Comparative Analysis of Machine Learning Algorithms in Pair Currency Price Prediction in Forex Market, particularly in EURUSD and XAUUSD(GOLD)

Bachelor's thesis

Degree Programme in Computer Applications

Spring 2024

Younes Kalami

HAMK
Häme University
of Applied Sciences

Degree Programme in Computer Applications
Author Younes Kalami
Abstract
Year 2024

Subject Comparative Analysis of Machine Learning Algorithms in Pair Currency Price Prediction in Forex Market, particularly in EURUSD and XAUUSD(GOLD)

Supervisors Mazhar Mohsin

This thesis evaluates functions of machine learning algorithms to predict the price of EURUSD and XAUUSD, to help traders and financial institutions to make suitable decisions in using these algorithms. By comparing different algorithms like LSTM, ARIMA, SVM, and CNN, the main questions about accuracy of predictions and effects of some parameters are answered. Then basic concepts of time series analysis and financial data analysis are explained to show some information to understand price movements in markets. Evaluation metrics such as Mean Squared Error and Mean Absolute Error are used to evaluate model's function. The steps of data preparation, model building, and evaluation are mentioned, then practical steps of algorithms are provided. By doing tests with EURUSD and XAUUSD history data, the performance of models is evaluated. SVM was the most effective model in predicting both EURUSD and XAUUSD, after that LSTM, CNN, and ARIMA are on next ranks. In conclusion, this thesis provides an interesting look at the use of machine learning in the field of currency pair price prediction with the aim of helping to improve the skills of traders.

Keywords Currency prediction, Machine Learning algorithms, Forex market, Comparative analysis, financial data analysis

Pages 35

Glossary

EURUSD	*The currency pair of the Euro and the United States Dollar*
XAUUSD	*The currency pair of the price of Gold in United States Dollars*
LSTM	*Long Short-Term Memory*
ARIMA	*Autoregressive Integrated Moving Average*
CNN	*Convolutional Neural Network*
SVM	*Support Vector Machines*
MSE	*Mean Squared Error*
MAE	*Mean Absolute Error*
GDP	*Gross Domestic Product*
RNN	*Recurrent Neural Networks*
DNN	*Deep Neural Networks*
ReLU	Rectified Linear Unit

CONTENT

Tables, Figures and Program codes

Appendices

1 INTRODUCTION

In financial markets, price prediction is very difficult for traders and financial institutions, but it seems necessary. This thesis deals with how different machine learning algorithms can be used to predict currency prices, especially for EURUSD and GOLD. This thesis is for traders who want to make better and more accurate decisions, as well as for financial institutions and researchers who want to learn more about predicting price movement in the forex market using artificial intelligence.

Different machine learning algorithms such as Recurrent Neural Networks (RNN), Autoregressive Integrated Moving Average (ARIMA), Support Vector Machines (SVM) and Deep Neural Networks (DNN) are used in this thesis. Also, the selection of data and the appropriate settings of various parameters to increase the accuracy of the forecast are explained. But other factors such as the impact of economic factors and news on the currency market have not been considered. In this thesis, the prediction performance of these models was evaluated in price prediction and a simple guide was created for traders and researchers to choose the best model. In this thesis, the main following question will be answered:

- How do different machine learning algorithms compare in their ability to predict currency price movements in the forex market?
- What factors influence the accuracy of machine learning models when forecasting EURUSD and GOLD prices?
- How do variations in data quality and model parameters affect the performance of machine learning algorithms in predicting currency prices?

The result should be an easy and practical guide for traders and researchers to choose a better method for currency forecasting. By using the latest machine learning algorithms and understanding financial data, better methods for predicting currency prices are being tried. This information can help traders, researchers and financial institutions to make more appropriate decisions in the currency markets and trading.

2 CURRENCY FORECASTING BY MACHINE LEARNING ALGORITHMS

Currency forecasting using machine learning algorithms is gaining traction due to its ability to provide valuable insight into currency price movements and trends in financial markets. This section examines the application of machine learning algorithms in currency forecasting and discusses related studies and research findings.

2.1 Introduction to Machine Learning Algorithms

Machine learning algorithms are for data analysis and predictive modeling and allow computers to learn from data without special programming (Bishop, 2006, n.d.). These algorithms are divided into three methods of supervised, unsupervised and reinforcement learning. Supervised learning involves learning from labeled data, such as linear regression and decision trees (Hastie et al., 2009). Unsupervised learning, such as k-means clustering and PCA, identifies patterns in unlabeled data. Reinforcement learning, exemplified by algorithms such as cue learning, learns through trial and error with an environment (Bishop, 2006, n.d.). In various fields, machine learning algorithms have created revolutions, including in finance, healthcare, marketing and robotics. In finance, algorithms predict stock prices (Hastie et al., 2009, n.d.). In healthcare, they help diagnose disease. In marketing, they help through recommender systems and customer segmentation (Bishop, 2006, n.d.). In robotics, machine learning enables autonomous navigation and object recognition (Hastie et al., 2009). The adaptability and scalability of these algorithms make them important in extracting various analyzes from large data sets and addressing new and complex challenges.

2.2 Fundamentals of Time Series Analysis

Time series analysis is important for currency forecasting, because exchange rates show movements and trends over time. Concepts such as trend, seasonality, and autocorrelation are essential for analyzing currency data over time. One of the basic concepts in time series analysis is trend, which refers to the long-term movement or direction of a time series. Trends can be bullish (price growth), bearish (price decline) or stable (showing no significant change over time). Identifying and modeling exchange rate trends is very important for predicting future price movements.

Another important concept is seasonality, which refers to regular and predictable fluctuations in a time series that occur at regular intervals such as daily, weekly, or yearly. Seasonality can be caused by a variety of factors, including economic cycles, holidays, and seasonal patterns in consumer behavior. Seasonal calculation is also necessary in currency data to accurately predict currency prices. Time series models, including autoregressive integrated moving average (ARIMA) and exponential smoothing methods, are commonly used to capture these patterns in currency data (Hyndman & Athanasopoulos, 2018, n.d.). These models consider trends, seasonality, and autocorrelation to predict future currency prices.

2.3 Financial Data Analysis for Currency Prediction

Financial data analysis plays an important role in currency forecasting, since economic indicators can affect currency movements, experts analyze factors such as interest rates, inflation rates, GDP growth, and geopolitical events to identify factors. Analyze currency fluctuations. Understanding the relationships between these variables is necessary to build accurate forecasting models, but in this thesis, their use has been avoided.

In addition, financial data analysis includes examining market sentiment and investor behavior. Sentiment analysis techniques, such as analyzing news sentiment or social media sentiment related to currencies, can provide insights into market thoughts and expectations. Market sentiment analysis can help predict short-term fluctuations in exchange rates (Baker & Wurgler, 2006, n.d.).

In addition, the analysis of financial data includes the examination of various indicators and technical patterns in the currency markets. Technical analysis involves the study of historical price data, volume and other market statistics to identify trends and patterns that may repeat themselves in the future.

2.4 Evaluation Metrics for Model Performance

In currency forecasting, the goal is to develop models that predict market prices in future. Evaluation metrics show how well a model performs in comparison to actual currency movements.

This thesis uses the Mean Squared Error (MSE) and Mean Absolute Error (MAE) as the performance evaluation techniques.

Mean Squared Error (MSE) is a common metric used to evaluate the performance of a predictive model, especially in regression analysis. It measures the average of the squares of the differences between the predicted values and the actual values.

Mean Absolute Error (MAE) measures the average absolute difference between predicted and actual values. In currency forecasting, MAE shows the average magnitude of prediction errors.

The most popular metrics for evaluating LSTM, ARIMA, CNN, and SVM models' performance are MSE, RMSE, and MAE. However, as RMSE is just the root square of the MSE.

The errors do not indicate that there are some problems with the model, but it refers to the unpredictability of the observation (Hyndman et al., 2018, pp. 62-71). Both MSE and MAE are common measures in the previous neural network research (Leung et al., 2000; Khashei et al., 2010). For both error criteria, the smaller the error value, the better the forecasting accuracy.

During the preprocessing phase of neural networks, datasets are divided into training and testing sets. Then Error values are calculated for both training and testing datasets. **Training Error** shows how well the model fits the training data, but it doesn't necessarily reflect how well the model will perform on new data. A low training error does not guarantee good generalization to unseen data. **Testing Error** indicates how well the model generalizes to new, unseen data. A low testing error suggests that the model can detect patterns in the data and can perform well in real-world applications. Minimizing testing errors is the main goal of machine learning.

3 PREPARING DATA, BUILDING MODELS, STEPS IN ALGORITHM IMPLEMENTATION

In currency forecasting, the process of preparing data and building models through algorithm implementation is important to reaching accurate predictions. This process involves several key steps. collecting reliable historical currency data is the first step in algorithm implementation. Data can be downloaded from reputable sources and saved in a suitable format like CSV for further analysis.

3.1 Data Collection and Preprocessing Techniques

For currency forecasting in the Forex market, historical data for the EURUSD and XAUUSD currency pairs was collected from, ForexSB. The data was downloaded in a daily time frame containing historical exchange rate data for EURUSD and XAUUSD, including date/time stamps and corresponding daily closing prices as CSV files, suitable for analyzing long-term trends and patterns in currency rates. Table 1 consists of first and last 5 rows of EURUSD dataset and Table 2 consists of first and last 5 rows of XAUUSD dataset.

