Configuration Manual

MSc Research Project
Data Analytics

Hilal Ozcelik
Student ID: X23218274

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Hilal Ozcelik……………………………………………………………………………………………
Student ID:	X23218274……………………………………………………………………………………….……
Programme:	Data Analytics…………………………………………… **Year:** 2024……………………..
Module:	MSc Research Project………………………………………………………………….………
Lecturer:	Jorge Basilio……………………………………………………………………………….………
Submission Due Date:	12/12/2024……………………………………………………………………………….………
Project Title:	Stock Price Prediction Using Machine Learning Methods: An Example of Turkish Banks ………………………………………………………………………….………
Word Count:	684……………………………… **Page Count:** …14……………………………….…….………

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: Hilal Ozcelik…………………………………………………………………………………

Date: 12/12/2024…………………………………………………………………………………

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

Hilal Ozcelik
Student ID: x23218274

1 Overview

This configuration manual outlines the hardware and software requirements for the graduation thesis project titled Stock Price Prediction Using Machine Learning Methods: An Example of Turkish Banks. It also provides a step-by-step explanation of the project's implementation.

In Chapter 2, the system installation process is explained; in Chapter 3, the necessary tools for the project are explained; in Chapter 4, the preparation of the datasets used in the project is outlined; and finally, in Chapter 5, the implementation of the project is described.

2 System

For this project, the computer described in Figure 1 was used.

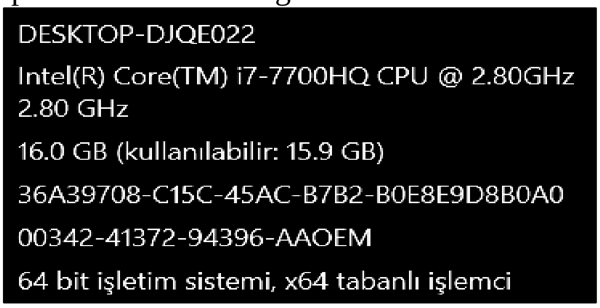

Figure 1: System Features.

3 Essential Tools for the Project

The tools used in the project are listed below.

1. Microsoft Excel: Since the data processed in this project was in .csv and .xlsx formats, Microsoft Excel 2016 was utilized for data handling.
2. Python: In this project, the Python programming language was employed throughout all stages, including data preparation, data reading, modelling, and finalization. To accomplish this, Jupyter Notebook 7.2.2 on the Anaconda platform (version details provided in Figure 2) was utilized. Furthermore, all code was executed using Python version 3.12.7.

Figure 2: Anaconda Navigator Version 2.6.3.

Libraries: A new environment was set up for this project on the Anaconda platform, and the necessary libraries were installed as listed below.

1. Pandas
2. Numpy
3. Matplotlib
4. Sklearn
5. Math
6. Keras

The codes shown in Figure 3 below were used to import the relevant libraries into the notebook.

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import plotly.graph_objects as go
import numpy as np
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, r2_score
import math
```

Figure 3: Importing the Relevant Libraries.

4 Preparation of Datasets

In this project, datasets were gathered from three different sources. The first group comprises historical stock data of banks, the second group includes financial variables, and the third group consists of textual data containing notifications from banks. All datasets are publicly available and accessible to everyone. This section provides a step-by-step explanation of the dataset preparation process, accompanied by figures between 4 and 9.

```
df_akbank = pd.read_csv("AKBNK Historical Data.csv")
```

```
df_akbank
```

	Date	Price	Open	High	Low	Vol.	Change %
0	08/29/2024	58.30	58.55	59.40	57.55	59.99M	-0.68%
1	08/28/2024	58.70	59.15	59.35	58.10	44.04M	-0.68%
2	08/27/2024	59.10	58.55	59.40	57.70	50.35M	1.03%
3	08/26/2024	58.50	58.65	59.65	57.85	52.43M	0.69%
4	08/23/2024	58.10	59.50	59.90	57.35	45.43M	-2.43%
...	...	...	...	...	...	...	...

Figure 4: Historical Stock Prices Dataset (df_akbank).

```
# Other Financial Datasets & Variables

df_spx = pd.read_csv("^spx_d_sp500.csv")
df_brent_oil = pd.read_csv("Brent Oil Futures Historical Data.csv")
df_crude_oil = pd.read_csv("Crude Oil WTI Futures Historical Data.csv")
df_ishares = pd.read_csv("tur_us_d_ishares.csv")
df_usdtry = pd.read_csv("usdtry_d.csv")
```

Figure 5: Loading Financial Variables Datasets.

