Portfolio Management

1. Business Understanding

Business Understanding of the problem which addresses the following questions.

1. What is the business problem that you are trying to solve?
2. What data do you need to answer the above problem?
3. What are the different sources of data?
4. What kind of analytics task are you performing?

1. **Business problem** To develop an intelligent portfolio management system that leverages data analysis, feature engineering, and machine learning to provide investors with actionable insights for informed decision-making and risk mitigation in the stock market.

- **Core Functionalities:**

Comprehensive Stock Analysis: Evaluate individual stocks using financial ratios, Beta, and historical prices to assess their financial health, risk, and potential returns.

- **Portfolio Optimization:**

Employ correlation analysis and machine learning to build diversified portfolios that minimize risk and maximize returns.

- **Decision Support:**

Offer data-driven recommendations on stock selection, portfolio allocation, and risk management strategies.

1. The data includes key financial ratios like PE Ratio, Dividend Yield, Return on Equity, and Debt to Equity Ratio which can be used to evaluate the financial health and potential of different companies for investment.

   Beta can help assess the systematic risk associated with a particular stock compared to the overall market.

   52 Week High and 52 Week Low provide insights into the stock's recent price performance and potential volatility.

2. **Dataset:** We have used a program for random generation of the dataset with 20 attributes and 11500 entries.

3. **Analytic tasks used are:**

- Data Preparation- Identified and removed data inconsistencies
- Data Exploration- Using Scatter plot, histogram, heatmaps, pair plot etc

- Data Wrangling- Using mutual information, gini index, gain ration, Chi square tests and strength of association
- Use of ML techniques- Classification & Clustering to help us management of the portfolio

2. Data Acquisition

For the problem identified , Data set we are taking are unique with minimum **20 features and 10k rows**) from any public data source.

2.1 Download the data directly

```
## we have used the below code to generate random data for us ##

import pandas as pd
import numpy as np
import random

# Generate data for 11000 stocks
num_stocks = 11000
data = {
    'Company Name': [f'Company_{i}' for i in range(1, num_stocks +
1)],
    'Industry': [random.choice(['Technology', 'Finance', 'Healthcare',
'Retail', 'Energy']) for _ in range(num_stocks)],
    'Open': [round(random.uniform(10, 1000), 2) for _ in
range(num_stocks)],
    'Close': [round(random.uniform(10, 1000), 2) for _ in
range(num_stocks)],
    'High': [round(random.uniform(10, 1000), 2) for _ in
range(num_stocks)],
    'Low': [round(random.uniform(10, 1000), 2) for _ in
range(num_stocks)],
    'Volume': [random.randint(10000, 1000000) for _ in
range(num_stocks)],
    'Market Cap': [random.randint(10000000, 1000000000) for _ in
range(num_stocks)],
    'EPS': [round(random.uniform(0, 10), 2) for _ in
range(num_stocks)],
    'PE Ratio': [round(random.uniform(10, 50), 2) for _ in
range(num_stocks)],
    'Dividend Yield': [round(random.uniform(0, 5), 2) for _ in
range(num_stocks)],
    'Debt to Equity Ratio': [round(random.uniform(0, 2), 2) for _ in
range(num_stocks)],
    'Return on Equity': [round(random.uniform(0, 30), 2) for _ in
```

```
range(num_stocks)],
    'Current Ratio': [round(random.uniform(1, 3), 2) for _ in
range(num_stocks)],
    'Quick Ratio': [round(random.uniform(0.5, 2), 2) for _ in
range(num_stocks)],
    'Cash Flow from Operations': [random.randint(1000000, 100000000)
for _ in range(num_stocks)],
    'Free Cash Flow': [random.randint(1000000, 100000000) for _ in
range(num_stocks)],
    'Beta': [round(random.uniform(0.5, 1.5), 2) for _ in
range(num_stocks)],
    '52 Week High': [round(random.uniform(10, 1000), 2) for _ in
range(num_stocks)],
    '52 Week Low': [round(random.uniform(10, 1000), 2) for _ in
range(num_stocks)]
}

df = pd.DataFrame(data)

# Introduce data inconsistencies
# Replace some values with NaN
df.loc[random.sample(range(num_stocks), 1000), 'Open'] = np.nan
df.loc[random.sample(range(num_stocks), 500), 'Close'] = np.nan
df.loc[random.sample(range(num_stocks), 800), 'EPS'] = np.nan
df.loc[random.sample(range(num_stocks), 700), 'PE Ratio'] = np.nan

# Introduce some incorrect data types
df.loc[random.sample(range(num_stocks), 300), 'Volume'] = 'N/A'
df.loc[random.sample(range(num_stocks), 200), 'Market Cap'] = 'Not
Available'

# Duplicate some rows
df = pd.concat([df, df.sample(n=500)], ignore_index=True)
```

```
<ipython-input-33-a89b09529815>:44: FutureWarning: Setting an item of
incompatible dtype is deprecated and will raise in a future error of
pandas. Value 'N/A' has dtype incompatible with int64, please
explicitly cast to a compatible dtype first.
  df.loc[random.sample(range(num_stocks), 300), 'Volume'] = 'N/A'
<ipython-input-33-a89b09529815>:45: FutureWarning: Setting an item of
incompatible dtype is deprecated and will raise in a future error of
pandas. Value 'Not Available' has dtype incompatible with int64,
please explicitly cast to a compatible dtype first.
  df.loc[random.sample(range(num_stocks), 200), 'Market Cap'] = 'Not
Available'
```

2.2 Code for converting the above downloaded data into a dataframe

```
## Note: conversion to df is in previous step itself ##

# Save the DataFrame to an Excel file
df.to_excel('indian_stocks6.xlsx', index=False)
```

