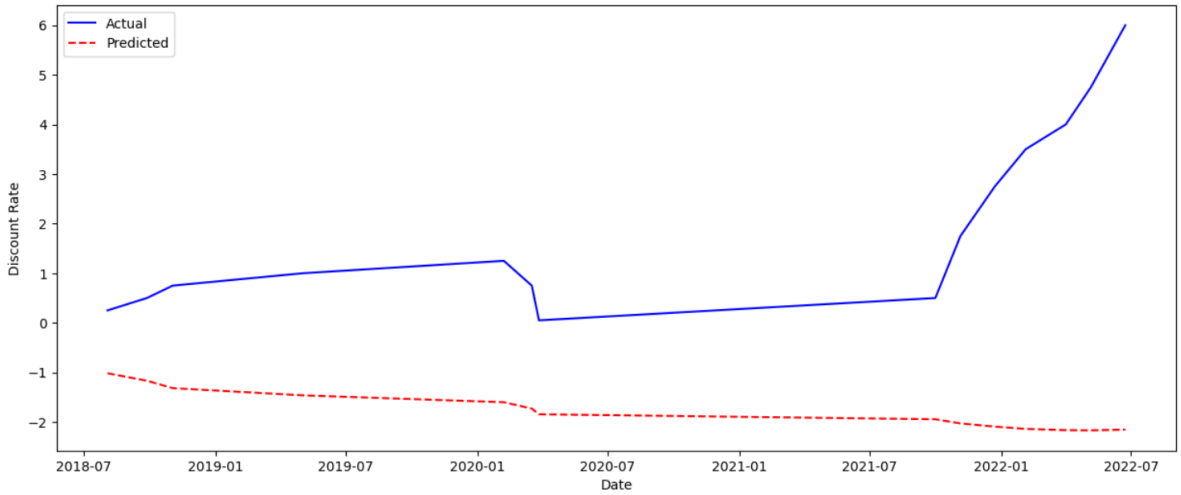

Fig. 7.5: Difference between Actual and predicted data

Here as shown in the figure the accuracy is not good and the pattern of predicted data compared to the actual data is not accurate this means that this dataset didn't have trends or seasonality in its pattern that's why the Prophet wasn't the best to fit that dataset.

7.2.2 Dataset 2 - Number of accidents

The second dataset will be the same as the previous dataset First, the code renames the columns of the dataframe to match the required format of Prophet, where 'ds' is the column containing the dates, and 'y' is the column containing the target variable.

Next, the code creates an instance of the Prophet model and fits it to the data in df2. This involves learning the trends, seasonality, and other patterns in the data that will be used to make future predictions.

Then, the code generates a new dataframe future_dates containing a range of dates for which we want to make predictions. In this case, the code generates dates for the next 361 days, starting from the last date in df2.

Finally, the code as shown below in Listing 7.6 uses the predict() method of the Prophet model to generate predictions for the dates in future_dates, and stores the predicted values in the predictions variable. These predictions can then be used for further analysis or visualization.

```
df2 = pd.read_excel(path+"/nehody.xlsx", header=4)
df2['Date'] = pd.to_datetime(df2['Date'], format='%d/%m/%Y')
df2.columns = ['ds', 'y']

# Create and fit the Prophet model
model = Prophet()
model.fit(df2)

# Generate future dates for prediction
future_dates = model.make_future_dataframe(periods=361)  # Predict
for 61 days (May and June 2023)

# Make predictions
predictions = model.predict(future_dates)
```

Listing 7.6: Creating the model

And here in the figure of the comparison of the predictions and actual data, the accuracy of the predictions is very high. In this case of this dataset which makes this model fits this dataset because it has some seasonality which makes this model performs better and predicts more accurate results as shown in Figure 7.6 below.

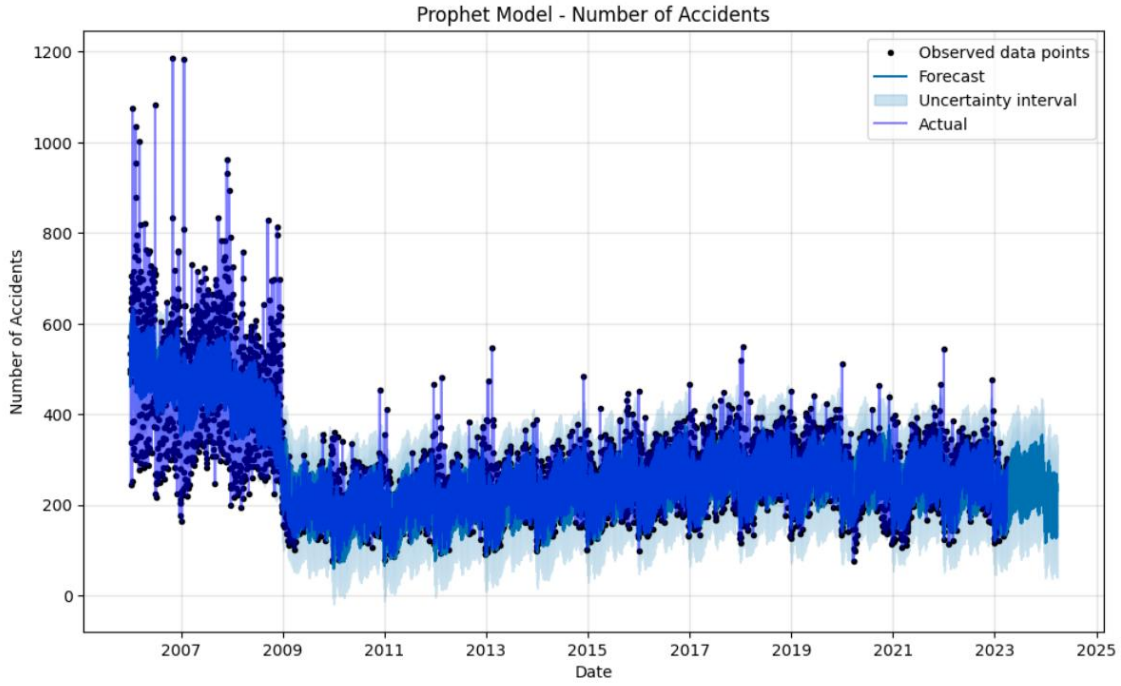

Fig. 7.6: The forecasted data

7.2.3 Dataset 3 - Yahoo Finance

Here also the Prophet model will be applied to the third dataset to make predictions on a test set, which can be used to evaluate the accuracy of the model. But in this dataset as could be observed in Figure 7.7, the accuracy of the predictions wasn't so good it could be possible that the dataset itself doesn't fit this model because of the pattern of data like the trend and seasonality. Also, it could be possible to be related to the generated dataset itself wasn't pre-processed in a good way to deal with this model.

```
# Prepare the data for Prophet
train_data_prophet = train_data.reset_index()
train_data_prophet.columns = ['ds', 'y']

# Create and fit the Prophet model
model = Prophet()
model.fit(train_data_prophet)

# Forecast on the test set
future = model.make_future_dataframe(periods=len(test_data))
forecast = model.predict(future)

# Extract the predicted values
predictions = forecast['yhat'][train_size:]
```

Listing 7.7: Preparing the model

Fig. 7.7: Difference between Actual and predicted data

7.3 LSTM

LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) that has grown in preference for tasks involving time series forecasting. It is a deep learning model that is best suited for evaluating and predicting time series since it is made to capture persistent dependencies and patterns in sequential data.

When dealing with time series data that have complicated temporal dependencies, non-linear relationships, and variable-length sequences, LSTM models perform very well. They can identify patterns in the data, including trends, seasonality, and irregularities.

And here in the project after applying this model 3 times with different 3 datasets. Its performance was almost the best and the most accurate as will be discussed next in the next parts.

7.3.1 Dataset 1 - Discount Rate

For this model it will be a little more complex than the previous models but eventually with higher performance and more accuracy. This code (Listing 7.8 below) first uses the MinMaxScaler from the scikit-learn library to scale the data between 0 and 1. And then splits the scaled data into training and testing sets, with 80% of the data used for training and 20% used for testing as mentioned before. Then the pre-processing part and the reshaping to adjust the dataset and make it ready for the model by reshaping the training and testing sets into the format of input (X) and output (Y) pairs for an LSTM model. Specifically, the input is the current value (X=t) and the output is the next value (Y=t+1).

After that the building of the LSTM model with two layers of 50 neurons each, followed by a single dense layer with one output. The model is compiled with a mean squared error loss function and the Adam optimizer. The model is then trained on the training set for 10 epochs with a batch size of 1 and a verbose level of 2.

This makes this model trains the data many times and the more epochs are the more accurate predictions till some certain point it could reach some overtraining and could lead to unacceptable inaccurate predictions.

