National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Hamza Pasking Akhtar
Student ID: X23115033

School of Computing
National College of Ireland

Supervisor: Abid Yaqoob

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Hamza Pasking Akhtar
Student ID:	X23115033
Programme:	MSc in Data Analytics **Year:** 2024 - 2025
Module:	MSc Research Project
Lecturer:	Abid Yaqoob
Submission Due Date:	12/12/2024
Project Title:	Forex Price Prediction Using Deep Learning and Comparative Analysis with Traditional Time Series Models
Word Count:	836 **Page Count:** 18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Hamza Pasking Akhtar
Date:	12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Hamza Pasking Akhtar
Student ID: x23115033

Introduction

Forex is the largest financial market in the world and being able to identify the trend effectively is crucial to gain insight and benefit from it. This configuration manual provides a detailed step-by-step guide to the configuration and the implementation of the research project.

1 Environment

This section provides details about the environment setup and the system requirements necessary for implementing the project, in this case, the code is executed using Jupyter Notebook which is provided by Anaconda Navigator

1.1 System Configuration

Table 1. Software Configuration Specifications

Requirement	**Specification**
Programming Language Used	Python 3
Tools	Anaconda Navigator, Jupyter Notebook, Word
Yahoo Finance API	Access Yahoo Finance API to get dataset
Operating System	Windows 11

Table 2. Hardware Configuration Specifications

Requirement	**Specification**
Processor	8th Gen Intel i7-8665U @ 1.90 GHz
RAM	16 GB Ram
Storage	256 GB SSD

1.2 Dataset

The dataset was taken from Yahoo Finance API which consist of 5188 entries.

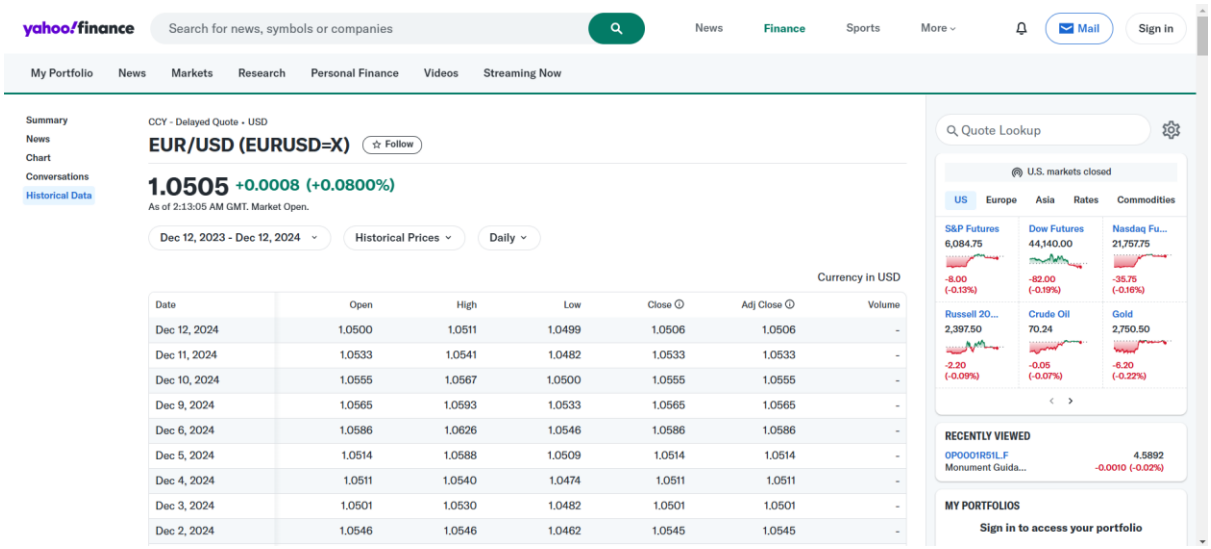

Figure 1 Data Source

1.3 Anaconda Navigator

The models utilized in this research was implemented using the Python programming language. The programming code portion of the script is completed using Jupyter Notebook, which has been configured by Anaconda Navigator.