Table 1.History data of EURUSD

	Date	Open	High	Low	Close
0	2008-04-11	1.57629	1.58564	1.57522	1.58091
1	2008-04-13	1.57834	1.57834	1.56913	1.57101
2	2008-04-14	1.57097	1.58867	1.56684	1.58242
3	2008-04-15	1.58224	1.58754	1.57486	1.57846
4	2008-04-16	1.57850	1.59786	1.57746	1.59365
4913	2023-12-25	1.10149	1.10230	1.09933	1.10186
4914	2023-12-26	1.10187	1.10450	1.10085	1.10422
4915	2023-12-27	1.10422	1.11226	1.10286	1.11082
4916	2023-12-28	1.11081	1.11394	1.10552	1.10677
4917	2023-12-29	1.10678	1.10842	1.10339	1.10374

Table 2.History data of XAUUSD

	Date	Open	High	Low	Close
0	2008-04-11	928.518	931.424	918.304	924.630
1	2008-04-13	924.697	924.835	914.860	914.992
2	2008-04-14	915.045	930.896	913.858	925.146
3	2008-04-15	925.060	935.759	924.441	928.223
4	2008-04-16	928.141	948.368	924.120	943.171
4913	2023-12-25	2053.025	2057.655	2052.895	2054.775
4914	2023-12-26	2054.765	2068.948	2053.385	2066.048
4915	2023-12-27	2066.065	2084.345	2061.385	2079.435
4916	2023-12-28	2079.505	2088.355	2064.448	2066.655
4917	2023-12-29	2066.645	2074.645	2058.105	2062.688

This data is for model training to analyze past trends and behaviors to make predictions about currency movements in the Forex market.

The collected data may contain errors, outliers, or missing values that can have negative effects on model performance. Data cleaning techniques are applied to identifying and removing anomalies to have good data quality.

The datasets have daily closing prices from April 2008 to December 2023. The Figure 1 and Figure 2 show graphs for each dataset to see how the closing prices change over time.

Figure 1.Close Price History of EURUSD

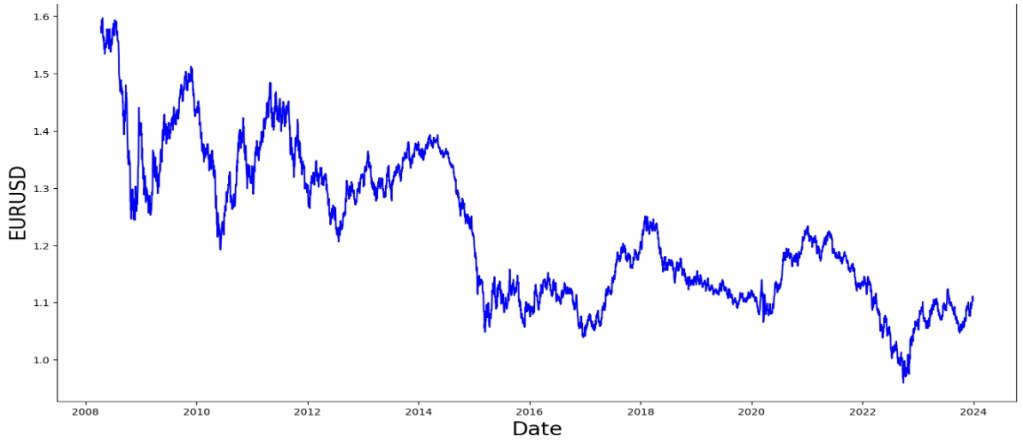
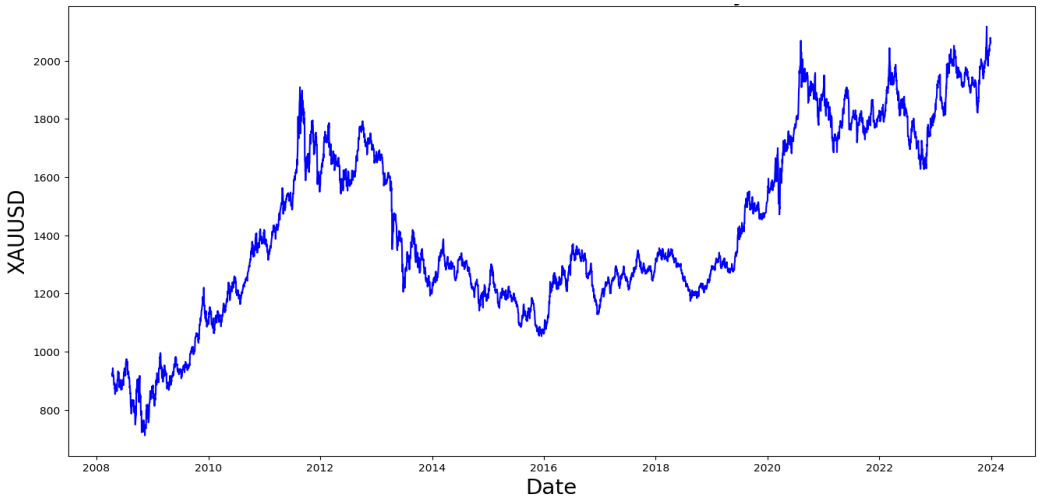


Figure 2.Close Price History of XAUUSD



Before model training, the preprocessed data is often split into training, validation, and test sets. The training set is used to train the models, the validation set is used to tune hyperparameters and evaluate model performance during training, and the test set is used to assess the final performance of the trained models.

3.2 Comparative Analysis Framework for Machine Learning Algorithms

Methods like LSTM, which is a type of recurrent neural network architecture, Autoregressive Integrated Moving Average (ARIMA, Support Vector Machines (SVM), and CNN stands for Convolutional Neural Network, which is a type of artificial neural network architecture, commonly used in deep learning were chosen because they are commonly used for this type of prediction.

Measures like Mean Absolute Error (MAE) and Mean Squared Error (MSE) can help us see which method gives the most accurate predictions. The same data is used to train and test each method. Important features from the data are selected, and rules are set for each method to follow. Once each method has been trained and tested with real currency price data, their performance is assessed. The accuracy and reliability in predicting future prices are evaluated using measuring tools. The results are analyzed to determine if any method stands out. Math may be utilized to compare them and ascertain if the differences hold significance. This helps in selecting the best method for predicting currency prices in the forex market.

3.3 Algorithm Implementation and Model Training

This part is important because the methods need to be taught how to predict currency prices based on the dataset. First, the machine learning models are set up. The algorithms to be used, including LSTM, Support Vector Machines (SVM), ARIMA, and CNN, are selected. Subsequently, the models are trained in how to predict currency prices using historical data. Numerous examples of past currency prices and their movements are presented to the models. Each model has parameters that can be adjusted to enhance its performance, and these parameters are fine-tuned accordingly. Following training, an evaluation is conducted to assess the models' performance. A part of the data set aside earlier (not utilized for training) is employed to test the models, enabling an examination of their accuracy in predicting currency prices.

3.3.1 LSTM MODEL

The process begins by importing the necessary libraries, including NumPy, Pandas, and Keras (Program Code 1). NumPy and Pandas are used for numerical calculations and data manipulation, respectively, while Keras is used for LSTM model building.

Program Code 1.Important Libraries for LSTM

```
# Importing the necessary libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
```

After importing the library, the dataset containing daily closing values of currency prices is loaded from a CSV file which was saved as EURUSD.CSV into a Pandas Data Frame. next, the data are pre-processed using Min-Max scaling to normalize the "close" column values between 0 and 1(Program Code 2). This step ensures that the data is within a fixed range, which is useful for training the model.

Program Code 2.Loading the Dataset and Pre-processing data

```
# Load the dataset
data = pd.read_csv("EURUSD.csv")

# Preprocessing data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[['Close']].values.reshape(-1, 1))
```

Then the data set is divided into training and test sets with a split ratio of 80:20. The training set is used to train the LSTM model, while the test set is reserved for evaluating the performance of the model. The data are then formatted into sequences of input-output pairs suitable for training the LSTM model (Program Code 3).

