National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analysis

Soumiya Kanwar
Student ID: X21218323

School of Computing
National College of Ireland

Supervisor: Abid Yaqoob

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Soumiya Kanwar……

Student ID: ……x21218323……

Programme: ……Msc in Data Analytics…… **Year:** …2023……..

Module: ……Research project……

Lecturer: …… Abid Yaqoob ……

Submission Due Date: ……14 dec 2023……

Project Title: … Comparative Analysis of Machine Learning Models for S&P 500 Prediction ……

Word Count: ……768…… **Page Count:** ……10……

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: ……Soumiya Kanwar……

Date: ……14/12/2023……

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

Soumiya Kanwar
Student ID: x21218323

1 Introduction

This project of Comparative Analysis of Machine Learning Models for S&P 500 Prediction involved using Python in building deep learning techniques and machine learning algorithms. This handbook spells out the specification of the system and the way it may be applied.

2 Hardware and Software Configuration

The System requirements that are required on the host device to implement this research is shown in Figure 1.

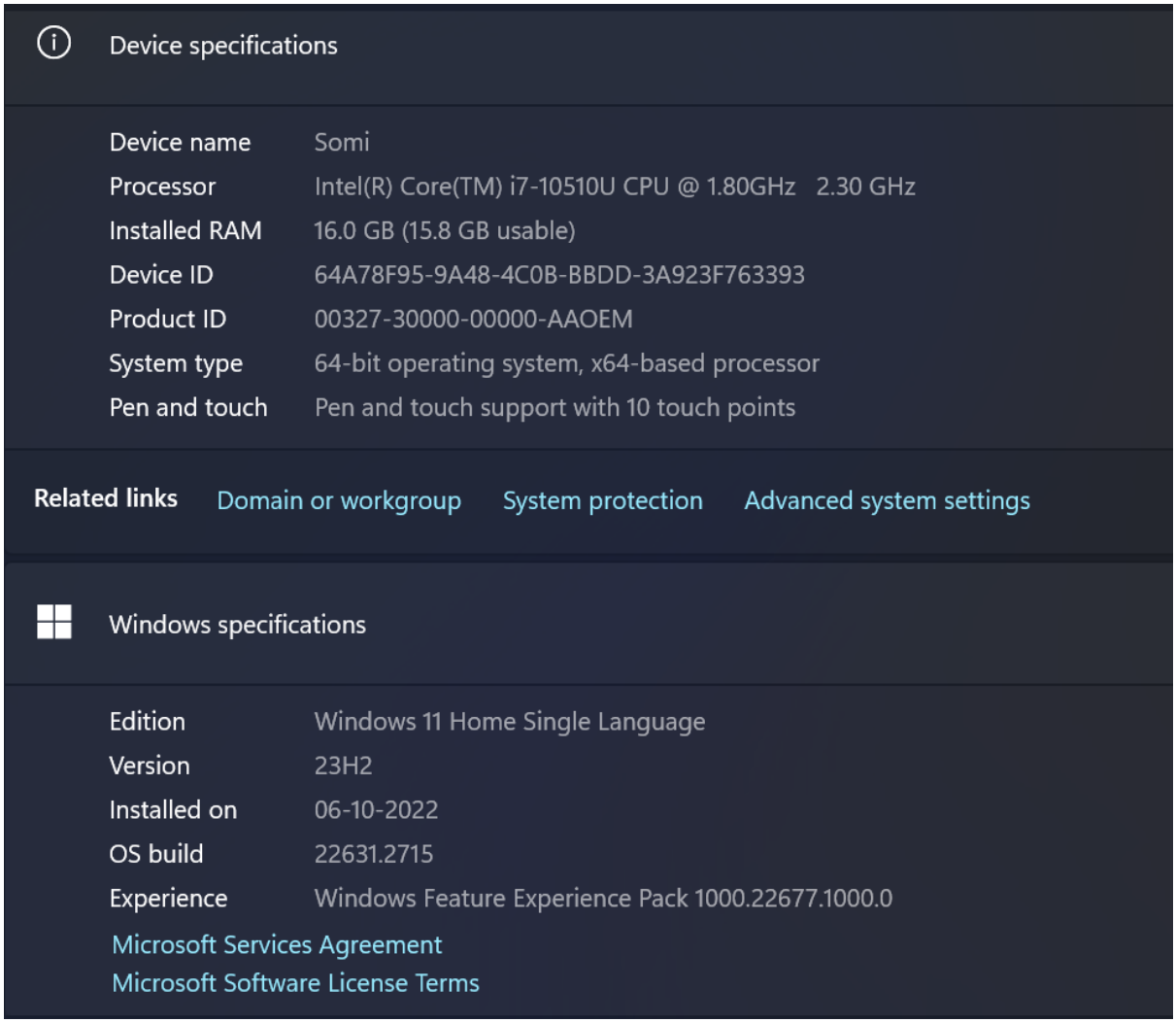

Figure 1: System Configuration

Python 3.10.9 is used for implementation. All the frameworks and libraries required are given below in Table 1.

IDE	Anaconda Navigator() and Jupiter Notebook
Computation	GPU
Number of GPU	1
Programming language	Python
Modues	Pandas, numpy, matplotlib, seaborn, yfinance, sklearn
Framework	tensorflow

Table 1: Setup configuration

