Sri Lanka Institute of Information Technology

Assignment – 01

Machine Learning (IT4060)

2023

Stock Price Prediction using Different Machine Learning Models and Compare the Performance of the Models

Submitted by:

Student ID	Student Name
IT20610098	Madhuwantha M.G. P
IT20613204	D.R.G.C.W Kodithuwakku
IT20613136	Jayarathne A.H. B
IT20609030	Karunanayake M. L

Table of Contents

1. **DESCRIPTION OF THE PROBLEM ADDRESSED:**

Stock price forecasting stands out as a favored and compelling topic within the scientific community, yet it presents distinct challenges compared to other forecasting domains. Despite concerted efforts by scholars and industry experts across various disciplines such as Computer Science, Economics, Business Arithmetic, and Marketing, achieving consensus on stock market price predictions remains elusive. The erratic nature of stock prices, often characterized by abrupt fluctuations akin to random walk behavior, complicates forecasting endeavors significantly. These unpredictable movements pose a substantial hurdle for analysts seeking to develop efficient and accurate forecasting models.

Nonetheless, the pursuit of effective stock market forecasting models persists due to the substantial implications for managers, investors, and decision-makers. By leveraging advanced machine learning techniques, analysts delve into extensive historical data to discern underlying patterns and trends that may provide insights into future stock price movements. This involves meticulous analysis of both current and past information to identify the most suitable predictive models. While the challenges are considerable, ongoing research efforts continue to refine methodologies, aiming to enhance decision-making processes and optimize investment strategies in the dynamic realm of stock markets.

1.1 What is the Stock Market?

The stock market is a public arena where people exchange shares, which represent ownership in publicly listed corporations. These shares, often known as stocks, are purchased and sold on stock exchanges, which facilitate these transactions. Essentially, investing in the stock market means acquiring a stake in a firm. The value of these ownership shares can fluctuate owing to a variety of causes, making it a dynamic and significant component within the larger framework of the global financial system.

1.2 Importance of Stock Market:

The stock market is extremely important in the financial world for a variety of compelling reasons. For starters, it provides a key platform for businesses to obtain funds by providing shares to investors, allowing them to expand and improve their operations. Second, it provides a major opportunity for individuals to possibly build personal wealth through the growth of their stock assets. Furthermore, stock markets are important indices of economic vitality, providing insights into the general state of the economy. Furthermore, they make it possible for individuals to invest in firms that are primed for development, allowing them to share in the future success of these businesses.

1.3 Stock Price Prediction:

Utilizing machine learning to forecast stock prices entails the endeavor to unveil the future worth of company stocks and other financial assets traded on stock exchanges. The overarching aim of such predictions is to realize substantial profits, though it presents an inherently formidable challenge. Stock price prognostication is subject to an array of influences, encompassing tangible and intangible factors such as physical and psychological elements, alongside rational and irrational investor behavior. These intricate dynamics contribute to the volatility inherent in the stock market, rendering the accurate prediction of stock prices a notably challenging feat. Nonetheless, amidst these hurdles, machine learning methodologies provide indispensable tools for endeavoring to make informed predictions within this intricate and ever-evolving financial milieu.

2. DATASET

2.1 Description of the Dataset

The "yahoo_finance_dataset(2018-2023)" represents a treasure trove of financial data, offering a detailed view of daily stock market activity from April 1, 2018, to March 31, 2023. With a substantial dataset comprising 1257 rows and 7 columns, it provides a wealth of insights into various financial instruments like equities, ETFs, and indexes. Sourced meticulously from Yahoo Finance, its primary aim is to empower researchers, analysts, and investors with the tools they need to navigate the complexities of the stock market effectively.

This data set caters to a broad spectrum of financial analysis tasks, from predicting stock prices to analyzing trends, optimizing portfolios, and managing risks. Its accessibility in XLSX format ensures seamless integration into popular data analysis platforms like Python, R, and Excel, making it a versatile resource for professionals across the financial landscape. At its core, the dataset comprises seven essential columns, each offering valuable insights into daily market dynamics: Date, Open, High, Low, Close, Adj Close, and Volume. These attributes provide a comprehensive view of asset performance, capturing opening and closing prices, intraday trading ranges, adjusted closing prices factoring in corporate actions, and trading volumes. In essence, it serves as a valuable compass for informed decision-making amid the ever-changing currents of financial markets.

Description	This dataset is contained in kaggle
Data Set Link	https://www.kaggle.com/datasets/suruchiarora/yahoo-finance-dataset-2018-2023
Size of Data	1257 *7
Related Task	Stock Market Prediction
Data Set Stored In	kaggle

2.2 Data Set Parameters

Characteristic	Information
Number of columns	7
Number of rows	1257
Columns	Date, Open, High, Low, Close, Volume, Adj Close

2.3 Sample Images of the Dataset

	Date	Open	High	Low	Close	Adj Close	Volume	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Returns
2	2023-04-26	33596.34	33645.83	33235.85	33301.87	33301.87	321170000	0	1	1	-0.015500
3	2023-04-25	33828.34	33875.49	33525.39	33530.83	33530.83	297880000	0	0	1	0.006875
4	2023-04-24	33805.04	33891.15	33726.09	33875.40	33875.40	252020000	1	0	0	0.010276
5	2023-04-21	33793.60	33858.83	33688.57	33808.96	33808.96	291080000	1	0	0	-0.001961
6	2023-04-20	33740.60	33875.39	33677.74	33786.62	33786.62	307910000	0	1	1	-0.000661
...	...	...	...	...	...	...	...	...	...	...	...
1253	2018-05-07	24317.66	24479.45	24263.42	24357.32	24357.32	307670000	1	0	0	-0.000119
1254	2018-05-04	23865.22	24333.35	23778.87	24262.51	24262.51	329480000	1	0	0	-0.003892
1255	2018-05-03	23836.23	23996.15	23531.31	23930.15	23930.15	389240000	0	1	0	-0.013699
1256	2018-05-02	24097.63	24185.52	23886.30	23924.98	23924.98	385350000	0	1	1	-0.000216
1257	2018-05-01	24117.29	24117.29	23808.19	24099.05	24099.05	380070000	0	0	0	0.007276

1256 rows × 11 columns

```
#View Data Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1256 entries, 2 to 1257
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Date               1256 non-null   datetime64[ns]
 1   Open               1256 non-null   float64
 2   High               1256 non-null   float64
 3   Low                1256 non-null   float64
 4   Close              1256 non-null   float64
 5   Adj Close          1256 non-null   float64
 6   Volume             1256 non-null   int64
 7   Increase_Decrease  1256 non-null   int64
 8   Buy_Sell_on_Open   1256 non-null   int64
 9   Buy_Sell           1256 non-null   int64
 10  Returns            1256 non-null   float64
dtypes: datetime64[ns](1), float64(6), int64(4)
memory usage: 117.8 KB
```

3. SELECTED MACHINE LEARNING ARCHITECTURES

To anticipate fluctuations in stock prices, we employ an array of machine learning models, meticulously evaluating their accuracy and loss function values to discern the most effective technique. These models encompass:

1. **Bayesian Ridge Regression**: This strategy provides a probabilistic framework for linear regression, adept at accommodating data uncertainty by leveraging probability distributions as opposed to singular estimates. It proves particularly advantageous when dealing with scant or disparately distributed data, bolstering the robustness of relationships modeled for stock price prognostication.

2. **Linear Regression**: This methodology seeks to establish correlations between two variables by fitting a linear equation to observed data. Typically, one variable acts as an explanatory factor while the other is deemed the outcome. For instance, within the realm of stock price prediction, linear regression may scrutinize historical price data in conjunction with various influencing factors.

3. **Decision Tree Regression**: Decision trees construct models in a tree-like format, fragmenting the dataset into smaller subsets predicated on feature values. Through this iterative process, decision nodes and leaf nodes emerge, furnishing insights into the intricate connections between input features and target variables.

4. **Logistic Regression**: Logistic regression is a statistical tool that predicts the likelihood of a binary event occurrence grounded on independent variables. In the context of forecasting stock prices, logistic regression might gauge the probability of a specific price movement, such as an escalation or downturn.

4. METHODOLOGY

We used TensorFlow, Keras as main framework for this and use normal python libraries like matplotlib, NumPy, pandas, sklearn to other purposes. They are an open-source machine learning library for Python, mainly developed by the Facebook AI Research team. These are the main steps we followed to build and train the models.

1. Import Libraries

2. Load data into a Data Frame

3. Data set Preprocessing

4. Dataset Cleaning and Null Value Testing

5. Analyze the Data

6. Define X and Y

7. Split Data to train and test

8. Training the Model

9. Comparison of Actual Values and Predicted Values

10. Accuracy and Loss Function Values of the Model

5. IMPLEMENTATION

5.1 Bayesian Ridge Regression Model - IT20610098 -Madhuwantha M.G.P.

Import Libraries

Import Libraries

```python
# Import Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# MATPLOTLIB & SEABORN FOR GRAPH-PLOTTING
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Load Dataset

Load Data into the Data Frame

[6]

```python
import pandas as pd

# Path to your CSV file in Google Drive
file_path = '/content/drive/MyDrive/yahoo_data.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame and format the date column
df.head().style.format({'Date': lambda x: pd.to_datetime(x).strftime('%Y-%m-%d')})

```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2023-04-28	33,797.43	34,104.56	33,728.40	34,098.16	34,098.16	354,310,000
1	2023-04-27	33,381.66	33,859.75	33,374.65	33,826.16	33,826.16	343,240,000
2	2023-04-26	33,596.34	33,645.83	33,235.85	33,301.87	33,301.87	321,170,000
3	2023-04-25	33,828.34	33,875.49	33,525.39	33,530.83	33,530.83	297,880,000
4	2023-04-24	33,805.04	33,891.15	33,726.09	33,875.40	33,875.40	252,020,000

Data Preprocessing

Data Preprocessing

```python
import pandas as pd
import numpy as np

# Assuming 'df' is the DataFrame containing your dataset

# Convert 'Date' column to datetime type
df['Date'] = pd.to_datetime(df['Date'])

# Remove commas and convert columns to numeric type
df['Open'] = df['Open'].replace(',', '', regex=True).astype(float)
df['High'] = df['High'].replace(',', '', regex=True).astype(float)
df['Low'] = df['Low'].replace(',', '', regex=True).astype(float)
df['Close'] = df['Close'].replace(',', '', regex=True).astype(float)
df['Adj Close'] = df['Adj Close'].replace(',', '', regex=True).astype(float)
df['Volume'] = df['Volume'].replace(',', '', regex=True).astype(int)

# Calculate the new columns based on the modified dataset
df['Increase_Decrease'] = np.where(df['Volume'].shift(-1) > df['Volume'], 1, 0)
df['Buy_Sell_on_Open'] = np.where(df['Open'].shift(-1) > df['Open'], 1, 0)
df['Buy_Sell'] = np.where(df['Adj Close'].shift(-1) > df['Adj Close'], 1, 0)
df['Returns'] = df['Adj Close'].pct_change()

# Drop rows with NaN values
df = df.dropna()

# Display the modified dataset
print(df.head())
```

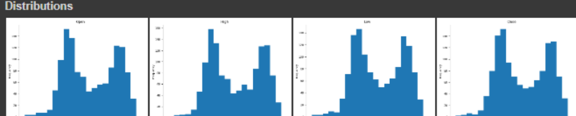
```
        Date      Open      High       Low     Close  Adj Close     Volume  \
2 2023-04-26  33596.34  33645.83  33235.85  33301.87   33301.87  321170000
3 2023-04-25  33828.34  33875.49  33525.39  33530.83   33530.83  297880000
4 2023-04-24  33805.04  33891.15  33726.09  33875.40   33875.40  252020000
5 2023-04-21  33793.60  33858.83  33688.57  33808.96   33808.96  291080000
6 2023-04-20  33740.60  33875.39  33677.74  33786.62   33786.62  307910000

   Increase_Decrease  Buy_Sell_on_Open  Buy_Sell   Returns
2                  0                 1         1 -0.015500
3                  0                 0         1  0.006875
4                  1                 0         0  0.010276
5                  1                 0         0 -0.001961
6                  0                 1         1 -0.000661
```

```python
#View Dataset
df
```

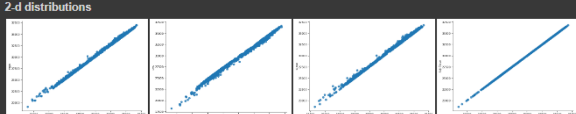
	Date	Open	High	Low	Close	Adj Close	Volume	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Returns
2	2023-04-26	33596.34	33645.83	33235.85	33301.87	33301.87	321170000	0	1	1	-0.015500
3	2023-04-25	33828.34	33875.49	33525.39	33530.83	33530.83	297880000	0	0	1	0.006875
4	2023-04-24	33805.04	33891.15	33726.09	33875.40	33875.40	252020000	1	0	0	0.010276
5	2023-04-21	33793.60	33858.83	33688.57	33808.96	33808.96	291080000	1	0	0	-0.001961
6	2023-04-20	33740.60	33875.39	33677.74	33786.62	33786.62	307910000	0	1	1	-0.000661
...	...	...	...	...	...	...	...	...	...	...	...
1253	2018-05-07	24317.66	24479.45	24263.42	24357.32	24357.32	307670000	1	0	0	-0.000119
1254	2018-05-04	23865.22	24333.35	23778.87	24262.51	24262.51	329480000	1	0	0	-0.003892
1255	2018-05-03	23836.23	23996.15	23531.31	23930.15	23930.15	389240000	0	1	0	-0.013699
1256	2018-05-02	24097.63	24185.52	23886.30	23924.98	23924.98	385350000	0	1	1	-0.000216
1257	2018-05-01	24117.29	24117.29	23808.19	24099.05	24099.05	380070000	0	0	0	0.007276

1256 rows × 11 columns

Distributions



2-d distributions



Time series

Values

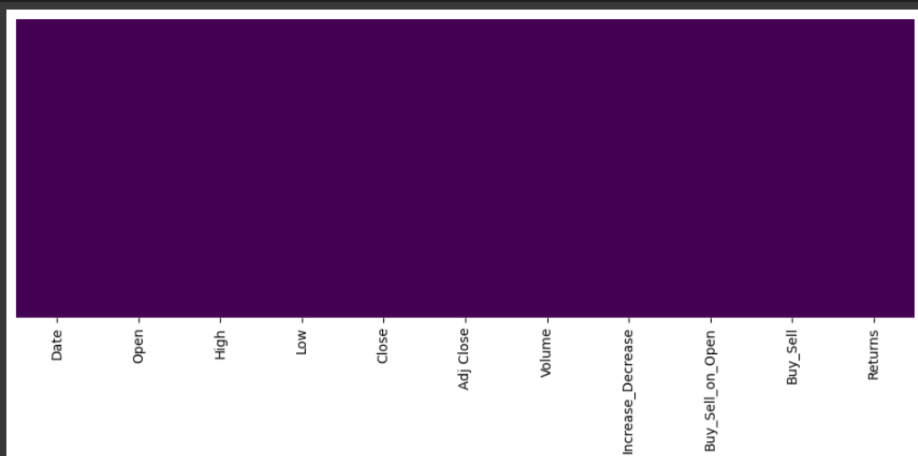
Dataset Cleaning and Null Value Testing

Dataset Cleaning and Null Value Testing

```
[ ]  1 # See how many null values in each column
     2
     3 df.isnull().sum()
```

```
Date                0
Open                0
High                0
Low                 0
Close               0
Adj Close           0
Volume              0
Increase_Decrease   0
Buy_Sell_on_Open    0
Buy_Sell            0
Returns             0
dtype: int64
```

```
[ ]  1 plt.figure(figsize=(12,4))
     2 sns.heatmap(df.isnull(), yticklabels=False,cbar=False,cmap='viridis')
     3 plt.savefig("Figure 1: Heatmap for Null Values")
```



Analyze the data

Analyze the Data

```python
# see number of rows, number of columns
df.shape
```

```
(1256, 11)
```

```python
#TOTAL NUMBER OF RECORDS
df.size
print("Total number of records = ",df.size)
```

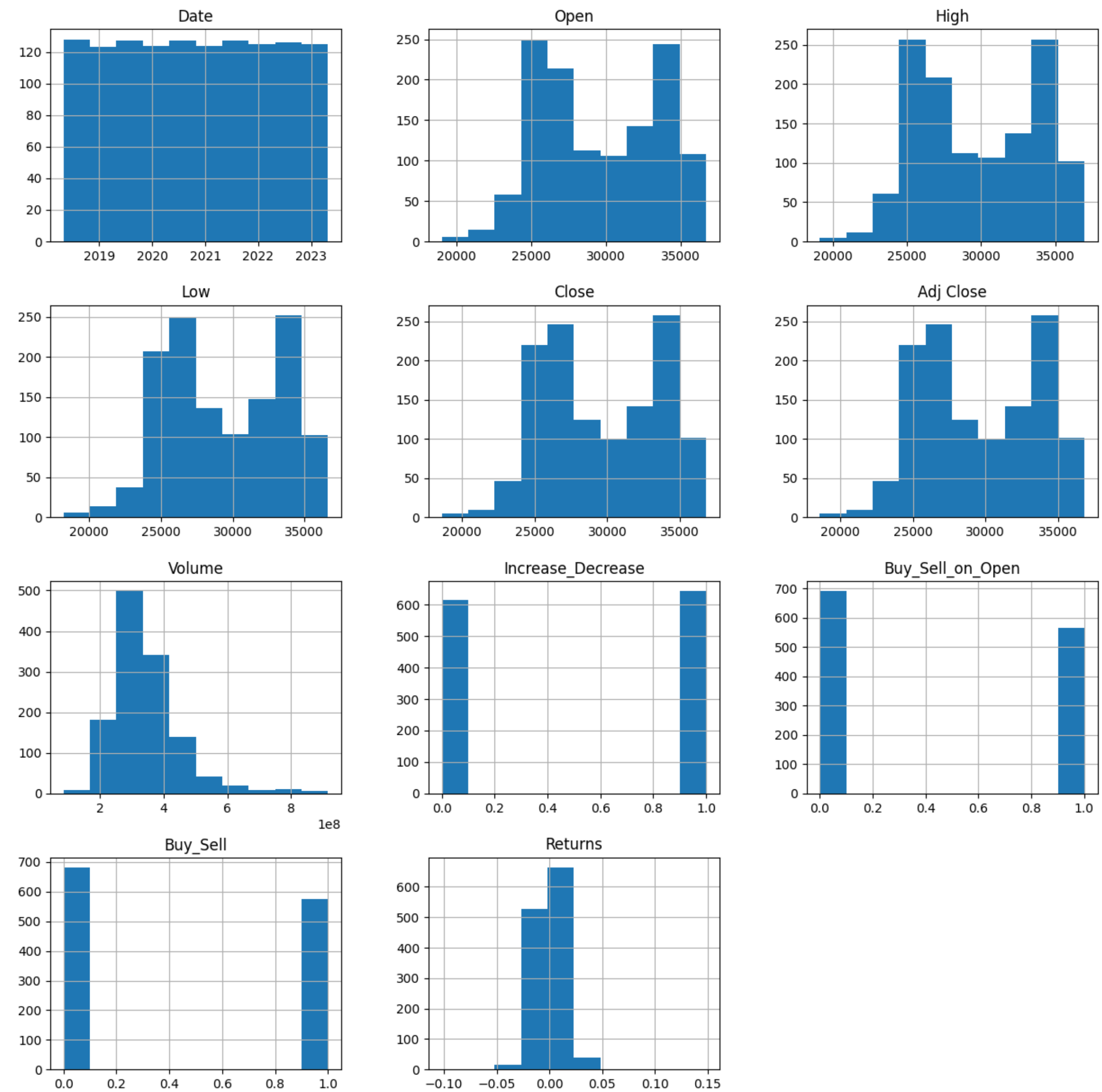
```
Total number of records =  13816
```