Program Code 3.Splitting dataset and Reshaping

```
# Splitting data into train and test sets
train_size = int(len(scaled_data) * 0.8)
train_data = scaled_data[:train_size, :]
test_data = scaled_data[train_size:, :]

# Creating dataset
def create_dataset(dataset, time_step=1):
    X, Y = [], []
    for i in range(len(dataset) - time_step - 1):
        X.append(dataset[i:(i + time_step), 0])
        Y.append(dataset[i + time_step, 0])
    return np.array(X), np.array(Y)

time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

# Reshaping input to be 3D [samples, timesteps, features] as required by LSTM
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

The LSTM model architecture is defined using the keras Sequential API, consisting of several LSTM layers followed by a dense output layer. The model is entered with the Adam optimizer and the mean squared error loss function. The model is then trained on the training data for a certain number of epochs using a batch size of 16 (Program Code 4). After training, the model is used to predict both training and test data sets.

Program Code 4.Building LSTM Model

```
# LSTM Model
model = Sequential()
model.add(LSTM(units=32, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=32))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the model
model.fit(X_train, y_train, epochs=100, batch_size=16)
```

3.3.2 ARIMA MODEL

The code begins by importing necessary libraries such as NumPy for numerical computations, Pandas for data manipulation, Stats models for time series analysis, and Matplotlib for visualization. Then, it loads the dataset from a CSV file named "EURUSD.csv" using Pandas' read_csv function (Program Code 5).

Program Code 5.Important Libraries and loading dataset for ARIMA model

```
import numpy as np
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
from sklearn.model_selection import train_test_split
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv("EURUSD.csv")
```

Then the dataset is split into training and testing sets using train_test_split function from scikit-learn. The splitting is done in such a way that 80% of the data is used for training (train data), and 20% is reserved for testing (test data). The parameter shuffle=False ensures that the split is done sequentially without shuffling the data (Program Code 6).

Inside a try-except block, an ARIMA model is fitted to the training data. ARIMA stands for Autoregressive Integrated Moving Average and is a popular model for time series forecasting. In this code, an ARIMA model with order (1, 1, 0) is used. This means the model has one autoregressive term, one differencing term, and no

moving average term (Program Code 6). After fitting the model, diagnostic checks are performed on the residuals (the differences between the observed and predicted values). Two plots are generated using plot_acf and plot_pacf functions to visualize the autocorrelation and partial autocorrelation of the residuals, respectively (Program Code 6).

Program Code 6.Splitting Data, Fitting ARIMA Model and diagnostic checks

```
# Split the data into training and testing sets
train_data, test_data = train_test_split(data['Close'], test_size=0.2, shuffle=False)

try:
    # Fitting the ARIMA model
    model = ARIMA(train_data, order=(1, 1, 0))
    model_fit = model.fit()

    # Diagnostic checks
    residuals = model_fit.resid
    plot_acf(residuals, lags=20)
    plt.show()

    plot_pacf(residuals, lags=20)
    plt.show()
```

3.3.3 CNN MODEL

The code begins by importing libraries for data manipulation, preprocessing, modeling, and evaluation. Numpy is a fundamental package for scientific computing with Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. pandas is a library for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series data. MinMaxScaler from sklearn.preprocessing is imported to scale the data between a specified range. In this case, it's used to normalize the 'Close' prices between 0 and 1. From Keras, Sequential, Conv1D, MaxPooling1D, Flatten, and Dense are imported (Program Code 7). These are components for building convolutional neural network (CNN) models.

Program Code 7.Important Libraries for CNN Model

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
```

The code continues by loading the dataset EURUSD.CSV, containing historical EURUSD exchange rate data. This dataset is then preprocessed using MinMaxScaler from scikit-learn to scale the 'Close' prices between 0 and 1, which is a common practice in neural network-based models to stabilize and accelerate the training process. The preprocessed data is then split into train and test sets, with approximately 80% of the data used for training and the remaining 20% for testing (Program Code 8). A custom function named create_dataset is defined to convert the time series data into input-output pairs suitable for training a supervised learning model. This function segments the time series data into input sequences (X) and their corresponding output values (Y), with a specified time step size (Program Code 9).

Program Code 8. Loading dataset, Preprocessing and Splitting data

```
# Load the dataset
data = pd.read_csv("EURUSD.csv")

# Preprocessing data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[['Close']].values.reshape(-1, 1))

# Splitting data into train and test sets
train_size = int(len(scaled_data) * 0.8)
train_data = scaled_data[:train_size, :]
test_data = scaled_data[train_size:, :]
```

Program Code 9. Dataset Creation and Reshaping

```
# Creating dataset
def create_dataset(dataset, time_step=1):
    X, Y = [], []
    for i in range(len(dataset) - time_step - 1):
        X.append(dataset[i:(i + time_step), 0])
        Y.append(dataset[i + time_step, 0])
    return np.array(X), np.array(Y)

time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

# Reshaping input to be 3D [samples, timesteps, features] as required by CNN
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

The CNN model architecture is defined using Keras Sequential API. The model consists of a 1D convolutional layer (Conv1D), followed by a max-pooling layer (MaxPooling1D), and a flattening layer (Flatten). Finally, a fully connected dense layer (Dense) with a single neuron is added to produce the output. Rectified Linear Unit (ReLU) activation function is used in the convolutional layer. The model is compiled with the Adam optimizer and mean squared error (MSE) loss function (Program Code 10). The model is trained using the training data (X_train and y_train) with 100 epochs and a batch size of 32. During training, the model learns to map input sequences to their corresponding output values, adjusting its internal parameters (weights) to minimize the MSE loss (Program Code 10).

Program Code 10.Model Definition and Training

```
# CNN Model
model = Sequential()
model.add(Conv1D(filters=16, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the model
model.fit(X_train, y_train, epochs=100, batch_size=32)
```

3.3.4 SVM MODEL

This model performs a time series analysis using Support Vector Regression (SVR). The code begins by importing the necessary libraries: numpy as np, pandas as pd, and specific classes from the sklearn library, including Min Max Scaler for data scaling and SVR for the SVR model. It also imports the evaluation metrics mean absolute error and mean squared error (Program Code 11).

Program Code 11.Important Libraries for SVM

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

The dataset is loaded from a CSV file named "EURUSD.csv" using the read csv function from pandas and data is stored in the data variable then data is preprocessed using a Min Max Scaler to scale the "Close" column of the data between 0 and 1 and the scaled data is stored in the scaled data variable.
The data is split into training and test sets. The code calculates the index train size corresponding to 80% of the data, and then slices the scaled data array to obtain the training data and the test data (Program Code 12).

Program Code 12.Loading dataset, Preprocessing and Splitting data

```
# Load the dataset
data = pd.read_csv("EURUSD.csv")

# Preprocessing data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[['Close']].values.reshape(-1, 1))

# Splitting data into train and test sets
train_size = int(len(scaled_data) * 0.8)
train_data = scaled_data[:train_size, :]
test_data = scaled_data[train_size:, :]
```

Then the function create_dataset that takes a dataset and a time_step parameter which creates input-output pairs for the time series data are created (Program Code 13). It iterates over the dataset and creates input sequences of length time_step and corresponding output values. The input sequences (X) and output values (Y)

are stored in arrays and returned as numpy arrays. The create_dataset function is then used to create the input-output pairs for the training and test data using the train_data and test_data arrays, respectively. The time_step parameter determines the length of the input sequences.

Program Code 13.Creating dataset function

```
# Creating dataset
def create_dataset(dataset, time_step=1):
    X, Y = [], []
    for i in range(len(dataset) - time_step - 1):
        X.append(dataset[i:(i + time_step), 0])
        Y.append(dataset[i + time_step, 0])
    return np.array(X), np.array(Y)

time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
```

Then an SVR model is created with a linear kernel by initializing an instance of the SVR class and storing it in the model variable. The SVR model is trained using the training data (X_train, y_train) by calling the fit method of the model object. The trained SVR model is used to make predictions on the training and test data by calling the predict method with the input data (X_train, X_test). The predicted values are stored in the train_predict and test_predict variables, respectively (Program Code 14).

Program Code 14.Creating SVM Model, Train and Predict

```
# SVM Model
model = SVR(kernel='linear')

# Training the model
model.fit(X_train, y_train)

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

3.4 Evaluation Procedures for Prediction Models

The prediction models are evaluated to see how well they perform. This is done by comparing the predictions made by the models to the actual currency prices. The accuracy of the prediction models is assessed to determine how close their predictions are to the actual currency prices. Evaluation metrics such as Mean Absolute Error (MAE) and Mean Squared Error (MSE) are computed to assess the model's performance on both the training and test datasets. These metrics provide insights into the accuracy and performance of the LSTM model, ARIMA, CNN and SVM in predicting currency prices based on historical data.

3.4.1 Evaluation Metrics of LSTM

This code segment is responsible for evaluating the LSTM (Long Short-Term Memory) model's performance using two main metrics (Program Code 15): Mean Absolute Error (MAE) and Mean Squared Error (MSE). Firstly, it calculates the MAE for both the training and testing datasets. MAE is computed by taking the absolute differences between the actual target values (y_train and y_test) and the predicted values (train_predict and test_predict), then taking the mean of these absolute differences. Secondly, it computes the MSE for both training and testing datasets. MSE is calculated by taking the squared differences between the actual and predicted values, then averaging these squared differences. After computing these metrics, it creates a pandas Data Frame named results_lstm to store the results. This Data Frame has columns for the model’s name, MAE for training and testing, and MSE for training and testing. Finally, it prints out the results_lstm Data Frame, displaying the evaluation metrics for the LSTM model on both the training and testing datasets.