To access the Jupiter Notebook, you must install Anaconda navigator. You can install it from https://docs.anaconda.com/free/navigator/index.html. To open the Jupiter notebook, you can navigate to Jupiter notebook app. It will open a new tab or window in your browser.

For this project we have used 6.5.2 versions of Jupiter Notebook. As in Figure 2

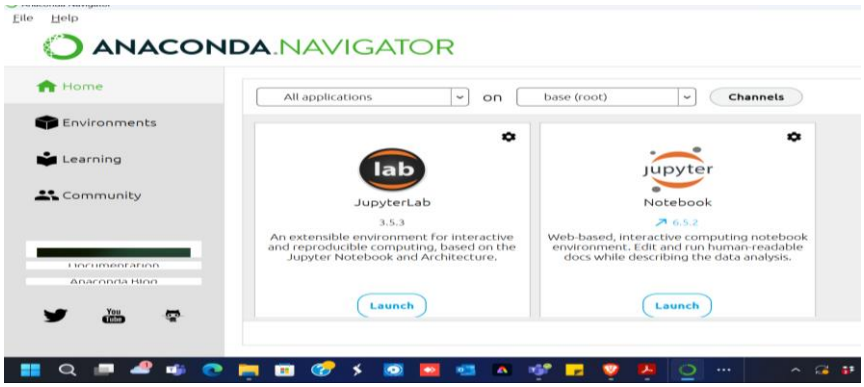

Figure 2: Anaconda Navigator

3 Dataset

We have used a library yfinance to import our data (Figure 3).

```
# Load the dataset
sp500 = yf.Ticker("^GSPC")
```

```
# Fetching the historical stock data for the S&P 500 with the period set to "max"
sp500 = sp500.history(period="max")
sp500.head()
```

...

```
#Deleting the "Dividends" and "Stock Splits" column from the DataFrame
del sp500["Dividends"]
del sp500["Stock Splits"]
sp500
```

Figure 3: Data loading

The data set is divided into two distance timeline the first one is covering from 2000 to present and the second one is covering from the recession. From 2007 to 2009.

We have two different code files:The first one is the data from 2000 to the present and the other is from 2007 to 2009 that is the recession peroid. (Figure 4 and 5)

```
# Using .loc to filter the DataFrame to include only rows from January 1, 2000, onwards
sp500 = sp500.loc["2000-01-01":].copy()
sp500
```

Figure 4: Data from 2000

```
sp500 = sp500.loc["2007-01-01":"2009-12-31"].copy()
sp500
```

Figure 5: Data from 2000

4 Exploratory Data Analysis

Exploratory Data Analysis has done on both the datasets. Including descriptive statistics as it could give us better understanding of the data. We can see the descriptive statistics for both the datasets in Figure 6, 7 and 8.

```
sp500.index
```