```python
# see columns names

df.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
       'Increase_Decrease', 'Buy_Sell_on_Open', 'Buy_Sell', 'Returns'],
      dtype='object')
```

```python
#View Data Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1256 entries, 2 to 1257
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Date               1256 non-null   datetime64[ns]
 1   Open               1256 non-null   float64
 2   High               1256 non-null   float64
 3   Low                1256 non-null   float64
 4   Close              1256 non-null   float64
 5   Adj Close          1256 non-null   float64
 6   Volume             1256 non-null   int64
 7   Increase_Decrease  1256 non-null   int64
 8   Buy_Sell_on_Open   1256 non-null   int64
 9   Buy_Sell           1256 non-null   int64
 10  Returns            1256 non-null   float64
dtypes: datetime64[ns](1), float64(6), int64(4)
memory usage: 117.8 KB
```



Define X and Y

Define X and Y

```
1 X = df['Open'].values.reshape(1257,-1)
2 y = df['Adj Close'].values.reshape(1257,-1)
```

Split Data to train and test

Split Train Data and Test Data

```
[ ]  from sklearn.model_selection import train_test_split
```

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Dataset Training and Model Training

Dataset Traning and Model Traning

```
model = BayesianRidge(compute_score=True)
model.fit(X_train, y_train)
```

```
BayesianRidge
BayesianRidge(compute_score=True)
```

```
[ ]  model.coef_
```

```
array([0.99834636])
```

```
[ ]  model.scores_
```

```
array([-8676.11227658, -6619.41180156, -6619.4117878 , -6619.4117878 ])
```

Evaluate model.

Comparison of Actual Values and Predicted Values

```
y_pred = model.predict(X_test)
```

Accuracy and Loss Function Values of the Model

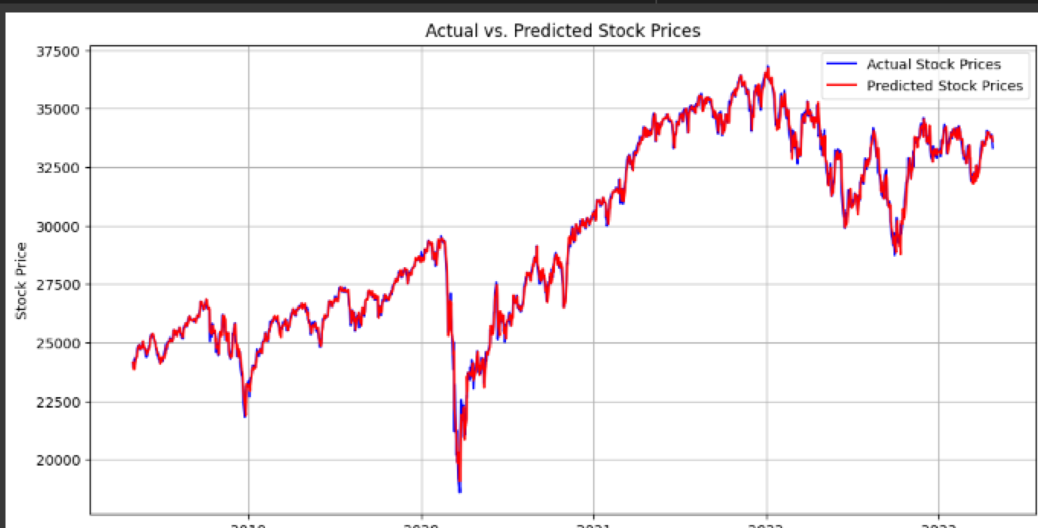
```
from sklearn import metrics
print('Mean_Absolute_Error(MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Mean_Squared_Error(MSE):', metrics.mean_squared_error(y_test, y_pred))
print('Root_Mean_Squared_Error(RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

```

```
Mean_Absolute_Error(MAE): 212.1244901111817
Mean_Squared_Error(MSE): 88499.93282967183
Root_Mean_Squared_Error(RMSE): 297.48938271755486
```

```
print('Accuracy Score:', model.score(X_test, y_test))
```

```
Accuracy Score: 0.9947976502144797
```

```
import matplotlib.pyplot as plt

# Plotting the actual stock prices
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Adj Close'], label='Actual Stock Prices', color='blue')

# Plotting the predicted stock prices for the entire dataset
plt.plot(df['Date'], model.predict(X), label='Predicted Stock Prices', color='red')

plt.title('Actual vs. Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid(True)
plt.show()

```



5.2 Linear Regression Model – IT20613204- Kodithuwakku D.R.G.C.W

Import Libraries

```
[2] import numpy as np
    import pandas as pd
    import seaborn as sns
    from sklearn import metrics
    import matplotlib.pyplot as plt
    from sklearn.linear_model import LinearRegression
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import train_test_split

    import warnings
    warnings.filterwarnings("ignore")
```

Load Dataset

Load Data into the Data Frame

```
[ ] # Path to your CSV file in Google Drive
    file_path = '/content/drive/MyDrive/yahoo_data.csv'

    # Read the CSV file into a DataFrame
    dataset = pd.read_csv(file_path)

    # Display the first few rows of the DataFrame and format the date column
    dataset.head().style.format({'Date': lambda x: pd.to_datetime(x).strftime('%Y-%m-%d')})
```

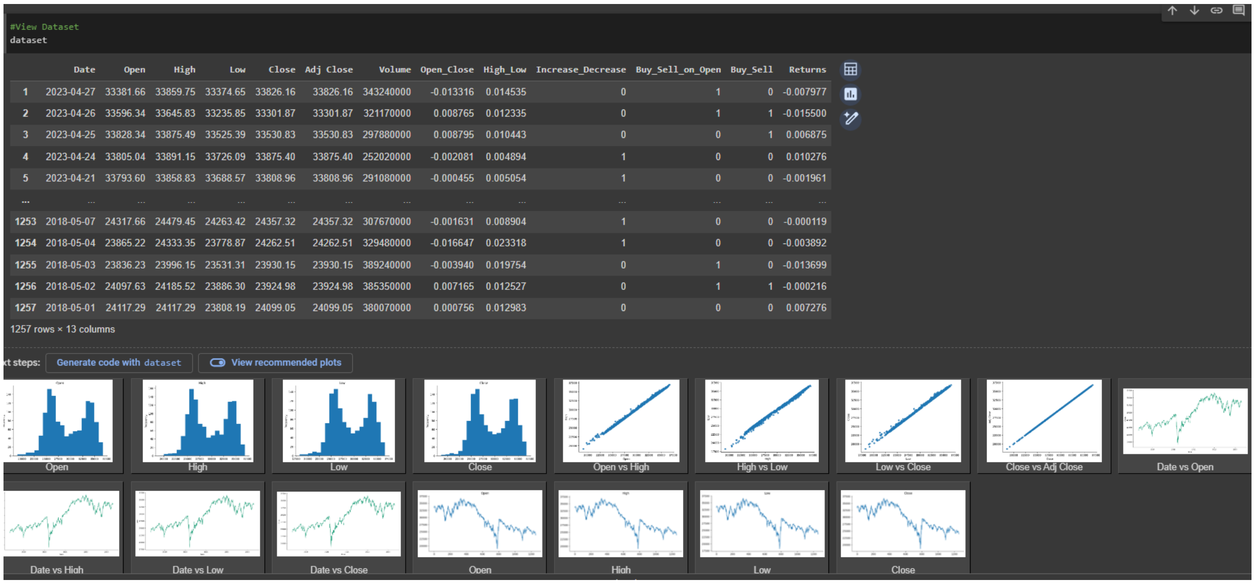
	Date	Open	High	Low	Close	Adj Close	Volume
0	2023-04-28	33,797.43	34,104.56	33,728.40	34,098.16	34,098.16	354,310,000
1	2023-04-27	33,381.66	33,859.75	33,374.65	33,826.16	33,826.16	343,240,000
2	2023-04-26	33,596.34	33,645.83	33,235.85	33,301.87	33,301.87	321,170,000
3	2023-04-25	33,828.34	33,875.49	33,525.39	33,530.83	33,530.83	297,880,000
4	2023-04-24	33,805.04	33,891.15	33,726.09	33,875.40	33,875.40	252,020,000

Date Preprocessing

Data Preprocessing

```python
import pandas as pd
import numpy as np

# Assuming 'df' is the DataFrame containing your dataset

# Convert 'Date' column to datetime type
dataset['Date'] = pd.to_datetime(dataset['Date'])

# Remove commas and convert columns to numeric type
dataset['Open'] = dataset['Open'].replace(',', '', regex=True).astype(float)
dataset['High'] = dataset['High'].replace(',', '', regex=True).astype(float)
dataset['Low'] = dataset['Low'].replace(',', '', regex=True).astype(float)
dataset['Close'] = dataset['Close'].replace(',', '', regex=True).astype(float)
dataset['Adj Close'] = dataset['Adj Close'].replace(',', '', regex=True).astype(float)
dataset['Volume'] = dataset['Volume'].replace(',', '', regex=True).astype(int)

# Calculate the new columns based on the modified dataset
dataset['Open_Close'] = (dataset['Open'] - dataset['Adj Close']) / dataset['Open']
dataset['High_Low'] = (dataset['High'] - dataset['Low']) / dataset['Low']
dataset['Increase_Decrease'] = np.where(dataset['Volume'].shift(-1) > dataset['Volume'], 1, 0)
dataset['Buy_Sell_on_Open'] = np.where(dataset['Open'].shift(-1) > dataset['Open'], 1, 0)
dataset['Buy_Sell'] = np.where(dataset['Adj Close'].shift(-1) > dataset['Adj Close'], 1, 0)
dataset['Returns'] = dataset['Adj Close'].pct_change()

# Drop rows with NaN values
dataset = dataset.dropna()

# Display the modified dataset
print(dataset.head())
```

```
        Date      Open      High       Low     Close  Adj Close     Volume  \
1 2023-04-27  33381.66  33859.75  33374.65  33826.16   33826.16  343240000
2 2023-04-26  33596.34  33645.83  33235.85  33301.87   33301.87  321170000
3 2023-04-25  33828.34  33875.49  33525.39  33530.83   33530.83  297880000
4 2023-04-24  33805.04  33891.15  33726.09  33875.40   33875.40  252020000
5 2023-04-21  33793.60  33858.83  33688.57  33808.96   33808.96  291080000

   Open_Close  High_Low  Increase_Decrease  Buy_Sell_on_Open  Buy_Sell  \
1   -0.013316  0.014535                  0                 1         0
2    0.008765  0.012335                  0                 1         1
3    0.008795  0.010443                  0                 0         1
4   -0.002081  0.004894                  1                 0         0
5   -0.000455  0.005054                  1                 0         0

    Returns
1 -0.007977
2 -0.015500
3  0.006875
4  0.010276
5 -0.001961
```

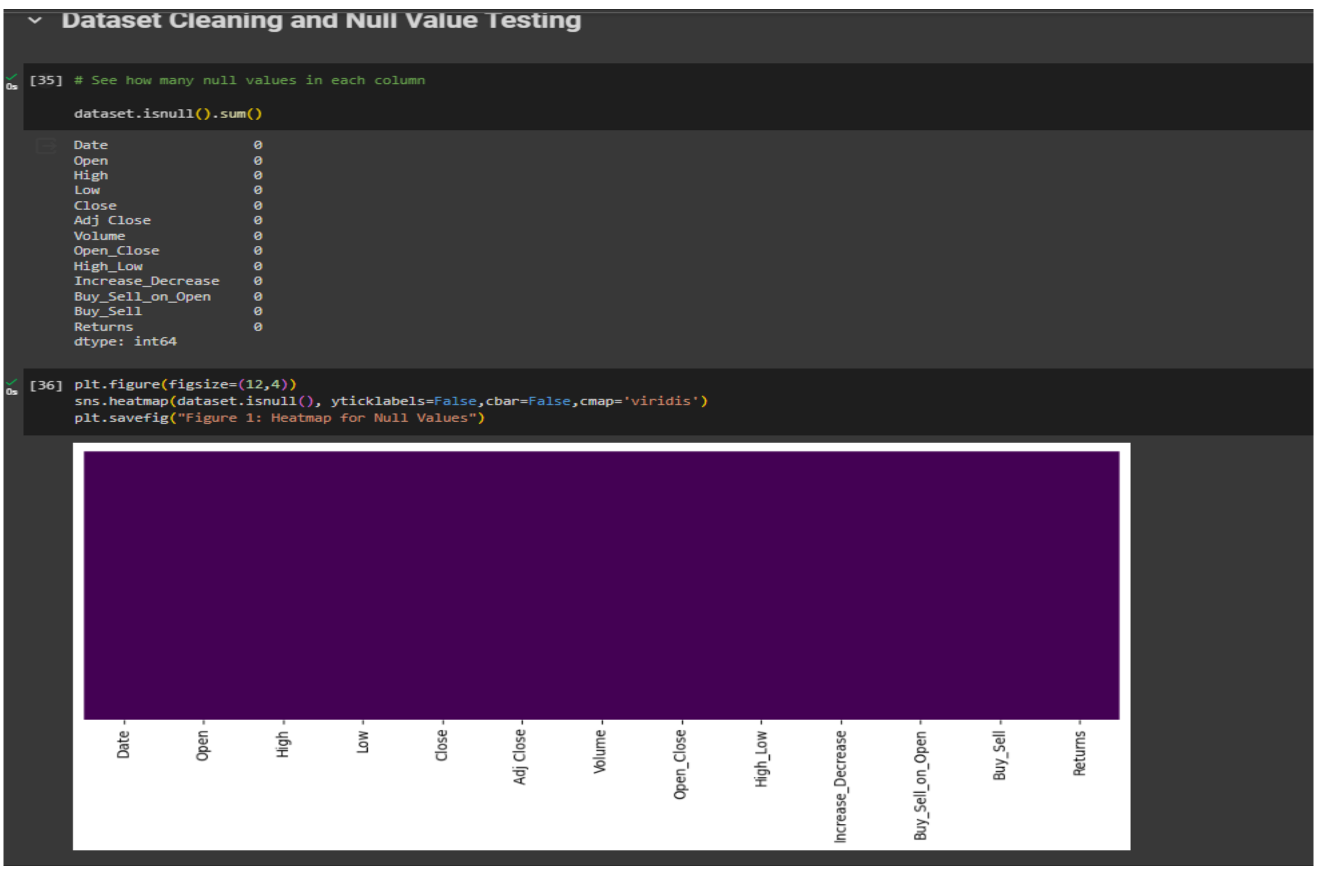


```
#View Dataset
dataset
```

	Date	Open	High	Low	Close	Adj Close	Volume	Open_Close	High_Low	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Returns
1	2023-04-27	33381.66	33859.75	33374.65	33826.16	33826.16	343240000	-0.013316	0.014535	0	1	0	-0.007977
2	2023-04-26	33596.34	33645.83	33235.85	33301.87	33301.87	321170000	0.008765	0.012335	0	1	1	-0.015500
3	2023-04-25	33828.34	33875.49	33525.39	33530.83	33530.83	297880000	0.008795	0.010443	0	0	1	0.006875
4	2023-04-24	33805.04	33891.15	33726.09	33875.40	33875.40	252020000	-0.002081	0.004894	1	0	0	0.010276
5	2023-04-21	33793.60	33858.83	33688.57	33808.96	33808.96	291080000	-0.000455	0.005054	1	0	0	-0.001961
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1253	2018-05-07	24317.66	24479.45	24263.42	24357.32	24357.32	307670000	-0.001631	0.008904	1	0	0	-0.000119
1254	2018-05-04	23865.22	24333.35	23778.87	24262.51	24262.51	329480000	-0.016647	0.023318	1	0	0	-0.003892
1255	2018-05-03	23836.23	23996.15	23531.31	23930.15	23930.15	389240000	-0.003940	0.019754	0	1	0	-0.013699
1256	2018-05-02	24097.63	24185.52	23886.30	23924.98	23924.98	385350000	0.007165	0.012527	0	1	1	-0.000216
1257	2018-05-01	24117.29	24117.29	23808.19	24099.05	24099.05	380070000	0.000756	0.012983	0	0	0	0.007276

1257 rows × 13 columns

Dataset Cleaning and Null Value Testing



Dataset Cleaning and Null Value Testing

```
[35] # See how many null values in each column

     dataset.isnull().sum()
```

```
Date                 0
Open                 0
High                 0
Low                  0
Close                0
Adj Close            0
Volume               0
Open_Close           0
High_Low             0
Increase_Decrease    0
Buy_Sell_on_Open     0
Buy_Sell             0
Returns              0
dtype: int64
```

```
[36] plt.figure(figsize=(12,4))
     sns.heatmap(dataset.isnull(), yticklabels=False,cbar=False,cmap='viridis')
     plt.savefig("Figure 1: Heatmap for Null Values")
```

Analyze the data.

Analyze the Dataset

```
[37] # view number of rows, number of columns
dataset.shape
```

```
(1257, 13)
```

```
[38] # total number of records
dataset.size
print("Total number of records = ",dataset.size)
```

```
Total number of records =  16341
```

```
[39] # view columns names
dataset.columns
```

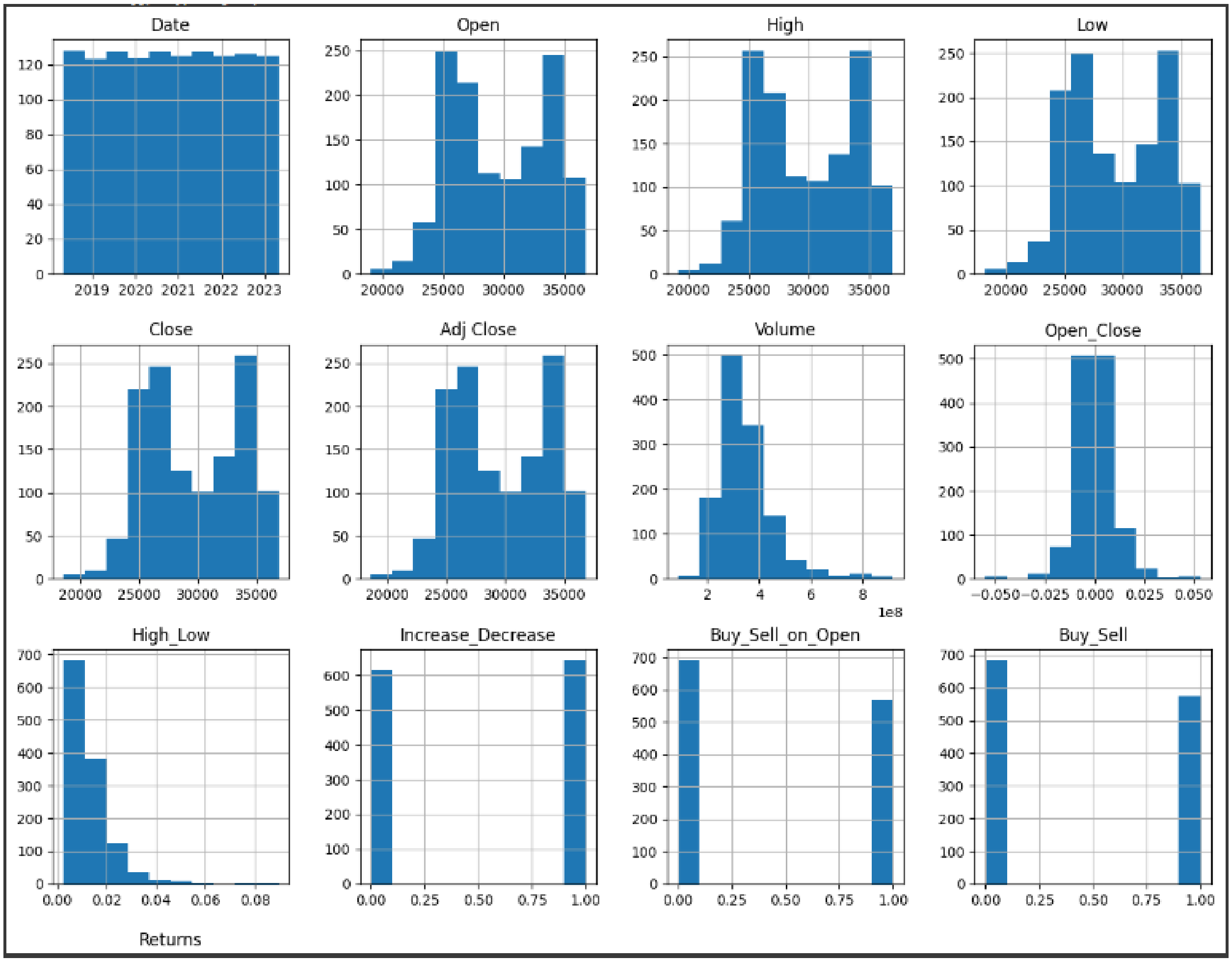
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
       'Open_Close', 'High_Low', 'Increase_Decrease', 'Buy_Sell_on_Open',
       'Buy_Sell', 'Returns'],
      dtype='object')
```

```
[40] # view data types of the columns
dataset.dtypes
```

```
Date                 datetime64[ns]
Open                        float64
High                        float64
Low                         float64
Close                       float64
Adj Close                   float64
Volume                        int64
Open_Close                  float64
High_Low                    float64
Increase_Decrease             int64
Buy_Sell_on_Open              int64
Buy_Sell                      int64
Returns                     float64
dtype: object
```

```
[41] # view dataset info
dataset.info()
```