The default parameters were used for most steps, but the key parameters such as the number of time steps, LSTM units, epochs, and batch size adjusted a little based on this dataset specific requirements.

LSTM Layers:

50 Units: Each LSTM layer has 50 units. This number adjusted based on the model complexity and data characteristics. More units can capture more complex patterns but may require more data and computational power.

Fitting the Model:

Epochs=10: The number of epochs is set to 10, meaning the model will iterate over the entire training set 10 times. The choice of 10 epochs and a batch size of 1 is a trade-off between training time and performance. More epochs can lead to better performance but risk overfitting, while a larger batch size can speed up training but may reduce model accuracy.

Batch_size=1: This means the model weights are updated after each training example. This can be set higher to improve training speed but may affect convergence.

```
# Scale the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_values = scaler.fit_transform(df.values.reshape(-1, 1))

# Build LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(1, 1)))
```

```
model.add(LSTM(50, return_sequences=False))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, Y_train, validation_data=(X_test, Y_test),
epochs=10, batch_size=1, verbose=2)

# Make predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
```

Listing 7.8: Building the LSTM model

Here is the final graph as it can be seen in Figure 7.8 that the accuracy of this model on this dataset with almost identical to the actual data which means the performance of this model was very good but this will be discussed in more detail in the next section.

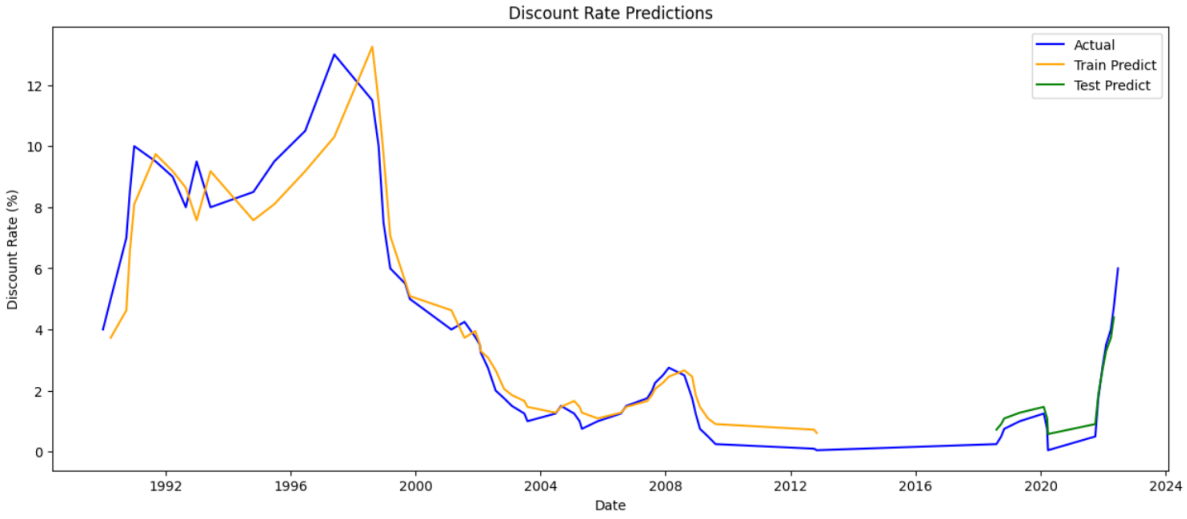

Fig. 7.8: Difference between Actual, Trained and tested data

7.3.2 Dataset 2 - Number of accidents

In the second dataset the data are bigger than the first data so that means more possible training to the model and is supposed to be with a better performance and accuracy.

As mentioned before, the dataset will be imported, pre-processed and split into training and test. Normalize the data using the MinMaxScaler from scikit-learn. Normalizing the

data is a common pre-processing step that scales the values to a range of 0 to 1. And then Define a function called (create_sequences) that creates input/output sequences for the LSTM model. The function takes in a time series dataset and a number of time steps as input. It returns two numpy arrays: one containing the input sequences and one containing the output sequences. Define the number of time steps to use in the LSTM model. The time steps represent the number of past values that the model will use to predict the next value in the sequence. Using the (create_sequences) function to create input/output sequences for the training data. The input sequences are created by selecting a window of time steps from the training data which created before, and the output sequences are created by selecting the value that follows the end of each input sequence.

Using the Sequential API of Keras to construct the LSTM model. Each of the model's two LSTM layers includes 50 memory units. Sequences are returned by the first LSTM layer but not by the second. A dense layer with only one output unit makes up the top layer. The optimizer is Adam., while mean squared error serves as the loss metric.

Train the LSTM model on the training data using the fit method. The model is trained for 5 epochs, with a batch size of 1. Verbose is set to 2 to print progress updates. Use the predict method of the LSTM model to generate predictions for the test data.

Rescale the predicted values using the inverse of the scaler used for the training data. The predicted values now represent the original scale of the time series.

The default parameters were used for most steps, but the key parameters such as the number of time steps, LSTM units, epochs, and batch size adjusted a little based on this dataset specific requirements.

Model Architecture:

First LSTM Layer: 50 units with return_sequences=True to return the full sequence to the next LSTM layer.

Second LSTM Layer: 50 units without return_sequences as it is the last LSTM layer.

Dense Layer: A single neuron to output the prediction.

Sequence Creation: A function is defined to create sequences of a specified number of time steps (7 in this case). This means the model will look back 7 days to make a prediction.

Time Steps: The choice of 7 time steps can be based on domain knowledge (e.g., weekly patterns)

```
# Create the training data sequences
def create_sequences(data, time_steps):
    X, Y = [], []
    for i in range(len(data) - time_steps - 1):
        X.append(data[i:(i + time_steps), 0])
        Y.append(data[i + time_steps, 0])
    return np.array(X), np.array(Y)

# Define the number of time steps
time_steps = 7

# Create the training sequences
X_train, Y_train = create_sequences(train_data, time_steps)

# Reshape the input data for LSTM
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],
1))

# Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(time_steps,
1)))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, Y_train, epochs=5, batch_size=1, verbose=2)

# Predict on the test data
inputs = df2[-len(test_data)-time_steps:].values.reshape(-1, 1)
inputs = scaler.transform(inputs)
X_test, Y_test = create_sequences(inputs, time_steps)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted = model.predict(X_test)
predicted = scaler.inverse_transform(predicted)
```

Listing 7.9: The prediction of the model

7.3.3 Dataset 3 - Yahoo Finance

Here also like the second dataset the data are bigger than the first dataset which means more training and more accuracy.

First as usual the importing and pre-processing of the data, then the splitting as mentioned many times before. After that Two arguments are passed to the function create_sequences and seq_length, the length of the input sequence, and data, a time series of data. The function generates input-output pairs from the time series data, with each input sequence having length seq_length and the associated output being the subsequent value in the time series. The result of the function is two numpy arrays: y, which includes the corresponding outputs, and X, which contains the input sequences.

The create_sequences function is then used by the algorithm to produce training and testing sequences for the train and test sets of data, respectively. The two numpy arrays that are produced, X_train and X_test, have the following shapes: (n_samples, sequence_length, 1), where n_samples is the total number of samples in the dataset. Numpy arrays of shape (n_samples,) make up the y_train and y_test variables.

The input data X_train and X_test are then reshaped into a 3D array of shape (n_samples, sequence_length, 1) to be compatible with the LSTM model.

The Sequential class from Keras is then used in the code to create an LSTM model. Two LSTM layers with 50 memory units each make up the model. To guarantee that a sequence rather than a single value is output from the first LSTM layer, the return_sequences parameter is set to True. At the network's ends, a dense layer with a single output is added. The mean_squared_error loss function and the Adam optimizer are used in the model's compilation.

The model is trained on the training data using the fit method of the model object. The training is run for 50 epochs with a batch size of 32 as described below in the Listing 7.10.

After training, the model is used to make predictions on the test data using the predict method of the model object. The resulting (predictions).

Finally, the predicted values and the actual values are inverse transformed using the inverse_transform method of the scaler object used to scale the data.

```
# Define the function to create input and output sequences
def create_sequences(data, seq_length):
    X = []
    y = []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length])
    return np.array(X), np.array(y)

# Set the sequence length
sequence_length = 10

# Create training sequences
X_train, y_train = create_sequences(train_data, sequence_length)

# Create testing sequences
X_test, y_test = create_sequences(test_data, sequence_length)

# Reshape the input data for LSTM
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],
1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True,
input_shape=(sequence_length, 1)))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32)

# Perform predictions on the test set
predictions = model.predict(X_test)

# Inverse transform the predictions and actual values
```

```
predictions = scaler.inverse_transform(predictions)
y_test = scaler.inverse_transform(y_test)
```

Listing 7.10: The LSTM model

And the graph in Figure 7.9 here shows that the predicted data is almost identical to the actual data and follows its pattern and that's because of the long training and also training using epochs which allows the model to improve its performance over time by adjusting its weights to better fit the training data. In general, as the number of epochs increases, the model has the potential to learn more complex patterns and relationships in the data, which can improve its accuracy on both the training and testing data.