...

```
sp500.info
```

...

```
# Descriptive statistics
sp500.describe()
```

...

```
# Visualize the distribution of 'Close' prices over time
plt.figure(figsize=(12, 6))
sns.lineplot(x='Date', y='Close', data=sp500)
plt.title('Closing Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.show()
```

Figure 5: Exploratory Data Analysis

	Open	High	Low	Close	Volume
count	6025.000000	6025.000000	6025.000000	6025.000000	6.025000e+03
mean	1967.789252	1979.423061	1955.286397	1968.061445	3.325329e+09
std	1054.895258	1060.225735	1049.332851	1055.138051	1.508857e+09
min	679.280029	695.270020	666.789978	676.530029	3.560700e+08
25%	1191.170044	1198.479980	1184.160034	1191.329956	2.082360e+09
50%	1456.630005	1464.939941	1446.060059	1456.630005	3.442760e+09
75%	2577.750000	2586.500000	2565.939941	2579.850098	4.159470e+09
max	4804.509766	4818.620117	4780.040039	4796.560059	1.145623e+10

Figure 7: Descriptive statistics(2000-present)

	Open	High	Low	Close	Volume
count	756.000000	756.000000	756.000000	756.000000	7.560000e+02
mean	1214.998584	1225.432288	1203.390212	1214.750793	4.613678e+09
std	253.106839	251.246626	254.737279	252.949807	1.575373e+09
min	679.280029	695.270020	666.789978	676.530029	1.219310e+09
25%	953.327515	979.697479	943.437500	954.457520	3.387888e+09
50%	1288.234985	1300.164978	1274.859985	1288.664978	4.381810e+09
75%	1443.935028	1450.224945	1433.212463	1444.347504	5.632375e+09
max	1564.979980	1576.089966	1555.459961	1565.150024	1.145623e+10

Figure 8: Descriptive statistics (2007-2009)

5 Implementation

5.1 Long short-term Memory

A Sequential model using Keras is defined and returned by the create_lstm_model function. The model consists of two layers: a single unit sigmoid activation function after a fifth layer of LSTM containing 50 units, with a ReLU activation function preceding it. According to Figure 9.

```python
 # Creating LSTM
def create_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(units=50, input_shape=input_shape, activation='relu'))
    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

```python
# Preprocess data for LSTM
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(sp500[["Close", "Volume", "Open", "High", "Low"]])
```

```python
# Create sequences for LSTM
sequence_length = 10
sequences = []
targets = []
```

```python
for i in range(len(scaled_data) - sequence_length):
    sequence = scaled_data[i : (i + sequence_length)]
    target = sp500["Target"].values[i + sequence_length]
    sequences.append(sequence)
    targets.append(target)
```

Figure 9: creating LSTM model.

The Model was Trained on different epochs. (Figure 10)

```python
# Train LSTM model
lstm_model = create_lstm_model(input_shape=(X_train.shape[1], X_train.shape[2]))
lstm_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

...

```python
# Evaluate LSTM model
lstm_scores = lstm_model.evaluate(X_test, y_test)
print(f"LSTM Model - Loss: {lstm_scores[0]}, Accuracy: {lstm_scores[1]}")
```

...

```python
# Make predictions with LSTM model
lstm_preds = (lstm_model.predict(X_test) > 0.5).astype(int)
```

Figure 10: Training model (epochs = 10)

5.2 Random Forest Classifier and Logistic Regression

```
model = rfc(n_estimators=200, min_samples_split=50, random_state=1)
```

...

```
train = sp500.iloc[:-100]
test = sp500.iloc[-100:]
```

```
predictors = ["Close","Volume", "Open", "High","Low"]
model.fit(train[predictors], train["Target"])
```

...

```
preds= model.predict(test[predictors])
```

```
preds = pd.Series(preds, index=test.index)
preds
```

...

```
precision_score(test["Target"], preds)
```

...

```
combined = pd.concat ([test["Target"], preds], axis=1)
combined.plot()
```

Figure 11. Traning Random Forest Classifier (2000- present)

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=1)
```

```
model.fit(train[predictors], train["Target"])
```

...