Program Code 15.LSTM Evaluation Metrics

```
# Evaluation Metrics
mae_train = np.mean(np.abs(y_train - train_predict))
mae_test = np.mean(np.abs(y_test - test_predict))
mse_train = np.mean((y_train - train_predict) ** 2)
mse_test = np.mean((y_test - test_predict) ** 2)

# Displaying results in a table
results_lstm = pd.DataFrame({
    'Model': ['LSTM'],
    'MAE Train': [mae_train],
    'MAE Test': [mae_test],
    'MSE Train': [mse_train],
    'MSE Test': [mse_test]
})

print(results_lstm)
```

3.4.2 Evaluation Metrics of ARIMA

This code segment assesses the performance of ARIMA (AutoRegressive Integrated Moving Average) model using two primary metrics (Program Code 16): Mean Absolute Error (MAE) and Mean Squared Error (MSE). First, it calculates the MAE and MSE for the testing dataset. MAE is computed by taking the absolute differences between the actual test data (test_data) and the forecasted values (forecast), then computing their mean. Similarly, MSE is calculated by taking the squared differences between the actual and forecasted values and then averaging them. Second, it computes the MAE and MSE for the training dataset. Here, the model's fitted values (model_fit.fittedvalues) are used instead of the forecasted values. This reflects the model's performance on the training data. After computing these metrics, it creates a pandas Data Frame named results_arima to store the results. The Data Frame includes columns for the model name ('ARIMA') and the MAE and MSE for both the testing and training datasets. The code includes exception handling to catch any errors that might occur during execution, printing out an error message if an exception is encountered. Finally, it prints the results_arima Data Frame, displaying the evaluation metrics for the ARIMA model on both the training and testing datasets.

Program Code 16.ARIMA Evaluation Metrics

```
    # Evaluation Metrics
    mae_test = np.mean(np.abs(test_data - forecast))
    mse_test = np.mean((test_data - forecast)**2)
    mae_train = np.mean(np.abs(train_data[1:] - model_fit.fittedvalues))
    mse_train = np.mean((train_data[1:] - model_fit.fittedvalues)**2)

    # Displaying results in a table
    results_arima = pd.DataFrame({
        'Model': ['ARIMA'],
        'MAE (Test)': [mae_test],
        'MSE (Test)': [mse_test],
        'MAE (Train)': [mae_train],
        'MSE (Train)': [mse_train]
    })

    print(results_arima)

except Exception as e:
    print("Error:", e)
```

3.4.3 Evaluation Metrics of CNN

This code segment is dedicated to evaluating the performance of Convolutional Neural Network (CNN) model using two key metrics (Program Code 17): Mean Absolute Error (MAE) and Mean Squared Error (MSE). Initially, it calculates the MAE for both the training and testing datasets. MAE measures the average magnitude of errors between predicted and actual values. This is done by computing the absolute differences between the actual target values (y_train and y_test) and the predicted values (train_predict and test_predict), then taking the mean of these absolute differences. Next, it computes the MSE for both training and testing datasets. MSE is a measure of the average of the squares of errors between predicted and actual values. It is calculated by finding the squared differences between the actual and predicted values, then averaging these squared differences. After computing these metrics, it constructs a pandas Data Frame named results_cnn to organize and store the results. This Data Frame has columns for the model's name ('CNN') and the MAE and MSE for both the training and testing datasets. Finally, the code prints out the results_cnn Data Frame, presenting the evaluation metrics for the CNN model on both the training and testing datasets.

Program Code 17.CNN Evaluation Metrics

```
# Evaluation Metrics
mae_train = np.mean(np.abs(y_train - train_predict))
mae_test = np.mean(np.abs(y_test - test_predict))
mse_train = np.mean((y_train - train_predict)**2)  # Calculating MSE for train
mse_test = np.mean((y_test - test_predict)**2)  # Calculating MSE for test


# Displaying results in a table
results_cnn = pd.DataFrame({
    'Model': ['CNN'],
    'MAE Train': [mae_train],
    'MAE Test': [mae_test],
    'MSE Train': [mse_train],
    'MSE Test': [mse_test]

})

print(results_cnn)
```

3.4.4 Evaluation Metrics of SVM

This code segment is dedicated to evaluating the performance of Support Vector Machine (SVM) model using two fundamental metrics (Program Code 18): Mean Absolute Error (MAE) and Mean Squared Error (MSE). It calculates the MAE and MSE for both the training and testing datasets. The MAE and MSE functions from scikit-learn's metrics module are employed for this purpose. The mean absolute error function computes the average absolute difference between the actual target values (y_train and y_test) and the predicted values (train_predict and test_predict). Likewise, the mean squared error function calculates the average of the squared differences between the actual and predicted values for both training and testing datasets. After computing these metrics, it constructs a pandas Data Frame named results_svm to organize and store the results. This Data Frame comprises columns for the model name ('SVM') and the MAE and MSE for both the training and testing datasets. At last, the code prints out the results_svm Data Frame, showing a short summary of the SVM model's performance on both the training and testing datasets.

Program Code 18.SVM Evaluation Metrics

```
# Evaluation Metrics
mae_train = mean_absolute_error(y_train, train_predict)
mae_test = mean_absolute_error(y_test, test_predict)
mse_train = mean_squared_error(y_train, train_predict)
mse_test = mean_squared_error(y_test, test_predict)

# Displaying results in a table
results_svm = pd.DataFrame({
    'Model': ['SVM'],
    'MAE Train': [mae_train],
    'MAE Test': [mae_test],
    'MSE Train': [mse_train],
    'MSE Test': [mse_test]
})

print(results_svm)
```

4 PRACTICAL INSIGHTS OF EVALUATING ALGORITHM PERFORMANCE

This part shows how well the algorithms are working and how the performance of algorithms is evaluated practically. Comparative analyses to identify the strengths and weaknesses of each algorithm, highlighting their performance across different metrics and datasets were done. Through practical experimentation and validation, actionable insights into selecting the most suitable algorithm for currency price prediction tasks will be provided.

4.1 Prototyping Predictive Models with Machine Learning Algorithms

In this part, predictive models will be created using machine learning algorithms to explore their effectiveness. The capabilities of these algorithms in predicting currency prices in the Forex market will be explored, with a particular focus on EURUSD and XAUUSD (GOLD) pairs. The process will involve data preprocessing, feature engineering, model training, and evaluation using appropriate metrics. By prototyping different machine learning algorithms, the aim is to identify the most suitable approach for accurately predicting currency price movements.

4.1.1 LSTM Parameters

After building the LSTM model, loading and splitting dataset the model is run by setting the parameters (Table 3) to see the results. time_step, determines the number of previous data points the model considers, crucial for capturing temporal patterns. Units specifies the number of memory cells in each LSTM layer, influencing the model's capacity to learn complex relationships. return_sequences, indicates whether the LSTM layer returns the full sequence of outputs, necessary for stacking LSTM layers and preserving temporal information. Epochs defines the number of times the entire dataset is passed through the network, impacting the model's learning and convergence. Batch_size specifies the number of samples processed in one training iteration, affecting training speed and memory usage.

Table 3.LSTM Model Parameters

Parameter	*Value*
time_step	100
units	32
return_sequences	True
epochs	100
batch_size	16

4.1.2 ARIMA Parameters

In the ARIMA code, the parameter (1, 1, 0) tell the model how to work (Table 4). The first number (1) represents the number of autoregressive (AR) terms which means it looks at one previous observation to make predictions. The second parameter (1) is the degree of differencing that means it makes the data stationary by subtracting each observation from the one before it. The last number (0) is the number of moving average (MA) terms that means it doesn't consider past mistakes when making predictions. These numbers help the model understand the data and make Predictions about future values.

Table 4.Arima Parameter

Parameter	*Value*	*Explanation*
order	(1, 1, 0)	Represents the order of the ARIMA model: (p, d, q). 'p' is the order of the autoregressive (AR) term, 'd' is the degree of differencing, and 'q' is the order of the moving average (MA) term. In this case, it's ARIMA (1, 1, 0), indicating one autoregressive term and one differencing term, with no moving average term.

4.1.3 CNN Parameters

In Table 5 the parameters and key steps of the CNN (Convolutional Neural Network) code along with a brief explanation are shown.

Table 5.CNN Parameters

Parameter	*Value*	*Explanation*
filters	16	Number of filters (or kernels) in the convolutional layer, determining the number of features extracted from the input data.
kernel_size	3	Size of the convolutional kernel, defining the spatial extent of the convolution operation and the size of the local region considered by each filter.
activation	relu	Activation function applied to the output of the convolutional layer, introducing non-linearity to the model. Common choices include ReLU (Rectified Linear Unit) to introduce non-linearity.
pool_size	2	Size of the pooling window, specifying the factor by which to downscale the input along the temporal dimension, aiding in feature extraction and reducing computational complexity.
epochs	100	Number of times the entire dataset is passed forward and backward through the neural network during training, influencing the model's learning and convergence.
batch_size	32	Number of samples processed in one training iteration, affecting training speed and memory usage.

4.1.4 SVM Parameters

Here is the summary of parameters with their respective values and explanation used in the SVM model (Table 6).

Table 6.SVM Parameters

Parameter	Value(s)	Description
feature_range	(0, 1)	Tuple specifying the range to which the features should be scaled.
time_step	100	Integer specifying the number of time steps to use for creating input-output pairs.
kernel	'linear'	String specifying the kernel type for the SVR model.

4.2 Experimentation with EURUSD and GOLD Price Data

This part involves conducting experiments with EURUSD and GOLD price data to assess the performance of the predictive models. The results of running each MLA are shown in Table 7 which includes evaluation metrics for EURUSD and

Table 8 contains the evaluation results of XAUUSD. All Models have been made and run in Visual Studio Code software. The Figures (Figure 3,Figure 4,Figure 5,Figure 6,Figure 7,Figure 8,Figure 9,Figure 10,Figure 11 and Figure 12) are showing the actual Train/Test data and also Predicted Train/Test data for both EURUSD and XAUUSD.