The number of epochs must be balanced with the possibility of overfitting, though. Overfitting happens when a model loses its ability to generalize to new data because it becomes too concentrated on the training set of data. The model may overfit the training data and perform badly on the testing data if the number of epochs is too high.

Therefore, it is important to choose the number of epochs carefully. This hyperparameter is frequently modified during the model selection and optimization process. The size of the dataset, the complexity of the issue, and the design of the LSTM model can all affect the perfect number of epochs.

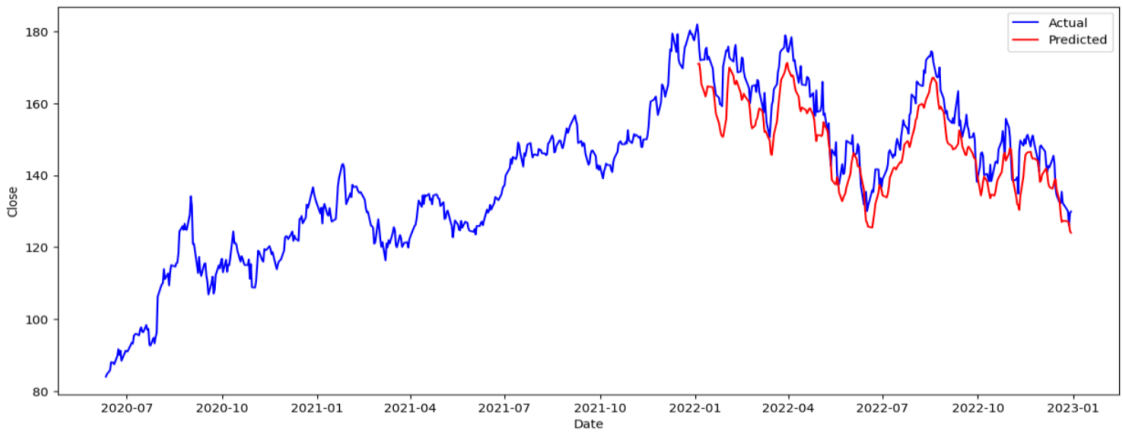

Fig. 7.9: Difference between Actual and predicted data

8. TIME SERIES MODELS EVALUATION

There are many ways or approaches to evaluate and modify our model and dataset we could use until we get the perfect and most performance output

Time series metrics

To go further with model selection, you will need to define a metric to evaluate your models. A very often used model in forecasting is the **Mean Squared Error**. This metric measures the error at each point in time and takes the square of it. The average of those squared errors is called the Mean Squared Error. Also, there is an often-used alternative is the **Root Mean Squared Error**: the square root of the Mean Squared Error.

Another frequently used metric is the **Mean Absolute Error**: rather than taking the square of each error, it takes the absolute value here. The Mean Absolute Percent Error is a variation on this where the Absolute Error at each point in time is expressed as a percentage of the actual value. This yields a metric that is a percentage, which is very easy to interpret [42].

8.1 Mean Squared Error

MSE stands for Mean Squared Error, and it is a widely used metric for evaluating the performance of regression models. It measures the average squared difference between the predicted and actual values in a regression problem.

The formula for MSE is:

$$MSE = 1/n * \Sigma(yi - \hat{y}i)^2. \quad (8.1)$$

Where:

n: the number of samples in the dataset.

yi: the actual value of the target variable for the i-th sample.

ŷi: the predicted value of the target variable for the i-th sample.

A measurement error of 0 indicates a perfect fit, where the predicted and actual values are the same for all samples. The MSE value is always non-negative. The performance of the model is worse the higher the MSE as this shows greater differences between the predicted and actual values.

As a result, it is suggested that using MSE along with other metrics to evaluate the performance of a regression model. These metrics can offer additional information on the model's performance and assist in the detection of potential problems. For example, MAE can be used to examine the amount of errors, and R-squared can be used to evaluate the model's overall goodness-of-fit. So it should be used in combination with other metrics to evaluate the performance of a regression model [43].

8.2 Root Mean Squared Error

Root Mean Squared Error (RMSE) is another metric used to evaluate the performance of a regression model. It is very similar to Mean Squared Error (MSE), but it has one additional step - taking the square root of the MSE.

$$RMSE = sqrt(MSE). \quad (8.2)$$

Where MSE is the mean of the squared differences between the predicted and actual values of the target variable.

Therefore, it is always suggested that you use MSE alongside with other metrics to evaluate a regression model's performance as mentioned before. These metrics can give additional information about the model's performance and aid in problem detection. For instance, R-squared can be utilized to evaluate the overall goodness-of-fit of the model, and MAE can be used to evaluate the size of errors [44].

8.3 Mean Absolute Error

Mean Absolute Error (MAE) is also a metric used to evaluate the performance of regression models like the previous models. It measures the average absolute difference

between the predicted and actual values in a dataset. But unlike Mean Squared Error (MSE), MAE is not sensitive to outliers because it does not involve squaring the errors.

The mathematical equation for MAE is as follows:

$$MAE = (1/n) * \Sigma|i = 1\ to\ n|\ |yi - \hat{y}i|. \quad (8.3)$$

where n is the number of data points, yi is the actual value of the target variable, and ŷi is the predicted value of the target variable.

It provides an idea of how far on average the predictions deviate from the actual numbers. A lower MAE shows that the model is more accurate in its predictions.

When it comes to the direction of the errors, or whether the forecasts are overestimating or underestimating the actual values, MAE isn't the perfect metric to do that.

8.4 Evaluation of the performance of the datasets

So eventually the best scenario is to use the 3 metrics with each other for each dataset evaluation and compare their performance with each other.

8.4.1 Evaluation of the first dataset

The MAE, MSE, and RMSE of every model would be calculated for the first dataset. Then a table would be implemented for the comparison and decide which model has the best performance and accuracy.

The following code (Listing 8.1) for the evaluation of the model for the first model of SARIMAX

```
# Evaluate the model
mae_1_2 = mean_absolute_error(test_data['CNB_DISCOUNT_RATE_IN_%'],
predictions)
mse_1_2 = mean_squared_error(test_data['CNB_DISCOUNT_RATE_IN_%'],
predictions)
rmse_1_2 = mean_squared_error(test_data['CNB_DISCOUNT_RATE_IN_%'],
predictions, squared=False)
```

Listing 8.1: Evaluation of the model

And here's the evaluation's results for SARIMAX model.

```
Evaluation Metrics:
Mean Absolute Error (MAE): 3.42
Mean Squared Error (MSE): 14.95
Root Mean Squared Error (RMSE): 3.87
```

Also for the next evaluation of the Prophet model will do almost the same in Listing 8.2 below.

```
# Evaluate the model
mae_1_2 = mean_absolute_error(test_data['CNB_DISCOUNT_RATE_IN_%'],
predictions)
mse_1_2 = mean_squared_error(test_data['CNB_DISCOUNT_RATE_IN_%'],
predictions)
rmse_1_2 = mean_squared_error(test_data['CNB_DISCOUNT_RATE_IN_%'],
predictions, squared=False)
```

Listing 8.2: Evaluation of the Prophet model

And here's again the evaluation's results for the Prophet model.

```
Evaluation Metrics:
Mean Absolute Error (MAE): 3.76
Mean Squared Error (MSE): 18.60
Root Mean Squared Error (RMSE): 4.31
```

Finally for the last model LSTM the following code for the evaluation of the model in the Listing 8.3.

```
# Calculate MSE
train_mse = mean_squared_error(y_train[:-1], train_predict)
mse_1_3 = mean_squared_error(y_test[:-1], test_predict)

# Calculate RMSE
train_rmse = math.sqrt(train_mse)
rmse_1_3 = math.sqrt(mse_1_3)

# Calculate MAE
train_mae = mean_absolute_error(y_train[:-1], train_predict)
mae_1_3 = mean_absolute_error(y_test[:-1], test_predict)
```

Listing 8.3: Evaluation of the LSTM model

And here's the evaluation's results for the LSTM model.

```
MSE: 0.4533803426218304
RMSE: 0.6733352379178075
MAE: 0.5531002342700959
```

And the Table 8.1 below is the evaluation trade-off for the First Dataset and the colour of the cells has been done according to this threshold

```
def style_cells(value):
    style = 'background-color: red' if value > 4 else  'background-color: green'
    return style
```

Listing 8.4: Visualising the results

	MSE	RMSE	MAE
SARIMAX	14.949495	3.866458	3.417446
Prophet	18.600330	4.312810	3.759066
LSTM	0.453380	0.673335	0.553100

Tab. 8.1: Comparison of the evaluation metrics for the 1st dataset with threshold 4

Looking at the given metrics, it seems that the LSTM model has performed better than the other two models. The MSE, RMSE and MAE values for LSTM are the lowest among the three models, indicating that the predictions made by the LSTM model are the closest to the actual values.