2.3 Confirm the data has been correctly by displaying the first 5 and last 5 records.

```
display(df.head(5))
display(df.tail(5))
```


```
{"summary":"{\n  \"name\": \"display(df\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"Company Name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"Company_2\",\n          \"Company_5\",\n          \"Company_3\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Industry\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"Energy\",\n          \"Finance\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Open\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 122.51291850249916,\n
\"min\": 132.44,\n        \"max\": 399.86,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          399.86,\n
223.34\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Close\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 65.15338824650642,\n        \"min\": 486.21,\n
\"max\": 642.99,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          642.99,\n          541.53\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"High\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 134.41797138031805,\n
\"min\": 56.96,\n        \"max\": 346.8,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          346.8,\n
102.31\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Low\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 295.93012089005066,\n        \"min\": 70.42,\n
\"max\": 710.6,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          282.44,\n          677.07\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Volume\",\n      \"properties\":
```

```
{\n        \"dtype\": \"date\",\n        \"min\": 470193,\n
\"max\": 990359,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          990359,\n          470193\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Market Cap\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
24116442,\n        \"max\": 599646935,\n        \"num_unique_values\":
5,\n        \"samples\": [\n          599646935,\n          118949662\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"EPS\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.333555398174369,\n        \"min\": 0.85,\n        \"max\":
4.15,\n        \"num_unique_values\": 5,\n        \"samples\": [\n
4.15,\n          0.85\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"PE
Ratio\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.64410816756887,\n        \"min\": 14.73,\n        \"max\":
18.01,\n        \"num_unique_values\": 4,\n        \"samples\": [\n
15.33,\n          14.73\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Dividend Yield\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.7172168430816442,\n
\"min\": 0.88,\n        \"max\": 2.84,\n        \"num_unique_values\":
5,\n        \"samples\": [\n          2.84,\n          1.98\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Debt to Equity Ratio\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.43218051783947875,\n        \"min\":
0.84,\n        \"max\": 1.83,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.97,\n          1.51\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Return on Equity\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8.480990508189477,\n        \"min\": 5.83,\n        \"max\": 27.78,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          24.23,\n
27.78\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Current Ratio\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.5663302923206563,\n        \"min\":
1.23,\n        \"max\": 2.62,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          2.35,\n          1.87\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Quick Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.2893959225697557,\n        \"min\": 0.98,\n        \"max\": 1.61,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1.57,\n
1.1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Cash Flow from Operations\",\n      \"properties\": {\n
```

```
\"dtype\": \"number\",\n        \"std\": 31935151,\n        \"min\":
15290373,\n        \"max\": 91643737,\n        \"num_unique_values\":
5,\n        \"samples\": [\n          23257338,\n          91643737\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Free Cash Flow\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
31501857,\n        \"min\": 1720999,\n        \"max\": 86993386,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
29821079,\n          86993386\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Beta\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.35400564967243103,\n        \"min\":
0.67,\n        \"max\": 1.48,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.67,\n          0.99\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"52 Week High\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
187.57221790553098,\n        \"min\": 291.39,\n        \"max\":
803.92,\n        \"num_unique_values\": 5,\n        \"samples\": [\n
803.92,\n          487.05\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"52 Week Low\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 294.00310734752446,\n
\"min\": 130.18,\n        \"max\": 866.78,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          866.78,\n
130.18\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

{"repr_error":"0","type":"dataframe"}
```


2.4 Display the column headings, statistical information, description and statistical summary of the data.

```
display(df.columns)
display(df.describe())
display(df.info())
display(f"The dataset has {df.shape[0]} rows and {df.shape[1]}
columns.")
display(df.dtypes)
display("Null data count per column:")
display(df.isnull().sum())

Index(['Company Name', 'Industry', 'Open', 'Close', 'High', 'Low',
'Volume',
       'Market Cap', 'EPS', 'PE Ratio', 'Dividend Yield',
       'Debt to Equity Ratio', 'Return on Equity', 'Current Ratio',
       'Quick Ratio', 'Cash Flow from Operations', 'Free Cash Flow',
'Beta',
```

```
       '52 Week High', '52 Week Low'],
      dtype='object')
```


```
{"summary":"{\n  \"name\": \"display(df\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"Open\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3540.4783945842446,\n        \"min\": 10.09,\n        \"max\":
10452.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
508.0867910447761,\n          511.95,\n          10452.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Close\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 3719.1859778841076,\
n        \"min\": 10.02,\n        \"max\": 10959.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
507.23467560908847,\n          507.58,\n          10959.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"High\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3911.3566402839347,\n        \"min\": 10.21,\n        \"max\":
11500.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
500.85152000000005,\n          500.82,\n          11500.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Low\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 3908.7061289107583,\
n        \"min\": 10.26,\n        \"max\": 11500.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
511.7969834782608,\n          513.55,\n          11500.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"EPS\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3766.513877872484,\n
\"min\": 0.0,\n        \"max\": 10658.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
4.987667479827359,\n          5.0,\n          10658.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PE Ratio\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3799.535178876305,\n        \"min\": 10.01,\n        \"max\":
10774.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
29.97861332838314,\n          30.11,\n          10774.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Dividend Yield\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4065.0369171156676,\n        \"min\": 0.0,\n        \"max\": 11500.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
2.488746086956522,\n          2.45,\n          11500.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Debt to Equity Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4065.532484872779,\n        \"min\": 0.0,\n        \"max\": 11500.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
```

```
0.9984121739130435,\n          0.99,\n          11500.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Return on Equity\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4060.8862309718643,\n        \"min\": 0.0,\n        \"max\": 11500.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
15.009861739130436,\n          14.94,\n          11500.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Current Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4065.2267712358575,\n        \"min\": 0.5759356579108988,\n
\"max\": 11500.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          2.0068686956521735,\n
2.0149999999999997,\n          11500.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Quick Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4065.4645833735804,\n        \"min\": 0.43311176093200693,\n
\"max\": 11500.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          1.245458260869565,\n          1.24,\n
11500.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Cash Flow from Operations\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 34854091.65443419,\n
\"min\": 11500.0,\n        \"max\": 99986477.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
50592881.43669565,\n          50697646.5,\n          11500.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Free Cash Flow\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 34768271.38507448,\n        \"min\":
11500.0,\n        \"max\": 99983013.0,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          50234686.12513044,\n
50153461.0,\n          11500.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Beta\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4065.5476132246204,\n        \"min\":
0.2886198057257535,\n        \"max\": 11500.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
0.9957217391304347,\n          0.99,\n          11500.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"52 Week High\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3910.8460268911263,\n        \"min\": 10.02,\n        \"max\":
11500.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
503.2010069565217,\n          503.065,\n          11500.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"52 Week Low\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

3911.45683888429,\n   \"min\": 10.28,\n   \"max\": 11500.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   500.6211530434782,\n   498.76,\n   11500.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11500 entries, 0 to 11499
Data columns (total 20 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Company Name                11500 non-null  object 
 1   Industry                    11500 non-null  object 
 2   Open                        10452 non-null  float64
 3   Close                       10959 non-null  float64
 4   High                        11500 non-null  float64
 5   Low                         11500 non-null  float64
 6   Volume                      11500 non-null  object 
 7   Market Cap                  11500 non-null  object 
 8   EPS                         10658 non-null  float64
 9   PE Ratio                    10774 non-null  float64
 10  Dividend Yield              11500 non-null  float64
 11  Debt to Equity Ratio        11500 non-null  float64
 12  Return on Equity            11500 non-null  float64
 13  Current Ratio               11500 non-null  float64
 14  Quick Ratio                 11500 non-null  float64
 15  Cash Flow from Operations   11500 non-null  int64  
 16  Free Cash Flow              11500 non-null  int64  
 17  Beta                        11500 non-null  float64
 18  52 Week High                11500 non-null  float64
 19  52 Week Low                 11500 non-null  float64
dtypes: float64(14), int64(2), object(4)
memory usage: 1.8+ MB

None

{"type":"string"}

Company Name                 object
Industry                     object
Open                        float64
Close                       float64
High                        float64
Low                         float64
Volume                       object
Market Cap                   object
EPS                         float64
PE Ratio                    float64
Dividend Yield              float64
Debt to Equity Ratio        float64
```

```
Return on Equity              float64
Current Ratio                 float64
Quick Ratio                   float64
Cash Flow from Operations       int64
Free Cash Flow                  int64
Beta                          float64
52 Week High                  float64
52 Week Low                   float64
dtype: object
```

```
{"type":"string"}
```

```
Company Name                     0
Industry                         0
Open                          1048
Close                          541
High                             0
Low                              0
Volume                           0
Market Cap                       0
EPS                            842
PE Ratio                       726
Dividend Yield                   0
Debt to Equity Ratio             0
Return on Equity                 0
Current Ratio                    0
Quick Ratio                      0
Cash Flow from Operations        0
Free Cash Flow                   0
Beta                             0
52 Week High                     0
52 Week Low                      0
dtype: int64
```