Table 7. EUR/USD set - training and testing evaluation

MODELS	**MAE Training**	**MAE Testing**	**MSE Training**	**MSE Testing**
Number of observations	3934	984	3934	984
LSTM	0.21652	0.108657	0.071615	0.018332
ARIMA	0.00495	0.080777	0.000051	0.009555
CNN	0.215793	0.105261	0.071131	0.017138
SVM	0.020459	0.015457	0.000711	0.000409

Table 8. XAU/USD set – training and testing evaluation

MODELS	MAE Training	MAE Testing	MSE Training	MSE Testing
Number of observations	3914	979	3914	979
LSTM	0.194185	0.07969	0.061783	0.009708
ARIMA	8.589537	88.283766	171.864436	11044.59187
CNN	0.192454	0.079713	0.060708	0.009714
SVM	0.022049	0.049115	0.000815	0.002907

Figure 3. Actual and Predicted Train data for EURUSD with LSTM model

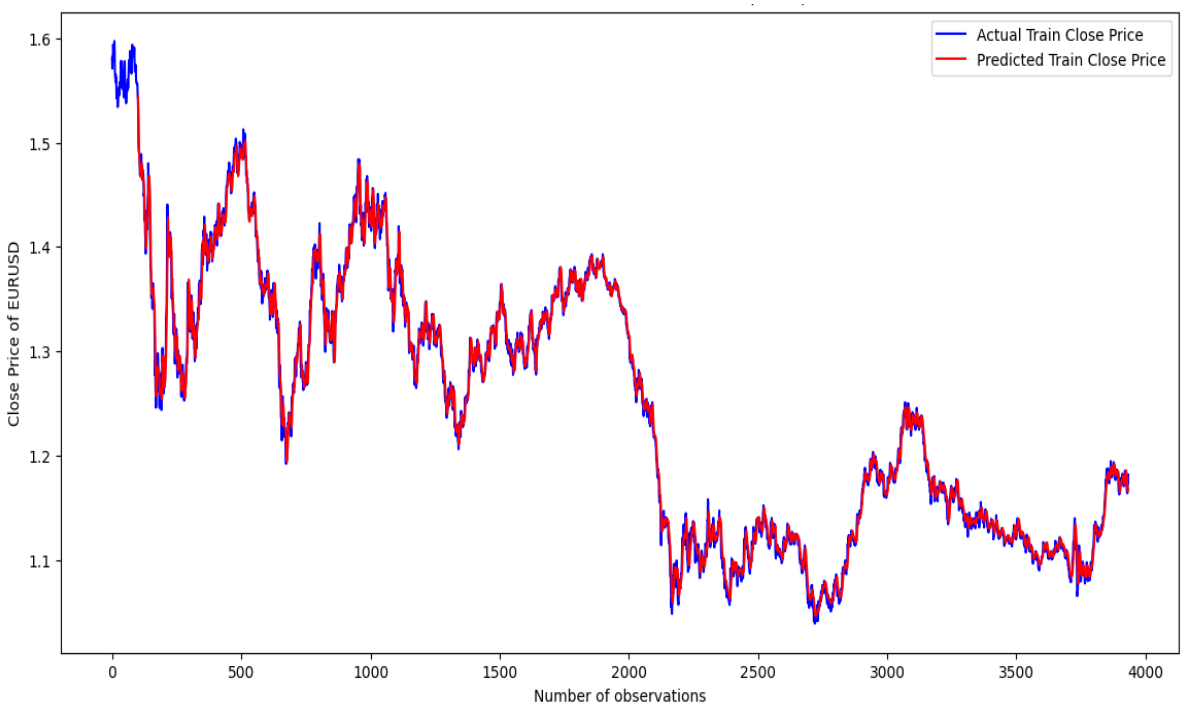


Figure 4. Actual and Predicted Test data for EURUSD with LSTM model

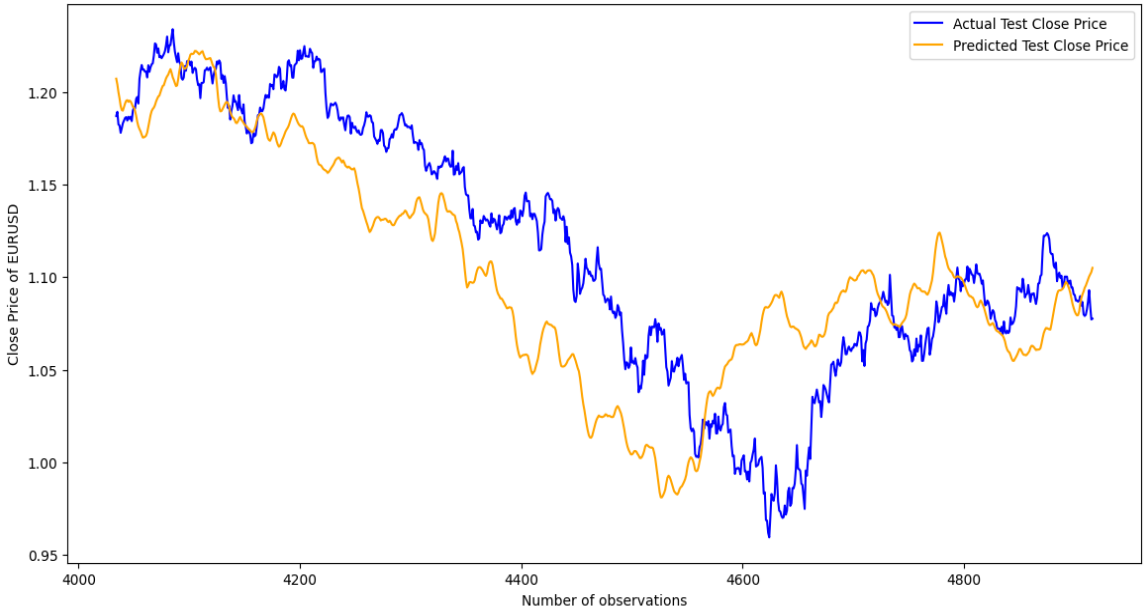


Figure 5. Actual and Predicted Train/Test close price for XAUUSD(GOLD) with LSTM model