```
# Converting date columns to datetime format
df_spx['Date'] = pd.to_datetime(df_spx['Date'])
df_brent_oil['Date'] = pd.to_datetime(df_brent_oil['Date'])
df_crude_oil['Date'] = pd.to_datetime(df_crude_oil['Date'])
df_ishares['Date'] = pd.to_datetime(df_ishares['Date'])
df_usdtry['Date'] = pd.to_datetime(df_usdtry['Date'])

# Rename price columns in each dataset
df_spx = df_spx[['Date', 'Close']].rename(columns={'Close': 'spx_close'})
df_brent_oil = df_brent_oil[['Date', 'Price']].rename(columns={'Price': 'brent_oil_close'})
df_crude_oil = df_crude_oil[['Date', 'Price']].rename(columns={'Price': 'crude_oil_close'})
df_ishares = df_ishares[['Date', 'Close']].rename(columns={'Close': 'ishares_close'})
df_usdtry = df_usdtry[['Date', 'Close']].rename(columns={'Close': 'usdtry_close'})

# Joining all datasets by Date column
df_financial = df_spx
df_financial = pd.merge(df_financial, df_brent_oil, on='Date', how='left')
df_financial = pd.merge(df_financial, df_crude_oil, on='Date', how='left')
df_financial = pd.merge(df_financial, df_ishares, on='Date', how='left')
df_financial= pd.merge(df_financial, df_usdtry, on='Date', how='left')

# Print
print(df_financial.head())

        Date  spx_close  brent_oil_close  crude_oil_close  ishares_close  \
0 2014-09-02    2002.28           100.34            92.88        44.4027
1 2014-09-03    2000.72           102.77            95.54        45.1391
2 2014-09-04    1997.65           101.83            94.45        45.5419
3 2014-09-05    2007.71           100.82            93.29        46.0468
4 2014-09-08    2001.54           100.20            92.66        45.4203

   usdtry_close
0        2.1720
1        2.1580
2        2.1628
3        2.1583
4        2.1724
```

Figure 6: Financial Variables Dataset (referred to as df_financial).

```python
# Converting date columns to datetime format
df_akbank['Date'] = pd.to_datetime(df_akbank['Date'])
df_financial['Date'] = pd.to_datetime(df_financial['Date'])

# Merge (left join)
df_akbank = pd.merge(df_akbank, df_financial, on='Date', how='left')

# Saving CVS
df_akbank.to_csv('df_akbank.csv', index=False)
```

Figure 7: Merging the Historical Stock Prices Dataset with the Financial Variables Dataset.

The step illustrated in Figure 7 was repeated for all banks by combining their historical stock price datasets with the financial variable dataset. Subsequently, all datasets were saved in csv format.

```python
df_kap_akbank = pd.read_excel("kap_akbank.xlsx")
df_kap_garanti = pd.read_excel("kap_garanti.xlsx")
df_kap_halk = pd.read_excel("kap_halk.xlsx")
df_kap_is = pd.read_excel("kap_is.xlsx")
df_kap_yapi = pd.read_excel("kap_yapi.xlsx")

df_kap_akbank
```

	date	info
0	2024-08-22	Applications Regarding the Issuance Limit of D...
1	2024-08-22	Applications Regarding the Issuance Limit of D...
2	2024-08-22	Applications Regarding the Issuance Limit of D...
3	2024-08-22	Authorization of General Directorate to Issue ...
4	2024-08-22	Authorization of The General Directorate For T...

Figure 8: Loading Bank Notification Datasets.

```python
df_kap_akbank = df_kap_akbank[df_kap_akbank['info'] != 'x']    # Removing rows include "x"

# Preserve the original order while grouping by 'Date' and merging 'Info' columns
df_kap_akbank = df_kap_akbank.reset_index()  # Save the current order as an index
df_kap_akbank['Original_Order'] = df_kap_akbank.index  # Store the original order in a new column

# Group by 'Date' and merge 'Info' columns
df_kap_akbank = (
    df_kap_akbank.groupby('date', sort=False)  # Keep the original date order with sort=False
    .agg({'info': ' '.join, 'Original_Order': 'min'})  # Retain the earliest original order in the group
    .reset_index()  # Convert back to DataFrame format
)