The reason for this difference in performance between the models could be due to the differences in the algorithms used by each model. SARIMAX and Prophet are both classical time series models that rely on statistical methods to make predictions, whereas LSTM is a type of deep learning model that uses neural networks to model the patterns in the data.

LSTM is known to perform well on time series data due to its ability to capture complex patterns and dependencies in the data. In contrast, classical time series models like SARIMAX and Prophet may struggle with capturing such patterns and may not perform as well on datasets with complex dependencies.

Additionally, the quality of the data and the features used by each model could also play a role in the differences in performance. It is possible that the LSTM model was able to learn more informative features from the data and therefore was able to make better predictions.

8.4.2 Evaluation of the Second dataset

	MSE	RMSE	MAE
SARIMAX	5813.758536	76.248007	54.865720
Prophet	3530.434766	59.417462	39.122976
LSTM	2507.393216	50.073878	37.646138

Tab. 8.2: Comparison of the evaluation metrics for the 2nd dataset with threshold 50

Here also looking at the results in the Table 8.2 , it seems like the LSTM model performed the best across all three metrics, with the lowest values of MSE, RMSE, and MAE. The SARIMAX and Prophet models, on the other hand, had higher values across all three metrics.

But here also it could be seen that in this time the Prophet's performance was better than SARIMAX and had better values. This could be SARIMAX has some assumptions regarding stationarity, linearity, and the normal distribution of residuals that need to be met for the model to work properly. These assumptions may not hold for this dataset On the other hand, Prophet does not have such assumptions and can handle non-linear and

non-stationary data better. Also Prophet sometimes is more flexible and can handle a wide range of data types and structures.

8.4.3 Evaluation of the Third dataset

And finally, The results in Table 8.3 shows the comparison of the evaluation metrics for the 3rd dataset with threshold 4, and the model of SARIMAX had the lowest values which means its performance was the best. This should be due to its ability to handle seasonality and trends effectively, and it has more effective tuning and parameter optimization compared to the other models in this dataset. Prophet is also designed to handle seasonality and trends, it might not be as precise as SARIMAX for this type of dataset.

	MSE	RMSE	MAE
SARIMAX	4.180719	2.044681	1.296307
Prophet	3782.017309	61.498108	58.879229
LSTM	38.821212	6.230667	5.282454

Tab. 8.3: Comparison of the evaluation metrics for the 3rd dataset with threshold 4

9. THE USER INTERFACE APPLICATION

In this project Prophet model would be applied to a Multivariate dataset of "Temperature Forecasting for IOT Device generated Data" this time, and there will be a creation of a user interface application with 2 features of choosing of IN/OUT Temperature status and the number of time points of the forecasting.

In this dataset, temperature readings were taken from IoT devices installed outside and inside of an anonymous room. Due to the testing phase of the device, it was uninstalled or shut off several times during the entire reading period, resulting in some outliers and missing values.

Building a time-series model to predict future temperature inside/outside the room by Prophet.

Prophet was chosen this time for the time-series modelling tool based on below reasons:

- Automatic detection of trend and seasonality.
- Robustness against outliers.
- Customizable seasonality.

Of course, The first steps would be done similarly to before with the previous datasets like Importing the libraries and loading the dataset, which is included in Listing 9.1 below.

---

https://colab.research.google.com/drive/1HX6OXR1bb-r61qVLtezXu4EQFvyys-Ws?usp=sharing

```
import numpy as np
import pandas as pd
import holoviews as hv
from holoviews import opts
hv.extension('bokeh')
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import os
from prophet import Prophet
from prophet.plot import add_changepoints_to_plot
import streamlit as st
df = pd.read_csv(r"./IOT-temp.csv"
```

Listing 9.1: Importing the libraries and loading the dataset

Then Pre-processing of the data as Column 'room_id/id' has only one value(Room Admin), so we don't need this column for analysis, and Change column names to understand easily.

```
df['room_id/id'].value_counts()
df.rename(columns={'noted_date':'date', 'out/in':'place'},
inplace=True)
df.head()
```

Listing 9.2: data pre-processing

Datetime column has a lot of information such as year, month, weekday and so on. To utilize this information in EDA and modelling phase, we need extract them from datetime column.

Hour variable can be broken into Night, Morning, Afternoon, and Evening based on its number.

- Night : 22:00 - 23:59 / 00:00 - 03:59.
- Morning : 04:00 - 11:59.
- Afternoon : 12:00 - 16:59.
- Evening : 17:00 - 21:59.

|Also for the Seasonal information, it could be broken into for example the Indian seasonal which has four climatological seasons as below.

- Winter : December to February.
- Summer : March to May.
- Monsoon : June to September.
- Post-monsoon : October to November.

Listing 9.2 below describing how to implement that.

```
def month2seasons(x):
    if x in [12, 1, 2]:
        season = 'Winter'
    elif x in [3, 4, 5]:
        season = 'Summer'
    elif x in [6, 7, 8, 9]:
        season = 'Monsoon'
    elif x in [10, 11]:
        season = 'Post_Monsoon'
    return season

def hours2timing(x):
    if x in [22,23,0,1,2,3]:
        timing = 'Night'
    elif x in range(4, 12):
        timing = 'Morning'
    elif x in range(12, 17):
        timing = 'Afternoon'
    elif x in range(17, 22):
        timing = 'Evening'
    else:
        timing = 'X'
    return timing
```

Listing 9.3: Season and hour variable

And after many other modifications to the dataset like checking whether any record is duplicated and if so duplicate records should be put into one unique record (Listing 9.4).

```
df[df.duplicated()]
df[df['id']=='__export__.temp_log_196108_4a983c7e']
df.drop_duplicates(inplace=True)
df[df.duplicated()]
```

Listing 9.4: Duplication detection

And many other pre-processing of the data, the semi-final shape of the data would be like this in Figure 9.1 below.

	id	date	temp	place	year	month	day	weekday	weekofyear	hour	minute	season	timing
84141	4000	2018-09-09 16:24:00	29	Out	2018	9	9	Sunday	36	16	24	Monsoon	Afternoon
84142	4002	2018-09-09 16:24:00	29	Out	2018	9	9	Sunday	36	16	24	Monsoon	Afternoon
84144	4004	2018-09-09 16:23:00	28	Out	2018	9	9	Sunday	36	16	23	Monsoon	Afternoon
84128	4006	2018-09-09 16:24:00	28	Out	2018	9	9	Sunday	36	16	24	Monsoon	Afternoon
84132	4007	2018-09-09 16:24:00	29	Out	2018	9	9	Sunday	36	16	24	Monsoon	Afternoon
84136	4009	2018-09-09 16:24:00	28	Out	2018	9	9	Sunday	36	16	24	Monsoon	Afternoon
84137	4010	2018-09-09 16:24:00	28	Out	2018	9	9	Sunday	36	16	24	Monsoon	Afternoon

Fig. 9.1: Dataset after pre-processing

As this is Multivariate so, Temperature clearly consists of multiple distributions of Place, Season, and Timing.

Monthly Readings by Place

```
pl_cnt = np.round(df['place'].value_counts(normalize=True) * 100)

in_month = np.round(df[df['place']=='In']['date'].apply(lambda x :
x.strftime("%Y-%m")).value_counts(normalize=True).sort_index() *
100, decimals=1)
out_month = np.round(df[df['place']=='Out']['date'].apply(lambda x
: x.strftime("%Y-%m")).value_counts(normalize=True).sort_index() *
100, decimals=1)
```

```
in_out_month =
pd.merge(in_month,out_month,right_index=True,left_index=True).renam
e(columns={'date_x':'In', 'date_y':'Out'})
in_out_month = pd.melt(in_out_month.reset_index(),
['index']).rename(columns={'index':'Month', 'variable':'Place'})
```

Listing 9.5: Monthly reading by place

Inside temperature is composed of a single distribution, while outside temperature is composed of multiple distributions as implemented in Listing 9.6 below.

Temperature by Season

```
season_agg = df.groupby('season').agg({'temp': ['min', 'max']})
season_maxmin =
pd.merge(season_agg['temp']['max'],season_agg['temp']['min'],right_
index=True,left_index=True)
season_maxmin = pd.melt(season_maxmin.reset_index(),
['season']).rename(columns={'season':'Season',
'variable':'Max/Min'})
```

Listing 9.6: Temperature by Season

Temperature by Timing

```
timing_agg = df.groupby('timing').agg({'temp': ['min', 'max']})
timing_maxmin =
pd.merge(timing_agg['temp']['max'],timing_agg['temp']['min'],right_
index=True,left_index=True)
timing_maxmin = pd.melt(timing_maxmin.reset_index(),
['timing']).rename(columns={'timing':'Timing',
'variable':'Max/Min'})
```

Listing 9.7: Temperature by Timing

The outside temperature has a larger time series change than the inside temperature.