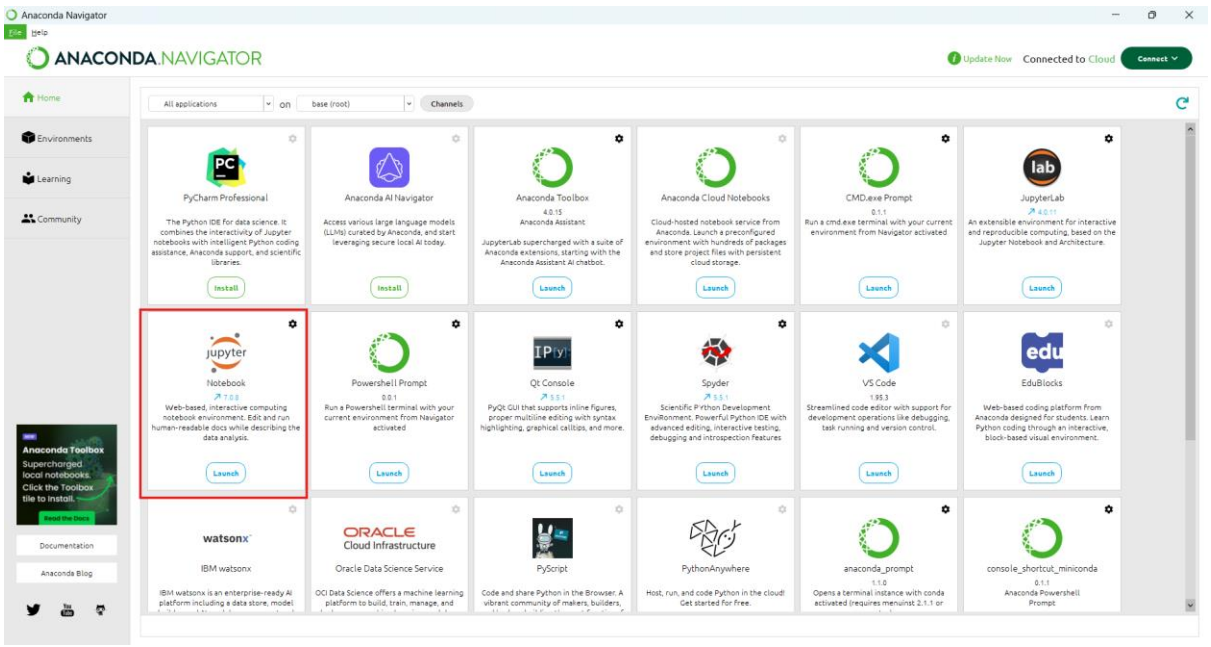

Figure 2 Anaconda Navigator

1.4 Importing Libraries

Different libraries required for the execution, evaluation and visualization of the models used in this research has been imported and installed.

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

from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error,mean_absolute_error

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

Figure 3 Libraries

2 Data Preprocessing

2.1 Data Loading

The dataset used in this research is taken from Yahoo Finance which is then saved as a CSV and read from there for further analysis.

```python
import yfinance as yf

# Defining the ticker symbol for the Yahoo API and the date range I want
ticker = "EURUSD=X"
start_date = "2004-01-01"
end_date = "2023-12-31"

# Downloading the daily interval
data = yf.download(ticker, start=start_date, end=end_date, interval="1d")

data.to_csv("EURUSD_Daily_2004-2023.csv")
print("Data saved to 'EURUSD_Daily_2004-2023.csv'")
```

```
[**********100%**********]  1 of 1 completed
Data saved to 'EURUSD_Daily_2004-2023.csv'
```

Figure 4 Getting and saving data as CSV

```python
import pandas as pd

# Loading the data
eurusd_data = pd.read_csv("EURUSD_Daily_2004-2023.csv")
```

Figure 5 Reading the data

2.2 Exploratory Data Analysis (EDA)

Before any processing is done the data is explored to have a better understanding of what the dataset contains.

```python
# Checking the first few rows to understand the data structure
eurusd_data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-01-01	1.259002	1.260796	1.247396	1.258194	1.258194	0
1	2004-01-02	1.258194	1.262802	1.252693	1.258194	1.258194	0
2	2004-01-05	1.263903	1.269406	1.263695	1.268698	1.268698	0
3	2004-01-06	1.268907	1.280803	1.267202	1.272103	1.272103	0
4	2004-01-07	1.272394	1.273999	1.262499	1.264095	1.264095	0

Figure 6 Initial 5 rows