# Restore the original order using the 'Original_Order' column
df_kap_akbank = df_kap_akbank.sort_values('Original_Order').drop(columns=['Original_Order']).reset_index(drop=True)

# Verify the result
print(df_kap_akbank)
```

```
           date                                              info
0    2024-08-22  Applications Regarding the Issuance Limit of D...
1    2024-08-09  AKBNK.E Circuit breaker has been activated in ...
2    2024-08-08    Redemption of Green/Sustainable Eurobond Abroad
3    2024-08-06  AKBNK.E Circuit breaker has been activated in ...
4    2024-08-01  Announcement Regarding Derivatives Market Tran...
...         ...                                               ...
1132 2014-09-26  About the Redemption of Akbank Bonds Dated 11....
1133 2014-09-24  Regarding CMB Approval for the Issuance of Deb...
1134 2014-09-16  Regarding the 11th Coupon Payment Interest Rat...
1135 2014-09-12  Regarding the 9th Coupon Payment of the bond w...
1136 2014-09-08  About the 31st Coupon Payment of the bond with...

[1137 rows x 2 columns]
```

Figure 9: Transforming of Bank Notification Datasets.

```python
# Pre-trained sentiment analysis modeli (BERT based)
nlp = pipeline('sentiment-analysis', model='distilbert-base-uncased-finetuned-sst-2-english')

import os
os.environ["HF_HUB_DISABLE_SYMLINKS_WARNING"] = "1"

# Sentiment analysis
df_kap_akbank['Sentiment_Score'] = df_kap_akbank['info'].apply(lambda x: 1 if nlp(x)[0]['label'] == 'POSITIVE' else 0)
df_kap_akbank['Sentiment_Confidence'] = df_kap_akbank['info'].apply(lambda x: nlp(x)[0]['score'])
```

```python
df_kap_akbank
```

	date	info	Sentiment_Score	Sentiment_Confidence
0	2024-08-22	Applications Regarding the Issuance Limit of D...	0	0.887888
1	2024-08-09	AKBNK.E Circuit breaker has been activated in ...	0	0.911154
2	2024-08-08	Redemption of Green/Sustainable Eurobond Abroad	1	0.998714
3	2024-08-06	AKBNK.E Circuit breaker has been activated in ...	0	0.911154
4	2024-08-01	Announcement Regarding Derivatives Market Tran...	0	0.964382
...	...	...	...	...
1132	2014-09-26	About the Redemption of Akbank Bonds Dated 11....	0	0.846472
1133	2014-09-24	Regarding CMB Approval for the Issuance of Deb...	1	0.941857
1134	2014-09-16	Regarding the 11th Coupon Payment Interest Rat...	0	0.942757

```python
# Saving CSV file
df_kap_akbank.to_csv('df_kap_akbank.csv', index=False)
```

Figure 9: Sentiment Analysis of Bank Notification Datasets Using Hugging Face.

The text data processing analysis shown in Figure 9 was repeated for each bank, and the new datasets, containing all sentiment confidence information, were saved in CSV format.

5 A Comprehensive Guide to Implementing the Project

This section explains all stages, from the preliminary preparation of the datasets to modelling and model results, with the support of Figures 10 to 21. Since similar processes are applied to each bank, figures from a single bank are used as examples.

5.1 Preprocessing

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import plotly.graph_objects as go
import numpy as np
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, r2_score
import math
```

```
df_akbank = pd.read_csv("df_akbank.csv")
df_garanti = pd.read_csv("df_garanti.csv")
df_halkbank = pd.read_csv("df_halkbank.csv")
df_isbank = pd.read_csv("df_isbank.csv")
df_yapi = pd.read_csv("df_yapi.csv")
```