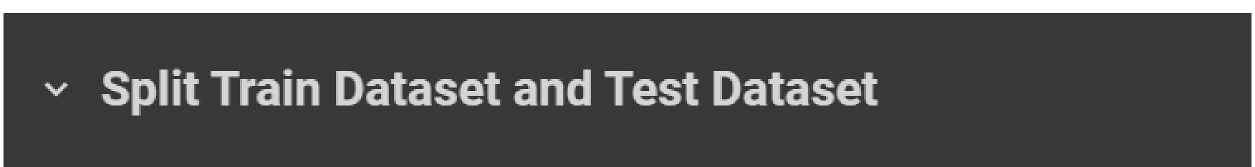
```
<class 'pandas.core.frame.DataFrame'>
Index: 1257 entries, 1 to 1257
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Date               1257 non-null   datetime64[ns]
 1   Open               1257 non-null   float64
 2   High               1257 non-null   float64
 3   Low                1257 non-null   float64
 4   Close              1257 non-null   float64
 5   Adj Close          1257 non-null   float64
 6   Volume             1257 non-null   int64
 7   Open_Close         1257 non-null   float64
 8   High_Low           1257 non-null   float64
 9   Increase_Decrease  1257 non-null   int64
 10  Buy_Sell_on_Open   1257 non-null   int64
 11  Buy_Sell           1257 non-null   int64
 12  Returns            1257 non-null   float64
dtypes: datetime64[ns](1), float64(8), int64(4)
memory usage: 137.5 KB
```



Define X and Y


```
[46] X = dataset[['Open', 'High', 'Low','Volume', 'Open_Close', 'High_Low', 'Returns']]
     y = dataset['Adj Close']
```

Split Data to train and test




```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                       test_size=0.4, random_state=101)
```

<u>Dataset Training and Model Training</u>

Dataset Traning and Model Traning

```
[48] lm = LinearRegression()
     lm.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[49] print(lm.intercept_)
```

```
-27.29076441367215
```

<u>Evaluate model.</u>

Accuracy and Loss Function Values of the Model

```
[56] print('Mean_Absolute_Error(MAE):', metrics.mean_absolute_error(y_test, y_pred))
     print('Mean_Squared_Error(MSE):', metrics.mean_squared_error(y_test, y_pred))
     print('Root_Mean_Squared_Error(RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

     Mean_Absolute_Error(MAE): 28.26542154620928
     Mean_Squared_Error(MSE): 2482.3811965426908
     Root_Mean_Squared_Error(RMSE): 49.82350844449598
```

```
[57] print("Accuracy score: {:.7f}".format(lm.score(X_test, y_test)))

     Accuracy score: 0.9998467
```

```
[58] import matplotlib.pyplot as plt

     # Plotting the actual stock prices
     plt.figure(figsize=(12, 6))
     plt.plot(dataset['Date'], dataset['Adj Close'], label='Actual Stock Prices', color='blue')

     # Plotting the predicted stock prices for the entire dataset
     plt.plot(dataset['Date'], lm.predict(X), label='Predicted Stock Prices', color='red')

     plt.title('Actual vs. Predicted Stock Prices')
     plt.xlabel('Date')
     plt.ylabel('Stock Price')
     plt.legend()
     plt.grid(True)
     plt.show()
```

5.3 Decision Tree Model - IT20613136 _ Jayarathne A. H. B

Import Libraries

Import libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# MATPLOTLIB FOR GRAPH-PLOTTING
import matplotlib.pyplot as plt
%matplotlib inline
```

Load Dataset

Load dataset

```
DATASET_FILE_PATH = './yahoo_data.csv'
```

```
# Read the CSV file into a DataFrame
df = pd.read_csv(DATASET_FILE_PATH)

# Display the first few rows of the DataFrame and format the date column
df.head().style.format({'Date': lambda x: pd.to_datetime(x).strftime('%Y-%m-%d')})
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2023-04-28	33,797.43	34,104.56	33,728.40	34,098.16	34,098.16	354,310,000
1	2023-04-27	33,381.66	33,859.75	33,374.65	33,826.16	33,826.16	343,240,000
2	2023-04-26	33,596.34	33,645.83	33,235.85	33,301.87	33,301.87	321,170,000
3	2023-04-25	33,828.34	33,875.49	33,525.39	33,530.83	33,530.83	297,880,000
4	2023-04-24	33,805.04	33,891.15	33,726.09	33,875.40	33,875.40	252,020,000

Date Preprocessing

Data Preprocessing

```python
# Convert 'Date' column to datetime type
df['Date'] = pd.to_datetime(df['Date'])

# Remove commas and convert columns to numeric type
df['Open'] = df['Open'].replace(',', '', regex=True).astype(float)
df['High'] = df['High'].replace(',', '', regex=True).astype(float)
df['Low'] = df['Low'].replace(',', '', regex=True).astype(float)
df['Close'] = df['Close'].replace(',', '', regex=True).astype(float)
df['Adj Close'] = df['Adj Close'].replace(',', '', regex=True).astype(float)
df['Volume'] = df['Volume'].replace(',', '', regex=True).astype(int)
```

```python
df['Increase_Decrease'] = np.where(df['Volume'].shift(-1) > df['Volume'], 1, 0)
df['Buy_Sell_on_Open'] = np.where(df['Open'].shift(-1) > df['Open'], 1, -1)
df['Buy_Sell'] = np.where(df['Adj Close'].shift(-1) > df['Adj Close'], 1, -1)
df['Return'] = df['Adj Close'].pct_change()
df = df.dropna()
```

Display preprocessed dataset

```python
df
```

	Date	Open	High	Low	Close	Adj Close	Volume	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Return
1	2023-04-27	33381.66	33859.75	33374.65	33826.16	33826.16	343240000	0	1	-1	-0.007977
2	2023-04-26	33596.34	33645.83	33235.85	33301.87	33301.87	321170000	0	1	1	-0.015500
3	2023-04-25	33828.34	33875.49	33525.39	33530.83	33530.83	297880000	0	-1	1	0.006875
4	2023-04-24	33805.04	33891.15	33726.09	33875.40	33875.40	252020000	1	-1	-1	0.010276
5	2023-04-21	33793.60	33858.83	33688.57	33808.96	33808.96	291080000	1	-1	-1	-0.001961
...	...	...	...	...	...	...	...	...	...	...	...
1253	2018-05-07	24317.66	24479.45	24263.42	24357.32	24357.32	307670000	1	-1	-1	-0.000119
1254	2018-05-04	23865.22	24333.35	23778.87	24262.51	24262.51	329480000	1	-1	-1	-0.003892
1255	2018-05-03	23836.23	23996.15	23531.31	23930.15	23930.15	389240000	0	1	-1	-0.013699
1256	2018-05-02	24097.63	24185.52	23886.30	23924.98	23924.98	385350000	0	1	1	-0.000216
1257	2018-05-01	24117.29	24117.29	23808.19	24099.05	24099.05	380070000	0	-1	-1	0.007276

1257 rows × 11 columns

Dataset Cleaning and Null Value Testing

Dataset cleaning and Null value testing

```
df = df.dropna()
df.isnull().sum()
```

```
Date                0
Open                0
High                0
Low                 0
Close               0
Adj Close           0
Volume              0
Increase_Decrease   0
Buy_Sell_on_Open    0
Buy_Sell            0
Return              0
dtype: int64
```

Analyze the dataset

```
df.columns
```

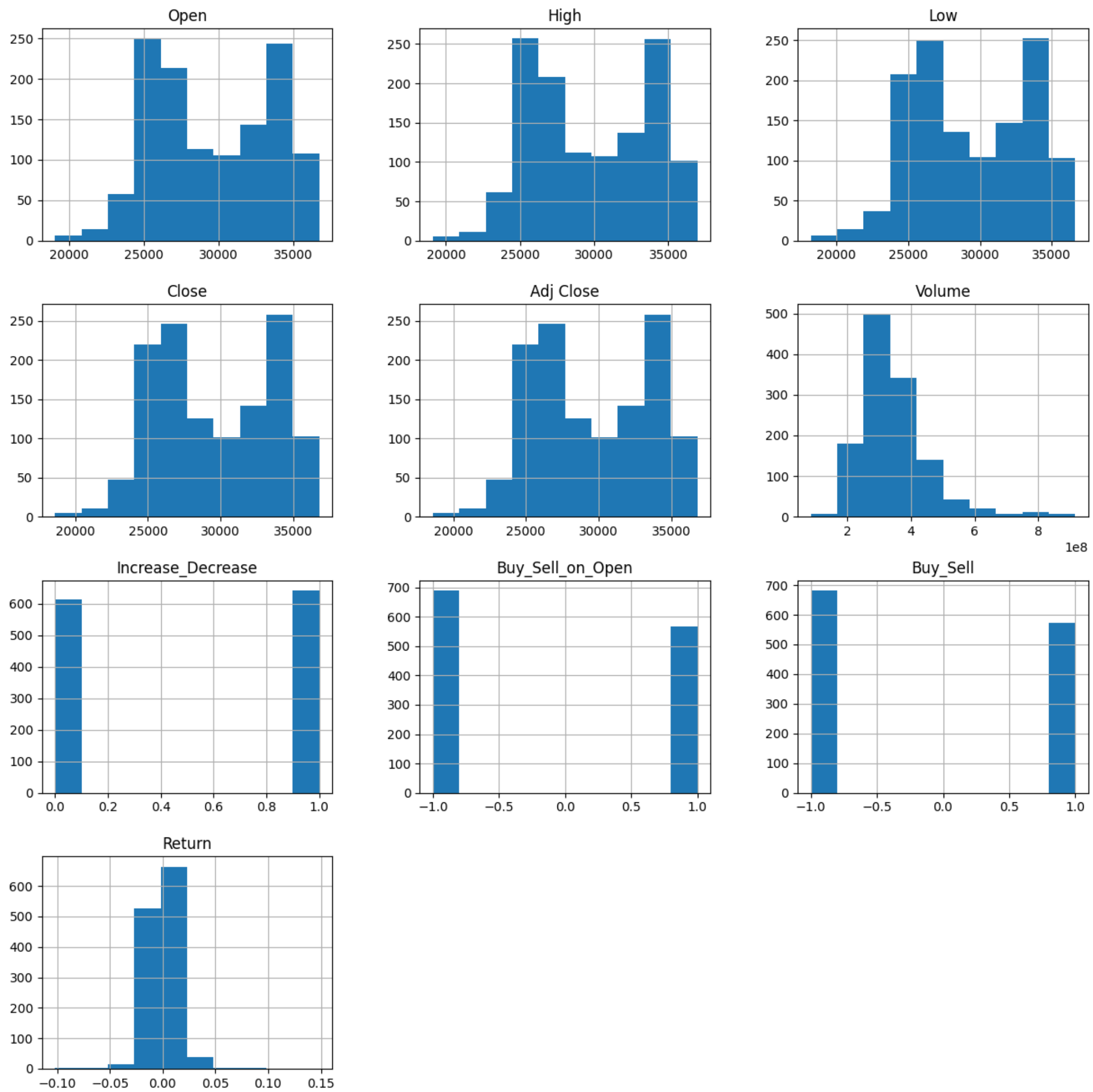
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
       'Increase_Decrease', 'Buy_Sell_on_Open', 'Buy_Sell', 'Return'],
      dtype='object')
```

```
df.shape
```

```
(1256, 11)
```

```
# Histogram per each numerical column
sort_columns = df.iloc[:, 1:]
sort_columns.hist(figsize=(15, 15))
# plt.savefig('histogram.png')
```

```
array([[<Axes: title={'center': 'Open'}>,
        <Axes: title={'center': 'High'}>,
        <Axes: title={'center': 'Low'}>],
       [<Axes: title={'center': 'Close'}>,
        <Axes: title={'center': 'Adj Close'}>,
        <Axes: title={'center': 'Volume'}>],
       [<Axes: title={'center': 'Increase_Decrease'}>,
        <Axes: title={'center': 'Buy_Sell_on_Open'}>,
        <Axes: title={'center': 'Buy_Sell'}>],
       [<Axes: title={'center': 'Return'}>, <Axes: >, <Axes: >]],
      dtype=object)
```




```python
# The statistics per each column
sort_columns.describe()
```

	Open	High	Low	Close	Adj Close	Volume	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Return
count	1257.000000	1257.000000	1257.000000	1257.000000	1257.000000	1.257000e+03	1257.000000	1257.000000	1257.000000	1257.000000
mean	29592.480477	29773.502928	29398.990724	29595.782681	29595.782681	3.450563e+08	0.510740	-0.097852	-0.086714	-0.000184
std	4005.917425	4008.742338	4004.681746	4007.052034	4007.052034	1.069564e+08	0.500084	0.995597	0.996630	0.013664
min	19028.360000	19121.010000	18213.650000	18591.930000	18591.930000	8.615000e+07	0.000000	-1.000000	-1.000000	-0.102052
25%	26040.300000	26162.280000	25877.240000	26026.320000	26026.320000	2.772300e+08	0.000000	-1.000000	-1.000000	-0.006289
50%	29198.920000	29330.160000	28995.660000	29196.040000	29196.040000	3.245800e+08	1.000000	-1.000000	-1.000000	-0.000725
75%	33596.340000	33817.960000	33343.430000	33597.920000	33597.920000	3.876100e+08	1.000000	1.000000	1.000000	0.004709
max	36722.600000	36952.650000	36636.000000	36799.650000	36799.650000	9.159900e+08	1.000000	1.000000	1.000000	0.148456

Define inputs (X) and targets (Y)

Define inputs (X) & targets(Y)

```
X = df.drop(['Date', 'Adj Close', 'Close'], axis=1)
y = df['Adj Close']
```

Split dataset to train set and test set

Split dataset into train set & validation set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Model Training

Model traning

```
model = DecisionTreeRegressor(random_state=0)
model.fit(X_train, y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=0)
```

Evaluate model.

Comparison of Actual Values and Predictions Values

```
y_pred = model.predict(X_test)
```

```
adj_value = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
print(adj_value.head(), "\n")
print(adj_value.tail())
```

```
       Actual  Predicted
6    33786.62   33745.40
495  34269.16   34496.51
53   33869.27   33826.69
985  24815.04   25014.86
187  32798.40   32953.46

        Actual  Predicted
1103  24423.26   24133.78
32    32155.40   32246.55
409   34869.63   34921.88
65    33743.84   33733.96
1030  25625.59   25473.23
```

```
print(y_test.shape)
print(y_pred.shape)
```

```
(252,)
(252,)
```

Accuracy and Loss function values of the model

```
from sklearn import metrics
print('Mean_Absolute_Error(MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Mean_Squared_Error(MSE):', metrics.mean_squared_error(y_test, y_pred))
print('Root_Mean_Squared_Error(RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean_Absolute_Error(MAE): 139.08472222222218
Mean_Squared_Error(MSE): 49705.407550396805
Root_Mean_Squared_Error(RMSE): 222.94709585548947
```

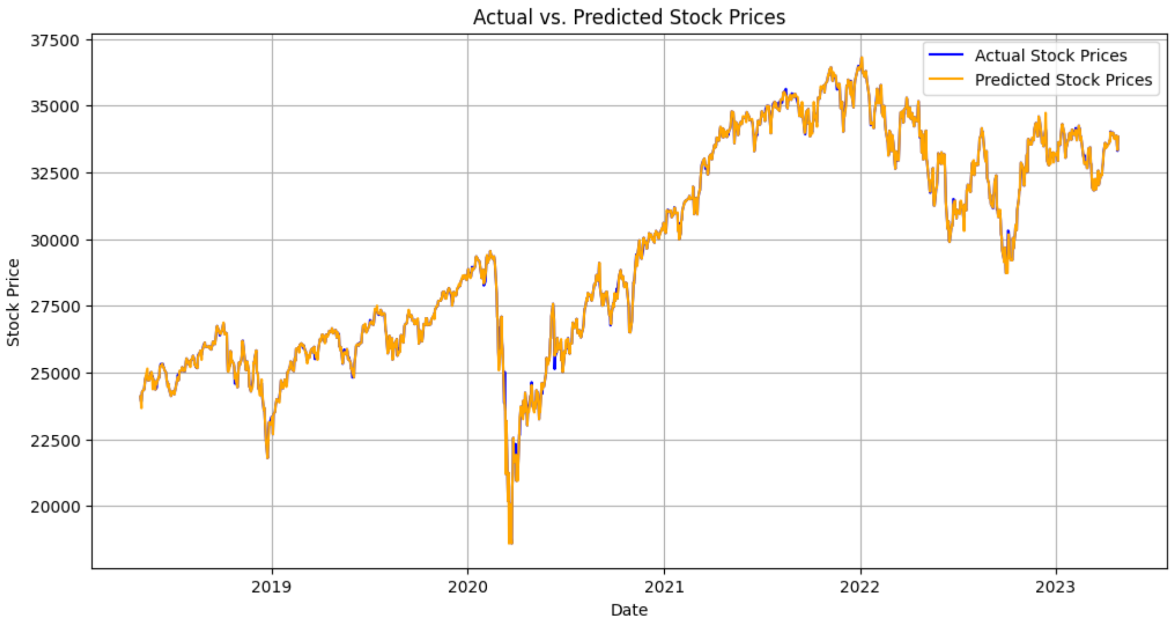
```
from sklearn.model_selection import cross_val_score

dt_fit = model.fit(X_train, y_train)
dt_scores = cross_val_score(dt_fit, X_train, y_train, cv = 5)
```

```
print("Accuracy score: {:.3f}".format(model.score(X_test, y_test)))
```

```
Accuracy score: 0.997
```

Plot the Actual and Predicted stock prices

```
# Plotting the actual stock prices
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Adj Close'], label='Actual Stock Prices', color='blue')

# Plotting the predicted stock prices for the entire dataset
plt.plot(df['Date'], model.predict(X), label='Predicted Stock Prices', color='orange')

plt.title('Actual vs. Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid(True)
plt.show()
```



5.4 Logistic Regression Model – IT20609030 – Karunanayake M.L

Import Libraries

Import Libraries

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

import warnings
warnings.filterwarnings("ignore")

# MATPLOTLIB & SEABORN FOR GRAPH-PLOTTING
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Load Dataset

Load Data into Data Frame