```
[12]: # Correlation Matrix to understand correlation
      plt.figure(figsize=(10, 6))
      sns.heatmap(eurusd_data[['Open', 'High', 'Low', 'Close']].corr(), annot=True, cmap='Blues', linewidths=0.5)
      plt.title('Correlation Matrix')
      plt.show()
```

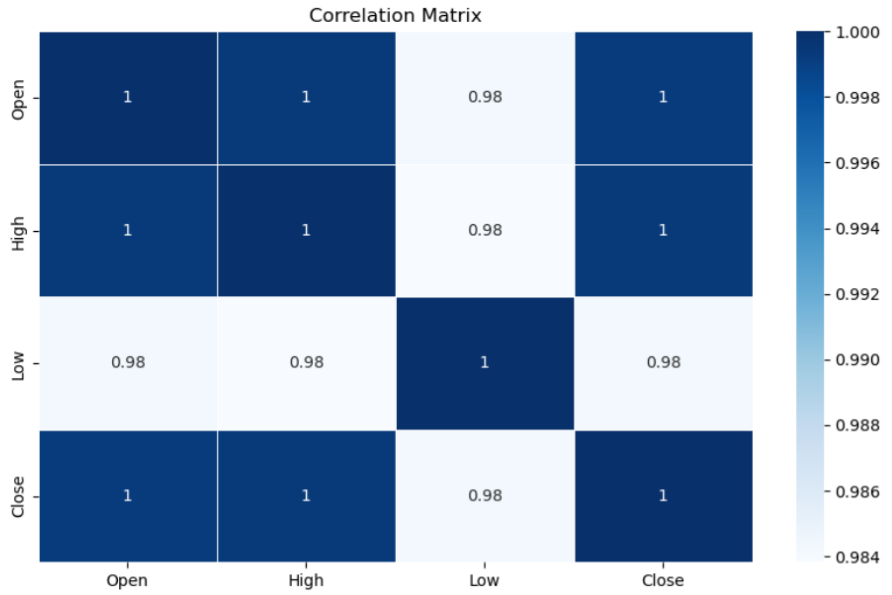

Figure 7 Correlation matrix of the columns

Checking the seasonal decomposition of the data.

```
decomposition = seasonal_decompose(eurusd_data['Close'], model='additive', period=365)

# Plotting the decomposition components
plt.figure(figsize=(12, 8))
plt.subplot(411)
plt.plot(eurusd_data['Date'], decomposition.observed, label='Observed', color='blue')
plt.legend(loc='upper left')

plt.subplot(412)
plt.plot(eurusd_data['Date'], decomposition.trend, label='Trend', color='orange')
plt.legend(loc='upper left')

plt.subplot(413)
plt.plot(eurusd_data['Date'], decomposition.seasonal, label='Seasonal', color='green')
plt.legend(loc='upper left')

plt.subplot(414)
plt.plot(eurusd_data['Date'], decomposition.resid, label='Residual', color='red')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```