Figure 10: Loading Necessary Libraries and Bank Datasets.

```
# All Datasets Name
datasets = [df_akbank, df_garanti, df_halkbank, df_isbank, df_yapi]
dataset_names = ['df_akbank', 'df_garanti', 'df_halkbank', 'df_isbank', 'df_yapi']

# Rows and Colums
n_rows = -(-len(datasets) // 3)
fig, axes = plt.subplots(n_rows, 3, figsize=(18, 5 * n_rows))

axes = axes.flatten()  # To make a flatten axes

# Line Graph for each datasets
for i, (df, name) in enumerate(zip(datasets, dataset_names)):
    df['Date'] = pd.to_datetime(df['Date'])   # Transform to Date Column as datetime

    ax = axes[i]
    ax.plot(df['Date'], df['Price'], label=f'{name} Price', color='#10a2dc')

    ax.set_title(f'{name} Price Over Time', fontsize=11)
    ax.set_xlabel('Date', fontsize=10)
    ax.set_ylabel('Price', fontsize=10)
    ax.grid(True, linestyle='--', alpha=0.6)
    ax.set_facecolor('white')
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))   # Date as year
    ax.xaxis.set_major_locator(mdates.YearLocator())
    ax.tick_params(axis='x', rotation=45)

for j in range(len(datasets), len(axes)):
    fig.delaxes(axes[j])

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

Figure 11: Plotting the Closing Prices of Each Bank.

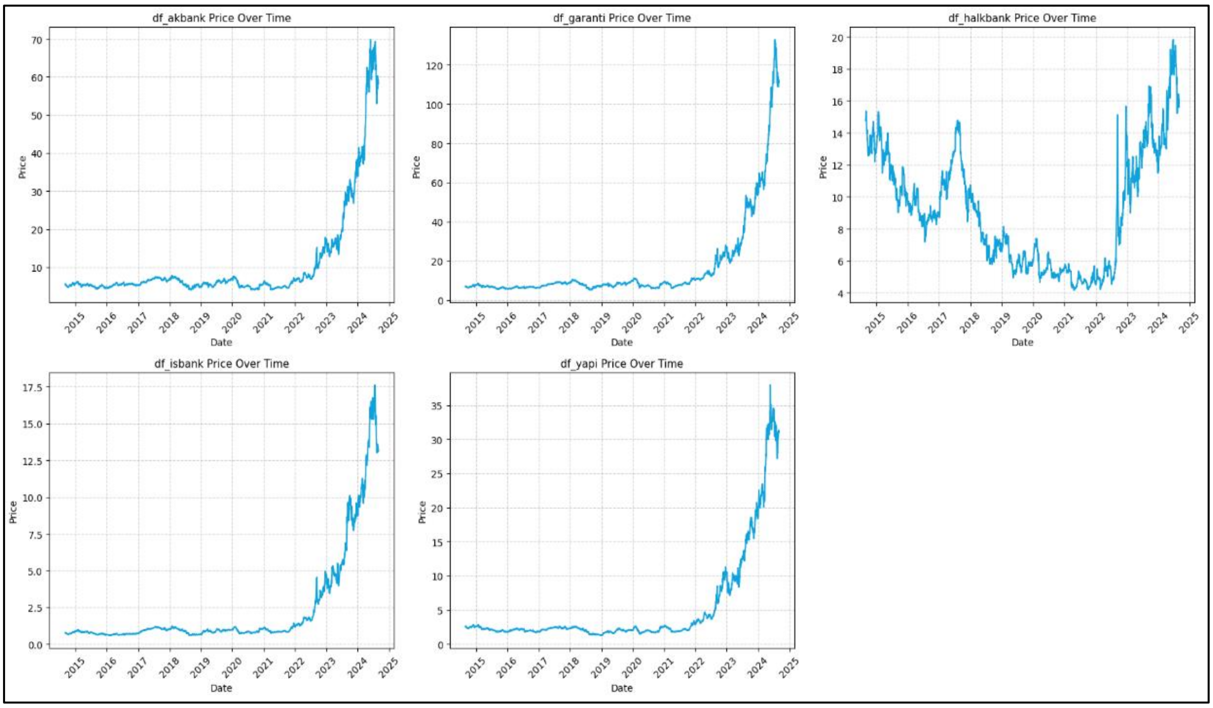

Figure 12: Closing Prices of Each Bank Over Time.

```
# Checking for missing values
print(df_akbank.isnull().sum())
```

```
Date              0
Price             0
Open              0
High              0
Low               0
Vol.              0
Change %          0
spx_close        83
brent_oil_close  83
crude_oil_close  83
ishares_close    83
usdtry_close     84
dtype: int64
```

```
# Droping the rows with missing values
df_akbank.dropna(inplace=True)
```

```
# Checking for duplicate rows
duplicate_rows = df_akbank.duplicated()
print(duplicate_rows.sum())
```