```
import pandas as pd

# Path to your CSV file in Google Drive
file_path = '/content/drive/MyDrive/yahoo_data.csv'

# Read the CSV file into a DataFrame
dataset = pd.read_csv(file_path)

# Display the first few rows of the DataFrame and format the date column
dataset.head().style.format({'Date': lambda x: pd.to_datetime(x).strftime('%Y-%m-%d')})
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2023-04-28	33,797.43	34,104.56	33,728.40	34,098.16	34,098.16	354,310,000
1	2023-04-27	33,381.66	33,859.75	33,374.65	33,826.16	33,826.16	343,240,000
2	2023-04-26	33,596.34	33,645.83	33,235.85	33,301.87	33,301.87	321,170,000
3	2023-04-25	33,828.34	33,875.49	33,525.39	33,530.83	33,530.83	297,880,000
4	2023-04-24	33,805.04	33,891.15	33,726.09	33,875.40	33,875.40	252,020,000

Data Preprocessing

Data Preprocessing

```
[ ] # Convert 'Date' column to datetime type
    dataset['Date'] = pd.to_datetime(dataset['Date'])

    # Remove commas and convert columns to numeric type
    dataset['Open'] = dataset['Open'].replace(',', '', regex=True).astype(float)
    dataset['High'] = dataset['High'].replace(',', '', regex=True).astype(float)
    dataset['Low'] = dataset['Low'].replace(',', '', regex=True).astype(float)
    dataset['Close'] = dataset['Close'].replace(',', '', regex=True).astype(float)
    dataset['Adj Close'] = dataset['Adj Close'].replace(',', '', regex=True).astype(float)
    dataset['Volume'] = dataset['Volume'].replace(',', '', regex=True).astype(int)
```

```
[ ] # Create a 'Buy_Sell' column based on the comparison of current 'Adj Close' and next day's 'Adj Close'
    # If next day's 'Adj Close' is higher, set 'Buy_Sell' to 1 (buy), otherwise set it to -1 (sell)
    dataset['Buy_Sell'] = np.where(dataset['Adj Close'].shift(-1) > dataset['Adj Close'],1,-1)
```

```
[ ] dataset.head()
```

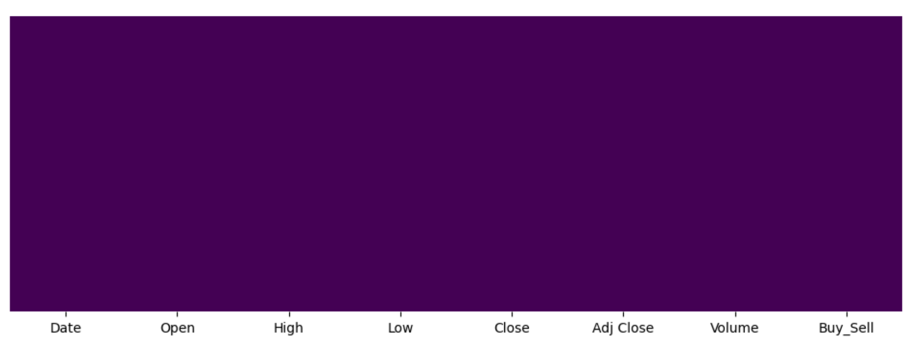
	Date	Open	High	Low	Close	Adj Close	Volume	Buy_Sell
0	2023-04-28	33797.43	34104.56	33728.40	34098.16	34098.16	354310000	-1
1	2023-04-27	33381.66	33859.75	33374.65	33826.16	33826.16	343240000	-1
2	2023-04-26	33596.34	33645.83	33235.85	33301.87	33301.87	321170000	1
3	2023-04-25	33828.34	33875.49	33525.39	33530.83	33530.83	297880000	1
4	2023-04-24	33805.04	33891.15	33726.09	33875.40	33875.40	252020000	-1

Dataset Cleaning and Null Value Testing

Dataset Cleaning and Null Value Testing

```
[ ] # See how many null values in each column
    dataset.isnull().sum()

    Date         0
    Open         0
    High         0
    Low          0
    Close        0
    Adj Close    0
    Volume       0
    Buy_Sell     0
    dtype: int64
```

```
plt.figure(figsize=(12,4))
sns.heatmap(dataset.isnull(), yticklabels=False,cbar=False,cmap='viridis')
plt.savefig("Figure 1: Heatmap for Null Values")
```



Analyze the data.

Analyze the Data

```
[ ] # number of rows and number of columns of the dataset
    dataset.shape
```

```
(1258, 8)
```

```
[ ] #Total number of records in the dataset
    print("Total number of records = ", len(dataset))
```

```
Total number of records =  1258
```

```
[ ] #Column names
    dataset.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
       'Buy_Sell'],
      dtype='object')
```

```
[ ] # Data types of the Columns
    dataset.dtypes
```

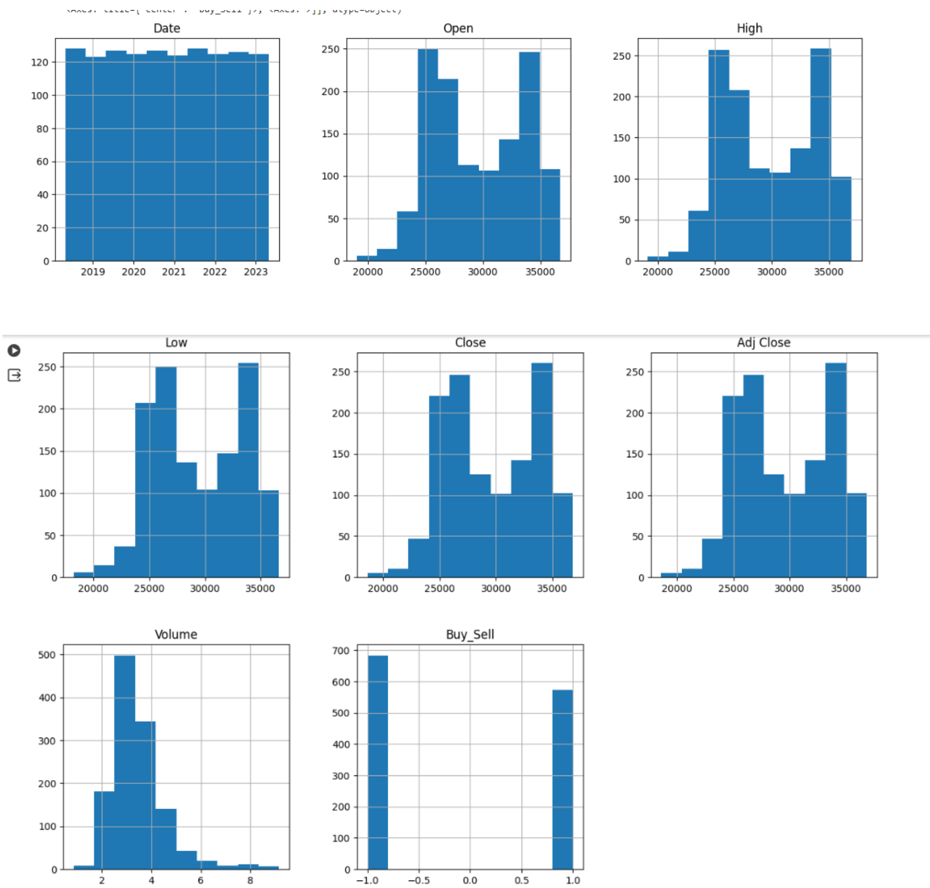
```
Date         datetime64[ns]
Open                float64
High                float64
Low                 float64
Close               float64
Adj Close           float64
Volume                int64
Buy_Sell              int64
dtype: object
```

```
[ ] # Summary of the dataset
    dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1258 non-null   datetime64[ns]
 1   Open       1258 non-null   float64
 2   High       1258 non-null   float64
 3   Low        1258 non-null   float64
 4   Close      1258 non-null   float64
 5   Adj Close  1258 non-null   float64
 6   Volume     1258 non-null   int64
 7   Buy_Sell   1258 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(2)
memory usage: 78.8 KB
```

```
[ ] # Histogram per each numerical column in dataset
    dataset.hist(figsize=(15, 15))
```

```
array([[<Axes: title={'center': 'Date'}>,
        <Axes: title={'center': 'Open'}>,
        <Axes: title={'center': 'High'}>],
       [<Axes: title={'center': 'Low'}>,
        <Axes: title={'center': 'Close'}>,
        <Axes: title={'center': 'Adj Close'}>],
       [<Axes: title={'center': 'Volume'}>,
        <Axes: title={'center': 'Buy_Sell'}>, <Axes: >]], dtype=object)
```




```python
# Calculate and display descriptive statistics for each numerical column
dataset.describe()
```

	Date	Open	High	Low	Close	Adj Close	Volume	Buy_Sell
count	1258	1258.000000	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03	1258.000000
mean	2020-10-28 09:39:12.305246464	29595.823045	29776.945739	29402.432226	29599.361677	29599.361677	3.450636e+08	-0.087440
min	2018-05-01 00:00:00	19028.360000	19121.010000	18213.650000	18591.930000	18591.930000	8.615000e+07	-1.000000
25%	2019-07-31 06:00:00	26041.267500	26163.155000	25877.872500	26027.120000	26027.120000	2.773125e+08	-1.000000
50%	2020-10-27 12:00:00	29201.410000	29335.685000	28996.500000	29199.460000	29199.460000	3.247250e+08	-1.000000
75%	2022-01-26 18:00:00	33604.027500	33825.445000	33346.827500	33600.342500	33600.342500	3.875100e+08	1.000000
max	2023-04-28 00:00:00	36722.600000	36952.650000	36636.000000	36799.650000	36799.650000	9.159900e+08	1.000000
std	NaN	4006.078299	4009.007573	4004.949066	4007.468822	4007.468822	1.069142e+08	0.996566

Define X and Y

Define X and Y

```
[ ] # Convert the 'Buy_Sell' column to integer data type
    dataset['Buy_Sell'] = dataset['Buy_Sell'].astype('int')
```

```
[ ] # Define X
    # Select the columns 'Open', 'High', 'Low', 'Adj Close', 'Volume' from the dataset
    # and convert them into a NumPy array X
    X = np.asarray(dataset[['Open', 'High', 'Low', 'Adj Close', 'Volume']])

    # Display the first 5 rows of the array X
    X[0:5]
```

```
array([[3.379743e+04, 3.410456e+04, 3.372840e+04, 3.409816e+04,
        3.543100e+08],
       [3.338166e+04, 3.385975e+04, 3.337465e+04, 3.382616e+04,
        3.432400e+08],
       [3.359634e+04, 3.364583e+04, 3.323585e+04, 3.330187e+04,
        3.211700e+08],
       [3.382834e+04, 3.387549e+04, 3.352539e+04, 3.353083e+04,
        2.978800e+08],
       [3.380504e+04, 3.389115e+04, 3.372609e+04, 3.387540e+04,
        2.520200e+08]])
```

```
[ ] # Define y
    # Select the 'Buy_Sell' column from the dataset
    # and convert it into a NumPy array y
    y = np.asarray(dataset['Buy_Sell'])

    # Display the first 5 elements of the array y
    y[0:5]
```

```
array([-1, -1,  1,  1, -1])
```

```
[ ] from sklearn import preprocessing
    X = preprocessing.StandardScaler().fit(X).transform(X)
    X[0:5]
```

```
array([[ 1.0492251 ,  1.07990201,  1.08058507,  1.1230499 ,  0.08651847],
       [ 0.94539904,  1.01881273,  0.99222173,  1.05514964, -0.01706368],
       [ 0.99900892,  0.96543167,  0.95755083,  0.9242694 , -0.223573  ],
       [ 1.05694394,  1.02274045,  1.02987513,  0.98142544, -0.44149787],
       [ 1.05112547,  1.02664821,  1.08000806,  1.06744159, -0.87061056]])
```

Split Data to train and test

Split Train data and Test data

```
[ ] # Splitting the dataset into the Training set and Test set
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

    print ('Train set:', X_train.shape,  y_train.shape)
    print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (943, 5) (943,)
Test set: (315, 5) (315,)
```

Dataset Training and Model Training

Dataset Traning and Model Traning

```
[ ] # Fitting Logistic Regression to the Training set
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import confusion_matrix
    LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
    LR
```

```
LogisticRegression
LogisticRegression(C=0.01, solver='liblinear')
```

Evaluate model.

Comparison of Actual Values and Predictions Values

```
[22] # Predicting the Test set results
     y_pred = LR.predict(X_test)
     y_pred
```

```
array([-1,  1, -1, -1, -1, -1,  1, -1, -1,  1,  1,  1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1,  1,  1, -1, -1, -1,
       -1, -1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1,  1, -1, -1, -1,  1, -1, -1, -1,  1,
       -1, -1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1, -1,  1, -1, -1,  1,
       -1, -1, -1,  1, -1, -1, -1, -1, -1, -1,  1,  1, -1, -1,  1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1,  1, -1, -1,  1,  1, -1,  1,  1,
       -1,  1, -1, -1,  1, -1,  1,  1,  1, -1, -1,  1, -1,  1, -1, -1, -1,
       -1, -1,  1, -1, -1, -1, -1, -1, -1,  1, -1, -1, -1, -1, -1,  1,  1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1, -1, -1, -1,
        1, -1, -1,  1,  1, -1, -1, -1,  1,  1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1,  1, -1,  1, -1, -1, -1, -1, -1, -1,
       -1, -1,  1, -1,  1, -1, -1, -1,  1, -1, -1, -1, -1, -1,  1, -1, -1,
        1, -1, -1,  1, -1, -1, -1, -1, -1, -1, -1,  1, -1,  1,  1,  1, -1,
        1,  1, -1, -1,  1, -1,  1, -1, -1, -1, -1, -1,  1,  1, -1, -1, -1,
        1, -1, -1, -1,  1, -1, -1, -1, -1])
```

```
[23] # Calculate the predicted probabilities for each class (0 and 1) using the test data
     y_pred_prob = LR.predict_proba(X_test)

     # Display the predicted probabilities
     y_pred_prob
```

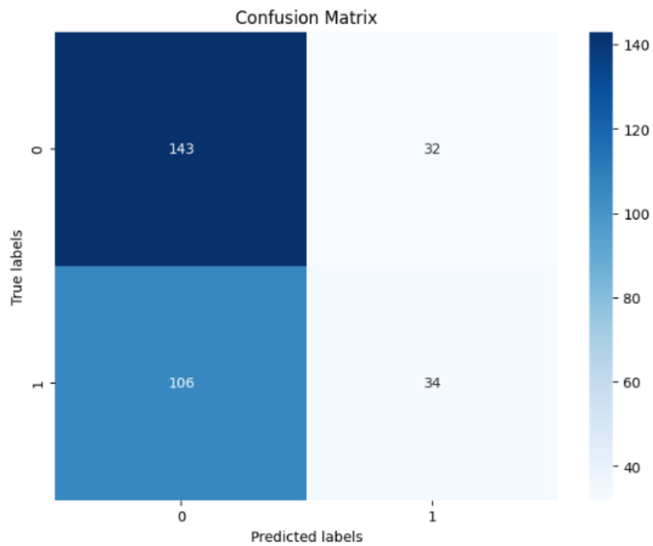
```
array([[0.54533889, 0.45466111],
       [0.49701402, 0.50298598],
       [0.54132405, 0.45867595],
       [0.56242227, 0.43757773],
       [0.53351917, 0.46648083],
       [0.53821733, 0.46178267],
       [0.36848801, 0.63151199],
```

```python
# Create a DataFrame to display y_test and y_pred
results_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Display the DataFrame
print(results_df)
```

```
     Actual  Predicted
0        -1         -1
1         1          1
2        -1         -1
3         1         -1
4         1         -1
..      ...        ...
310      -1          1
311       1         -1
312      -1         -1
313      -1         -1
314       1         -1

[315 rows x 2 columns]
```

Accuracy and Loss Function Values of the Model

```python
from sklearn.metrics import confusion_matrix, classification_report

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



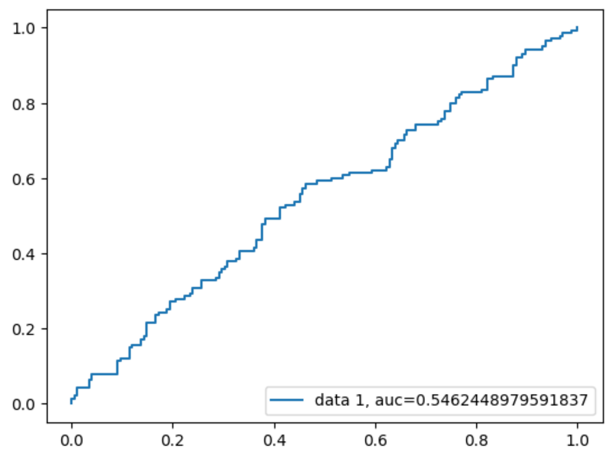
```
[26] # Print a classification report showing precision, recall, F1-score, and other metrics
     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          -1       0.57      0.82      0.67       175
           1       0.52      0.24      0.33       140

    accuracy                           0.56       315
   macro avg       0.54      0.53      0.50       315
weighted avg       0.55      0.56      0.52       315
```

```
[27] from sklearn import metrics

     print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
     print("Precision:",metrics.precision_score(y_test, y_pred))
     print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.5619047619047619
Precision: 0.5151515151515151
Recall: 0.24285714285714285
```

```
[30] # Calculate ROC curve and AUC for the logistic regression model
     y_pred_proba = LR.predict_proba(X_test)[::,1]
     fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
     auc = metrics.roc_auc_score(y_test, y_pred_proba)