2.5 Write observations from the above.

1. Size of the dataset
2. What type of data attributes are there?
3. Is there any null data that has to be cleaned?

3. Data Preparation

3.1 Check for

- duplicate data
- missing data
- data inconsistencies

```
duplicate_rows = df[df.duplicated()]
display("Number of duplicate rows:", duplicate_rows.shape[0])
display(duplicate_rows)

missing_data = df.isnull().sum()
display(missing_data)

non_numeric_volume = df[~df['Volume'].apply(lambda x: isinstance(x,
(int, float)))]
display("Rows with non-numeric values in 'Volume' column:")
display(non_numeric_volume)

non_numeric_market_cap = df[~df['Market Cap'].apply(lambda x:
isinstance(x, (int, float)))]
display("Rows with non-numeric values in 'Market Cap' column:")
display(non_numeric_market_cap)
```


```
{"type":"string"}
```


```
500
```


```
{"summary":"{\n  \"name\": \"duplicate_rows\",\n  \"rows\": 500,\n
\"fields\": [\n    {\n      \"column\": \"Company Name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 500,\n        \"samples\": [\n
\"Company_647\",\n          \"Company_3972\",\n
\"Company_1304\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Industry\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"Retail\",\n          \"Finance\",\n
\"Technology\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Open\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 286.5684203061002,\n        \"min\": 13.1,\n        \"max\":
999.94,\n        \"num_unique_values\": 450,\n        \"samples\": [\n
370.96,\n          752.51,\n          648.67\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Close\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 284.889075315621,\n
\"min\": 12.38,\n        \"max\": 998.34,\n
\"num_unique_values\": 459,\n        \"samples\": [\n          62.7,\n
323.82,\n          840.65\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"High\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 284.73820510204996,\n        \"min\":
10.64,\n        \"max\": 999.89,\n        \"num_unique_values\": 499,\
n        \"samples\": [\n          574.73,\n          14.05,\n
295.06\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"Low\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 289.2764535102166,\n        \"min\": 10.83,\n        \"max\":
997.26,\n        \"num_unique_values\": 500,\n        \"samples\": [\n
397.82,\n          914.39,\n          371.97\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Volume\",\n      \"properties\":
{\n        \"dtype\": \"string\",\n        \"num_unique_values\":
491,\n        \"samples\": [\n          864983,\n          923655,\n
400273\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Market Cap\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 491,\n        \"samples\":
[\n          908421322,\n          637442577,\n          900247433\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"EPS\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 2.8468221527574222,\
n        \"min\": 0.01,\n        \"max\": 9.99,\n
\"num_unique_values\": 370,\n        \"samples\": [\n          4.06,\n
0.82,\n          1.91\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"PE
Ratio\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 11.472727761467175,\n        \"min\": 10.12,\n
\"max\": 49.89,\n        \"num_unique_values\": 451,\n
\"samples\": [\n          40.89,\n          14.52,\n          27.38\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Dividend Yield\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.4356967872534436,\n        \"min\": 0.01,\n        \"max\": 4.97,\n
\"num_unique_values\": 320,\n        \"samples\": [\n          1.99,\n
3.03,\n          0.14\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Debt to Equity Ratio\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.588677374886624,\n        \"min\":
0.0,\n        \"max\": 1.99,\n        \"num_unique_values\": 184,\n
\"samples\": [\n          0.0,\n          1.37,\n          0.8\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Return on Equity\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8.848923148574563,\n        \"min\": 0.0,\n        \"max\": 29.97,\n
\"num_unique_values\": 454,\n        \"samples\": [\n          6.08,\n
1.72,\n          7.09\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Current Ratio\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.5740396874132058,\n        \"min\":
1.0,\n        \"max\": 2.99,\n        \"num_unique_values\": 186,\n
\"samples\": [\n          2.28,\n          2.83,\n          2.27\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Quick Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

0.43384661548932596,\n   \"min\": 0.5,\n   \"max\": 2.0,\n   \"num_unique_values\": 146,\n   \"samples\": [\n   0.57,\n   1.66,\n   1.03\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Cash Flow from Operations\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 28448185,\n   \"min\": 1081017,\n   \"max\": 99943687,\n   \"num_unique_values\": 500,\n   \"samples\": [\n   89180158,\n   28323876,\n   63161967\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Free Cash Flow\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 27315298,\n   \"min\": 1253861,\n   \"max\": 99809604,\n   \"num_unique_values\": 500,\n   \"samples\": [\n   65138160,\n   26587770,\n   18030139\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Beta\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.29423903687587627,\n   \"min\": 0.5,\n   \"max\": 1.5,\n   \"num_unique_values\": 100,\n   \"samples\": [\n   0.98,\n   0.62,\n   0.88\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"52 Week High\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 280.4886398236153,\n   \"min\": 10.02,\n   \"max\": 999.23,\n   \"num_unique_values\": 497,\n   \"samples\": [\n   699.85,\n   835.76,\n   803.89\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"52 Week Low\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 277.2163265244344,\n   \"min\": 10.77,\n   \"max\": 998.92,\n   \"num_unique_values\": 498,\n   \"samples\": [\n   505.19,\n   964.32,\n   935.66\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"duplicate_rows"}

Company Name	0
Industry	0
Open	1048
Close	541
High	0
Low	0
Volume	0
Market Cap	0
EPS	842
PE Ratio	726
Dividend Yield	0
Debt to Equity Ratio	0
Return on Equity	0
Current Ratio	0
Quick Ratio	0