```
0
```

```
# Changing 'Vol.' column

df_akbank['Vol.'] = (
    df_akbank['Vol.']
    .str.replace('M', '', regex=False)   # Remove 'M'
    .astype(float)
    .mul(1_000_000)
    .astype(int)
)

print(df_akbank[['Vol.']].head())
```

```
       Vol.
0  59990000
1  44040000
2  50350000
3  52430000
4  45430000
```

```
# Changing 'Change %' column

df_akbank['Change %'] = (
    df_akbank['Change %']
    .str.replace('%', '', regex=False)  # Remove '%'
    .astype(float)
)

print(df_akbank[['Change %']].head())
```

```
   Change %
0     -0.68
1     -0.68
2      1.03
3      0.69
4     -2.43
```

Figure 13: Checking for Missing Values and Modifying Certain Columns.

5.2 Feature Engineering

```
df_akbank['Price_change'] = df_akbank['Price'] - df_akbank['Open']  #Create a new feature 'Price_change'
df_akbank['Returns'] = df_akbank['Price'].pct_change()              #Create a new feature 'Returns'
df_akbank['Average_price'] = (df_akbank['Price'] + df_akbank['Open']) / 2         #Create a new feature 'Average_price'
df_akbank['Price_range'] = df_akbank['High'] - df_akbank['Low']      #Create a new feature 'Price_range'
df_akbank['Volume_change'] = df_akbank['Vol.'].diff()               #Create a new feature 'Volume_change'


## Features related to date
df_akbank['Date'] = pd.to_datetime(df_akbank['Date'])
# Year, Month, day
df_akbank['year'] = df_akbank['Date'].dt.year
df_akbank['month'] = df_akbank['Date'].dt.month
df_akbank['day'] = df_akbank['Date'].dt.day


## Lag Features
df_akbank['Price_lag1'] = df_akbank['Price'].shift(1)   #Price one day ago
df_akbank['Price_lag2'] = df_akbank['Price'].shift(2)   #Price two days ago
df_akbank['Price_change_lag'] = df_akbank['Price'] - df_akbank['Price_lag1']


## Moving Average of the Closing Price
# These features calculate the moving average of the closing price over the previous 5, 10 and 20 days respectively
df_akbank['SMA_5'] = df_akbank['Price'].rolling(5).mean().shift()
df_akbank['SMA_10'] = df_akbank['Price'].rolling(10).mean().shift()
df_akbank['SMA_20'] = df_akbank['Price'].rolling(20).mean().shift()


## The Exponential Moving Average of the Closing Price
# These features calculate the exponential moving average of the closing price over the previous 5, 10 and 20 days respectively,
df_akbank['EMA_5'] = df_akbank['Price'].ewm(span=5).mean()
df_akbank['EMA_10'] = df_akbank['Price'].ewm(span=10).mean()
df_akbank['EMA_20'] = df_akbank['Price'].ewm(span=20).mean()


## MACD (Moving Average Convergence Divergence)
# 'macd': This feature calculates the difference between the 12-day and 26-day exponential moving averages of the closing price
df_akbank['EMA_12'] = df_akbank['Price'].ewm(span=12).mean()
df_akbank['EMA_26'] = df_akbank['Price'].ewm(span=26).mean()
df_akbank['Macd'] = df_akbank['EMA_12'] - df_akbank['EMA_26']              #Create a new feature 'Macd'


## Macd Signal
df_akbank['Macd_signal'] = df_akbank['Macd'].rolling(window=9).mean()         #Create a new feature 'Macd_signal'
```

Figure 14: Feature Extraction.

```
## RSI (Relative Strength Index)

# Price Difference
df_akbank['Price_diff'] = df_akbank['Price'].diff()

# Gains and Loss
df_akbank['Gain'] = df_akbank['Price_diff'].where(df_akbank['Price_diff'] > 0, 0)
df_akbank['Loss'] = -df_akbank['Price_diff'].where(df_akbank['Price_diff'] < 0, 0)

# For 14 periods average gains and loss
period = 14
df_akbank['Avg_Gain'] = df_akbank['Gain'].rolling(window=period).mean()
df_akbank['Avg_Loss'] = df_akbank['Loss'].rolling(window=period).mean()

# Rsi
df_akbank['RS'] = df_akbank['Avg_Gain'] / df_akbank['Avg_Loss']
df_akbank['RSI'] = 100 - (100 / (1 + df_akbank['RS']))
```

Figure 15: Feature Extraction (Continued).