It is thought that the inside temperature is adjusted by air conditioner, but the outside temperature is affected by seasonal temperature fluctuations.

Time-series analysis can be easily conducted with unique time-index data. Thus, mean values need to be calculated by the 'date' column, and the 'id' column should be deleted.

Daily and Monthly Temperature Mean

```
in_month =
tsdf[tsdf['place']=='In'].groupby('month').agg({'temp':['mean']})
in_month.columns = [f"{i[0]}_{i[1]}" for i in in_month.columns]
out_month =
tsdf[tsdf['place']=='Out'].groupby('month').agg({'temp':['mean']})
out_month.columns = [f"{i[0]}_{i[1]}" for i in out_month.columns]

tsdf['daily'] = tsdf['date'].apply(lambda x :
pd.to_datetime(x.strftime('%Y-%m-%d')))
in_day =
tsdf[tsdf['place']=='In'].groupby(['daily']).agg({'temp':['mean']})
in_day.columns = [f"{i[0]}_{i[1]}" for i in in_day.columns]
out_day =
tsdf[tsdf['place']=='Out'].groupby(['daily']).agg({'temp':['mean']}
)
out_day.columns = [f"{i[0]}_{i[1]}" for i in out_day.columns]


import matplotlib.pyplot as plt

# Extracting data from HoloViews Curves
in_day_data = in_day.reset_index()
out_day_data = out_day.reset_index()

# Plotting with Matplotlib
plt.figure(figsize=(10, 6))

plt.plot(in_day_data['daily'], in_day_data['temp_mean'],
label='In', marker='o')
plt.plot(out_day_data['daily'], out_day_data['temp_mean'],
label='Out', marker='o')

# Adding labels and title
plt.title("Daily Temperature Mean")
plt.xlabel("Day")
plt.ylabel("Temperature")
plt.legend()
plt.grid(True)

# Show plot
plt.show()
```

Listing 9.8: Temperature Mean

And here is the Figure 9.2 showing the difference between In and Out.

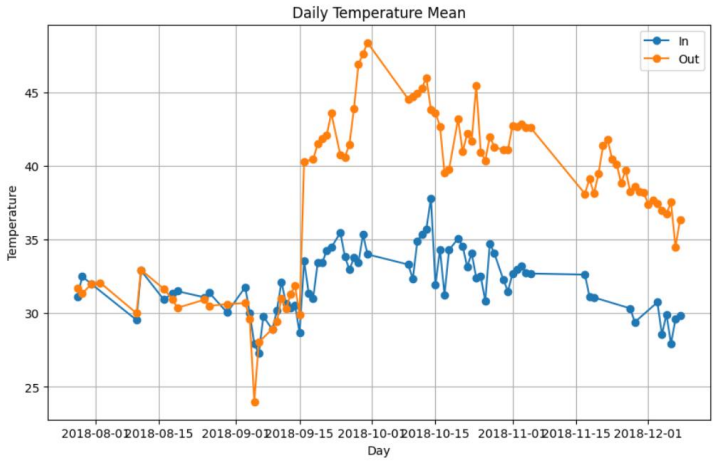

Fig. 9.2: Daily Temperature Mean

Missing data

Plotting overall data, it is found that there are some missing data points randomly throughout the whole period.

Interpolating with 'nearest' method looks better(yet far from best), but there are many gaps in interpolated data.

In order to forecast future temperature, it’s better to convert data into rough granularity.

Using interpolated daily mean data looks good solution as in Listing 9.9 below.

```
in_tsdf = tsdf[tsdf['place']=='In'].reset_index(drop=True)
in_tsdf.index = in_tsdf['date']
in_all = hv.Curve(in_tsdf['temp']).opts(title="[In] Temperature
All", ylabel="Temperature", xlabel='Time', color='red')

out_tsdf = tsdf[tsdf['place']=='Out'].reset_index(drop=True)
out_tsdf.index = out_tsdf['date']
out_all = hv.Curve(out_tsdf['temp']).opts(title="[Out] Temperature
All", ylabel="Temperature", xlabel='Time', color='blue')
```

```
in_tsdf_int =
in_tsdf['temp'].resample('1min').interpolate(method='nearest')
in_tsdf_int_all = hv.Curve(in_tsdf_int).opts(title="[In]
Temperature All Interpolated with 'nearest'", ylabel="Temperature",
xlabel='Time', color='red', fontsize={'title':11})
out_tsdf_int =
out_tsdf['temp'].resample('1min').interpolate(method='nearest')
out_tsdf_int_all = hv.Curve(out_tsdf_int).opts(title="[Out]
Temperature All Interpolated with 'nearest'", ylabel="Temperature",
xlabel='Time', color='blue', fontsize={'title':11})
inp_df = pd.DataFrame()
in_d_inp = in_day.resample('1D').interpolate('spline', order=5)
out_d_inp = out_day.resample('1D').interpolate('spline', order=5)
inp_df['In'] = in_d_inp.temp_mean
inp_df['Out'] = out_d_inp.temp_mean
```

Listing 9.9: Missing data

Hyperparameter tuning

Here, some hyperparameter tuning techniques would be applied to the model to see if the performance would change and to choose the best performer among them. Below in Listing 9.10 some snippet of the code with apply grid search to select best hyperparameters.

```
# Define the function to evaluate the model
def evaluate_model(place, changepoint_prior_scale,
yearly_seasonality, weekly_seasonality):
    prediction_periods = 30  # Define the number of periods to
predict
    result = run_prophet_with_params(place, prediction_periods,
changepoint_prior_scale, yearly_seasonality, weekly_seasonality)

    # Calculate MSE between actual and predicted values
    actual = org_df[place][-prediction_periods:].values
    predicted = result['yhat'][-prediction_periods:].values
    mse = mean_squared_error(actual, predicted)

    return mse

# Grid search parameters
```

```
param_grid = {
    'changepoint_prior_scale': [0.01, 0.1, 0.5],
    'yearly_seasonality': [2, 5, 10],
    'weekly_seasonality': [False, True]
}

# Generate all combinations of parameters
param_combinations =
list(product(param_grid['changepoint_prior_scale'],
param_grid['yearly_seasonality'],
param_grid['weekly_seasonality']))

# Perform grid search
best_params = None
best_mse = float('inf')
results = []

for params in param_combinations:
    changepoint_prior_scale, yearly_seasonality, weekly_seasonality
= params
    mse = evaluate_model("Out", changepoint_prior_scale,
yearly_seasonality, weekly_seasonality)
    results.append((params, mse))

    if mse < best_mse:
        best_mse = mse
        best_params = params

# Print the best parameters and MSE
print("Best Parameters:", best_params)
print("Best MSE:", best_mse)

# Optionally, you can convert results to a DataFrame for better
visualization
results_df = pd.DataFrame(results, columns=['Parameters', 'MSE'])
print(results_df)
```

Listing 9.10: Hyperparameter tunning

And here are the results with the best MSE as shown in Figure 9.3.

```
Best MSE: 4.5756250669266185
           Parameters         MSE
0    (0.01, 2, False)  132.697364
1     (0.01, 2, True)  153.003574
```

```
2     (0.01, 5, False)     4.630651
3      (0.01, 5, True)    14.546761
4    (0.01, 10, False)     5.546444
5     (0.01, 10, True)     6.167597
6      (0.1, 2, False)   139.604486
7       (0.1, 2, True)   136.147785
8      (0.1, 5, False)     4.575625
9       (0.1, 5, True)     5.723650
10   (0.1, 10, False)     6.163159
11    (0.1, 10, True)     6.170643
12     (0.5, 2, False)   371.156863
13      (0.5, 2, True)   408.270783
14     (0.5, 5, False)    20.028329
15      (0.5, 5, True)     9.234085
16   (0.5, 10, False)   213.200977

17    (0.5, 10, True)    20.372572
```