     # Plot ROC curve
     plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
     plt.legend(loc=4)
     plt.show()
```




```
[31] print("LogLoss: : %.2f" % log_loss(y_test, y_pred_prob))
```

```
LogLoss: : 0.68
```

```
[32] print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5619047619047619
```

6. RESULTS AND DISCUSSION

6.1 Accuracy Comparison of the Models

The performance of a model depends on its loss function values and accuracy values.

Three common loss functions:

- **MSE (Mean Squared Error):** MSE measures the average of the squared differences between predicted and actual values. Lower MSE indicates better model performance, as it means the model's predictions are closer to the actual values.
- **MAE (Mean Absolute Error):** MAE measures the average of the absolute differences between predicted and actual values. Like MSE, lower MAE indicates better model performance, and it is easier to interpret because it's in the same unit as the target variable.
- **RMSE (Root Mean Squared Error):** RMSE is the square root of MSE and provides a measure of the standard deviation of the errors. Like MSE and MAE, lower RMSE is better. RMSE is sensitive to outliers.

To compare different models, you can train each model on the same dataset, use the same evaluation metrics, and then compare their performance. The model with the lowest MSE, MAE, RMSE, and the highest R2 is generally considered the best model for the given task.

Log loss metric is used to evaluate Logistic regression model.

- **Log Loss:** Also known as logarithmic loss or cross-entropy loss, is a metric used to evaluate the performance of a classification model that outputs probabilities. It measures the difference between the predicted probabilities and the actual binary outcome. Log loss penalizes incorrect confident predictions more heavily than incorrect but less confident predictions.

The table below compares the values and accuracy of the loss functions applicable to the models we used,

Model	MAE	MSE	RMSE	Log Loss	Accuracy
Bayesian Ridge Regression	214.50602456644984	89056.62027878832	298.42355851840574	-	0.9948743553820713
Linear Regression	28.26542154620928	2482.3811965426908	49.82350044449598	-	0.9998467
Decision Tree Regression	139.08472222222218	49705.407550396805	222.94709585548947	-	0.9970342
Logistic Regression	-	-	-	0.68	0.561905

This analysis compared the performance of four machine learning algorithms when predicting the price of a stock market. The results indicated that Linear Regression had the highest accuracy score of 99.99 percent with a promising performance. Bayesian Ridge Regression with 99.49 and Decision Tree Regression with 99.70 also provided good performances although, in this context, accuracy is not the only necessary aspect to consider. Bayesian Ridge Regression also provided uncertainty estimates and is robust against collar linearity . Decision Tree Regression was highly effective because there were non-linear relationships present. In contrast, Logistic Regression scored the lowest performance of 56.19 percent, making it an ineffective measure when predicting the price of stock market prices, even though LR is widely used in binary classifications. Overall, the choice of algorithm should consider not only accuracy but also other factors such as interpretability and robustness to different data characteristics.

7. CRITICAL ANALYSIS & DISCUSSION

7.1 Possible Limitations

Using machine learning (ML) models to forecast stock market prices has numerous drawbacks. First, data quality and quantity are crucial, and insufficient or noisy data might result in skewed forecasts. Second, stock market behavior, which is characterized by volatility and nonlinearity, calls into question basic ML algorithm assumptions such as linearity. Another risk is overfitting, which occurs when models learn noise rather than genuine patterns in the data, particularly with sophisticated models or small datasets. Furthermore, many machine learning models lack interpretability, making it difficult to comprehend their choices.

Assumptions regarding data independence and identically distributed samples may not be valid in financial time series data, reducing model accuracy. Furthermore, ML models may detect connections without comprehending causation, resulting in incorrect predictions. Finally, model resilience in dynamic market situations, as well as the necessity for continuous adaptation, provide substantial obstacles. To improve prediction reliability and interpretability, these restrictions must be addressed by careful model selection, feature engineering, and the incorporation of domain expertise alongside ML approaches.

7.2 Challenges in Stock Price Prediction

- Data Quality: Accurate forecasts require high-quality, dependable data; noisy or inadequate data might provide biased findings.
- Market Volatility: Stock markets are naturally volatile, with sharp movements that can test the reliability of forecasting models.
- Non-linearity: Stock price fluctuations frequently follow non-linear patterns, making it challenging for classic linear models to represent market intricacies.
- Overfitting occurs when ML models capture noise in training data rather than genuine underlying patterns, resulting in poor generalization to unknown data.
- Interpretability: Many ML models lack interpretability, making it difficult to comprehend the reasoning behind their forecasts and obtain insights into market dynamics.
- Assumption Violation: ML algorithms may make assumptions about data independence and distribution that are not valid for financial time series data, resulting in erroneous forecast.

7.3 Future Work (Areas of Possible Improvement)

- Improving forecast accuracy through the integration of supplementary data sourced from original databases.
- Investigating models designed to predict fluctuations in stock prices, rather than solely focusing on stock prices themselves.
- Expanding predictive methodologies to anticipate forthcoming inventory expenses.
- Employing hybrid forecasting models that amalgamate multiple techniques to enhance accuracy.
- Confronting the difficulty of training a universally accurate model that considers country-specific factors, income disparities, and educational levels.

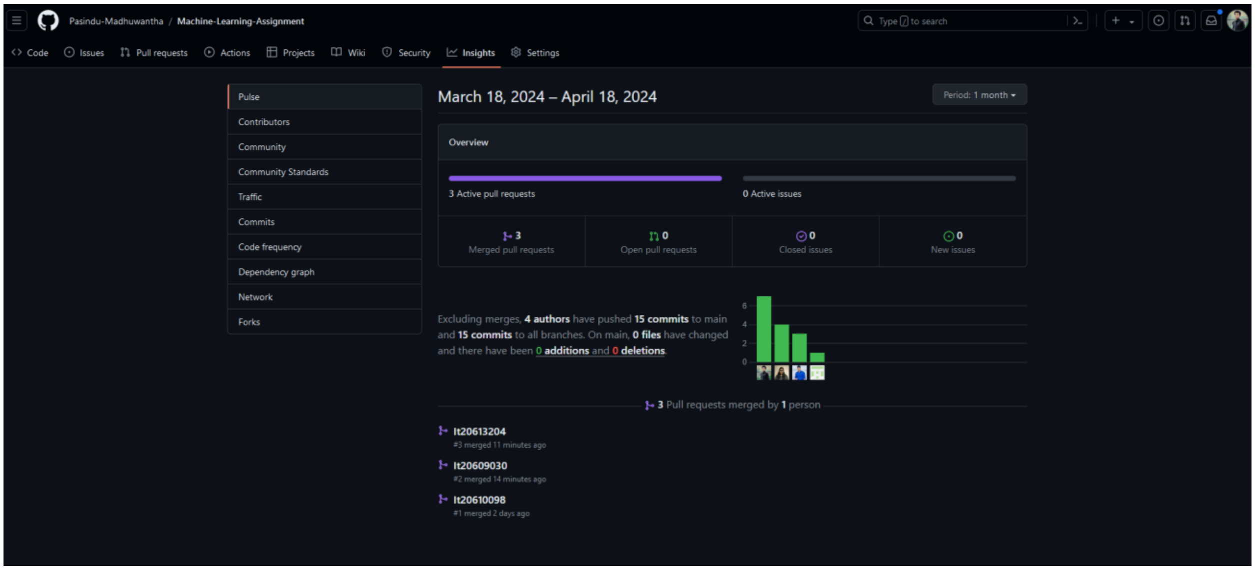
7.4 Solutions for Stock Price Prediction

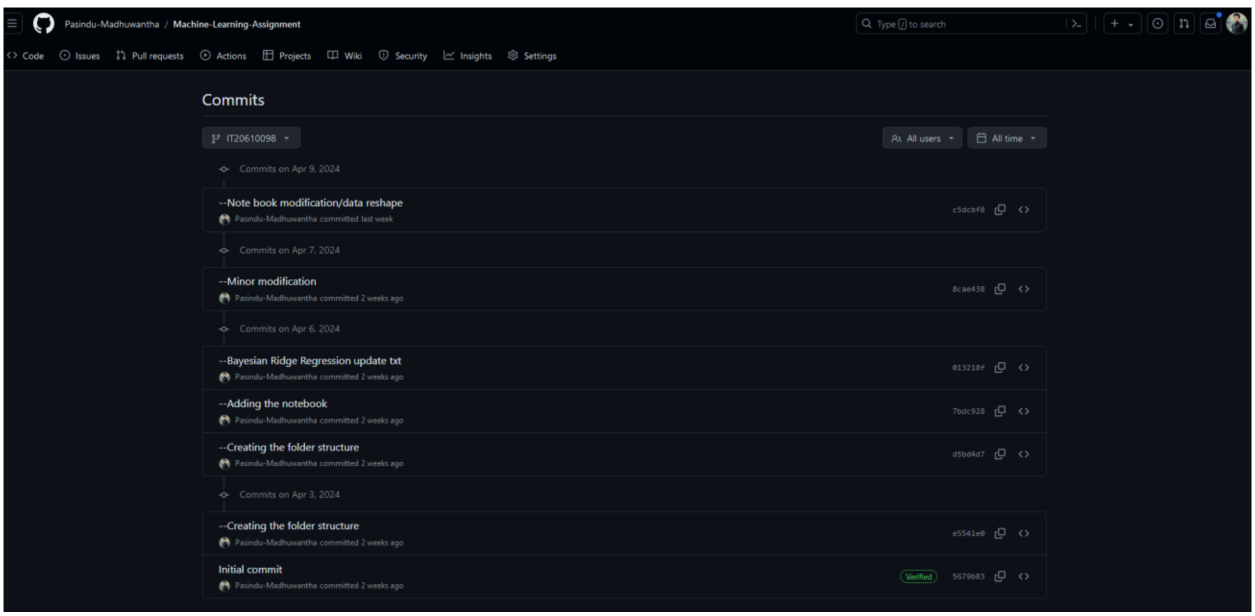
- Recognizing recurrent stock patterns to establish the groundwork for predictive modeling.
- Developing pattern networks capable of encapsulating intricate relationships within stock price datasets.
- Extracting critical characteristic variables that exert the most substantial influence on prediction outcomes.
- Fine-tuning model parameters to maximize performance tailored to specific datasets.
- Deploying predictive models in real-market settings to evaluate their real-world effectiveness.

This revised section provides a clearer and more structured overview of possible limitations, challenges, future research directions, and potential solutions in the domain of stock price prediction.

8.INDIVIDUAL CONTRIBUTION

8.1 GitHub Commits Screenshots – IT20610098 | Madhuwantha M.G.P





8.2 Individual References – IT20610098 | Madhuwantha M.G.P

[01] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8248663/

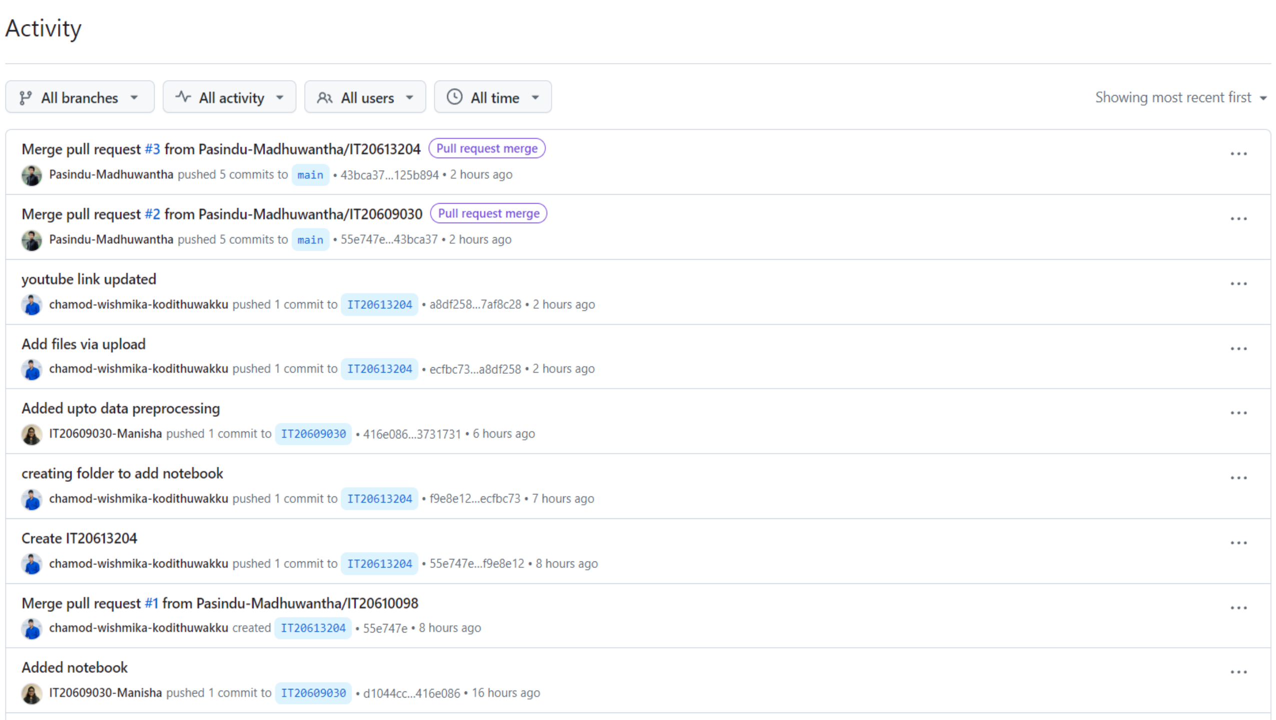
[02] https://www.researchgate.net/publication/363372013_A_Study_of_Stock_Portfolio_Strategy_Based_on_Machine_Learning

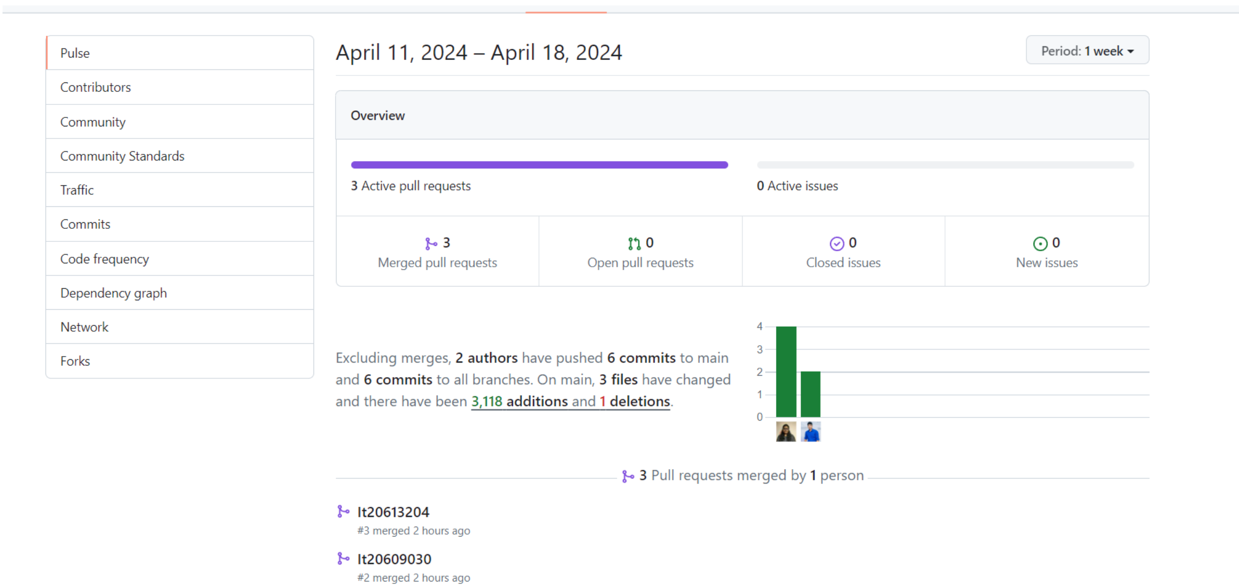
[03] https://www.academia.edu/104105137/Stock_Price_Prediction_of_PT_Kimia_Farma_Tbk_Using_Bayesian_Ridge_Algorithm?uc-sb-sw=86701160

[04] https://www.researchgate.net/publication/354727127_Application_of_Bayesian_Regression_Model_in_Financial_Stock_Market_Forecasting

[05] https://digital.wpi.edu/downloads/0k225b181

8.3 GitHub Commits Screenshots - IT20613204 Kodithuwakku D.R.G.C.W





8.4 Individual References – IT20613204 Kodithuwakku D.R.G.C.W

[01] https://vitalflux.com/popular-machine-learning-techniques-for-stock-price-movement-prediction/#:~:text=Machine%20learning%20techniques%20used%20for%20predicting%20stock%20prices%20involve%20analyzing,the%20best%20fit%20predictive%20models

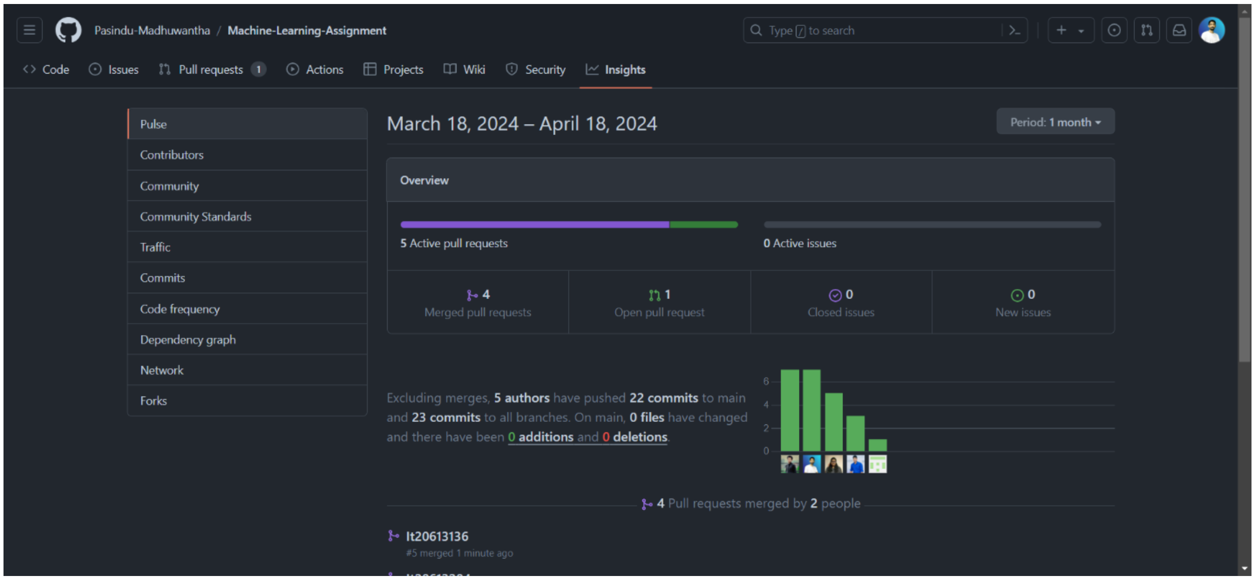
[02] https://www.projectpro.io/article/stock-price-prediction-using-machine-learning-project/571

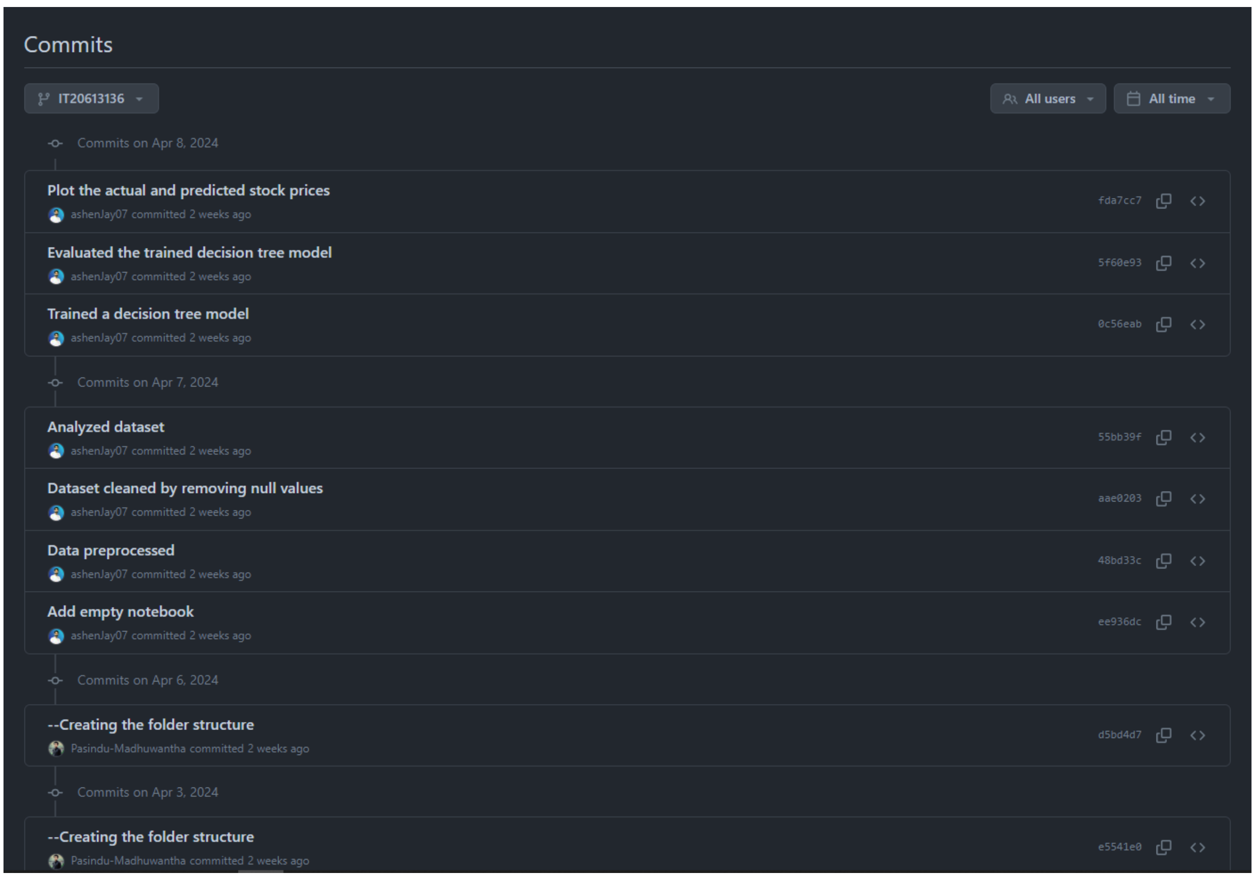
[03] https://www.sciencedirect.com/science/article/pii/S1877050918307828

[04] https://towardsdatascience.com/5-reasons-why-stock-prediction-projects-fail-a3dddf30d242

[05] https://www.android-examples.com/category/phpmyadmin/

8.5 GitHub Commits Screenshots – IT20613136 | Jayarathne A. H. B

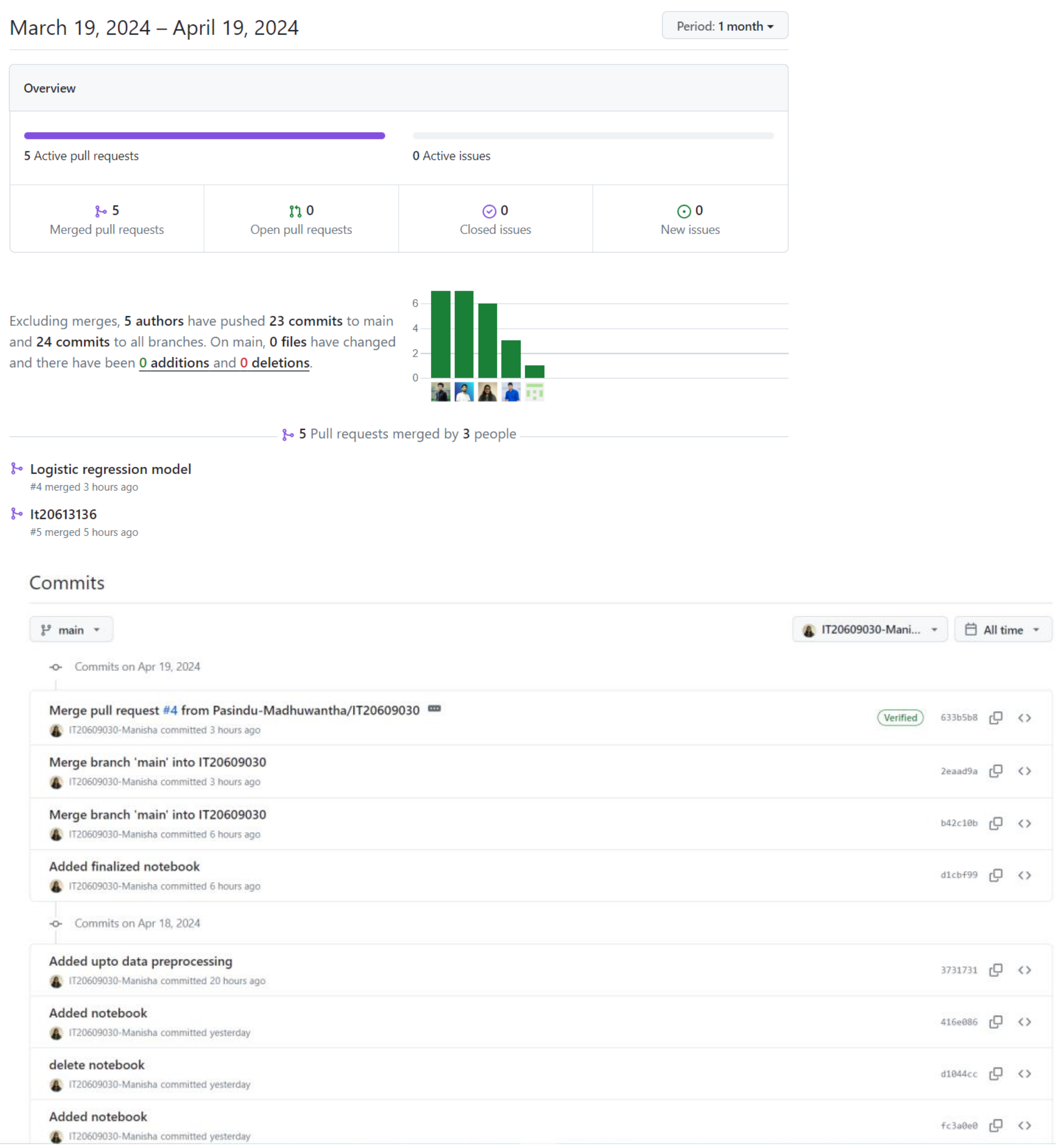




8.6 Individual References – IT20613136 | Jayarathne A. H. B

[01] https://www.coursera.org/articles/decision-tree-machine-learning
[02] https://www.analyticsvidhya.com/blog/2021/07/a-comprehensive-guide-to-decision-trees/
[03] https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/
[04] https://sci-hub.se/https://doi.org/10.1016/j.eswa.2005.09.026
[05] https://www.academia.edu/download/48838863/A_Decision_Tree-_Rough_Set_Hybrid_System20160914-7509-161mltw.pdf

8.7 GitHub Commits Screenshots – IT20609030 | Karunanayake M.L



8.8 Individual References – IT20609030 | Karunanayake M.L

[1] https://ieeexplore.ieee.org/document/8328543

[2]https://www.researchgate.net/publication/343438201_Stock_Market_Prediction_using_Logistic_Regression_Analysis_-_A_Pilot_Study.

[3] https://arxiv.org/ftp/arxiv/papers/2202/2202.09359.pdf

[4] https://www.proquest.com/docview/2596019108?sourcetype=Scholarly%20Journals

[5] https://www.analyticsvidhya.com/blog/2021/07/an-introduction-to-logistic-regression/

[6] https://www.ibm.com/topics/logistic-regression

9.REFERENCES

[01] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8248663/

[02] https://www.researchgate.net/publication/363372013_A_Study_of_Stock_Portfolio_Strategy_Based_on_Machine_Learning

[03] https://www.academia.edu/104105137/Stock_Price_Prediction_of_PT_Kimia_Farma_Tbk_Using_Bayesian_Ridge_Algorithm?uc-sb-sw=86701160

[04] https://www.researchgate.net/publication/354727127_Application_of_Bayesian_Regression_Model_in_Financial_Stock_Market_Forecasting

[05] https://digital.wpi.edu/downloads/0k225b181

[06]https://vitalflux.com/popular-machine-learning-techniques-for-stock-price-movement-prediction/#:~:text=Machine%20learning%20techniques%20used%20for%20predicting%20stock%20prices%20involve%20analyzing,the%20best%20fit%20predictive%20models

[07] https://www.projectpro.io/article/stock-price-prediction-using-machine-learning-project/571

[08] https://www.sciencedirect.com/science/article/pii/S1877050918307828

[09] https://towardsdatascience.com/5-reasons-why-stock-prediction-projects-fail-a3dddf30d242

[10] https://www.android-examples.com/category/phpmyadmin/

[11] https://www.coursera.org/articles/decision-tree-machine-learning

[12] https://www.analyticsvidhya.com/blog/2021/07/a-comprehensive-guide-to-decision-trees/

[13] https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/

[14] https://sci-hub.se/https://doi.org/10.1016/j.eswa.2005.09.026

[15] https://www.academia.edu/download/48838863/A_Decision_Tree-_Rough_Set_Hybrid_System20160914-7509-161mltw.pdf

10 APPENDIX

10.1 GitHub Repository Link

https://github.com/Pasindu-Madhuwantha/Machine-Learning-Assignment

10.2 Video Demonstration You tube Link.

https://www.youtube.com/playlist?list=PLfIkPB6l9EbJaboXP-ied64WrYohJfyPH

10.3 Sources codes

10.3.1 Bayesian Ridge Regression Model - IT20610098 -Madhuwantha M.G.P.