```python
# After calculating Rsi filling NaN values with 0
df_akbank['RSI'] = df_akbank['RSI'].fillna(0)
df_akbank
```

	Date	Price	Open	High	Low	Vol.	Change %	spx_close	brent_oil_close	crude_oil_close	...	Macd	Macd_signal	Macd_histogram	Price_diff	Gain	Lo
0	2024-08-29	58.30	58.55	59.40	57.55	59990000	-0.68	5591.96	78.82	74.67	...	0.000000	NaN	NaN	NaN	0.00	-0
1	2024-08-28	58.70	59.15	59.35	58.10	44040000	-0.68	5592.18	77.58	73.44	...	0.008974	NaN	NaN	0.40	0.40	-0
2	2024-08-27	59.10	58.55	59.40	57.70	50350000	1.03	5625.80	78.66	74.46	...	0.023839	NaN	NaN	0.40	0.40	-0
3	2024-08-26	58.50	58.65	59.65	57.85	52430000	0.69	5616.84	80.36	76.17	...	0.008353	NaN	NaN	-0.60	0.00	0
4	2024-08-23	58.10	59.50	59.90	57.35	45430000	-2.43	5634.61	78.15	73.93	...	-0.016155	NaN	NaN	-0.40	0.00	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2501	2014-09-08	5.59	5.57	5.63	5.55	17030000	0.72	2001.54	100.20	92.66	...	0.080968	0.014489	0.066479	0.23	0.23	-0
2502	2014-09-05	5.55	5.57	5.60	5.53	16710000	-0.72	2007.71	100.82	93.29	...	0.101830	0.030003	0.071826	-0.04	0.00	0
2503	2014-09-04	5.59	5.47	5.59	5.46	33630000	2.01	1997.65	101.83	94.45	...	0.120205	0.045362	0.074842	0.04	0.04	-0
2504	2014-09-03	5.48	5.47	5.53	5.45	26430000	0.74	2000.72	102.77	95.54	...	0.124456	0.059360	0.065096	-0.11	0.00	0
2505	2014-09-02	5.44	5.46	5.50	5.42	22050000	-0.37	2002.28	100.34	92.88	...	0.123178	0.072842	0.050336	-0.04	0.00	0

2423 rows × 41 columns

Figure 16: Final Overview of the Bank Dataset.

Similar data cleaning and feature extraction processes were applied to all bank datasets. The figures display examples from the Akbank dataset. In the first phase of the implementation, each bank's dataset, consisting of 41 variables, was used. For the second phase, the sentiment confidence variable was added, resulting in datasets with a total of 42 variables, which were utilized in this phase.

```python
# Merging two datasets

# Convert 'Date' columns to datetime format
df_halkbank['Date'] = pd.to_datetime(df_halkbank['Date'])
df_kap_halk['date'] = pd.to_datetime(df_kap_halk['date'])

# Adding the Sentiment_Confidence column from df_kap_akbank to the df_akbank dataset
# If there is no match on that date, we throw 0
df_halkbank['Sentiment_Confidence'] = df_halkbank['Date'].map(
    df_kap_halk.set_index('date')['Sentiment_Confidence'].to_dict()).fillna(0)
```

Figure 17: Merging the Sentiment Confidence Column with Bank Datasets.

```
print(df_akbank)

            Date  Price   Open   High    Low       Vol.  Change %  spx_close  \
0     2024-08-01  63.20  62.90  64.85  62.85   60900000      2.10    5446.68
1     2024-07-31  61.90  62.70  63.55  61.45   66379999     -1.35    5522.30
2     2024-07-30  62.75  64.50  65.00  62.50   58540000     -2.94    5436.44
3     2024-07-29  64.65  64.70  66.70  64.50   48240000     -1.52    5463.54
4     2024-07-26  65.65  65.75  66.50  64.95   50470000      0.15    5459.10
...          ...    ...    ...    ...    ...        ...       ...        ...
2398  2014-09-08   5.59   5.57   5.63   5.55   17030000      0.72    2001.54
2399  2014-09-05   5.55   5.57   5.60   5.53   16710000     -0.72    2007.71
2400  2014-09-04   5.59   5.47   5.59   5.46   33630000      2.01    1997.65
2401  2014-09-03   5.48   5.47   5.53   5.45   26430000      0.74    2000.72
2402  2014-09-02   5.44   5.46   5.50   5.42   22050000     -0.37    2002.28

      brent_oil_close  crude_oil_close  ...  Macd_signal  Macd_histogram  \
0               78.95            75.40  ...    -0.420530        0.656066
1               80.84            76.84  ...    -0.334160        0.815404
2               78.07            73.88  ...    -0.201000        0.924845
3               79.05            74.80  ...    -0.010813        1.045157
4               80.28            76.00  ...     0.209179        1.123963
...               ...              ...  ...          ...             ...
2398           100.20            92.66  ...     0.014489        0.066479
2399           100.82            93.29  ...     0.030003        0.071826
2400           101.83            94.45  ...     0.045362        0.074842
2401           102.77            95.54  ...     0.059360        0.065096
2402           100.34            92.88  ...     0.072842        0.050336