Figure 8 Seasonal decomposition

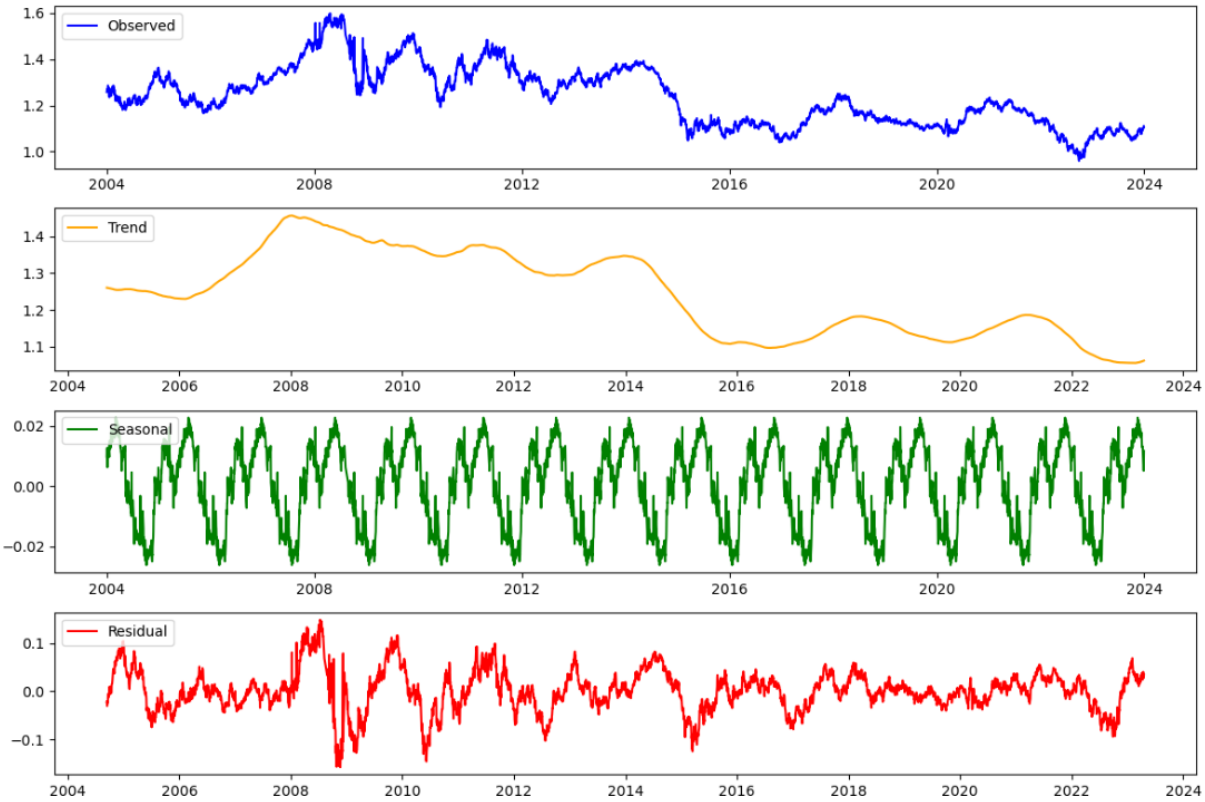

Figure 9 Seasonal decomposition

2.3 Data Preprocessing

Train and Test Split

```
[11]: import seaborn as sns

# Creating a split for train and test
train_data = eurusd_data[eurusd_data['Date'] < '2022-01-01']
test_data = eurusd_data[eurusd_data['Date'] >= '2022-01-01']

# Visualization of the Train and Test Data Closing Price
plt.figure(figsize=(12, 6))
plt.plot(train_data['Date'], train_data['Close'], label='Train Data (Close Price)', color='blue')
plt.plot(test_data['Date'], test_data['Close'], label='Test Data (Close Price)', color='red')
plt.title('Train and Test Split (EUR/USD Closing Price)')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 10 Data split to train and test

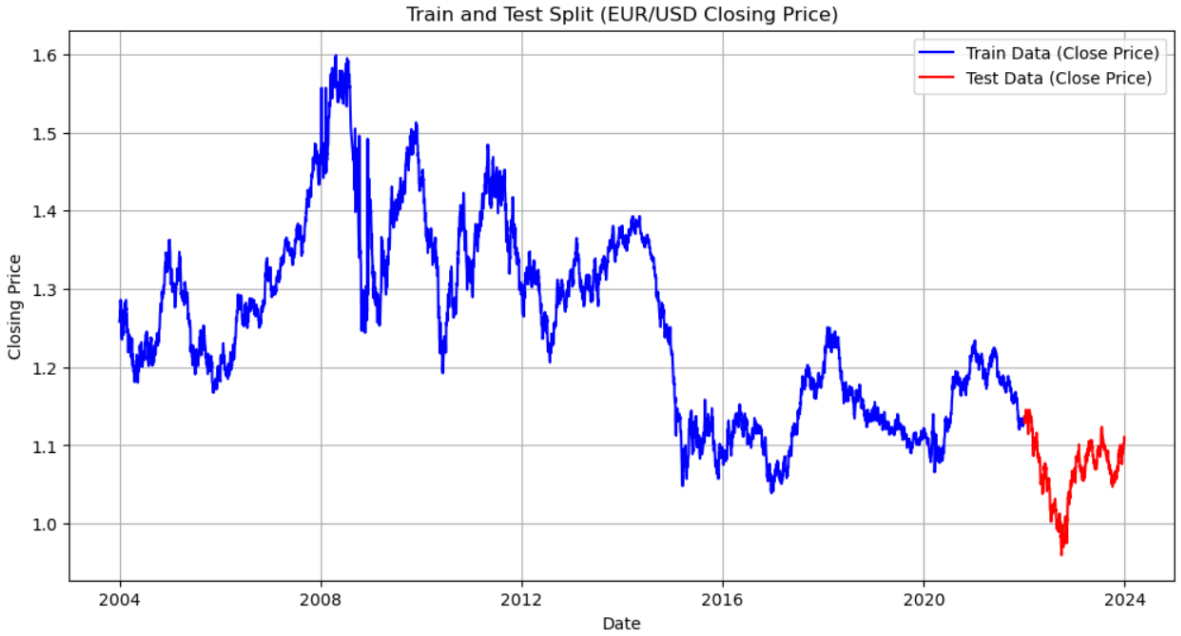

Figure 11 Split visualization

Now moving towards the implementation of the model

3 ARIMA

Checking how ARIMA does with the dataset. After checking if stationary or not by dickey fuller test we can use the PACF and ACF plot. Based on the stationarity situation we apply differencing as well.

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Using the differenced series to achieve stationarity
train_data_diff = train_data['Close'].diff().dropna()

# Plotting ACF and PACF
plt.figure(figsize=(14, 6))

# ACF Plot
plt.subplot(121)
plot_acf(train_data_diff, ax=plt.gca(), lags=40)
plt.title('Autocorrelation Function (ACF)')

# PACF Plot
plt.subplot(122)
plot_pacf(train_data_diff, ax=plt.gca(), lags=40)
plt.title('Partial Autocorrelation Function (PACF)')

plt.tight_layout()
plt.show()
```