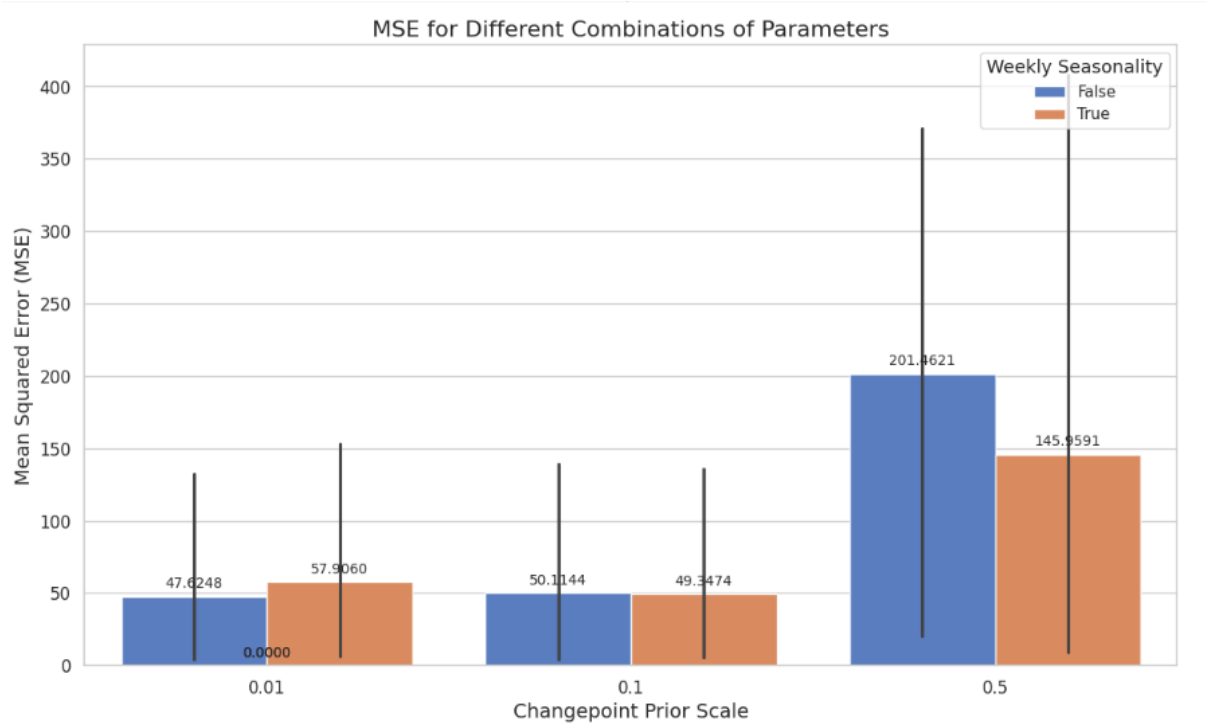

Fig. 9.3: MSE for Different Combinations of Parameters

Below in Listing 9.11 describes the build of the model and the prediction.

```
def run_prophet(place, prediction_periods, plot_comp=True):
    # make dataframe for training
    prophet_df = pd.DataFrame()
    prophet_df["ds"] = pd.date_range(start=org_df['daily'][0],
end=org_df['daily'][133])
    prophet_df['y'] = org_df[place]
```

```
    # add seasonal information
    prophet_df['monsoon'] = org_df['season_Monsoon']
    prophet_df['post_monsoon'] = org_df['season_Post_Monsoon']
    prophet_df['winter'] = org_df['season_Winter']

    # train model by Prophet
    m = Prophet(changepoint_prior_scale=0.1, yearly_seasonality=2,
weekly_seasonality=False)
    # include seasonal periodicity into the model
    m.add_seasonality(name='season_monsoon', period=124,
fourier_order=5, prior_scale=0.1, condition_name='monsoon')
    m.add_seasonality(name='season_post_monsoon', period=62,
fourier_order=5, prior_scale=0.1, condition_name='post_monsoon')
    m.add_seasonality(name='season_winter', period=93,
fourier_order=5, prior_scale=0.1, condition_name='winter')
    m.fit(prophet_df)

    # make dataframe for prediction
    future = m.make_future_dataframe(periods=prediction_periods)
    # add seasonal information
    future_season = pd.get_dummies(future['ds'].apply(lambda x :
month2seasons(x.month)))
    future['monsoon'] = future_season['Monsoon']
    future['post_monsoon'] = future_season['Monsoon']
    future['winter'] = future_season['Winter']

    # predict the future temperature
    prophe_result = m.predict(future)
```

Listing 9.11: The build

As it can be seen in Figures 9.4 and 9.5 The IN and OUT predictions of the Temperature during the Time.

The Red Line is the Linear introduction of the mean of the points.

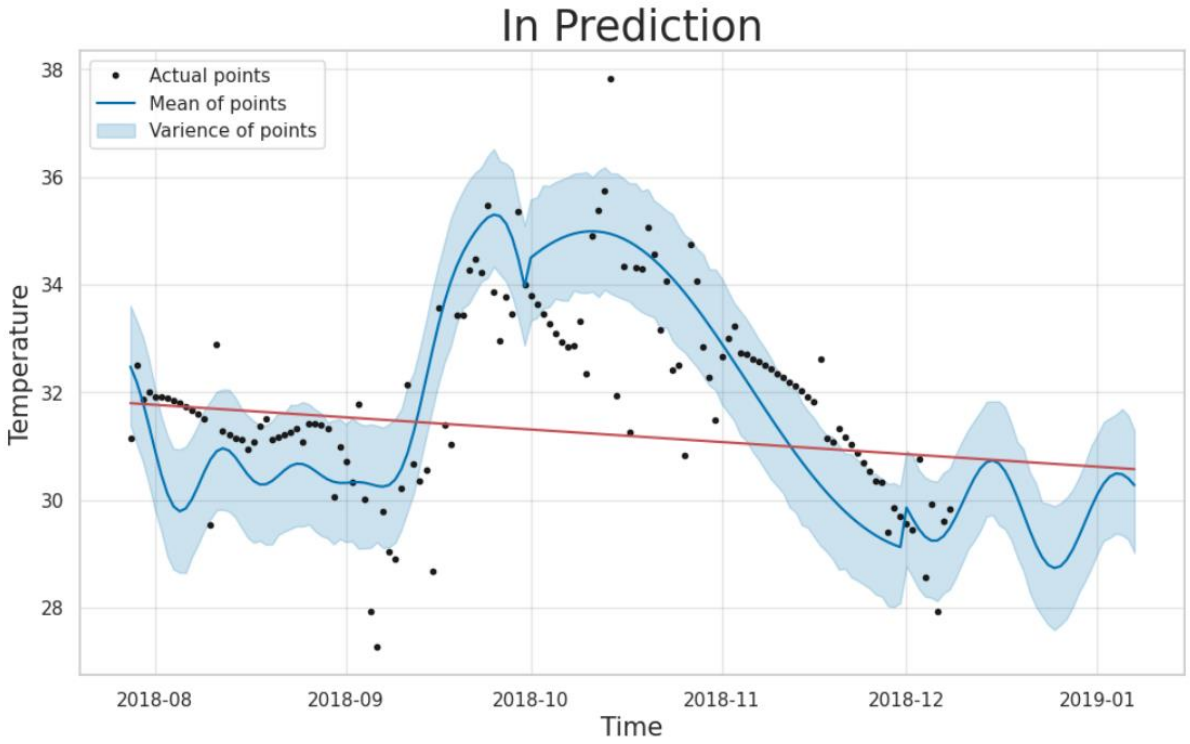

Fig. 9.4: Prediction of IN temperature

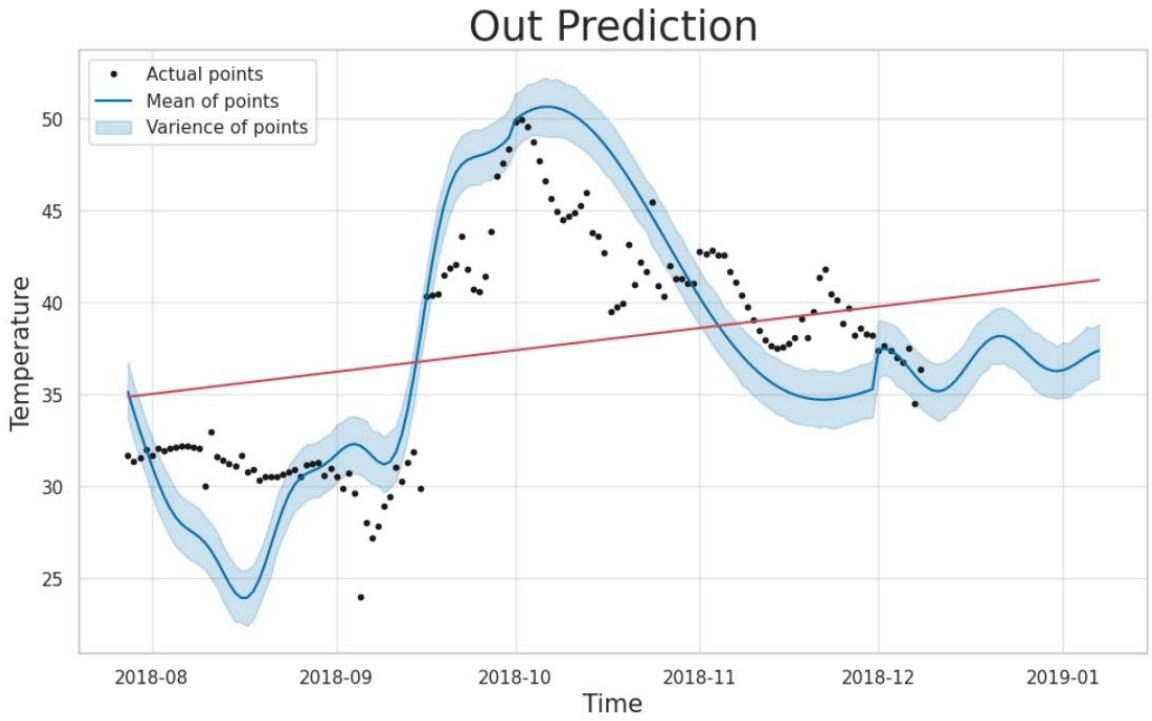

Fig. 9.5: Prediction of IN temperature

In the provided code, the features used for the Prophet model are:

Date-Time Features:

- Year: Extracted from the timestamp to capture any yearly trends or seasonality.
- Month: Extracted to capture seasonal patterns within each month.
- Day: Extracted to account for any potential daily variations or trends.
- Weekday: Extracted to capture any day-of-week patterns or fluctuations.
- Weekofyear: Derived to understand the week of the year, which might have its own patterns.
- Hour: Extracted to capture variations throughout the day.
- Minute: Extracted for more granular analysis, although not utilized in this specific model.