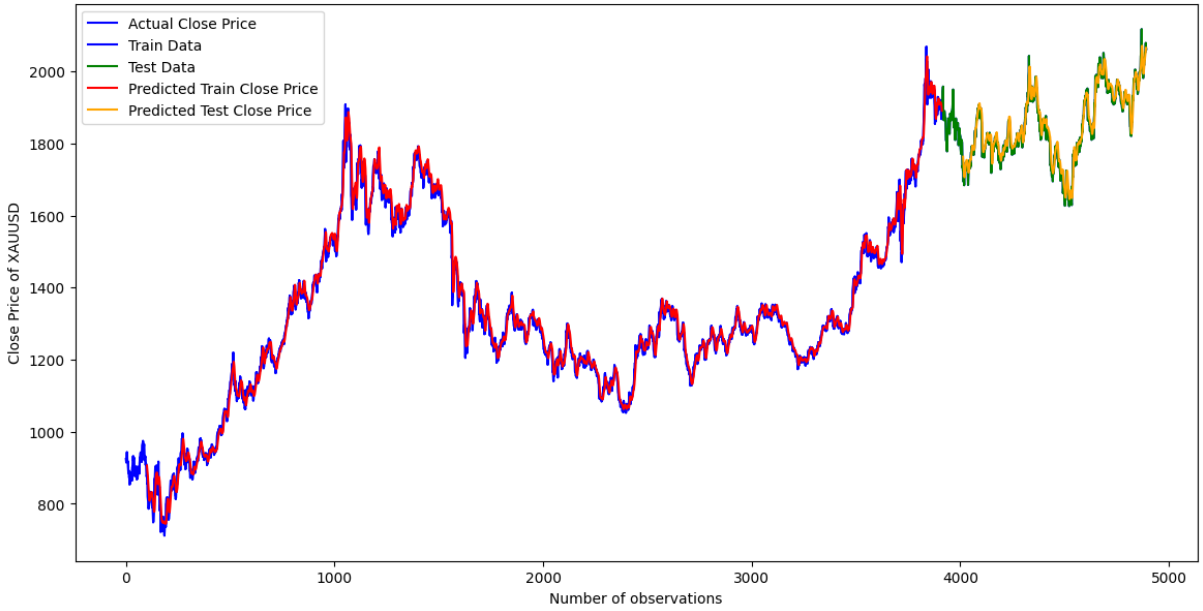


Figure 6. Actual and Predicted Train data for EURUSD with CNN model

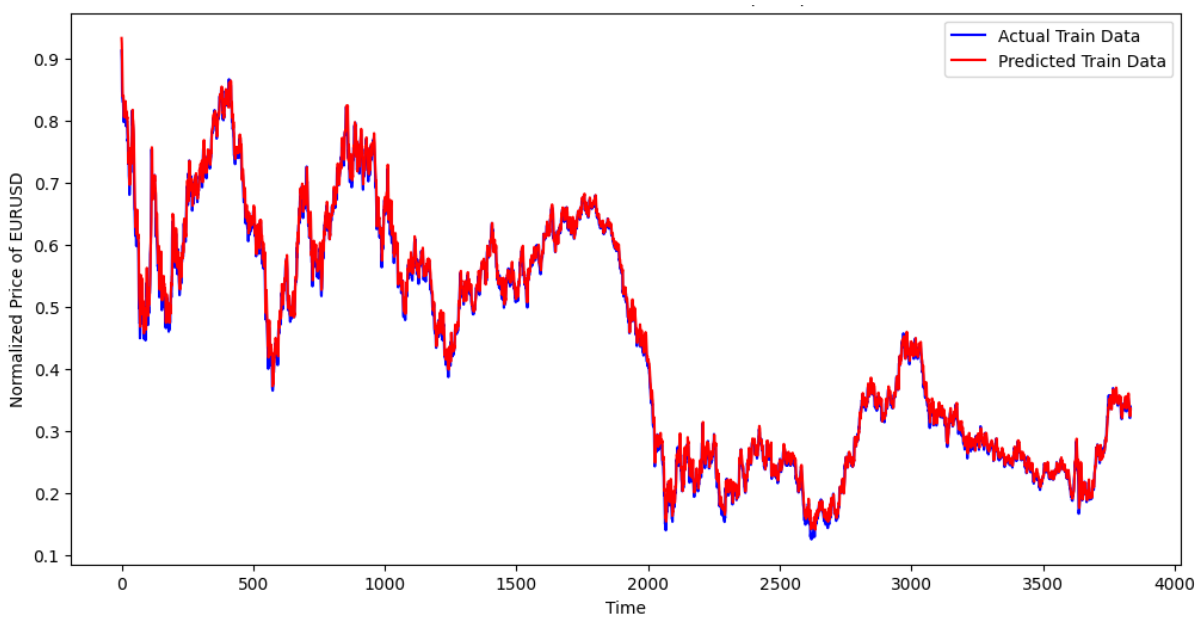


Figure 7. Actual and Predicted Test close price for EURUSD with CNN model

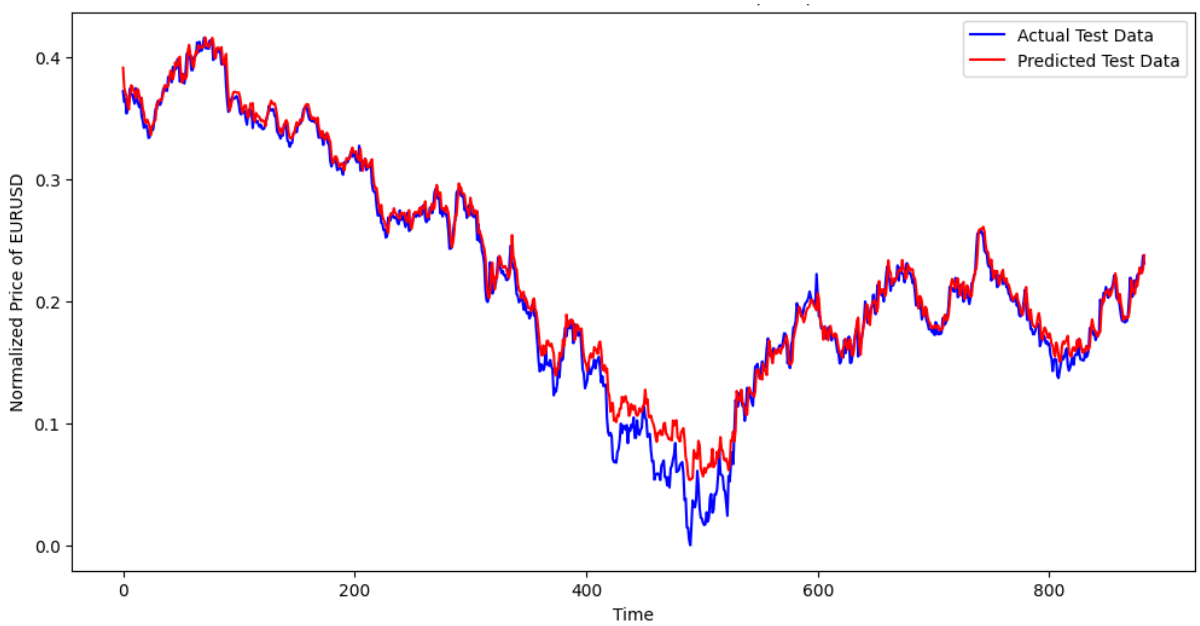


Figure 8. Actual and Predicted Train data for XAUUSD with CNN model

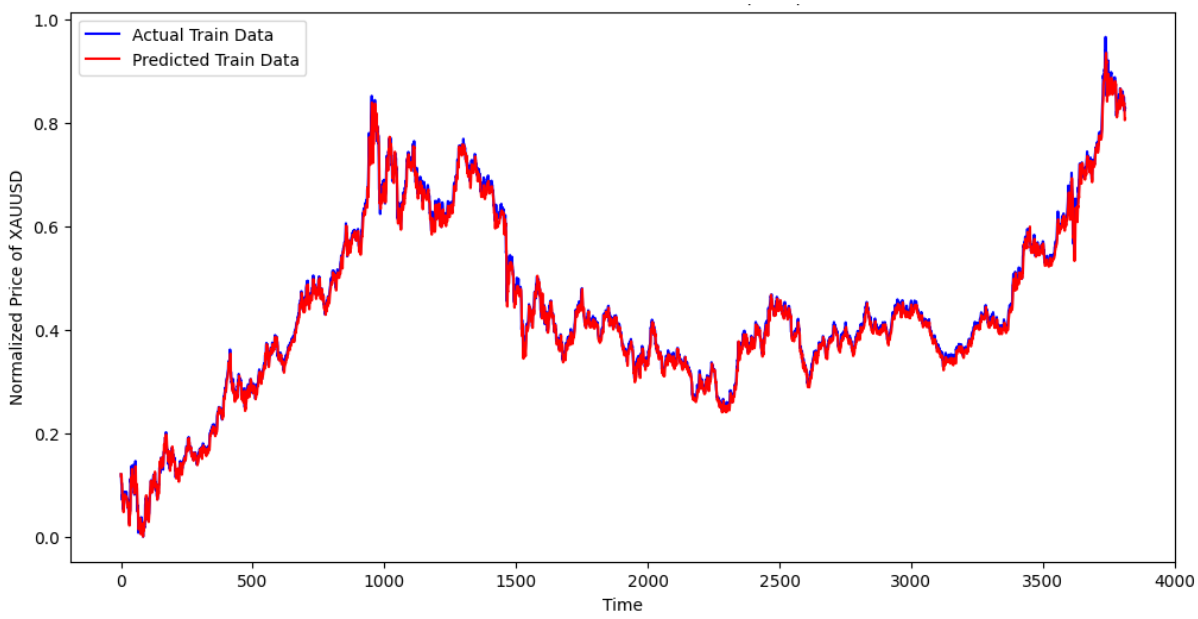


Figure 9. Actual and Predicted Test data for XAUUSD with CNN model