```
from google.colab import drive
drive.mount('/content/drive')
```

Bayesian regression allows a natural mechanism to survive insufficient data or poorly distributed data by formulating linear regression using probability distributors rather than point estimates.

Import Libraries

```
# Import Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

# MATPLOTLIB & SEABORN FOR GRAPH-PLOTTING
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Load Data into the Data Frame

```
import pandas as pd

# Path to your CSV file in Google Drive
file_path = '/content/drive/MyDrive/yahoo_data.csv'
```

```python
# Read the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame and format the date column
df.head().style.format({'Date': lambda x: pd.to_datetime(x).strftime('%Y-%m-%d')})
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2023-04-28	33,797.43	34,104.56	33,728.40	34,098.16	34,098.16	354,310,000
1	2023-04-27	33,381.66	33,859.75	33,374.65	33,826.16	33,826.16	343,240,000
2	2023-04-26	33,596.34	33,645.83	33,235.85	33,301.87	33,301.87	321,170,000
3	2023-04-25	33,828.34	33,875.49	33,525.39	33,530.83	33,530.83	297,880,000
4	2023-04-24	33,805.04	33,891.15	33,726.09	33,875.40	33,875.40	252,020,000

Data Preprocessing

```python
import pandas as pd
import numpy as np

# Assuming 'df' is the DataFrame containing your dataset

# Convert 'Date' column to datetime type
df['Date'] = pd.to_datetime(df['Date'])

# Remove commas and convert columns to numeric type
df['Open'] = df['Open'].replace(',', '', regex=True).astype(float)
df['High'] = df['High'].replace(',', '', regex=True).astype(float)
df['Low'] = df['Low'].replace(',', '', regex=True).astype(float)
df['Close'] = df['Close'].replace(',', '', regex=True).astype(float)
df['Adj Close'] = df['Adj Close'].replace(',', '', regex=True).astype(float)
df['Volume'] = df['Volume'].replace(',', '', regex=True).astype(int)

# Calculate the new columns based on the modified dataset
df['Increase_Decrease'] = np.where(df['Volume'].shift(-1) > df['Volume'], 1, 0)
df['Buy_Sell_on_Open'] = np.where(df['Open'].shift(-1) > df['Open'], 1, 0)
df['Buy_Sell'] = np.where(df['Adj Close'].shift(-1) > df['Adj Close'], 1, 0)
df['Returns'] = df['Adj Close'].pct_change()

# Drop rows with NaN values
df = df.dropna()

# Display the modified dataset
print(df.head())
```

Mounted at /content/drive

	Date	Open	High	Low	Close	Adj Close	Volume
0	2023-04-28	33,797.43	34,104.56	33,728.40	34,098.16	34,098.16	354,310,000
1	2023-04-27	33,381.66	33,859.75	33,374.65	33,826.16	33,826.16	343,240,000
2	2023-04-26	33,596.34	33,645.83	33,235.85	33,301.87	33,301.87	321,170,000
3	2023-04-25	33,828.34	33,875.49	33,525.39	33,530.83	33,530.83	297,880,000
4	2023-04-24	33,805.04	33,891.15	33,726.09	33,875.40	33,875.40	252,020,000

```
        Date      Open      High       Low     Close  Adj Close     Volume  \
2 2023-04-26  33596.34  33645.83  33235.85  33301.87   33301.87  321170000
3 2023-04-25  33828.34  33875.49  33525.39  33530.83   33530.83  297880000
4 2023-04-24  33805.04  33891.15  33726.09  33875.40   33875.40  252020000
5 2023-04-21  33793.60  33858.83  33688.57  33808.96   33808.96  291080000
6 2023-04-20  33740.60  33875.39  33677.74  33786.62   33786.62  307910000

   Increase_Decrease  Buy_Sell_on_Open  Buy_Sell   Returns
2                  0                 1         1 -0.015500
3                  0                 0         1  0.006875
4                  1                 0         0  0.010276
5                  1                 0         0 -0.001961
6                  0                 1         1 -0.000661
```

```
#View Dataset
df
```

	Date	Open	High	Low	Close	Adj Close	Volume	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Returns
2	2023-04-26	33596.34	33645.83	33235.85	33301.87	33301.87	321170000	0	1	1	-0.015500
3	2023-04-25	33828.34	33875.49	33525.39	33530.83	33530.83	297880000	0	0	1	0.006875
4	2023-04-24	33805.04	33891.15	33726.09	33875.40	33875.40	252020000	1	0	0	0.010276

	Date	Open	High	Low	Close	Adj Close	Volume	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Returns
5	2023-04-21	33793.60	33858.83	33688.57	33808.96	33808.96	291080000	1	0	0	-0.001961
6	2023-04-20	33740.60	33875.39	33677.74	33786.62	33786.62	307910000	0	1	1	-0.000661
...	...	...	...	...	...	...	...	...	...	...	...
1253	2018-05-07	24317.66	24479.45	24263.42	24357.32	24357.32	307670000	1	0	0	-0.000119
1254	2018-05-04	23865.22	24333.35	23778.87	24262.51	24262.51	329480000	1	0	0	-0.003892
1255	2018-05-03	23836.23	23996.15	23531.31	23930.15	23930.15	389240000	0	1	0	-0.013699
1256	2018-05-02	24097.63	24185.52	23886.30	23924.98	23924.98	385350000	0	1	1	-0.000216
1257	2018-05-01	24117.29	24117.29	23808.19	24099.05	24099.05	380070000	0	0	0	0.007276

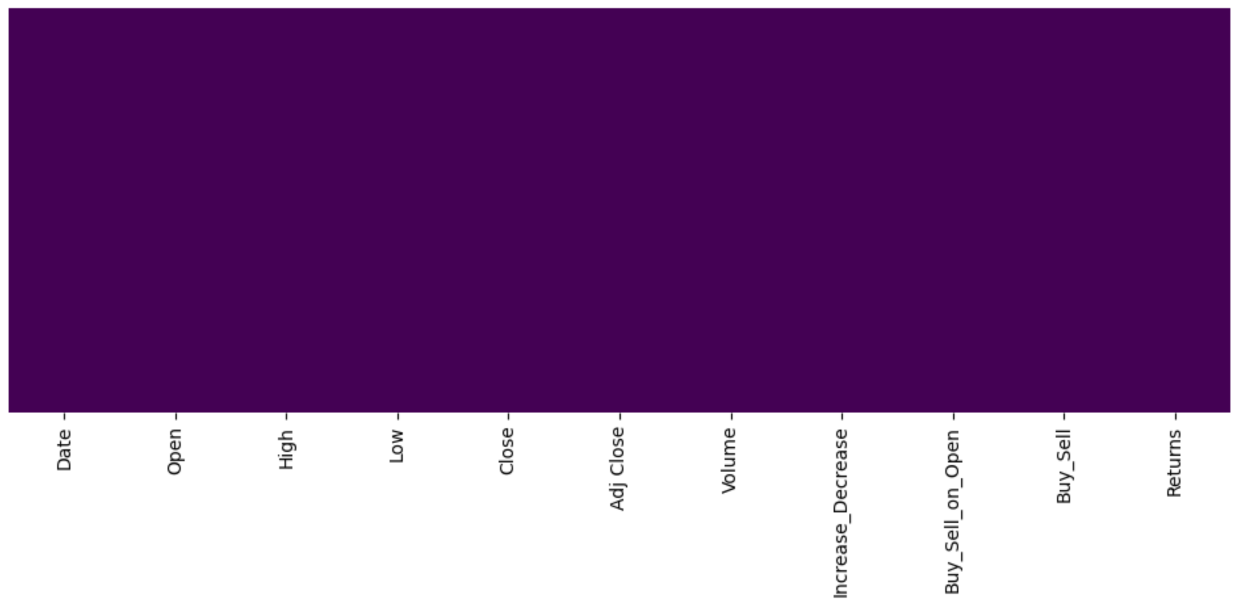
1256 rows × 11 columns

Dataset Cleaning and Null Value Testing

```
# See how many null values in each column

df.isnull().sum()
```

```
Date                 0
Open                 0
High                 0
Low                  0
Close                0
Adj Close            0
Volume               0
Increase_Decrease    0
Buy_Sell_on_Open     0
Buy_Sell             0
Returns              0
dtype: int64
```

```
plt.figure(figsize=(12,4))
sns.heatmap(df.isnull(), yticklabels=False,cbar=False,cmap='viridis')
plt.savefig("Figure 1: Heatmap for Null Values")
```



Analyze the Data

```
# see number of rows, number of columns
df.shape
```

```
(1256, 11)
```

```
#TOTAL NUMBER OF RECORDS
df.size
print("Total number of records = ",df.size)
```

```
Total number of records =  13816
```

```
# see columns names

df.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
       'Increase_Decrease', 'Buy_Sell_on_Open', 'Buy_Sell', 'Returns'],
      dtype='object')
```

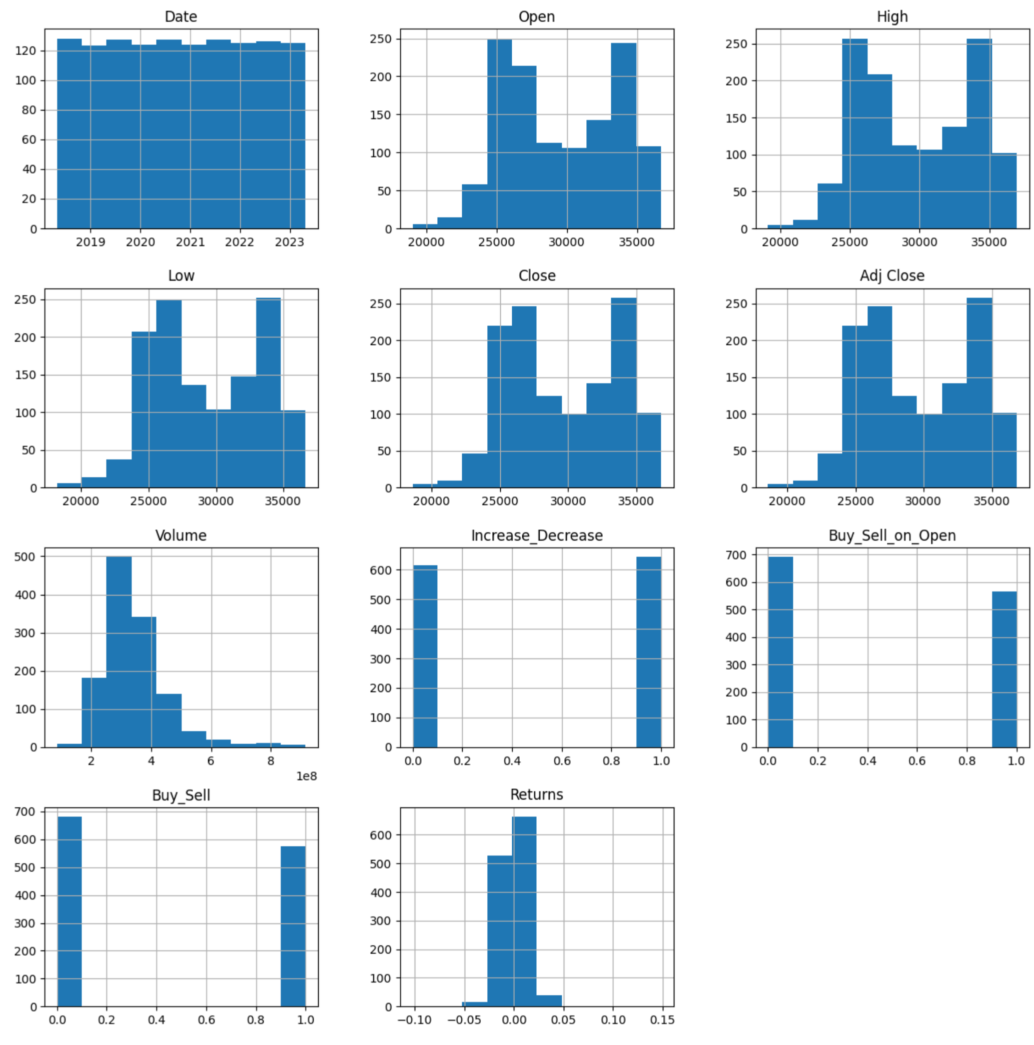
```
#View Data Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1256 entries, 2 to 1257
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Date               1256 non-null   datetime64[ns]
 1   Open               1256 non-null   float64
 2   High               1256 non-null   float64
 3   Low                1256 non-null   float64
 4   Close              1256 non-null   float64
 5   Adj Close          1256 non-null   float64
 6   Volume             1256 non-null   int64
 7   Increase_Decrease  1256 non-null   int64
 8   Buy_Sell_on_Open   1256 non-null   int64
 9   Buy_Sell           1256 non-null   int64
 10  Returns            1256 non-null   float64
dtypes: datetime64[ns](1), float64(6), int64(4)
```