```
Cash Flow from Operations       0
Free Cash Flow                  0
Beta                            0
52 Week High                    0
52 Week Low                     0
dtype: int64
```


```
{"type":"string"}
```

```
{"summary":"{\n  \"name\": \"non_numeric_volume\",\n  \"rows\": 310,\n
\"fields\": [\n    {\n      \"column\": \"Company Name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 300,\n        \"samples\": [\n
\"Company_7546\",\n          \"Company_9877\",\n
\"Company_5556\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Industry\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"Energy\",\n          \"Technology\",\n
\"Finance\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Open\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 275.03830963274515,\n        \"min\": 16.06,\n
\"max\": 991.15,\n        \"num_unique_values\": 264,\n
\"samples\": [\n          521.89,\n          581.87,\n
869.57\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Close\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 274.44022545445426,\n        \"min\": 21.16,\n
\"max\": 998.34,\n        \"num_unique_values\": 283,\n
\"samples\": [\n          677.34,\n          97.91,\n          654.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"High\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
294.77673744656283,\n        \"min\": 15.85,\n        \"max\":
997.42,\n        \"num_unique_values\": 298,\n        \"samples\": [\n
669.87,\n          809.77,\n          685.07\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Low\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 282.0774183048179,\n
\"min\": 14.49,\n        \"max\": 997.27,\n
\"num_unique_values\": 300,\n        \"samples\": [\n          62.71,\
n          933.77,\n          558.07\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Volume\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
1,\n        \"samples\": [\n          \"N/A\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Market Cap\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
```

```
\"num_unique_values\": 290,\n        \"samples\": [\n
500687880\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"EPS\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2.931918911525096,\n        \"min\": 0.05,\n        \"max\":
9.99,\n        \"num_unique_values\": 248,\n        \"samples\": [\n
0.08\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"PE
Ratio\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 11.419898452289406,\n        \"min\": 10.13,\n
\"max\": 49.94,\n        \"num_unique_values\": 268,\n
\"samples\": [\n          16.03\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Dividend Yield\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.492590610431285,\n        \"min\": 0.01,\n        \"max\": 4.99,\n
\"num_unique_values\": 231,\n        \"samples\": [\n          2.07\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Debt to Equity Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.595379001251317,\n        \"min\": 0.0,\n        \"max\": 1.99,\n
\"num_unique_values\": 153,\n        \"samples\": [\n          1.03\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Return on Equity\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8.761556619285349,\n        \"min\": 0.02,\n        \"max\": 30.0,\n
\"num_unique_values\": 291,\n        \"samples\": [\n          6.75\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Current Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.5589310409611843,\n        \"min\": 1.01,\n        \"max\": 2.99,\n
\"num_unique_values\": 156,\n        \"samples\": [\n          2.28\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Quick Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.43822174326828806,\n        \"min\": 0.51,\n        \"max\": 2.0,\n
\"num_unique_values\": 131,\n        \"samples\": [\n          1.13\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Cash Flow from Operations\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
28459724,\n        \"min\": 1323325,\n        \"max\": 99597456,\n
\"num_unique_values\": 300,\n        \"samples\": [\n
73183035\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Free Cash Flow\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 28256547,\n        \"min\": 1253861,\n
\"max\": 99769139,\n        \"num_unique_values\": 300,\n
\"samples\": [\n          57585563\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"Beta\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.29454974654422666,\n
\"min\": 0.5,\n        \"max\": 1.49,\n        \"num_unique_values\":
94,\n        \"samples\": [\n          0.6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"52 Week High\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
274.00726837942864,\n        \"min\": 13.07,\n        \"max\":
996.02,\n        \"num_unique_values\": 300,\n        \"samples\": [\n
913.51\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"52
Week Low\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 301.00628717295757,\n        \"min\": 10.52,\n
\"max\": 996.33,\n        \"num_unique_values\": 300,\n
\"samples\": [\n          646.78\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\
n}","type":"dataframe","variable_name":"non_numeric_volume"}

{"type":"string"}

{"summary":"{\n  \"name\": \"non_numeric_market_cap\",\n  \"rows\":
210,\n  \"fields\": [\n    {\n      \"column\": \"Company Name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 200,\n        \"samples\": [\n
\"Company_4970\",\n          \"Company_699\",\n
\"Company_1759\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Industry\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"Energy\",\n          \"Retail\",\n
\"Technology\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Open\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 296.70844353430607,\n        \"min\": 12.47,\n
\"max\": 995.4,\n        \"num_unique_values\": 184,\n
\"samples\": [\n          320.58,\n          797.72,\n
254.49\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Close\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 296.7539694207858,\n        \"min\": 11.95,\n        \"max\":
998.34,\n        \"num_unique_values\": 194,\n        \"samples\": [\n
445.18,\n          212.13,\n          231.83\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"High\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 297.45968393635934,\n
\"min\": 13.06,\n        \"max\": 995.67,\n
\"num_unique_values\": 200,\n        \"samples\": [\n
152.53,\n          485.55,\n          382.9\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"Low\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 268.4979428110082,\n
\"min\": 14.99,\n        \"max\": 993.56,\n
\"num_unique_values\": 200,\n        \"samples\": [\n
512.94,\n          483.9,\n          872.3\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Volume\",\n      \"properties\":
{\n        \"dtype\": \"string\",\n        \"num_unique_values\":
190,\n        \"samples\": [\n          895076,\n          244120,\n
816299\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Market Cap\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"Not Available\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"EPS\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2.91928267189184,\n        \"min\":
0.05,\n        \"max\": 9.99,\n        \"num_unique_values\": 177,\n
\"samples\": [\n          6.73\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"PE Ratio\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 11.737612397376427,\n        \"min\":
10.06,\n        \"max\": 49.5,\n        \"num_unique_values\": 186,\n
\"samples\": [\n          17.05\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Dividend Yield\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.4787323681868965,\n        \"min\": 0.01,\n        \"max\": 4.97,\n
\"num_unique_values\": 174,\n        \"samples\": [\n          0.01\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Debt to Equity Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.5598358257051739,\n        \"min\": 0.03,\n        \"max\": 1.99,\n
\"num_unique_values\": 121,\n        \"samples\": [\n          1.04\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Return on Equity\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8.951390768065911,\n        \"min\": 0.05,\n        \"max\": 29.97,\n
\"num_unique_values\": 197,\n        \"samples\": [\n          6.74\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Current Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.5561737939996101,\n        \"min\": 1.0,\n        \"max\": 2.99,\n
\"num_unique_values\": 128,\n        \"samples\": [\n          2.81\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Quick Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.42236604488464374,\n        \"min\": 0.52,\n        \"max\": 2.0,\n
\"num_unique_values\": 115,\n        \"samples\": [\n          1.17\n
```