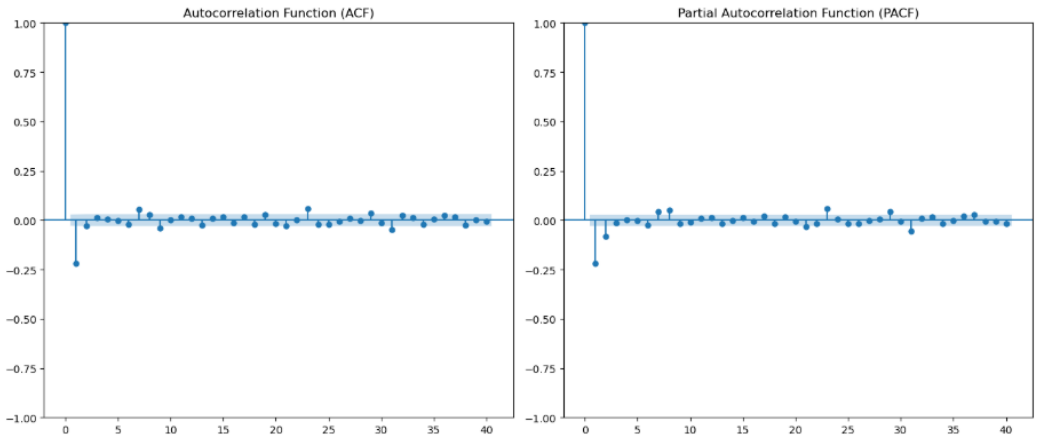

Figure 12 ACF and PACF Plot

Applying ARIMA Model

[21]:
```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error,mean_absolute_error
import warnings

# Ignoring warnings for better readability of output
warnings.filterwarnings('ignore')

# Applying ARIMA Model
# We'll use first-order differenced data for ARIMA model training
train_data_diff = train_data['Close'].diff().dropna()
test_data_diff = test_data['Close'].diff().dropna()


# Fit ARIMA model on training data
arima_order = (1, 1, 1)
arima_model = ARIMA(train_data_diff, order=arima_order)
arima_model_fit = arima_model.fit()
```

Forecasting using ARIMA

[23]:
```
# Forecasting using ARIMA

forecast_steps = len(test_data_diff)  # Number of steps to forecast
forecast = arima_model_fit.forecast(steps=forecast_steps)
forecast
```

[23]:
```
4667    0.000711
4668   -0.000221
```

Figure 13 Applying and forecasting using ARIMA

4 SARIMA

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Applying SARIMA Model

sarima_order = (1, 1, 1)
seasonal_order = (1, 1, 1, 12)

# Fitting SARIMA model on the entire training data
sarima_model = SARIMAX(train_data['Close'], order=sarima_order, seasonal_order=seasonal_order)
sarima_model_fit = sarima_model.fit(disp=False)
```

Forecasting using SARIMA

```
sarima_forecast = sarima_model_fit.forecast(steps=len(test_data))
```

Visualizing SARIMA Forecast

```
plt.figure(figsize=(12, 6))

plt.plot(train_data['Date'], train_data['Close'], label='Training Data', color='skyblue')

plt.plot(test_data['Date'], test_data['Close'], label='Test Data', color='blue')

plt.plot(test_data['Date'], sarima_forecast, label='SARIMA Forecast', color='orange', linestyle='--')

plt.title('EUR/USD Forex Price Prediction using SARIMA')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 14 SARIMA Forecasting and Visualization

5 LSTM

Tensorflow is installed as it is required for LSTM by doing “pip install tensorflow”.