```
# Histogram per each numerical column
```

```
df.hist(figsize=(15, 15))
```

```
array([[<Axes: title={'center': 'Date'}>,
        <Axes: title={'center': 'Open'}>,
        <Axes: title={'center': 'High'}>],
       [<Axes: title={'center': 'Low'}>,
        <Axes: title={'center': 'Close'}>,
        <Axes: title={'center': 'Adj Close'}>],
       [<Axes: title={'center': 'Volume'}>,
        <Axes: title={'center': 'Increase_Decrease'}>,
        <Axes: title={'center': 'Buy_Sell_on_Open'}>],
       [<Axes: title={'center': 'Buy_Sell'}>,
        <Axes: title={'center': 'Returns'}>, <Axes: >]], dtype=object)
```




```
# The statistics per each column
```

```
df.describe()
```

	Open	High	Low	Close	Adj Close	Volume	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Returns
count	1256.000000	1256.000000	1256.000000	1256.000000	1256.000000	1.256000e+03	1256.000000	1256.000000	1256.000000	1256.000000
mean	29589.463615	29770.249546	29395.825390	29592.414546	29592.414546	3.450577e+08	0.511146	0.450637	0.457006	-0.000178
std	4006.084312	4008.678662	4004.703535	4006.867706	4006.867706	1.069990e+08	0.500075	0.497756	0.498347	0.013667
min	19028.360000	19121.010000	18213.650000	18591.930000	18591.930000	8.615000e+07	0.000000	0.000000	0.000000	-0.102052
25%	26039.980000	26162.082500	25875.690000	26025.980000	26025.980000	2.772025e+08	0.000000	0.000000	0.000000	-0.006232
50%	29172.725000	29325.180000	28981.320000	29191.155000	29191.155000	3.245750e+08	1.000000	0.000000	0.000000	-0.000710
75%	33598.902500	33812.642500	33342.837500	33596.937500	33596.937500	3.876200e+08	1.000000	1.000000	1.000000	0.004718
max	36722.600000	36952.650000	36636.000000	36799.650000	36799.650000	9.159900e+08	1.000000	1.000000	1.000000	0.148456

Define X and Y

```
X = df['Open'].values.reshape(1256,-1)
y = df['Adj Close'].values.reshape(1256,-1)
```

```
from sklearn.linear_model import BayesianRidge, LinearRegression

# Fit the Bayesian Ridge Regression and an OLS for comparison
model = BayesianRidge(compute_score=True)
model.fit(X, y)
```

```
▾ BayesianRidge
BayesianRidge(compute_score=True)
```

```
BayesianRidge(compute_score=True)
```

```
model.coef_
```

```
array([0.99778593])
```

```
model.scores_
```

```
array([-11580.25045677,  -8855.55533932,  -8855.55533381])
```

Split Train Data and Test Data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Dataset Traning and Model Traning

```
model = BayesianRidge(compute_score=True)
model.fit(X_train, y_train)
```

```
▾ BayesianRidge
BayesianRidge(compute_score=True)
```

```
BayesianRidge(compute_score=True)
```

```
model.coef_
```

```
array([0.99834636])
```

```
model.scores_
```

```
array([-8676.11227658, -6619.41180156, -6619.4117878 , -6619.4117878 ])
```

Comparison of Actual Values and Predicted Values

```
y_pred = model.predict(X_test)
```

Accuracy and Loss Function Values of the Model

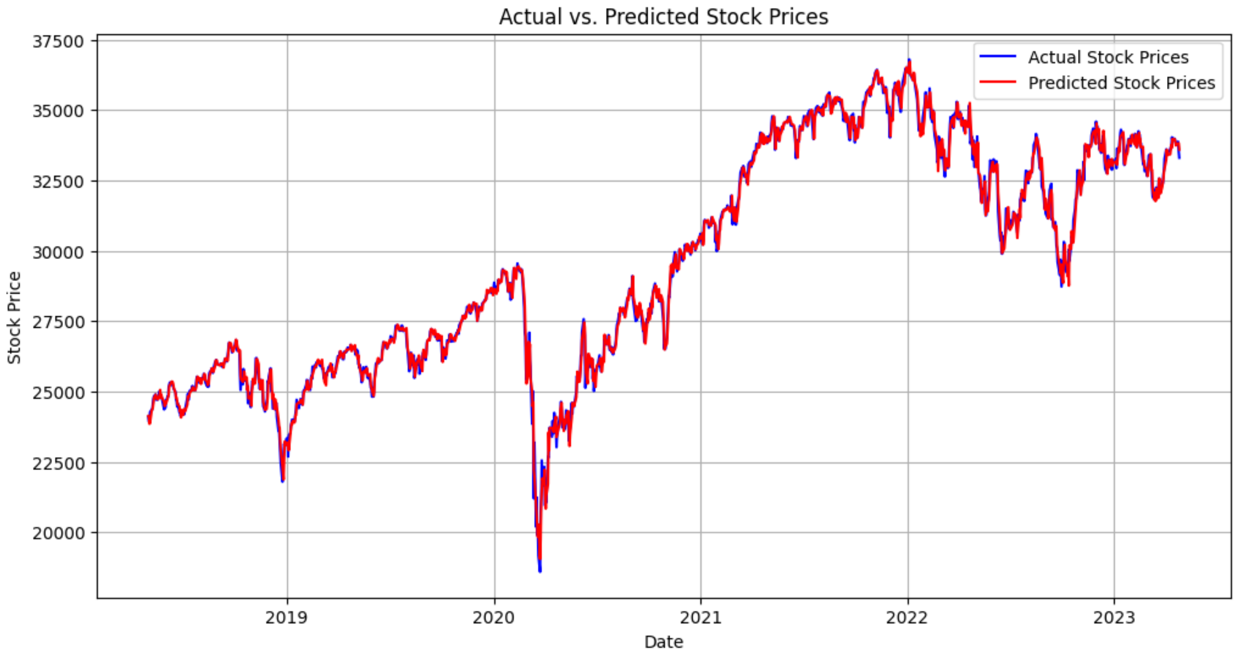
```
from sklearn import metrics
print('Mean_Absolute_Error(MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Mean_Squared_Error(MSE):', metrics.mean_squared_error(y_test, y_pred))
print('Root_Mean_Squared_Error(RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean_Absolute_Error(MAE): 212.1244901111817
Mean_Squared_Error(MSE): 88499.93282967183
Root_Mean_Squared_Error(RMSE): 297.48938271755486
```

```
print('Accuracy Score:', model.score(X_test, y_test))
```

```
Accuracy Score: 0.9947976502144797
```

```
import matplotlib.pyplot as plt

# Plotting the actual stock prices
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Adj Close'], label='Actual Stock Prices', color='blue')

# Plotting the predicted stock prices for the entire dataset
plt.plot(df['Date'], model.predict(X), label='Predicted Stock Prices', color='red')

plt.title('Actual vs. Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid(True)
plt.show()
```


Actual vs. Predicted Stock Prices
Actual Stock Prices
Predicted Stock Prices
Stock Price
Date
37500
35000
32500
30000
27500
25000
22500
20000
2019
2020
2021
2022
2023

10.3.2 Linear Regression Model – IT20613204 – Kodithuwakku D.R.G.C.W

```
from google.colab import drive
drive.mount('/content/drive')



import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore")



# Path to your CSV file in Google Drive
file_path = '/content/drive/MyDrive/yahoo_data.csv'

# Read the CSV file into a DataFrame
dataset = pd.read_csv(file_path)

# Display the first few rows of the DataFrame and format the date column
dataset.head().style.format({'Date': lambda x: pd.to_datetime(x).strftime('%Y-%m-
%d')})
```

Date	Open	High	Low	Close	Adj Close	Volume

0	2023-04-28	33,797.43	34,104.56	33,728.40	34,098.16	34,098.16	354,310,000
1	2023-04-27	33,381.66	33,859.75	33,374.65	33,826.16	33,826.16	343,240,000
2	2023-04-26	33,596.34	33,645.83	33,235.85	33,301.87	33,301.87	321,170,000
3	2023-04-25	33,828.34	33,875.49	33,525.39	33,530.83	33,530.83	297,880,000
4	2023-04-24	33,805.04	33,891.15	33,726.09	33,875.40	33,875.40	252,020,000

Data Preprocessing

```
import pandas as pd
import numpy as np

# Assuming 'df' is the DataFrame containing your dataset

# Convert 'Date' column to datetime type
dataset['Date'] = pd.to_datetime(dataset['Date'])

# Remove commas and convert columns to numeric type
dataset['Open'] = dataset['Open'].replace(',', '', regex=True).astype(float)
dataset['High'] = dataset['High'].replace(',', '', regex=True).astype(float)
dataset['Low'] = dataset['Low'].replace(',', '', regex=True).astype(float)
dataset['Close'] = dataset['Close'].replace(',', '', regex=True).astype(float)
dataset['Adj Close'] = dataset['Adj Close'].replace(',', '',
regex=True).astype(float)
dataset['Volume'] = dataset['Volume'].replace(',', '', regex=True).astype(int)

# Calculate the new columns based on the modified dataset
dataset['Open_Close'] = (dataset['Open'] - dataset['Adj Close']) /
dataset['Open']
dataset['High_Low'] = (dataset['High'] - dataset['Low']) / dataset['Low']
dataset['Increase_Decrease'] = np.where(dataset['Volume'].shift(-1) >
dataset['Volume'], 1, 0)
dataset['Buy_Sell_on_Open'] = np.where(dataset['Open'].shift(-1) >
dataset['Open'], 1, 0)
```

```python
dataset['Buy_Sell'] = np.where(dataset['Adj Close'].shift(-1) > dataset['Adj Close'], 1, 0)

dataset['Returns'] = dataset['Adj Close'].pct_change()


# Drop rows with NaN values

dataset = dataset.dropna()


# Display the modified dataset

print(dataset.head())



#View Dataset

dataset
```

	Date	Open	High	Low	Close	Adj Close	Volume	Open_Close	High_Low	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Returns
1	2023-04-27	33381.66	33859.75	33374.65	33826.16	33826.16	343240000	-0.013316	0.014535	0	1	0	-0.007977
2	2023-04-26	33596.34	33645.83	33235.85	33301.87	33301.87	321170000	0.008765	0.012335	0	1	1	-0.015500
3	2023-04-25	33828.34	33875.49	33525.39	33530.83	33530.83	297880000	0.008795	0.010443	0	0	1	0.006875
4	2023-04-24	33805.04	33891.15	33726.09	33875.40	33875.40	252020000	-0.002081	0.004894	1	0	0	0.010276
5	2023-04-21	33793.60	33858.83	33688.57	33808.96	33808.96	291080000	-0.000455	0.005054	1	0	0	-0.001961

...	...	...	...	...	...	...	...	...	...	...	...	...	...
1253	2018-05-07	24317.66	24479.45	24263.42	24357.32	24357.32	307670000	-0.001631	0.008904	1	0	0	-0.000119
1254	2018-05-04	23865.22	24333.35	23778.87	24262.51	24262.51	329480000	-0.016647	0.023318	1	0	0	-0.003892
1255	2018-05-03	23836.23	23996.15	23531.31	23930.15	23930.15	389240000	-0.003940	0.019754	0	1	0	-0.013699
1256	2018-05-02	24097.63	24185.52	23886.30	23924.98	23924.98	385350000	0.007165	0.012527	0	1	1	-0.000216
1257	2018-05-01	24117.29	24117.29	23808.19	24099.05	24099.05	380070000	0.000756	0.012983	0	0	0	0.007276

Dataset Cleaning and Null Value Testing

```
# See how many null values in each column


dataset.isnull().sum()
```

[35]

0s

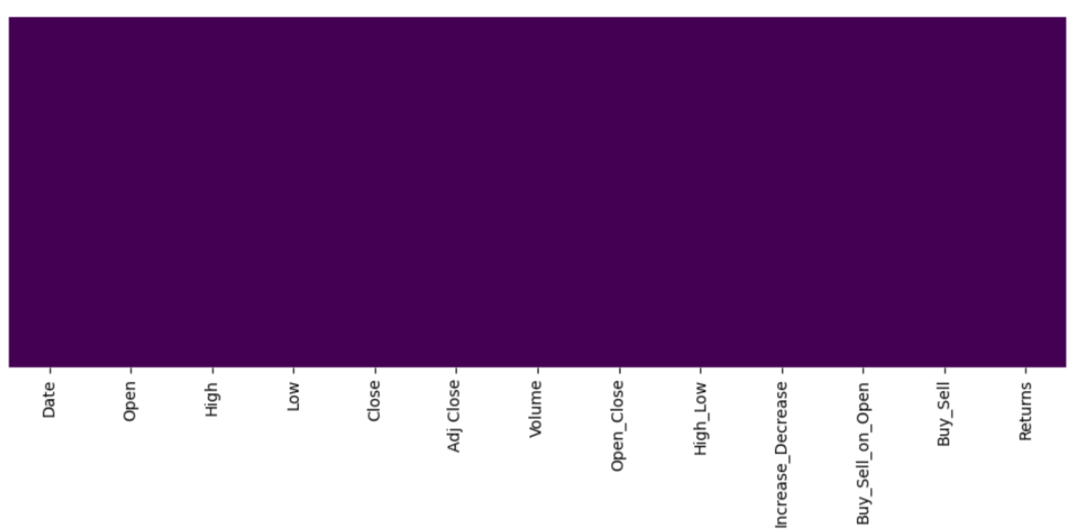
```
# See how many null values in each column
dataset.isnull().sum()
```

output

```
Date            0
Open            0
High            0
Low             0
Close           0
Adj Close       0
Volume          0
Open_Close      0
```

```
High_Low             0
Increase_Decrease    0
Buy_Sell_on_Open     0
Buy_Sell             0
Returns              0
```

```python
plt.figure(figsize=(12,4))
sns.heatmap(dataset.isnull(), yticklabels=False,cbar=False,cmap='viridis')
plt.savefig("Figure 1: Heatmap for Null Values")
```



Analyze the Dataset

```python
# view number of rows, number of columns
dataset.shape
```

```
(1257, 13)
```

```python
# total number of records
dataset.size
```

```python
print("Total number of records = ",dataset.size)
```

```
Total number of records = 16341
```

```python
# view columns names
dataset.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
       'Open_Close', 'High_Low', 'Increase_Decrease', 'Buy_Sell_on_Open',
       'Buy_Sell', 'Returns'],
      dtype='object')
```

```python
# view data types of the columns
dataset.dtypes
```

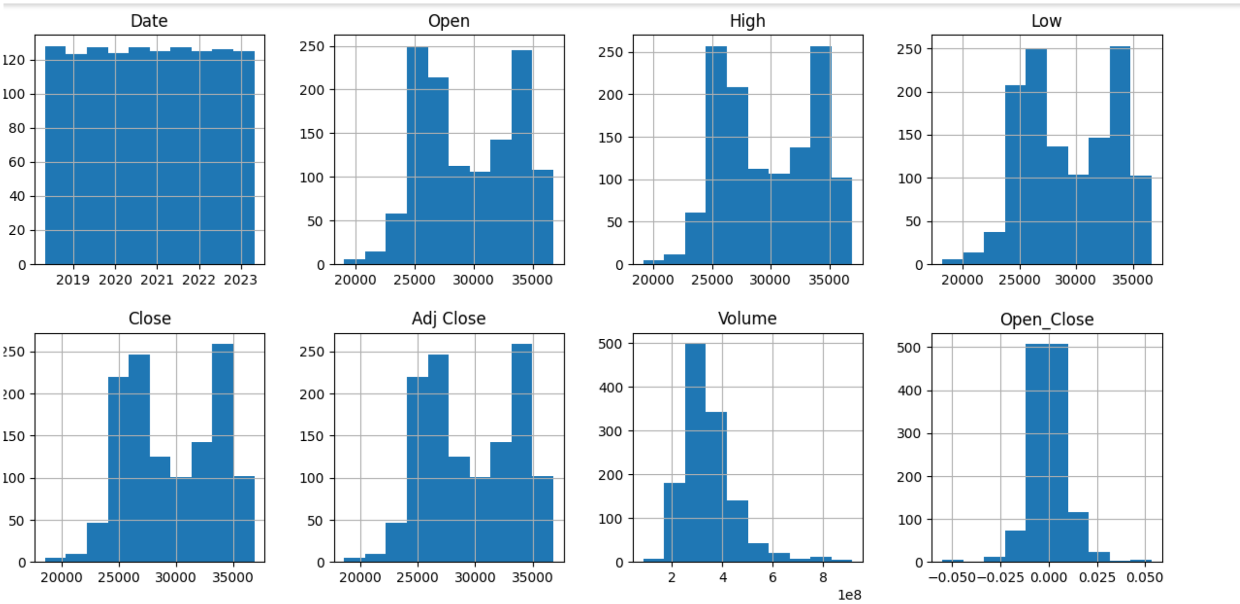
```
Date                 datetime64[ns]
Open                        float64
High                        float64
Low                         float64
Close                       float64
Adj Close                   float64
Volume                        int64
Open_Close                  float64
High_Low                    float64
Increase_Decrease             int64
Buy_Sell_on_Open              int64
Buy_Sell                      int64
Returns                     float64
```

```
# view dataset info
dataset.info()



# view the statistics per each column
dataset.describe()
```

	Date	Open	High	Low	Close	Adj Close	Volume	Open_Close	High_Low	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Returns
count	1257	1257.000000	1257.000000	1257.000000	1257.000000	1257.000000	1.257000e+03	1257.000000	1257.000000	1257.000000	1257.000000	1257.000000	1257.000000
mean	2020-10-27 16:14:53.556086016	29592.480477	29773.502928	29398.990724	29595.782681	29595.782681	3.450563e+08	-0.000157	0.013052	0.510740	0.451074	0.456643	-0.000184
min	2018-05-01 00:00:00	19028.360000	19121.010000	18213.650000	18591.930000	18591.930000	8.615000e+07	-0.055147	0.002525	0.000000	0.000000	0.000000	-0.102052
25%	2019-07-31 00:00:00	26040.300000	26162.280000	25877.240000	26026.320000	26026.320000	2.772300e+08	-0.005230	0.007149	0.000000	0.000000	0.000000	-0.006289
50%	2020-10-27 00:00:00	29198.920000	29330.160000	28995.660000	29196.040000	29196.040000	3.245800e+08	-0.000522	0.010461	1.000000	0.000000	0.000000	-0.000725
75%	2022-01-26 00:00:00	33596.340000	33817.960000	33343.430000	33597.920000	33597.920000	3.876100e+08	0.004157	0.015958	1.000000	1.000000	1.000000	0.004709
max	2023-04-27 00:00:00	36722.600000	36952.650000	36636.000000	36799.650000	36799.650000	9.159900e+08	0.053284	0.089469	1.000000	1.000000	1.000000	0.148456

std	NaN	4005.917425	4008.742338	4004.681746	4007.052034	4007.052034	1.069564e+08	0.009942	0.009684	0.500084	0.497799	0.498315	0.013664