```
preds= model.predict(test[predictors])
```

```
preds = pd.Series(preds, index=test.index)
preds
```

...

```
precision_score(test["Target"], preds)
```

...

```
combined = pd.concat ([test["Target"], preds], axis=1)
combined.plot()
```

```
: <Axes: xlabel='Date'>
```

Figure 12. Traning Logistic Regression

```
train = sp500.iloc[:-75]
test = sp500.iloc[-75:]
```

Figure 13.Creating training and testing datasets

Figure 14 shows, predict function accepts the training set, the test set, the predictors list, and a machine learning algorithm. It fits the model on training data, predicts on test data, and then returns a DataFrame with the actual targets and predictions of interest for additional investigation.

```
def predict(train, test, predictors, model):
    model.fit(train[predictors],train["Target"])
    preds = model.predict(test[predictors])
    preds = pd.Series(preds, index = test.index, name="Predictions")
    combined = pd.concat ([test["Target"], preds], axis=1)
    return combined
```

Figure 14: Define predict function

Figure 15 and 16 shows the backtest function performs a rolling-window backtest on a time series dataset using a machine learning model, generating predictions for specified predictors at intervals defined by the start index, step size, and concatenating the results into a Pandas DataFrame.

```
def backtest(data, model, predictors, start =250, step=25):
    all_predictions = []

    for i in range(start, data.shape[0], step):
        train = data.iloc[0:i].copy()
        test = data.iloc[i:(i+step)].copy()
        predictions = predict(train, test, predictors, model)
        all_predictions.append(predictions)
    return pd.concat(all_predictions)
```

Figure 15 : backtest funtion for (2007-2009)

```
def backtest(data, model, predictors, start =2500, step=250):
    all_predictions = []

    for i in range(start, data.shape[0], step):
        train = data.iloc[0:i].copy()
        test = data.iloc[i:(i+step)].copy()
        predictions = predict(train, test, predictors, model)
        all_predictions.append(predictions)
    return pd.concat(all_predictions)
```

Figure 16: backtest funtion for (2000 – present)

Figure 17 and 18 state that, the data under the “close” column is used to ratio against target-divided columns with rolling average/trends measured. These are additional features or information associated with raw data that highlight average values over a particular timescale.

```
# Calucating Mean
horizons = [2,5,60,250,1000]
new_predictors = []

for horizon in horizons:
    rolling_averages = sp500.rolling(horizon).mean()

    ratio_column = f"Close_Ratio_{horizon}"
    sp500[ratio_column] = sp500["Close"] / rolling_averages["Close"]

    trend_column = f"Trend_{horizon}"
    sp500[trend_column] = sp500.shift(1).rolling(horizon).sum()["Target"]

    new_predictors += [ratio_column, trend_column]
```

```
sp500=sp500.dropna()
sp500
```

Figure 17: Calucating Mean (2000 – present)

```
# Calucating Mean
horizons = [2,5,7,10,14]
new_predictors = []

for horizon in horizons:
    rolling_averages = sp500.rolling(horizon).mean()

    ratio_column = f"Close_Ratio_{horizon}"
    sp500[ratio_column] = sp500["Close"] / rolling_averages["Close"]

    trend_column = f"Trend_{horizon}"
    sp500[trend_column] = sp500.shift(1).rolling(horizon).sum()["Target"]

    new_predictors += [ratio_column, trend_column]
```

Figure 18: Calucating Mean (2007 –2009)

The function predict trains a machine learning model on the training data, makes predictions on the test data using specified predictors, converts predicted probabilities into binary predictions based on a threshold of 0.5, 0.6 and 0.7 creates a Pandas Series with the predictions, and returns a DataFrame combining the actual target values from the test data with the predicted values.as we can see in Figure 10.