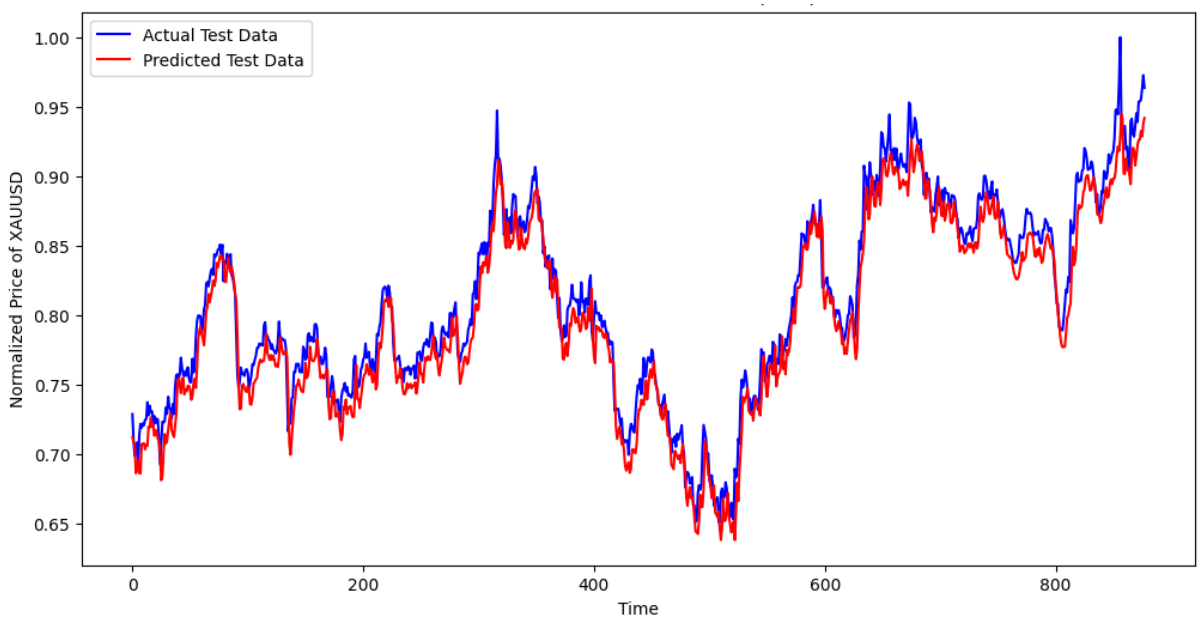


Figure 10. Actual and Predicted Train data for EURUSD with SVM model

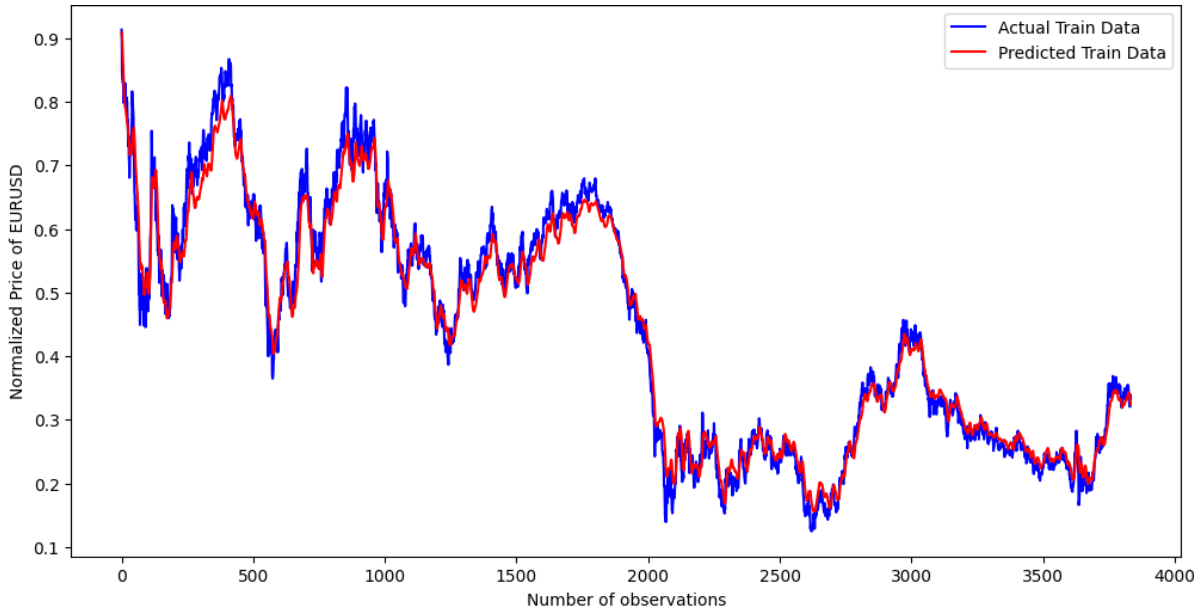


Figure 11. Actual and Predicted Test data for EURUSD with SVM model

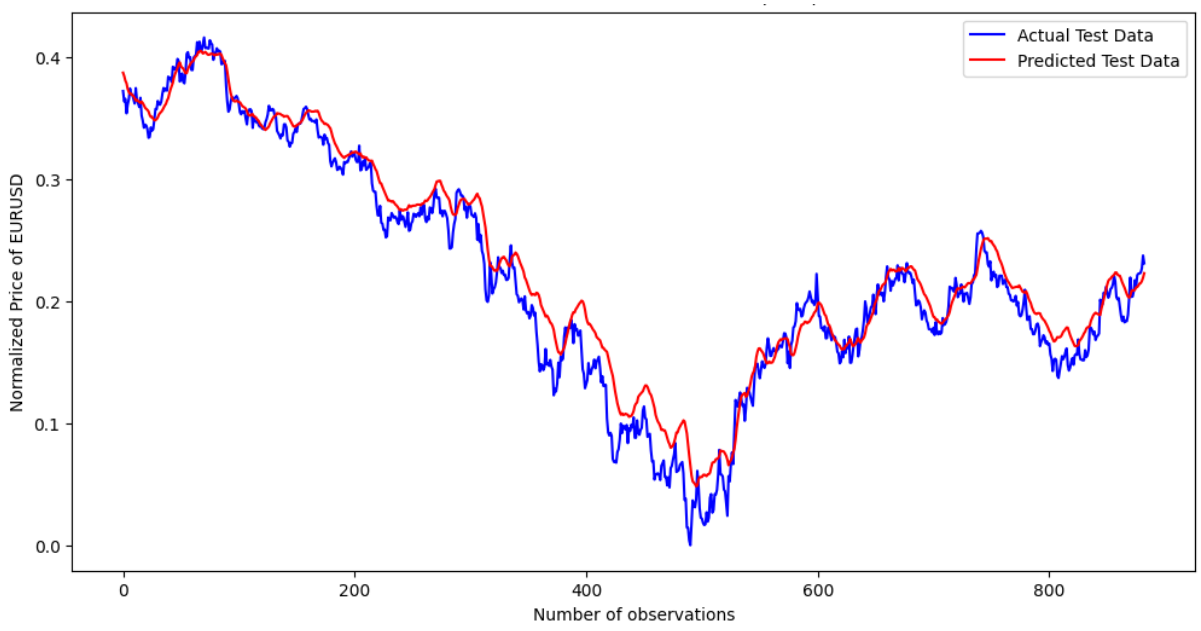


Figure 12. Actual and Predicted Training data for XAUUSD with SVM model

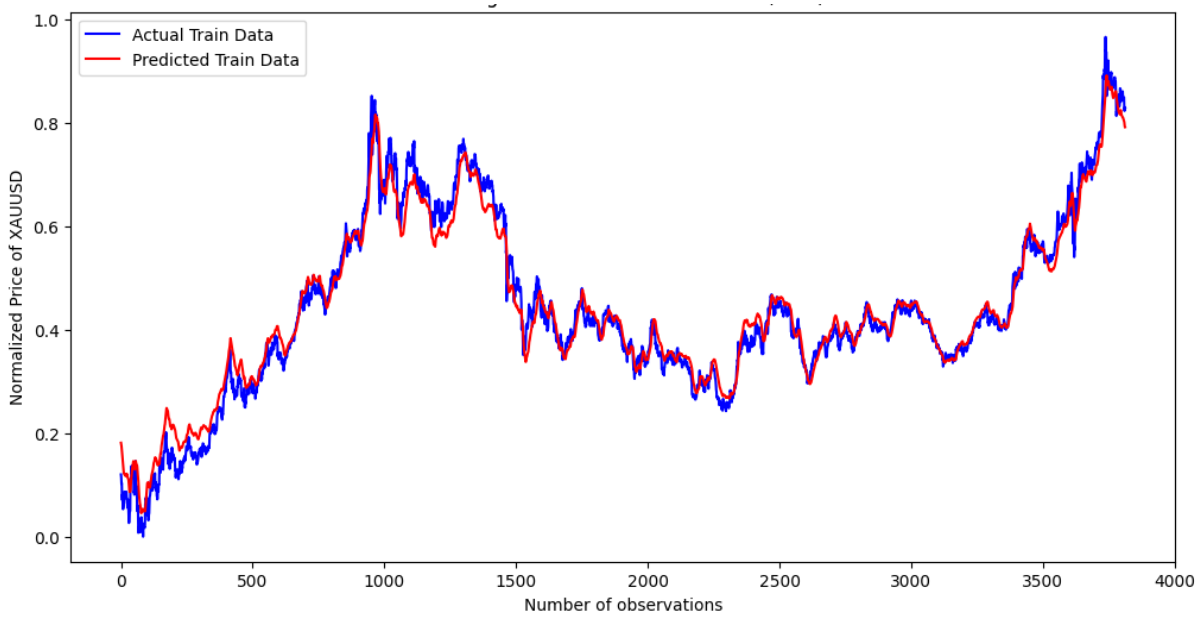
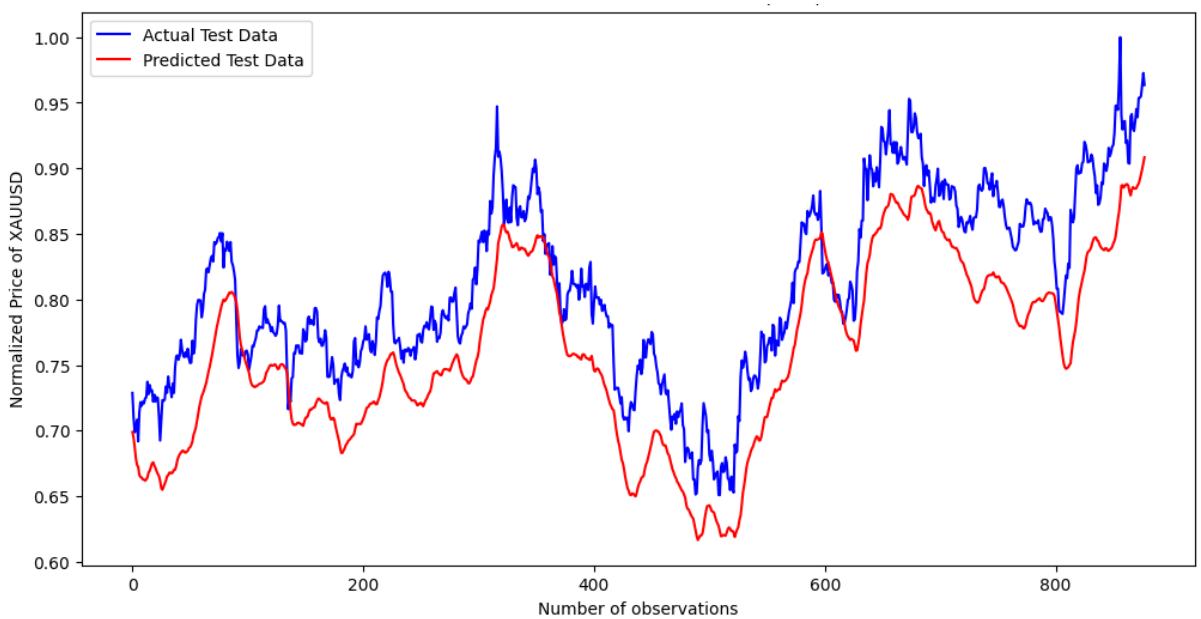