Seasonal Features:

- Season: Derived from the month to categorize data into different seasons (Winter, Summer, Monsoon, Post-Monsoon). This helps capture seasonal patterns and trends specific to each season.

Additional Temporal Features:

- Timing: Derived from the hour to categorize data into different time segments (Morning, Afternoon, Evening, Night). This adds another layer of temporal granularity, capturing potential variations based on the time of day.

These features are used to provide the model with as much relevant information as possible to capture the underlying patterns and trends in the temperature data. By including various temporal features, the model can learn and account for different patterns that may emerge at different time scales, such as daily, weekly, monthly, or seasonal patterns. Additionally, incorporating seasonal information allows the model to capture recurring patterns associated with different seasons, which can significantly impact temperature fluctuations.

And for the user interface application, Streamlit app would be used for this with the two main features of choosing the Temperature IN/OUT and the second one of choosing the

TimePoints as the days of the forecasting with the following piece of code (Listing 9.12).

```
# Streamlit app
def main():
    st.title("Temperature Prediction App")

    # Dropdown for Temperature status
    temp_status = st.selectbox("Temperature status:", ["IN",
"OUT"])

    # Number input for TimePoints
    time_points = st.number_input("TimePoints:", min_value=1,
step=1, value=30)

    # Button to execute the code
    if st.button("Run Prophet"):
        if temp_status == "IN":
            run_prophet("In", time_points)
        else:
            run_prophet("Out", time_points)
```

Listing 9.12: The user interface app

And the output of this application would be in the Figure 9.6 below

The main page will have two options that the user can change:

- Temperature statues: It has two options to choose from IN/OUT
- TimePoints: Each Point represents one day, and the user can change the days (points) according to their desire to forecast the desired duration.

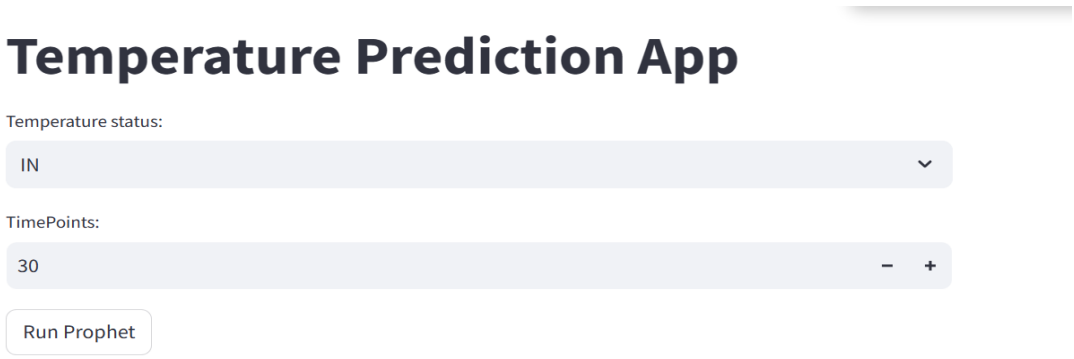

Fig. 9.6: The main page of the app

As the TimePoints could be increased or decreased

Here are the results of 30 TimePoint and 90 TimePoints in Figures 9.7 and 9.8 respectively below.

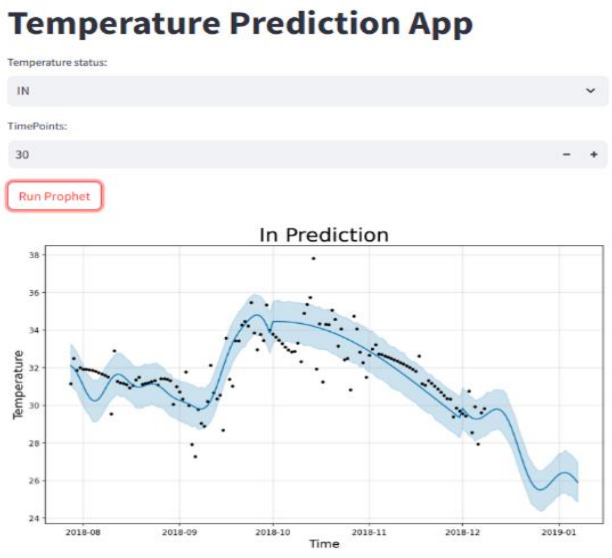

Fig. 9.7: Prediction of IN temperature with 30 TimePoints

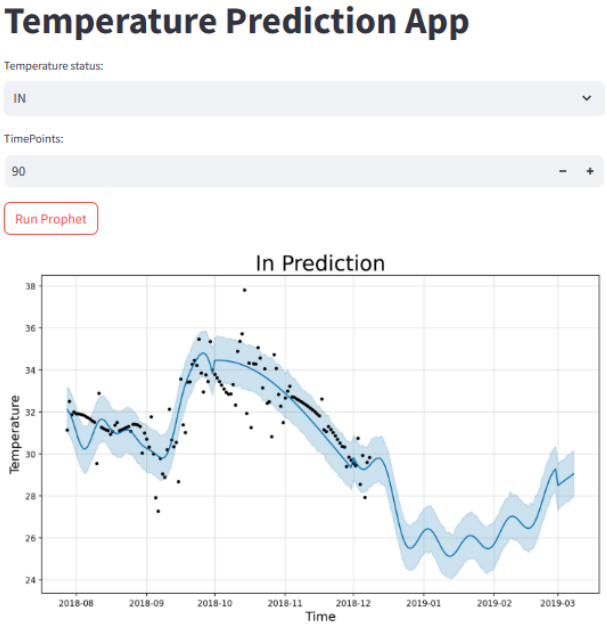

Fig. 9.8: Prediction of IN temperature with 90 TimePoints

So in conclusion Outside temperature is composed of multiple distributions, while inside temperature has a single distribution.

Inside temperature has flat trend, but outside temperature has the trend that is seemed to be affected by time-series factor such as seasonality.

So many drops in the data made it difficult to build model, so interpolating daily-mean data by 'spline' method worked.

Some outliers made it difficult to build forecasting model, but thanks to Prophet it is thought we built a robust model against outliers.

CONCLUSION

The Time Series Forecasting is a major topic nowadays for the future predictions, it can help with many different applications. In this paper, we explored the effectiveness of three prominent time series forecasting models with different backgrounds: SARIMAX, Prophet, and LSTM. These models were applied to various univariate and multivariate datasets to assess their performance across different types of data.

The first dataset, "Dataset 1 - Discount Rate," consisted of the discount rate changes over time in the Czech Republic, sourced from the Czech National Bank (CNB). This relatively simple dataset required minimal pre-processing. Among the models, the LSTM model showed superior performance, accurately capturing the temporal patterns in the data. SARIMAX also provided reasonable results, while the Prophet model lagged slightly behind.

The second dataset, "Dataset 2 - Number of Accidents," comprised a more complex dataset, detailing the number of car accidents in the Czech Republic on a daily basis since 2006. In this case, the LSTM model again showed the best performance, effectively capturing the intricate patterns within the data. Prophet performed better than SARIMAX, but both were less effective compared to LSTM.

The third dataset, "Dataset 3 - Yahoo Finance," was created using the Yahoo Finance API to generate stock data for Apple Inc. (AAPL). For this dataset, the SARIMAX model performed well, especially given the stock data's inherent volatility. The LSTM model provided intermediate results, showcasing its adaptability to different data complexities. The Prophet model, however, showed less effectiveness in handling the volatility of financial data.