      Price_diff  Gain  Loss  Avg_Gain  Avg_Loss        RS        RSI  \
0           3.00  3.00 -0.00  0.932143  0.642857  1.450000  59.183673
1          -1.30  0.00  1.30  0.878571  0.735714  1.194175  54.424779
2           0.85  0.85 -0.00  0.914286  0.735714  1.242718  55.411255
3           1.90  1.90 -0.00  1.050000  0.492857  2.130435  68.055556
4           1.00  1.00 -0.00  1.014286  0.492857  2.057971  67.298578
...          ...   ...   ...       ...       ...       ...        ...
2398        0.23  0.23 -0.00  0.062857  0.011429  5.500000  84.615385
2399       -0.04  0.00  0.04  0.060000  0.014286  4.200000  80.769231
2400        0.04  0.04 -0.00  0.057143  0.014286  4.000000  80.000000
2401       -0.11  0.00  0.11  0.050714  0.022143  2.290323  69.607843
2402       -0.04  0.00  0.04  0.050714  0.022857  2.218750  68.932039

      Sentiment_Confidence
0                 0.964382
1                 0.000000
2                 0.000000
3                 0.946517
4                 0.000000
...                    ...
2398              0.984091
2399              0.000000
2400              0.000000
2401              0.000000
2402              0.000000

[2403 rows x 42 columns]
```

Figure 18: Final Overview of the Bank Dataset with the Sentiment Confidence Column.

5.3 Train Test Splitting and Modelling

```python
# Train start-end dates and Test start-end dates
train_start_date = "2014-09-01"
train_end_date = "2021-09-01"
test_start_date = "2021-09-01"
test_end_date = "2024-09-01"

# Dates as datetime format
df_akbank['Date'] = pd.to_datetime(df_akbank['Date'])

# Train and Test Split
train_data = df_akbank[(df_akbank['Date'] >= train_start_date) & (df_akbank['Date'] < train_end_date)].copy()
test_data = df_akbank[(df_akbank['Date'] >= test_start_date) & (df_akbank['Date'] <= test_end_date)].copy()

fig = go.Figure()

# Traind and Test Data on Graph
fig.add_trace(go.Scatter(x=train_data['Date'], y=train_data['Price'], name='Training'))
fig.add_trace(go.Scatter(x=test_data['Date'], y=test_data['Price'], name='Test'))

fig.update_layout(title='Train and Test Data', xaxis_title='Date', yaxis_title='Price')
fig.show()

# Dates as Index
train_data.set_index('Date', inplace=True)
test_data.set_index('Date', inplace=True)

# Checking Dates
print("Train Data Dates:", train_data.index.min(), "-", train_data.index.max())
print("Test Data Dates:", test_data.index.min(), "-", test_data.index.max())
```

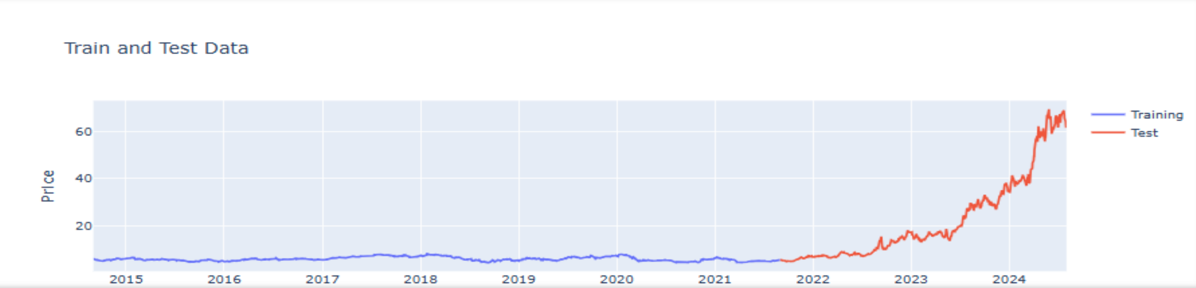

Figure 19: Train-Test Split of Bank Datasets.