```
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Cash Flow from Operations\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
28291796,\n        \"min\": 1599285,\n        \"max\": 99693321,\n
\"num_unique_values\": 200,\n        \"samples\": [\n
47825913\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Free Cash Flow\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 28441352,\n        \"min\": 1230280,\n
\"max\": 99929260,\n        \"num_unique_values\": 200,\n
\"samples\": [\n          45189791\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Beta\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.2980418206604756,\n
\"min\": 0.5,\n        \"max\": 1.5,\n        \"num_unique_values\":
87,\n        \"samples\": [\n          1.23\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"52 Week High\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
300.40523340459544,\n        \"min\": 11.48,\n        \"max\":
996.66,\n        \"num_unique_values\": 200,\n        \"samples\": [\n
689.11\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"52
Week Low\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 279.8804775086042,\n        \"min\": 16.59,\n        \"max\":
999.26,\n        \"num_unique_values\": 199,\n        \"samples\": [\n
705.8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"non_numeric_market_cap"}
```


3.2 Apply techiniques

- to remove duplicate data
- to impute or remove missing data
- to remove data inconsistencies

```
df.drop_duplicates(inplace=True)

numerical_cols = df.select_dtypes(include=np.number).columns
df[numerical_cols] =
df[numerical_cols].fillna(df[numerical_cols].mean())

df = df[df['Volume'].apply(lambda x: isinstance(x, (int, float)))]
df = df[df['Market Cap'].apply(lambda x: isinstance(x, (int, float)))]

duplicate_rows = df[df.duplicated()]
display(duplicate_rows)

missing_data = df.isnull().sum()
display(missing_data)
```

```
non_numeric_volume = df[~df['Volume'].apply(lambda x: isinstance(x,
(int, float)))]
display("Rows with non-numeric values in 'Volume' column:")
display(non_numeric_volume)

non_numeric_market_cap = df[~df['Market Cap'].apply(lambda x:
isinstance(x, (int, float)))]
display("Rows with non-numeric values in 'Market Cap' column:")
display(non_numeric_market_cap)
```

{"repr_error":"Out of range float values are not JSON compliant: nan","type":"dataframe","variable_name":"duplicate_rows"}

```
Company Name                  0
Industry                      0
Open                          0
Close                         0
High                          0
Low                           0
Volume                        0
Market Cap                    0
EPS                           0
PE Ratio                      0
Dividend Yield                0
Debt to Equity Ratio          0
Return on Equity              0
Current Ratio                 0
Quick Ratio                   0
Cash Flow from Operations     0
Free Cash Flow                0
Beta                          0
52 Week High                  0
52 Week Low                   0
dtype: int64
```

{"type":"string"}

{"repr_error":"Out of range float values are not JSON compliant: nan","type":"dataframe","variable_name":"non_numeric_volume"}

{"type":"string"}

{"repr_error":"Out of range float values are not JSON compliant: nan","type":"dataframe","variable_name":"non_numeric_market_cap"}

3.3 Encode categorical data

```
df = pd.get_dummies(df, columns=['Industry'])

display(df)
```

```
{"type":"dataframe","variable_name":"df"}

# Perform feature Engineering to capture relevant financial indicators

# Calculate moving averages
df['MA_50'] = df['Close'].rolling(window=50).mean()
df['MA_200'] = df['Close'].rolling(window=200).mean()

# Calculate volatility (standard deviation of closing price over a
period)
df['Volatility_30'] = df['Close'].rolling(window=30).std()

# Calculate relative strength index (RSI)
def calculate_rsi(data, window=14):
    delta = data['Close'].diff()
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=window).mean()
    avg_loss = loss.rolling(window=window).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

df['RSI_14'] = calculate_rsi(df)

# Display the updated DataFrame
display(df)

{"type":"dataframe","variable_name":"df"}
```

3.4 Report

Mention and justify the method adopted

- to remove duplicate data, if present
- to impute or remove missing data, if present
- to remove data inconsistencies, if present

OR for textdata

- How many tokens after step 3?
- how may tokens after stop words filtering?

If the any of the above are not present, then also add in the report below.

Remove duplicate data

Justification:

Duplicate data can skew analysis and model training. Removing duplicates ensures data integrity. Method: df.drop_duplicates(inplace=True) This method identifies and removes duplicate rows from the DataFrame.

Impute missing data

Justification:

Missing data can lead to biased results. Imputation replaces missing values with estimated values. Method: df[numerical_cols] = df[numerical_cols].fillna(df[numerical_cols].mean()) This method fills missing values in numerical columns with the mean of the respective column.

Remove data inconsistencies

Justification:

Inconsistent data types can cause errors in analysis and modeling. Method: df = df[df['Volume'].apply(lambda x: isinstance(x, (int, float)))] df = df[df['Market Cap'].apply(lambda x: isinstance(x, (int, float)))] This method filters out rows with non-numeric values in the 'Volume' and 'Market Cap' columns.

The dataset did not have textdata

3.5 Identify the target variables.

- Separate the data from the target such that the dataset is in the form of (X,y) or (Features, Label)
- Discretize / Encode the target variable or perform one-hot encoding on the target or any other as and if required.
- Report the observations

```
import pandas as pd

# 1. Identify Target and Features
X = df.drop('Close', axis=1)  # Features (all columns except 'Close')
y = df['Close']               # Target variable (Close price)

# 2. Discretize Target (example - create 3 price categories)
y_discretized = pd.cut(y, bins=3, labels=['Low', 'Medium', 'High'])

# 3. One-Hot Encode Discretized Target
y_encoded = pd.get_dummies(y_discretized, prefix='Close')

# Add encoded target back to feature DataFrame (for demonstration)
X = pd.concat([X, y_encoded], axis=1)

# Report observations
display("Shape of Features (X):", X.shape)
display("Sample Data with Encoded Target:")
display(X.head())
```