Additionally, we applied the Prophet model to another multivariate dataset, involving inside and outside temperature measurements, and performed hyperparameter tuning. The outside temperature showed seasonal trends, while the inside temperature had a flat trend. Challenges such as drops and outliers were addressed using interpolation and Prophet's robustness, resulting in an effective forecasting model.

Furthermore, it was observed that the LSTM model's performance improved with more extensive training, indicating its potential for further optimization.

In conclusion, the LSTM model consistently demonstrated superior performance across all datasets, highlighting its robustness and capability in capturing complex temporal patterns. While SARIMAX was effective for simpler datasets and certain types of data, it struggled with more complex datasets. The Prophet model, despite its ease of use and quick setup, generally exhibited lower performance, particularly in volatile and complex datasets. These findings emphasize the importance of selecting appropriate models based on the specific characteristics and complexities of the datasets in time series forecasting tasks. Moreover, the application of hyperparameter tuning and addressing data irregularities are crucial steps in improving model performance.

LITERATURE

[1] Hayes, A. (2022). Time Series. Investopedia. Available from URL: https://www.investopedia.com/terms/t/timeseries.asp

[2] Barkved, K. (2022). Introducing Obviously AI Time Series. Obviously AI. Available from URL: https://www.obviously.ai/post/introducing-obviously-ai-time-series

[3] Sharma, A. Different Approaches to Conventional Programming v/s Machine Learning. KDnuggets. Available from URL:

https://www.kdnuggets.com/2018/12/different-conventional-programming-machine-learning.html

[4] Brownlee, J. (2020, August 15). Time Series Forecasting. Machine Learning Mastery. Available from URL:

https://machinelearningmastery.com/time-series-forecasting/

[5] Kaur, M. (2020, February 22). Time Series Analysis. QuantInsti. Available from URL: https://blog.quantinsti.com/time-series-analysis/

[6] Machine Learning. GeeksforGeeks. Last updated January 25, 2024. Available from URL: https://www.geeksforgeeks.org/machine-learning/

[7] Manika. (2024, April 9). Time Series Forecasting Models. ProjectPro. Available from URL: https://www.projectpro.io/article/time-series-forecasting-models/559

[8] Palachy Affek, S. (2019, April 8). Stationarity in Time Series Analysis. Towards Data Science. Available from URL:

https://towardsdatascience.com/stationarity-in-time-series-analysis-90c94f27322

[9] Iordanova, T. (2022, January 5). Understanding Stationarity in Time Series Analysis. Investopedia. Retrieved from

https://www.investopedia.com/articles/trading/07/stationary.asp

[10] Pathak, P. P. (2021, September 8). Time Series Forecasting: A Complete Guide. Medium. Available from URL:

https://medium.com/analytics-vidhya/time-series-forecasting-a-complete-guide-d963142da33f

[11] Bobbitt, Z. (2022, April 25). Univariate vs. Multivariate Analysis. Statology. Available from URL:

https://www.statology.org/univariate-vs-multivariate-analysis/

[12] Baruah, I. D. (2020, September 28). Combining Time Series Analysis with Artificial Intelligence: The Future of Forecasting. Medium. Available from URL: https://medium.com/analytics-vidhya/combining-time-series-analysis-with-artificial-intelligence-the-future-of-forecasting-5196f57db913

[13] Autoregressive Moving Average (ARMA(p,q)) Models for Time Series Analysis: Part 2. QuantStart. Available from URL:

https://www.quantstart.com/articles/Autoregressive-Moving-Average-ARMA-p-q-Models-for-Time-Series-Analysis-Part-2/

[14] Shweta. (2021, July 30). Introduction to Time Series Forecasting: Part 2 - ARIMA Models. Towards Data Science. Available from URL:

https://towardsdatascience.com/introduction-to-time-series-forecasting-part-2-arima-models-9f47bf0f476b

[15] Statespace VARMAX. StatsModels. Available from URL: https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_varmax.html

[16] Brownlee, J. (2023, October 3). Supervised and Unsupervised Machine Learning Algorithms. Machine Learning Mastery. Available from URL: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[17] Linear Regression. Wikipedia. Available from URL: https://en.wikipedia.org/wiki/Linear_regression

[18] Johnson, P. (2024, Jan 23). Linear Relationship. WallStreetMojo. Available from URL: https://www.wallstreetmojo.com/linear-relationship/

[19] Bevans, R. (2020, February 20). Multiple Linear Regression. Scribbr. Available from URL: https://www.scribbr.com/statistics/multiple-linear-regression/

[20] Random Forest. IBM. Available from URL:

https://www.ibm.com/topics/random-forest

[21] Joseph. (2022, August 15). Machine Learning: Information Gain. Reason Town. Available from URL:

https://reason.town/machine-learning-information-gain/

[22] Jun M. (2020, November 13). Construct a Decision Tree and How to Deal With Overfitting. Towards Data Science. Available from URL:

https://towardsdatascience.com/construct-a-decision-tree-and-how-to-deal-with-overfitting-f907efc1492d

[23] XGBoost. GeeksforGeeks. Available from URL:

https://www.geeksforgeeks.org/xgboost/

[24] Stecanella, B. (2017, June 22). Introduction to Support Vector Machines (SVM). MonkeyLearn. Available from URL:

https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

[25] Support Vector Machine Algorithm. GeeksforGeeks. Available from URL: https://www.geeksforgeeks.org/support-vector-machine-algorithm/

[26] Naive Bayes. IBM. Available from URL:

https://www.ibm.com/topics/naive-bayes

[27] Binhuraib, T. (2020, September 30). The Mathematics Behind Naive Bayes Classifiers. Medium. Available from URL:

https://taha-huraibb99.medium.com/the-mathematics-behind-naive-bayes-classifiers-333b6486f39c

[28] Arik, S. O., & Pfister, T. (2021, December 13). Interpretable Deep Learning for Time Series Forecasting. Google Research Blog. Available from URL: https://research.google/blog/interpretable-deep-learning-for-time-series-forecasting/

[29] Aishwarya. (2023, December 4). Introduction to Recurrent Neural Network. GeeksforGeeks. Available from URL:

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

[30] In-Depth Explanation of Recurrent Neural Network. Analytics Vidhya. Available from URL: https://www.analyticsvidhya.com/blog/2021/07/in-depth-explanation-of-recurrent-neural-network/

[31] Understanding of LSTM Networks. GeeksforGeeks. Available from URL: https://www.geeksforgeeks.org/understanding-of-lstm-networks/

[32] Prophet. Facebook. Available from URL:

https://facebook.github.io/prophet/

[33] DeepAR Forecasting Algorithm. Amazon SageMaker Developer Guide. Available from URL: https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html

[34] Time Series and Forecasting Using R. GeeksforGeeks. Available from URL: https://www.geeksforgeeks.org/time-series-and-forecasting-using-r/

[35] Statsmodels. GitHub. Available from URL:

https://github.com/statsmodels/statsmodels

[36] Lenczuk, J. (2021, August 17). Why Start Using sktime for Forecasting. Towards Data Science. Available from URL:

https://towardsdatascience.com/why-start-using-sktime-for-forecasting-8d6881c0a518

[37] Heller, M. (2019, January 28). What is Keras? The Deep Neural Network API Explained. InfoWorld. Available from URL:

https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html

[38] Geisler Mesevage, T. (2021, May 24). Data Preprocessing: What It Is and How to Do It. MonkeyLearn. Available from URL:

https://monkeylearn.com/blog/data-preprocessing/

[39] Bains, R. (2022, February 16). Prepare Time Series Data with Amazon SageMaker Data Wrangler. Vedere AI. Available from URL:

https://www.vedereai.com/prepare-time-series-data-with-amazon-sagemaker-data-wrangler/

[40] isitapol. (2022, May 25). How to Split a Dataset into Train and Test Sets Using Python. GeeksforGeeks. Available from URL:

https://www.geeksforgeeks.org/how-to-split-a-dataset-into-train-and-test-sets-using-python/

[41] The Data Detective (Jan 31, 2020) The 80/20 Split Intuition and an Alternative Split Method. Available from URL:

https://towardsdatascience.com/finally-why-we-use-an-80-20-split-for-training-and-test-data-plus-an-alternative-method-oh-yes-edc77e96295d

[42] Kumar, A. (2022, April 2). Steps for Evaluating & Validating Time Series Models. Vitalflux. Available from URL:

https://vitalflux.com/steps-for-evaluating-validating-time-series-models/?utm_content=cmp-true

[43] Evaluation of Regression Models in Scikit-Learn. Data Courses. Available from URL: https://www.datacourses.com/evaluation-of-regression-models-in-scikit-learn-846/

[44] Priya, J. (2022, January 24). Complete List of Performance Metrics for Monitoring Regression Models. Qualdo AI. Available from URL:

https://www.qualdo.ai/blog/complete-list-of-performance-metrics-for-monitoring-regression-models/