```python
def predict(train, test, predictors, model):
    model.fit(train[predictors],train["Target"])
    preds = model.predict_proba(test[predictors]) [:,1]
    preds[preds>= .5] = 1
    preds[preds< .5] = 0
    preds = pd.Series(preds, index = test.index, name="Predictions")
    combined = pd.concat ([test["Target"], preds], axis=1)
    return combined
```

Figure 19: define predict function (thershold = 0.5)

Note: For Logistic recession model simalier steps have been taken as Random Forest Classifier

6 Conclusion

In order to execute the entire code in Jupyter, Just click on Run All from the Cell Menu. As shown in the Figure 20.

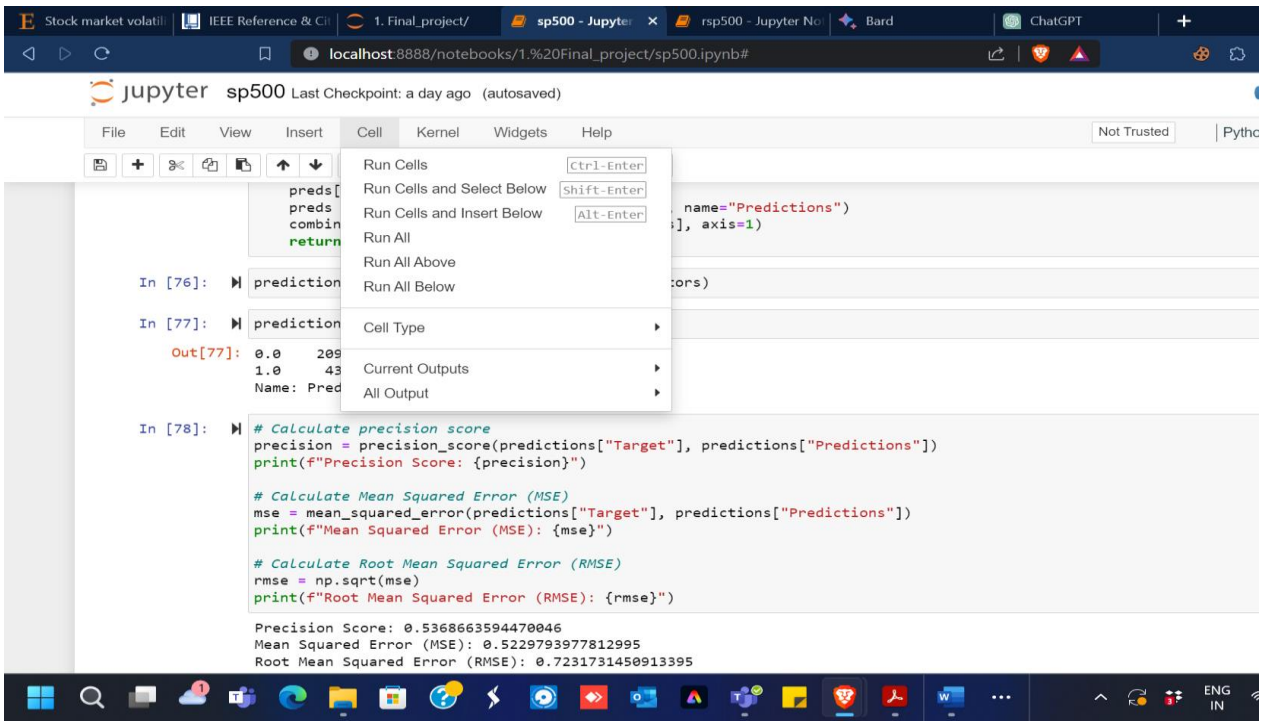

Figure 20: Jupiter File Menu

7 Exploratory Data Analysis Visualization

The line graph of the closing charge over the years is shown in Figures 21 and 22. As we will see, prices commenced to fall in the middle of 2007 and reached rock bottom within the middle of 2009.

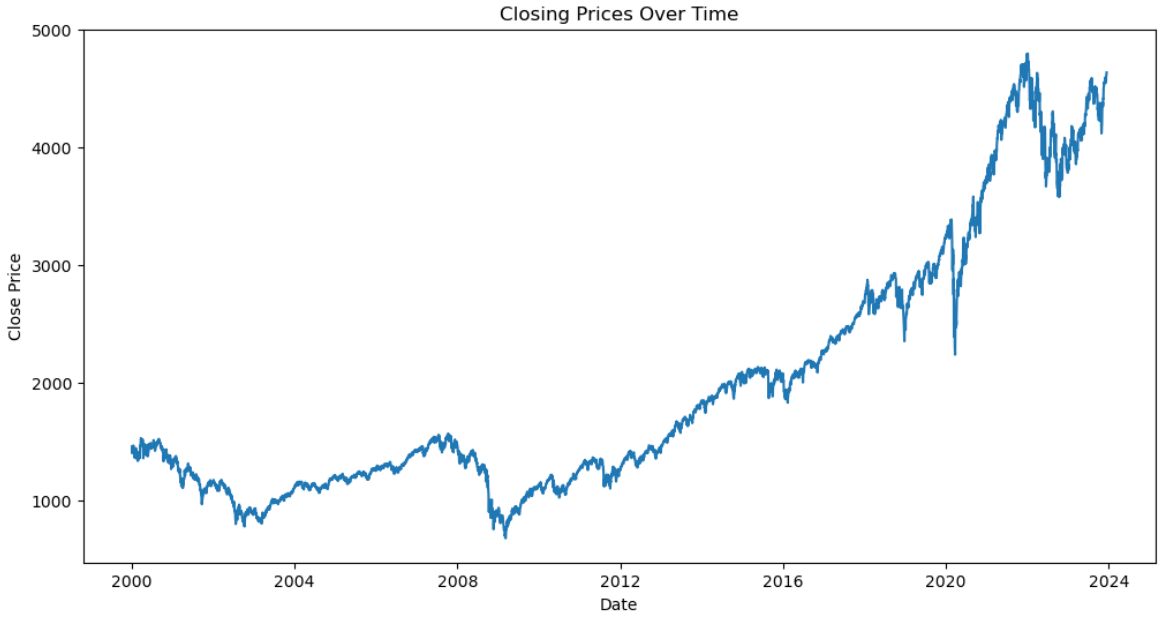

Figure 21: Closing price over time (2000-present)

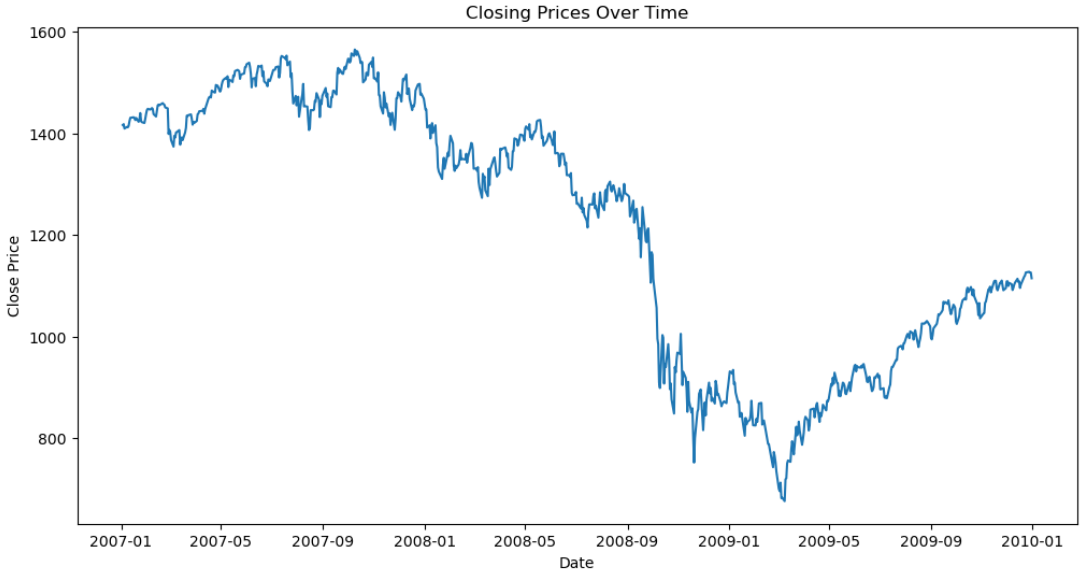

Figure 22: Closing price over time (2007-2009)