```
{"type":"string"}
```

```
(10800, 30)
```

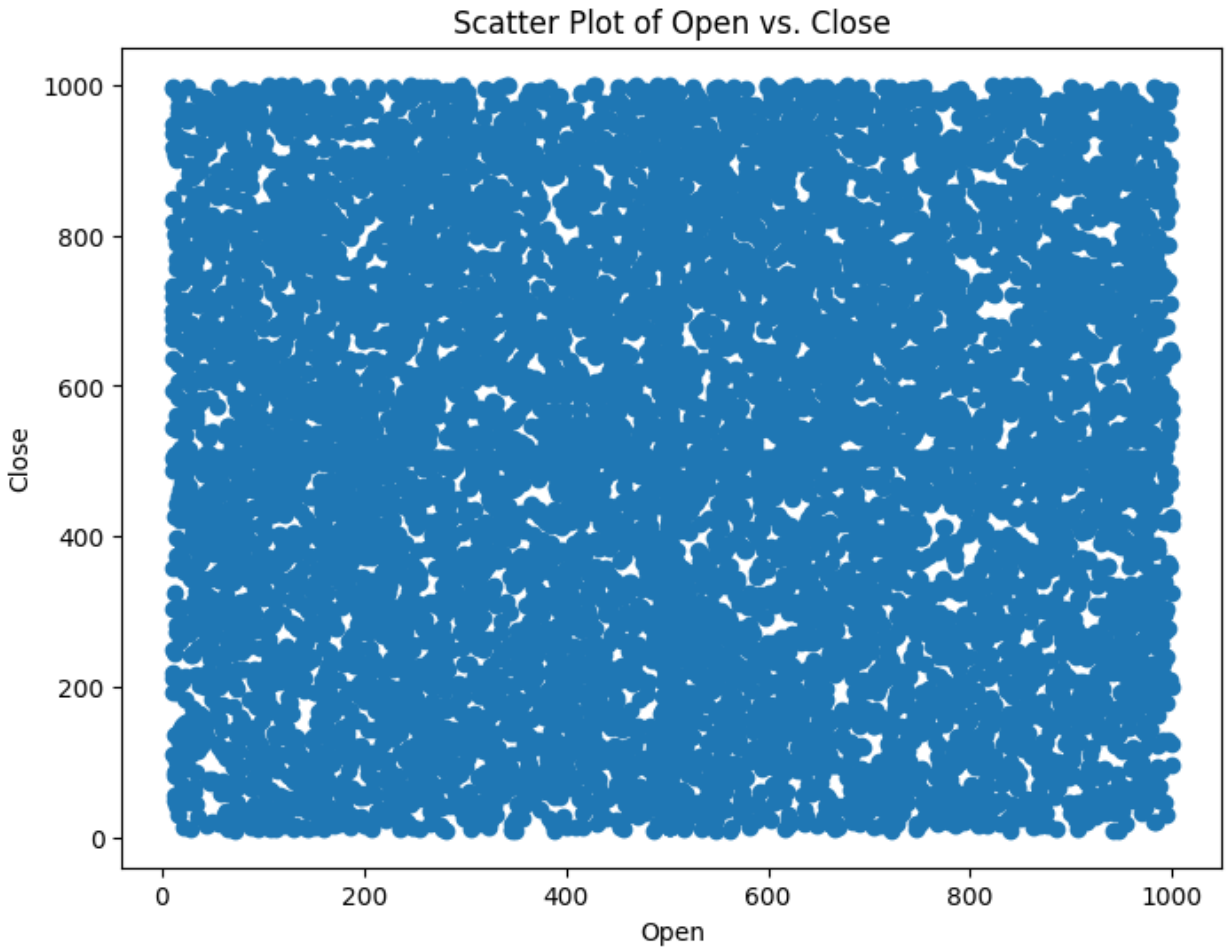
```
{"type":"string"}
```

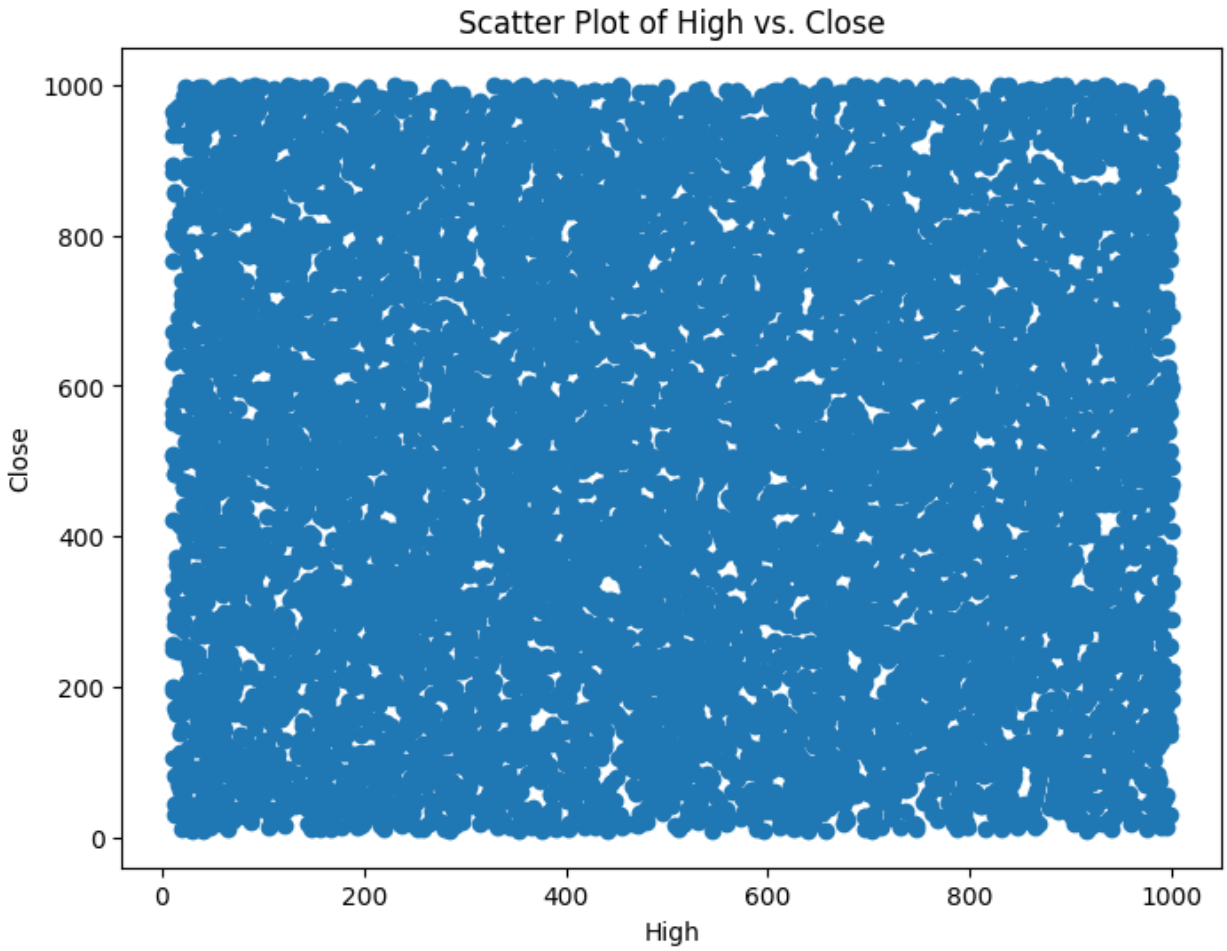
```
{"type":"dataframe"}
```

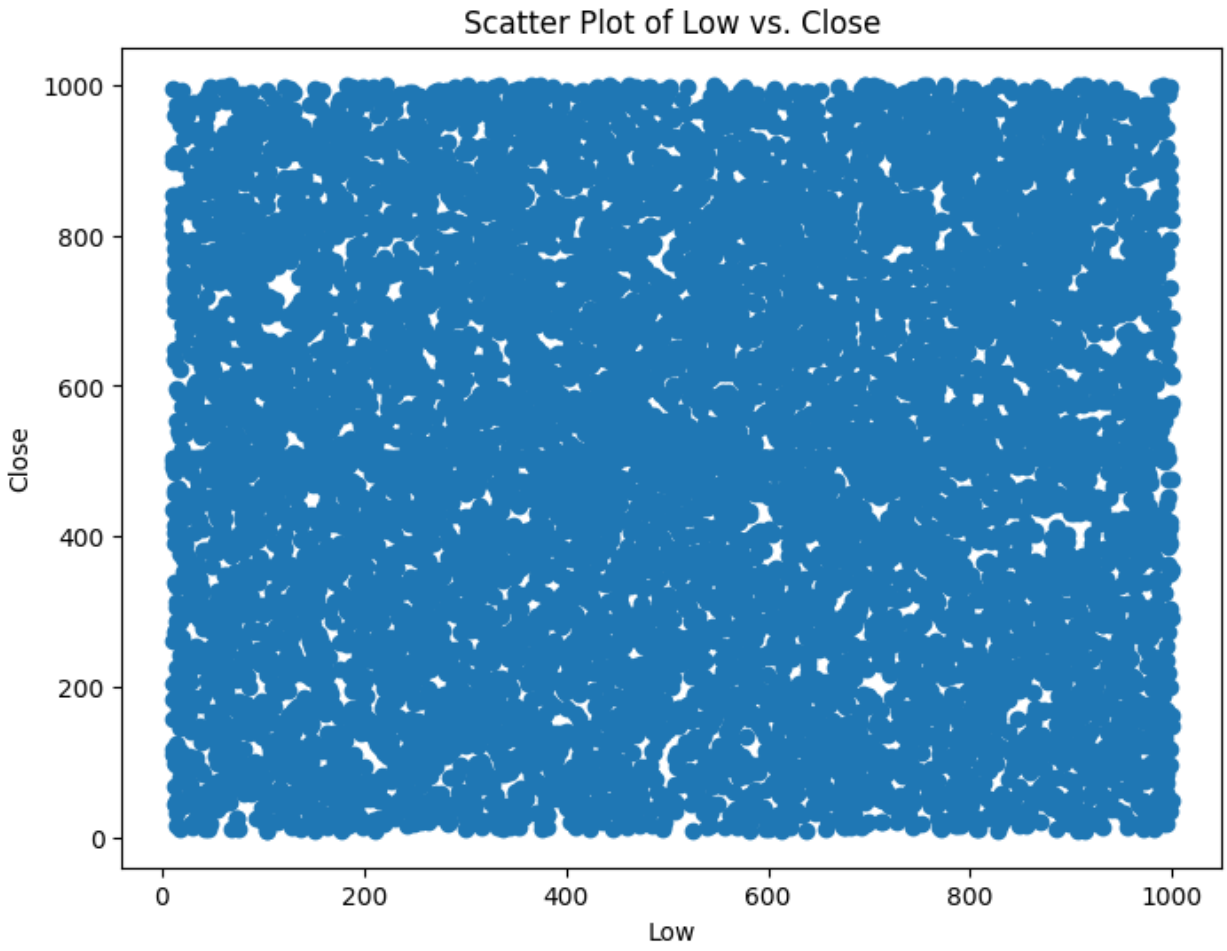
4. Data Exploration using various plots