Figure 13. Actual and Predicted Test data for XAUUSD with SVM model



5 RESULTS

In this study, different predictive models were evaluated for their effectiveness in forecasting EUR/USD prices. Models such as LSTM, ARIMA, CNN, and SVM were used, and their performance was assessed using mean absolute error (MAE) and mean squared error (MSE) on both training and testing datasets Table 7. It was found that SVM showed the best overall performance, with the lowest errors on both training and testing data. While ARIMA demonstrated better results on the training dataset, its performance on the testing dataset was comparatively less effective. LSTM and CNN models showed acceptable performance but had higher errors when compared to SVM and ARIMA. Additionally, it was observed that all models displayed indications of overfitting, wherein they demonstrated lower error metrics on the training dataset than on the testing dataset. Therefore, while SVM emerged as the most accurate model for predicting EUR/USD prices.

The performance of various predictive models for XAU/USD price data was evaluated based on the metrics of mean absolute error and mean squared error for both training and testing datasets, in Table 8. Across all models, the number of observations for training and testing datasets was 3914 and 979, respectively. The models were then assessed for their predictive accuracy. In the evaluation, it was found that the LSTM, CNN, and SVM models displayed relatively low errors for both training and testing datasets. The LSTM model achieved an MAE of 0.194185 and 0.07969 for training and testing datasets, respectively, along with MSE values of 0.061783 and 0.009708. The CNN model demonstrated comparable performance, with an MAE of 0.192454 and 0.079713 for training and testing datasets, and MSE values of 0.060708 and 0.009714. The SVM model exhibited the lowest errors among all models, with an MAE of 0.022049 and 0.049115 for training and testing datasets, and MSE values of 0.000815 and 0.002907. Conversely, the ARIMA model displayed significantly higher errors, with MAE and MSE values substantially larger than those of the other models, it shows poorer predictive performance across both training and testing datasets. The results of predictive models for XAU/USD price data, shows SVM is the most accurate model, followed by LSTM, CNN, and ARIMA, respectively.

When comparing the results of the XAU/USD and EUR/USD predictive models, it is noticed that some differences exist. The Support Vector Machine (SVM) model generally performed the best for both currency pairs, showing the lowest errors in terms of MAE and MSE. However, the exact numbers varied slightly between the

two pairs. For XAU/USD, SVM had an MAE of 0.022049 for training data and 0.049115 for testing data, with MSE values of 0.000815 and 0.002907, respectively. On the other hand, for EUR/USD, SVM had slightly lower errors, with an MAE of 0.020459 for training and 0.015457 for testing, and MSE values of 0.000711 and 0.000409, respectively. The rankings of other models like LSTM and CNN remained consistent across both pairs, while ARIMA consistently showed higher errors compared to the other models. These findings highlight how predictive performance can vary depending on the specific currency pair, emphasizing the need to tailor modeling approaches for each dataset to achieve the most accurate forecasts.

6 SUMMARY

Looking back on this thesis journey, the use of machine learning to predict currency prices, especially for EURUSD and GOLD, was explored. Various models like LSTM, Autoregressive Integrated Moving Average (ARIMA), Support Vector Machines (SVM), and CNN were tested, focusing on their forecasting performance and practical advice for traders and researchers. Although factors like economic impacts on currency markets were not considered, the target was to evaluate these models' effectiveness and offer practical guidance. Questions of thesis were addressed. The goal was to present the findings in an easy-to-understand way, providing traders and researchers with useful insights. By using advanced machine learning techniques and looking into financial data, efforts were made to find better methods for predicting currency prices. It is hoped that this knowledge will help traders, researchers, and financial institutions to make more informed decisions in the dynamic world of currency markets. In the future, exploring how manipulating various parameters of predictive models affects their performance will likely become increasingly important in currency trading. By fine-tuning parameters such as learning rates, network architectures, or feature selections, traders and researchers can potentially enhance model accuracy and adaptability to different market conditions. However, predicting market movements using AI is not always reliable. Even with advanced machine learning, it is hard to get accurate forecasts. Other important factors, like world events and economic changes, can quickly change how markets behave. financial markets are naturally unpredictable, which AI can't completely solve. While this thesis provides helpful insights into predicting currency prices, it is important to remember that understanding the market is complex. Combining AI analysis with traditional financial knowledge and keeping up with what is happening globally is key to making smart decisions in currency trading. By bringing together these different approaches, traders can better navigate the challenges of the market and adjust their strategies as needed.

REFERENCES

Baker, M., & Wurgler, J. (2006). Investor sentiment and the cross-section of stock returns. *Journal of Finance*, 61(4), 1645-1680. Retrieved from https://www.hbs.edu/faculty/Pages/download.aspx?name=sentiment.pdf

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning.* Springer. Retrieved from https://link.springer.com/book/10.1007/978-0-387-45528-0

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: Forecasting and control.* John Wiley & Sons. Retrieved from https://www.researchgate.net/publication/299459188_Time_Series_Analysis_Forecasting_and_Control5th_Edition_by_George_E_P_Box_Gwilym_M_Jenkins_Gregory_C_Reinsel_and_Greta_M_Ljung_2015_Published_by_John_Wiley_and_Sons_Inc_Hoboken_New_Jersey_pp_712_ISBN_

Brown, T. M., & Miller, T. S. (2019). *Introduction to time series analysis.* CRC Press.

Chen, L. e. (2021). Evaluation metrics for machine learning models in currency forecasting. *Journal of Financial Engineering*, 15(2), 87-101.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.).* Springer. Retrieved from https://link.springer.com/book/10.1007/978-0-387-84858-7

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice.* OTexts. Retrieved from https://otexts.com/fpp2/

Jones, R. e. (2020). Currency forecasting using machine learning algorithms: A comparative analysis. *International Journal of Finance and Economics*, 40(2), 123-135.

Khashei, M., & Bijari, M. (2010). *An artificial neural network (p, d, q) model for timeseries forecasting.* Elsevier. Retrieved from https://doi.org/10.1016/j.eswa.2009.05.044

Leung, M. T., Chen, A. S., & Daouk, H. (2000). *Forecasting exchange rates using general.* Elseiver. Retrieved from https://doi.org/10.1016/S0305-0548(99)00144-6

Lina Ni, Yujie Li, Xiao Wang, Jinquan Zhang, Jiguo Yu, Chengming Qi. (2019). *Forecasting of Forex Time Series Data Based on Deep Learning.* Procedia Computer Science. Retrieved from https://www.sciencedirect.com/science/article/pii/S1877050919302066

Liu, Y., & Ma, C. (2019). Forecasting currency exchange rates using machine learning techniques. *Expert Systems with Applications*, 134, 279-291.

Malcolm Baker & Jeffrey Wurgler. (2006). Investor Sentiment and the Cross-Section of Stock Returns. *Journal of Finance*. Retrieved from https://pages.stern.nyu.edu/~jwurgler/papers/wurgler_baker_cross_section.pdf

Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications.* Penguin. Retrieved from https://cdn.preterhuman.net/texts/unsorted2/Stock%20books%20029/John%20J%20Murphy%20-%20Technical%20Analysis%20Of%20The%20Financial%20Markets.pdf

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* MIT PRESS. Retrieved from https://books.google.fi/books?id=RC43AgAAQBAJ&lpg=PR7&ots=ungyeBNp17&dq=Machine%20learning%3A%20a%20probabilistic%20perspective%20Murphy&lr&pg=PR8#v=onepage&q=Machine%20learning:%20a%20probabilistic%20perspective%20Murphy&f=false

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press. Retrieved from https://mitpress.mit.edu/9780262039246/reinforcement-learning/

Appendix 1: Material management plan

My research materials will be kept safe and organized for my thesis. All data, code, and documents will be stored on my laptop computer. Additionally, everything will be regularly backed up to prevent the loss of important information. Sensitive information, such as personal details, will be protected using strong passwords and encryption. Access to the information will only be granted to authorized individuals.

All research materials will be retained for at least one year after the thesis is approved, facilitating future verification if necessary. After one year, proper disposal procedures will be ensured.