We then proceed to preprocess by doing a split and scaling using MinMaxScaler

```
# Using MinMaxScaler to scale the data between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(closing_prices[['Close']])

# Splitting the data into training and test sets
train_size = len(closing_prices[closing_prices['Date'] < '2022-01-01'])
train_data_scaled = scaled_data[:train_size]
test_data_scaled = scaled_data[train_size:]
```

Figure 15 LSTM Preprocessing

```
# Defining the sequence length
# e.g., using previous 60 days to predict the next value
sequence_length = 60

# Function to create input-output sequences
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(seq_length, len(data)):
        X.append(data[i-seq_length:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

# Create training and test sequences
X_train, y_train = create_sequences(train_data_scaled, sequence_length)
X_test, y_test = create_sequences(test_data_scaled, sequence_length)

# Reshape input data as required for 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))
```

Figure 16 LSTM Sequence

After creating the LSTM Sequence we build the LSTM model

```
# Initializing the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=1))

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the model
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/20
144/144 ━━━━━━━━━━━━━━━━━━━━ 8s 29ms/step - loss: 0.0382 - val_loss: 6.9099e-04
Epoch 2/20
144/144 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - loss: 9.1888e-04 - val_loss: 4.8936e-04
Epoch 3/20
144/144 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - loss: 7.9786e-04 - val_loss: 4.1949e-04
Epoch 4/20
144/144 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - loss: 6.8666e-04 - val_loss: 3.6874e-04
Epoch 5/20
144/144 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - loss: 6.1625e-04 - val_loss: 3.7282e-04
Epoch 6/20
144/144 ━━━━━━━━━━━━━━━━━━━━ 5s 38ms/step - loss: 6.4642e-04 - val_loss: 3.1760e-04
Epoch 7/20
144/144 ━━━━━━━━━━━━━━━━━━━━ 5s 30ms/step - loss: 5.8889e-04 - val_loss: 3.2408e-04
Epoch 8/20
```

Figure 17 LSTM Model Building

After building we do our prediction and visualization

Predictions

```
# Predicting on test data
predictions = model.predict(X_test)

# Inverse transform the predictions and test data to original scale
predictions_unscaled = scaler.inverse_transform(predictions.reshape(-1, 1))
y_test_unscaled = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step
```

Visualization

```
plt.figure(figsize=(12, 6))

# Plotting actual test data
plt.plot(closing_prices['Date'][-len(y_test_unscaled):], y_test_unscaled, label='Actual Test Data', color='blue')

# Plotting LSTM predicted values
plt.plot(closing_prices['Date'][-len(predictions_unscaled):], predictions_unscaled, label='LSTM Forecast', color='orange', linestyle='--')

# Customizing the plot
plt.title('EUR/USD Forex Price Prediction using LSTM')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 18 LSTM prediction and visualization

6 Fine Tuned LSTM

LSTM is then fine tuned using hyperparameter tuning to improve its performance

Preparing LSTM Sequence

[53]:
```
# Define sequence length
# Using previous 60 days to predict the next value
sequence_length = 60

def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(seq_length, len(data)):
        X.append(data[i-seq_length:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

# Create training and test sequences
X_train, y_train = create_sequences(train_data_scaled, sequence_length)
X_test, y_test = create_sequences(test_data_scaled, sequence_length)

# Reshape input data as required for 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))
```

Building and Optimizing Model

[55]:
```
# Initializing the LSTM model
model = Sequential()
model.add(LSTM(units=100, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.3))
model.add(LSTM(units=100, return_sequences=True))
model.add(Dropout(0.3))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.3))
model.add(Dense(units=1))

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Early Stopping and Model Checkpoint Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_lstm_model.keras', save_best_only=True, monitor='val_loss', mode='min')

# Training the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1, callbacks=[early_stopping, model_checkpoint])
```

```
Epoch 1/50
144/144 ━━━━━━━━━━━━━━━━━━━━ 16s 72ms/step - loss: 0.0223 - val_loss: 6.6821e-04
```

Figure 19 LSTM Model building and optimization

Prediction

```
# Predict on test data
predictions = model.predict(X_test)

# Inverse transform the predictions and test data to original scale
predictions_unscaled = scaler.inverse_transform(predictions.reshape(-1, 1))
y_test_unscaled = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step
```

Visualization