```python
# Normalization on Price column
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train_data[['Price']])
test_scaled = scaler.transform(test_data[['Price']])
```

```python
# Separeting time series as X (input) and y (output)
def create_dataset(data, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step):
        X.append(data[i:(i + time_step), 0])
        y.append(data[i + time_step, 0])
    return np.array(X), np.array(y)


time_step = 30 # Looking at 30 days before to prediction

X_train, y_train = create_dataset(train_scaled, time_step)          # Train and Test
X_test, y_test = create_dataset(test_scaled, time_step)

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)    # Reshaping train and test
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
```

```
X_train shape: (1669, 30, 1)
X_test shape: (674, 30, 1)
```

```python
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout

model = Sequential()        # Model building
model.add(LSTM(units=50, return_sequences=False, input_shape=(X_train.shape[1], 1)))   # Lstm Layer
model.add(Dropout(0.2))     # Dropout Layer
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')  # Compiling the model

model.summary()    # Model Summary
```

Figure 19: Normalization of the Price Column and Modelling.

```python
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=10, mode='min', restore_best_weights=True)    # Early stop

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

Figure 20: Training the Model.

5.4 Model Results

```python
# Prediction on test data
test_predictions = model.predict(X_test)

# Predictions transform to previous scale
test_predictions = scaler.inverse_transform(test_predictions)
y_test_actual = scaler.inverse_transform(y_test.reshape(-1, 1))

# Visualization of results
plt.figure(figsize=(10,4))
plt.plot(test_data.index[time_step:], y_test_actual, color='skyblue', label='Actual Price')
plt.plot(test_data.index[time_step:], test_predictions, color='mediumpurple', label='Predicted Price')
plt.title('Akbank Stock Price Prediction')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```

```
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```

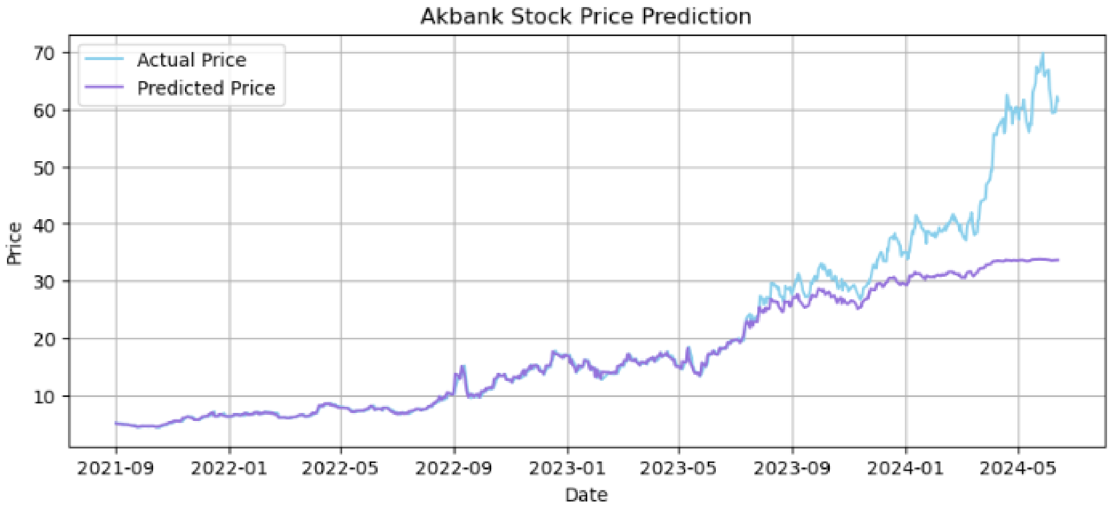

Figure 20: Actual vs. Predicted Prices Using LSTM.

```python
# RMSE (Root Mean Squared Error)
rmse = math.sqrt(mean_squared_error(y_test_actual, test_predictions))
print("Test RMSE:", rmse)

# R2 (R-Squared)
r2 = r2_score(y_test_actual, test_predictions)
print("Test R²:", r2)
```

```
Test RMSE: 7.736845964414174
Test R²: 0.7531285841453703
```

Figure 21: RMSE and R-Squared Results.