```
# view the histogram per each numerical column
dataset.hist(figsize=(15, 15))
```



Define X and Y

```
X = dataset[['Open', 'High', 'Low','Volume', 'Open_Close', 'High_Low',
'Returns']]
y = dataset['Adj Close']
```

Split Train Dataset and Test Dataset

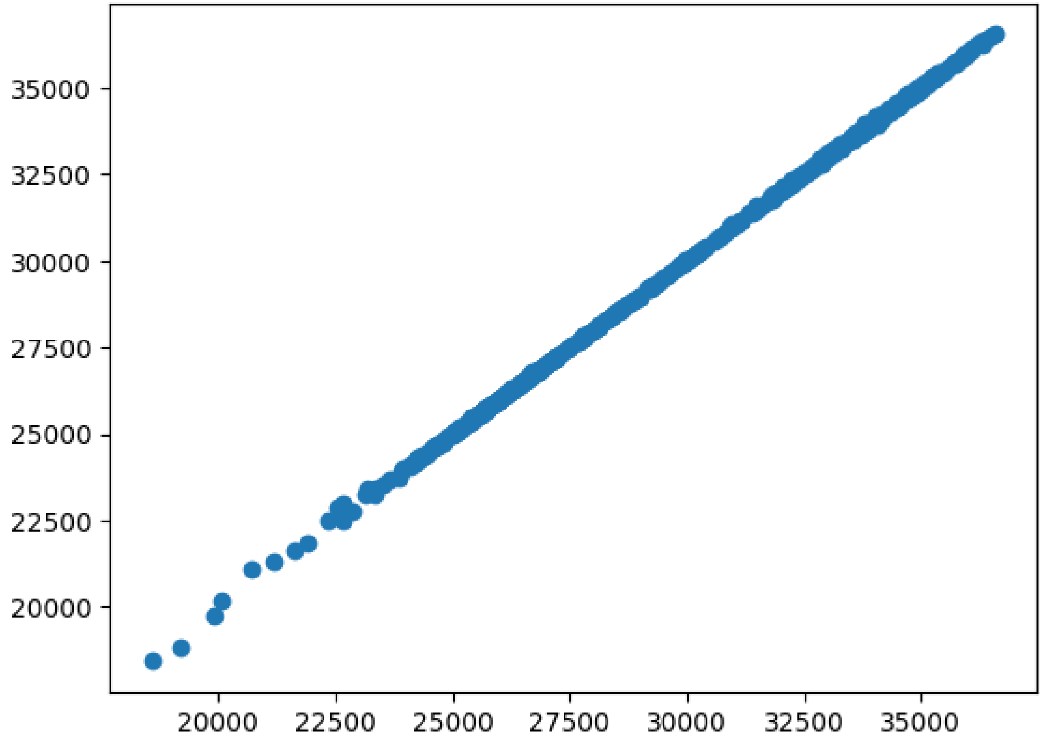
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                   test_size=0.4, random_state=101)
```

Dataset Traning and Model Traning

```
lm = LinearRegression()
lm.fit(X_train,y_train)

print(lm.intercept_)

-27.29076441367215

# Assuming lm.coef_ is an array of coefficients and X_train.columns is the list of feature names
coeff_df = pd.DataFrame(lm.coef_, columns=['Coefficient'], index=X_train.columns)
coeff_df
# Assuming lm.coef_ is an array of coefficients and X_train.columns is the list of feature names
coeff_df = pd.DataFrame(lm.coef_, columns=['Coefficient'], index=X_train.columns)
coeff_df
```

Comparison of Actual Values and Predicted Values

```
y_pred = lm.predict(X_test)
plt.scatter(y_test,y_pred)
plt.savefig("Figure: Comparison of Actual Values and Predictions Values")
```




```
df = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
print(df.head())
print(df.tail())
```

```
        Actual     Predicted
1025  26218.13  26214.107697
1042  25650.88  25699.515051
211   31438.26  31441.567756
1157  26743.50  26746.146514
1     33826.16  33772.665984
       Actual     Predicted
510  33821.30  33849.930692
188  32845.13  32815.078898
192  31990.04  31976.533031
988  25347.77  25328.541531
266  34496.51  34495.039217
```

```
print(y_test.shape)
print(y_pred.shape)
```

```
(503,) (503,)
```

```python
lm_fit = lm.fit(X_train, y_train)
lm_scores = cross_val_score(lm_fit, X_train, y_train, cv = 5)

print("Mean cross validation score: {}".format(np.mean(lm_scores)))
print("Score without cv: {}".format(lm_fit.score(X_train, y_train)))
```

```
Mean cross validation score: 0.9998945905244245
Score without cv: 0.999908520002656
```

Accuracy and Loss Function Values of the Model

```python
print('Mean_Absolute_Error(MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Mean_Squared_Error(MSE):', metrics.mean_squared_error(y_test, y_pred))
print('Root_Mean_Squared_Error(RMSE):',
np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean_Absolute_Error(MAE): 28.26542154620928
 Mean_Squared_Error(MSE): 2482.3811965426908
 Root_Mean_Squared_Error(RMSE): 49.82350044449598
```

```python
print("Accuracy score: {:.7f}".format(lm.score(X_test, y_test)))
```

```
Accuracy score: 0.9998467
```

```python
import matplotlib.pyplot as plt

# Plotting the actual stock prices
```

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

plt.plot(dataset['Date'], dataset['Adj Close'], label='Actual Stock Prices',
color='blue')


# Plotting the predicted stock prices for the entire dataset

plt.plot(dataset['Date'], lm.predict(X), label='Predicted Stock Prices',
color='red')


plt.title('Actual vs. Predicted Stock Prices')

plt.xlabel('Date')

plt.ylabel('Stock Price')

plt.legend()

plt.grid(True)

plt.show()
```

10.3.3 Decision Tree Model – IT20613136 _ Jayarathne A. H. B

Import Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# MATPLOTLIB FOR GRAPH-PLOTTING
import matplotlib.pyplot as plt
%matplotlib inline
```

Load Dataset

```
DATASET_FILE_PATH = './yahoo_data.csv'
```

```
# Read the CSV file into a DataFrame
df = pd.read_csv(DATASET_FILE_PATH)

# Display the first few rows of the DataFrame and format the date column
df.head().style.format({'Date': lambda x: pd.to_datetime(x).strftime('%Y-
%m-%d')})
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2023-04-28	33,797.43	34,104.56	33,728.40	34,098.16	34,098.16	354,310,000
1	2023-04-27	33,381.66	33,859.75	33,374.65	33,826.16	33,826.16	343,240,000
2	2023-04-26	33,596.34	33,645.83	33,235.85	33,301.87	33,301.87	321,170,000
3	2023-04-25	33,828.34	33,875.49	33,525.39	33,530.83	33,530.83	297,880,000
4	2023-04-24	33,805.04	33,891.15	33,726.09	33,875.40	33,875.40	252,020,000

Data Preprocessing

```
# Convert 'Date' column to datetime type
df['Date'] = pd.to_datetime(df['Date'])

# Remove commas and convert columns to numeric type
df['Open'] = df['Open'].replace(',', '', regex=True).astype(float)
```

```
df['High'] = df['High'].replace(',', '', regex=True).astype(float)
df['Low'] = df['Low'].replace(',', '', regex=True).astype(float)
df['Close'] = df['Close'].replace(',', '', regex=True).astype(float)
df['Adj Close'] = df['Adj Close'].replace(',', '',
regex=True).astype(float)
df['Volume'] = df['Volume'].replace(',', '', regex=True).astype(int)
```

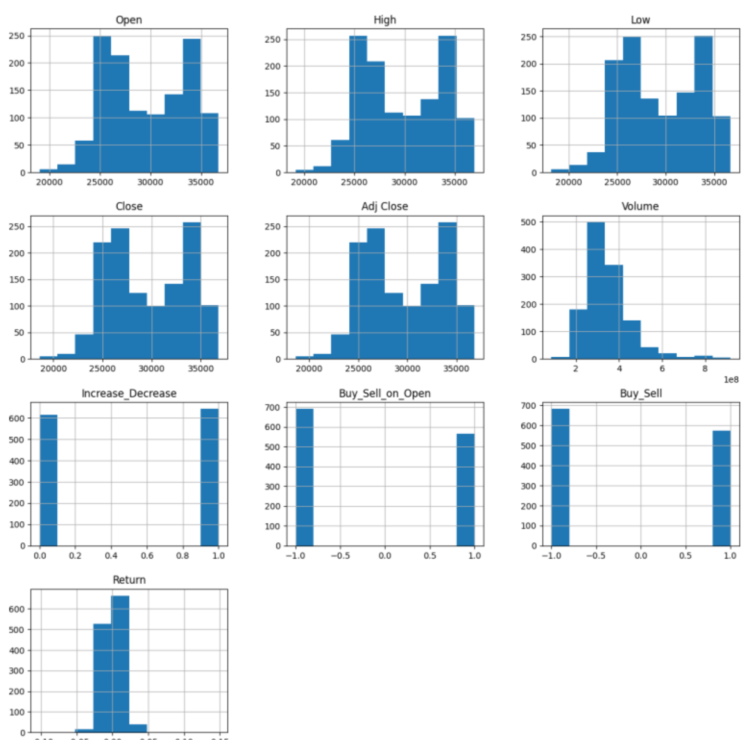
```
df['Increase_Decrease'] = np.where(df['Volume'].shift(-1) > df['Volume'],
1, 0)
df['Buy_Sell_on_Open'] = np.where(df['Open'].shift(-1) > df['Open'], 1, -
1)
df['Buy_Sell'] = np.where(df['Adj Close'].shift(-1) > df['Adj Close'], 1,
-1)
df['Return'] = df['Adj Close'].pct_change()
```

Dataset cleaning and null value testing

```
df = df.dropna()
df.isnull().sum()
```

Analyze the dataset

```
# Histogram per each numerical column
sort_columns = df.iloc[:, 1:]
sort_columns.hist(figsize=(15, 15))
# plt.savefig('histogram.png')
```



```python
# The statistics per each column
sort_columns.describe()
```

Open	High	Low	Close	Adj Close	Volume	Increase_Decrease	Buy_Sell_on_Open	Buy_Sell	Return	
count	1257.000000	1257.000000	1257.000000	1257.000000	1257.000000	1.257000e+03	1257.000000	1257.000000	1257.000000	1257.000000
mean	29592.480477	29773.502928	29398.990724	29595.782681	29595.782681	3.450563e+08	0.510740	-0.097852	-0.086714	-0.000184
std	4005.917425	4008.742338	4004.681746	4007.052034	4007.052034	1.069564e+08	0.500084	0.995597	0.996630	0.013664
min	19028.360000	19121.010000	18213.650000	18591.930000	18591.930000	8.615000e+07	0.000000	-1.000000	-1.000000	-0.102052
25%	26040.300000	26162.280000	25877.240000	26026.320000	26026.320000	2.772300e+08	0.000000	-1.000000	-1.000000	-0.006289
50%	29198.920000	29330.160000	28995.660000	29196.040000	29196.040000	3.245800e+08	1.000000	-1.000000	-1.000000	-0.000725
75%	33596.340000	33817.960000	33343.430000	33597.920000	33597.920000	3.876100e+08	1.000000	1.000000	1.000000	0.004709
max	36722.600000	36952.650000	36636.000000	36799.650000	36799.650000	9.159900e+08	1.000000	1.000000	1.000000	0.148456

Define inputs (X) & targets(Y)

```python
X = df.drop(['Date', 'Adj Close', 'Close'], axis=1)
y = df['Adj Close']
```

Split dataset into train set & validation set

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

Model Training

```python
model = DecisionTreeRegressor(random_state=0)
model.fit(X_train, y_train)
```

Comparison of Actual Values and Predictions Values

```python
y_pred = model.predict(X_test)
```

```python
adj_value = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
print(adj_value.head(), "\n")
print(adj_value.tail())
```

```
      Actual  Predicted
6    33786.62   33745.40
495  34269.16   34496.51
53   33869.27   33826.69
985  24815.04   25014.86
187  32798.40   32953.46

        Actual  Predicted
1103  24423.26   24133.78
32    32155.40   32246.55
409   34869.63   34921.88
65    33743.84   33733.96
1030  25625.59   25473.23
```

Accuracy and Loss function values of the model

```
from sklearn import metrics
print('Mean_Absolute_Error(MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Mean_Squared_Error(MSE):', metrics.mean_squared_error(y_test, y_pred))
print('Root_Mean_Squared_Error(RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean_Absolute_Error(MAE): 139.08472222222218
Mean_Squared_Error(MSE): 49705.407550396805
Root_Mean_Squared_Error(RMSE): 222.94709585548947
```

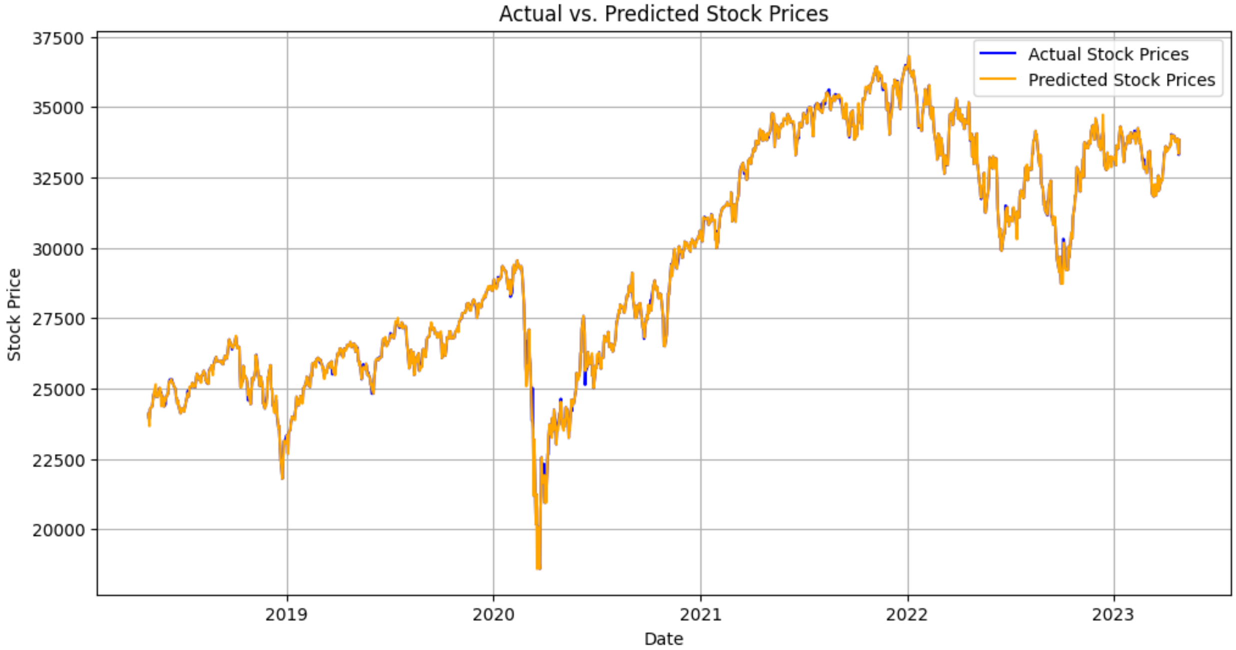
```
from sklearn.model_selection import cross_val_score

dt_fit = model.fit(X_train, y_train)
dt_scores = cross_val_score(dt_fit, X_train, y_train, cv = 5)
```

```
print("Accuracy score: {:.7f}".format(model.score(X_test, y_test)))
```

```
Accuracy score: 0.9970342
```

Plot the Actual and Predicted stock prices

```
# Plotting the actual stock prices
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Adj Close'], label='Actual Stock Prices',
color='blue')

# Plotting the predicted stock prices for the entire dataset
plt.plot(df['Date'], model.predict(X), label='Predicted Stock Prices',
color='orange')

plt.title('Actual vs. Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid(True)
plt.show()
```



10.3.4 – Logistic Regression Model – IT20609030 – Karunanayake M.L

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

import warnings
warnings.filterwarnings("ignore")

# MATPLOTLIB & SEABORN FOR GRAPH-PLOTTING
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Path to your CSV file in Google Drive
file_path = '/content/drive/MyDrive/yahoo_data.csv'

# Read the CSV file into a DataFrame
dataset = pd.read_csv(file_path)

# Display the first few rows of the DataFrame and format the date column
dataset.head().style.format({'Date': lambda x:
pd.to_datetime(x).strftime('%Y-%m-%d')})
```

```
# Convert 'Date' column to datetime type
dataset['Date'] = pd.to_datetime(dataset['Date'])

# Remove commas and convert columns to numeric type
dataset['Open'] = dataset['Open'].replace(',', '',
regex=True).astype(float)
dataset['High'] = dataset['High'].replace(',', '',
regex=True).astype(float)
dataset['Low'] = dataset['Low'].replace(',', '', regex=True).astype(float)
dataset['Close'] = dataset['Close'].replace(',', '',
regex=True).astype(float)
dataset['Adj Close'] = dataset['Adj Close'].replace(',', '',
regex=True).astype(float)
dataset['Volume'] = dataset['Volume'].replace(',', '',
regex=True).astype(int)
dataset['Buy_Sell'] = np.where(dataset['Adj Close'].shift(-1) >
dataset['Adj Close'],1,-1)
# See how many null values in each column
dataset.isnull().sum()
plt.figure(figsize=(12,4))
```

```
sns.heatmap(dataset.isnull(), yticklabels=False,cbar=False,cmap='viridis')
plt.savefig("Figure 1: Heatmap for Null Values")
# number of rows and number of columns of the dataset
dataset.shape
#Total number of records in the dataset
print("Total number of records = ", len(dataset))
#Column names
dataset.columns
# Data types of the Columns
dataset.dtypes
# Summary of the dataset
dataset.info()
# Histogram per each numerical column in dataset
dataset.hist(figsize=(15, 15))
# Calculate and display descriptive statistics for each numerical column
dataset.describe()
# Convert the 'Buy_Sell' column to integer data type
dataset['Buy_Sell'] = dataset['Buy_Sell'].astype('int')
# Define X
# Select the columns 'Open', 'High', 'Low', 'Adj Close', 'Volume' from the
dataset
# and convert them into a NumPy array X
X = np.asarray(dataset[['Open', 'High', 'Low', 'Adj Close', 'Volume']])

# Display the first 5 rows of the array X
X[0:5]
# Define y
# Select the 'Buy_Sell' column from the dataset
# and convert it into a NumPy array y
y = np.asarray(dataset['Buy_Sell'])

# Display the first 5 elements of the array y
y[0:5]
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.25, random_state = 0)

print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
```

```python
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
# Predicting the Test set results
y_pred = LR.predict(X_test)
y_pred
# Calculate the predicted probabilities for each class (0 and 1) using the
test data
y_pred_prob = LR.predict_proba(X_test)

# Display the predicted probabilities
y_pred_prob
# Create a DataFrame to display y_test and y_pred
results_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Display the DataFrame
print(results_df)
from sklearn.metrics import confusion_matrix, classification_report

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
# Print a classification report showing precision, recall, F1-score, and
other metrics
print(classification_report(y_test, y_pred))
from sklearn import metrics

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
# Calculate and print the log loss between the true labels and predicted
probabilities.
from sklearn.metrics import log_loss
log_loss(y_test, y_pred_prob)
# Calculate ROC curve and AUC for the logistic regression model
y_pred_proba = LR.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
```

```
# Plot ROC curve
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
print("LogLoss: : %.2f" % log_loss(y_test, y_pred_prob))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```