```
plt.figure(figsize=(12, 6))

# Plotting actual test data
plt.plot(closing_prices['Date'][-len(y_test_unscaled):], y_test_unscaled, label='Actual Test Data', color='blue')

# Plotting LSTM predicted values
plt.plot(closing_prices['Date'][-len(predictions_unscaled):], predictions_unscaled, label='Optimized LSTM Forecast', color='orange', linestyle='--')

# Customizing the plot
plt.title('EUR/USD Forex Price Prediction using Optimized LSTM')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 20 Prediction and Visualization

7 Bidirectional LSTM

```
# Define sequence length
# using previous 60 days to predict the next value
sequence_length = 60

# Function to create input-output sequences
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(seq_length, len(data)):
        X.append(data[i-seq_length:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

# Create training and test sequences
X_train, y_train = create_sequences(train_data_scaled, sequence_length)
X_test, y_test = create_sequences(test_data_scaled, sequence_length)

# Reshape input data for 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))
```

```
# Initializing the Bi-Directional LSTM model
model = Sequential()
model.add(Bidirectional(LSTM(units=100, return_sequences=True, input_shape=(X_train.shape[1], 1))))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(units=100, return_sequences=False)))
model.add(Dropout(0.3))
model.add(Dense(units=1))

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the model
model.fit(X_train, y_train, epochs=30, batch_size=32, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/30
144/144 ━━━━━━━━━━━━━━━━━━━━ 21s 83ms/step - loss: 0.0198 - val_loss: 6.2890e-04
```

Figure 21 Bidirectional LSTM model building

```
# Predicting on test data
predictions = model.predict(X_test)

# Inverse transform the predictions and test data to original scale
predictions_unscaled = scaler.inverse_transform(predictions.reshape(-1, 1))
y_test_unscaled = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 2s 81ms/step
```

```
plt.figure(figsize=(12, 6))

# Plotting actual test data
plt.plot(closing_prices['Date'][-len(y_test_unscaled):], y_test_unscaled, label='Actual Test Data', color='blue')

# Plotting Bi-Directional LSTM predicted values
plt.plot(closing_prices['Date'][-len(predictions_unscaled):], predictions_unscaled, label='Bi-Directional LSTM Forecast', color='orange', linestyle='--'

# Customizing the plot
plt.title('EUR/USD Forex Price Prediction using Bi-Directional LSTM')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 22 Bidirectional LSTM prediction and visualization

8 Technical Indicators LSTM

```
# Indicator 1
# Moving Average Convergence Divergence (MACD)
eurusd_data['EMA_12'] = eurusd_data['Close'].ewm(span=12, adjust=False).mean()  # 12-day EMA
eurusd_data['EMA_26'] = eurusd_data['Close'].ewm(span=26, adjust=False).mean()  # 26-day EMA
eurusd_data['MACD'] = eurusd_data['EMA_12'] - eurusd_data['EMA_26']  # MACD Line
eurusd_data['Signal_Line'] = eurusd_data['MACD'].ewm(span=9, adjust=False).mean()  # Signal Line

# MACD Histogram which is difference between MACD and Signal line
eurusd_data['MACD_Hist'] = eurusd_data['MACD'] - eurusd_data['Signal_Line']

# Indicator 2
# Moving Average (50-day and 200-day)
eurusd_data['MA_50'] = eurusd_data['Close'].rolling(window=50).mean()  # 50-day moving average
eurusd_data['MA_200'] = eurusd_data['Close'].rolling(window=200).mean()  # 200-day moving average

# Indicator 3
# Relative Strength Index (RSI)
delta = eurusd_data['Close'].diff(1)
gain = delta.where(delta > 0, 0)
loss = -delta.where(delta < 0, 0)
average_gain = gain.rolling(window=14).mean()
average_loss = loss.rolling(window=14).mean()
rs = average_gain / average_loss
eurusd_data['RSI'] = 100 - (100 / (1 + rs))

# Drop rows with NaN values resulting from moving average and RSI calculations
eurusd_data = eurusd_data.dropna()
```

```
# Features to be used in the model are Close, MACD, Signal_Line, MA_50, MA_200, RSI
features = eurusd_data[['Close', 'MACD', 'Signal_Line', 'MA_50', 'MA_200', 'RSI']]

# Using MinMaxScaler to scale the data between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_features = scaler.fit_transform(features)

# Splitting the data into training and test sets
train_size = len(eurusd_data[eurusd_data['Date'] < '2022-01-01'])
train_data_scaled = scaled_features[:train_size]
test_data_scaled = scaled_features[train_size:]
```

Figure 23 LSTM with Technical Indicators

```
# Define sequence length
sequence_length = 60

# Function to create input-output sequences
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(seq_length, len(data)):
        X.append(data[i-seq_length:i])  # Input sequence of length `seq_length`
        y.append(data[i, 0])  # Predict the closing price (first column)
    return np.array(X), np.array(y)

# Creating training and test sequences
X_train, y_train = create_sequences(train_data_scaled, sequence_length)
X_test, y_test = create_sequences(test_data_scaled, sequence_length)

# Reshaping input data for LSTM
# The input shape should be [samples, time steps, features]
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], X_train.shape[2]))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], X_test.shape[2]))
```

```
# Initializing the LSTM model
model = Sequential()
model.add(LSTM(units=100, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.3))  # Dropout to reduce overfitting
model.add(LSTM(units=100, return_sequences=False))
model.add(Dropout(0.3))
model.add(Dense(units=1))  # Output layer to predict the closing price

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the model
model.fit(X_train, y_train, epochs=30, batch_size=32, validation_data=(X_test, y_test), verbose=1)
```

Figure 24 LSTM Indicators initialization

```
# Predicting on test data
predictions = model.predict(X_test)

# Inversing transform the predictions and test data to original scale
predictions_unscaled = scaler.inverse_transform(np.hstack((predictions, np.zeros((predictions.shape[0], X_test.shape[2] - 1)))))[..., 0]
y_test_unscaled = scaler.inverse_transform(np.hstack((y_test.reshape(-1, 1), np.zeros((y_test.shape[0], X_test.shape[2] - 1)))))[..., 0]
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step
```

```
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(14, 16), gridspec_kw={'height_ratios': [3, 1, 1]})

# Plotting actual test data and LSTM forecast on the first subplot
ax1.plot(eurusd_data['Date'][-len(y_test_unscaled):], y_test_unscaled, label='Actual Test Data', color='blue')
ax1.plot(eurusd_data['Date'][-len(predictions_unscaled):], predictions_unscaled, label='LSTM with Technical Indicators Forecast', color='orange', linest
ax1.plot(eurusd_data['Date'][-len(y_test_unscaled):], eurusd_data['MA_50'][-len(y_test_unscaled):], label='50-Day MA', color='red', linestyle='-.')
ax1.plot(eurusd_data['Date'][-len(y_test_unscaled):], eurusd_data['MA_200'][-len(y_test_unscaled):], label='200-Day MA', color='purple', linestyle='-.')
ax1.set_title('EUR/USD Forex Price Prediction with LSTM and Moving Averages')
ax1.set_xlabel('Date')
ax1.set_ylabel('Closing Price')
ax1.legend()
ax1.grid(True)

# Plotting MACD lines and histogram on the second subplot
ax2.plot(eurusd_data['Date'][-len(y_test_unscaled):], eurusd_data['MACD'][-len(y_test_unscaled):], label='MACD Line', color='blue')
ax2.plot(eurusd_data['Date'][-len(y_test_unscaled):], eurusd_data['Signal_Line'][-len(y_test_unscaled):], label='Signal Line', color='red')
ax2.bar(eurusd_data['Date'][-len(y_test_unscaled):], eurusd_data['MACD_Hist'][-len(y_test_unscaled):], color=['green' if val >= 0 else 'red' for val in
ax2.axhline(0, color='black', linewidth=0.5, linestyle='--')  # Zero line for MACD
ax2.set_title('MACD Indicator')
ax2.set_xlabel('Date')
ax2.set_ylabel('MACD Value')
ax2.legend()
ax2.grid(True)

# Plotting RSI on the third subplot
ax3.plot(eurusd_data['Date'][-len(y_test_unscaled):], eurusd_data['RSI'][-len(y_test_unscaled):], label='RSI', color='green')
ax3.axhline(70, color='red', linestyle='--', linewidth=0.5, label='Overbought (70)')
ax3.axhline(30, color='blue', linestyle='--', linewidth=0.5, label='Oversold (30)')
ax3.set_title('RSI Indicator')
ax3.set_xlabel('Date')
ax3.set_ylabel('RSI Value')
ax3.legend()
ax3.grid(True)

plt.tight_layout()
plt.show()
```

Figure 25 Prediction and Visualization of LSTM with Technical indicators