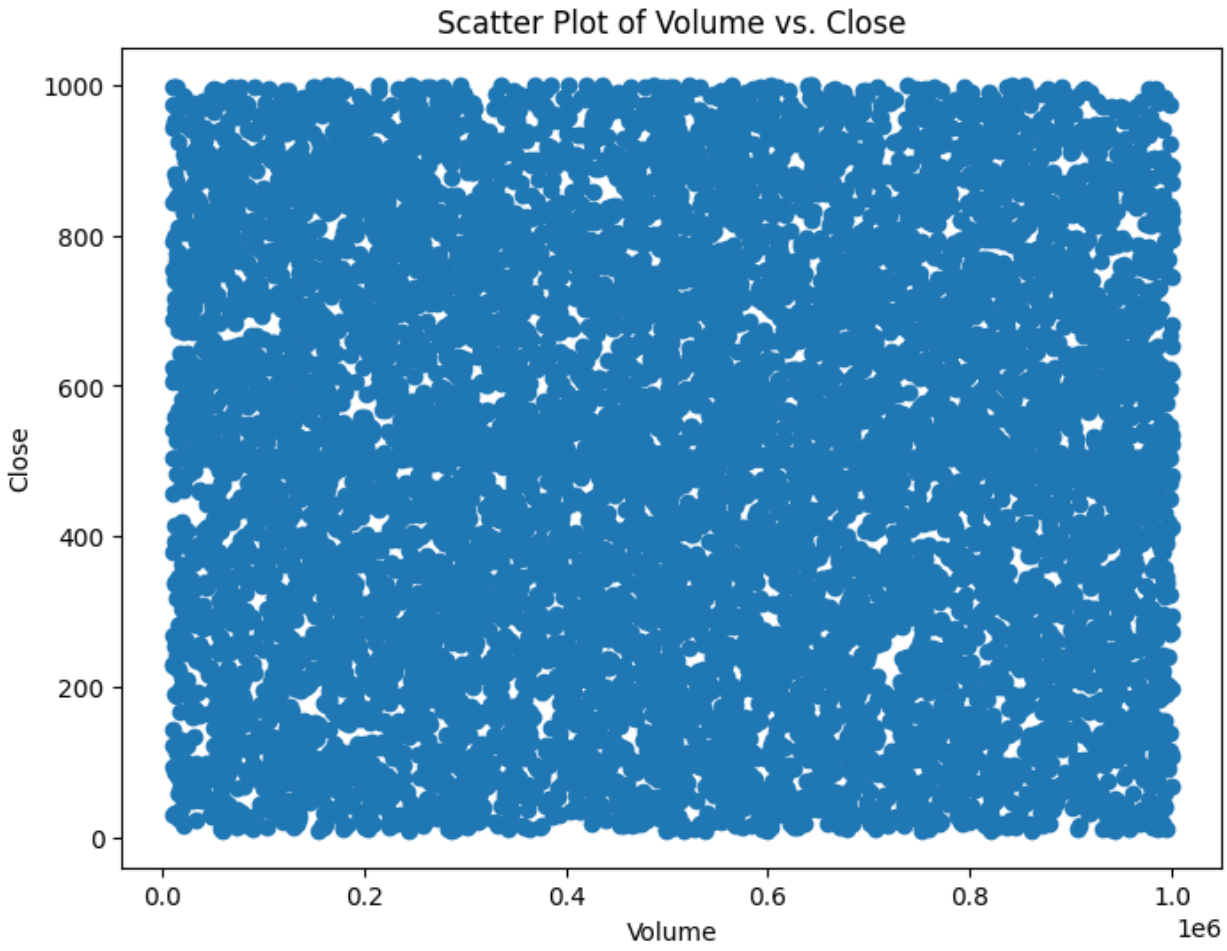
4.1 Scatter plot of each quantitative attribute with the target.

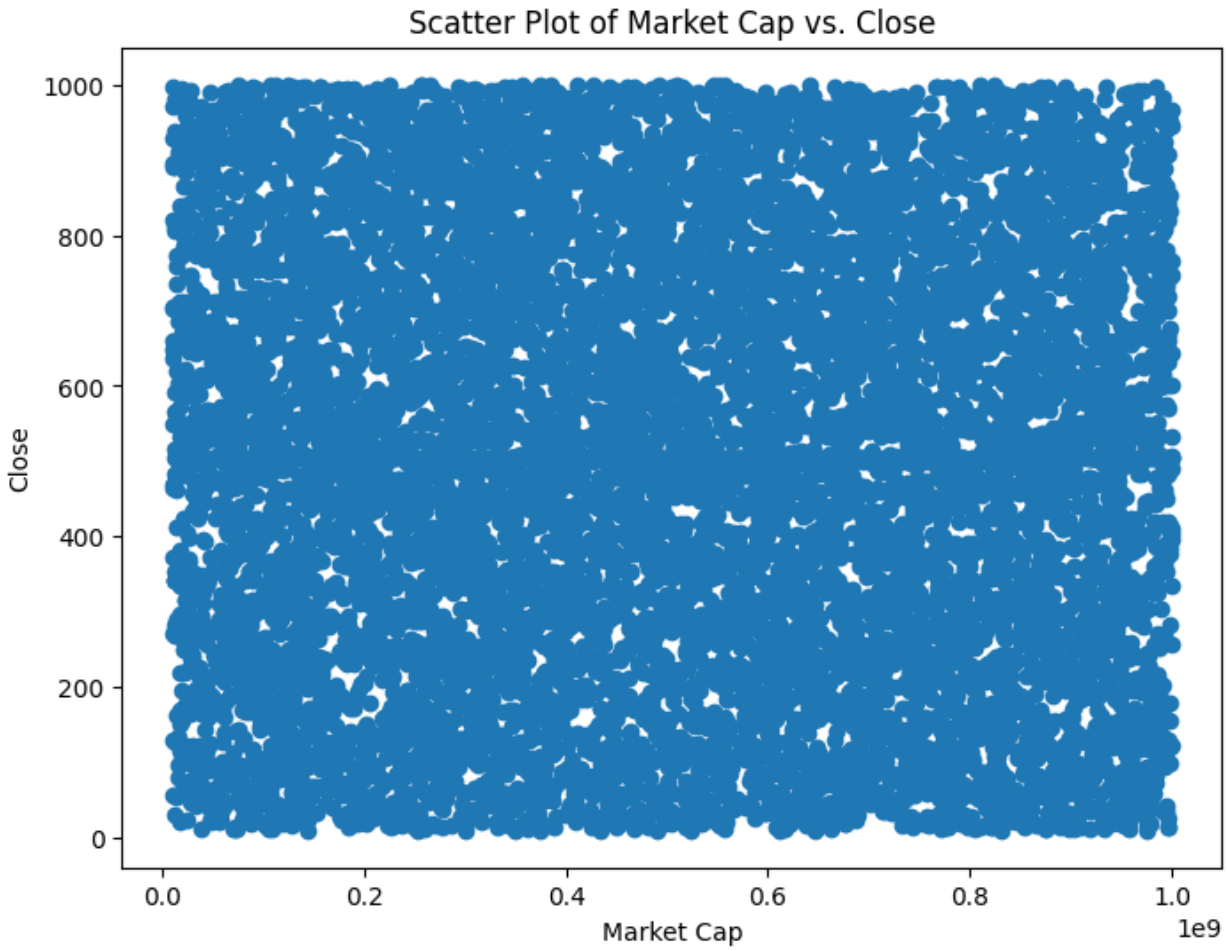
```
import matplotlib.pyplot as plt
import seaborn as sns
# Select quantitative attributes
quantitative_attributes = ['Open', 'High', 'Low', 'Volume', 'Market
Cap', 'EPS', 'PE Ratio', 'Dividend Yield',
                           'Debt to Equity Ratio', 'Return on Equity',
'Current Ratio', 'Quick Ratio',
                           'Cash Flow from Operations', 'Free Cash
Flow', 'Beta', '52 Week High', '52 Week
Low','MA_50','MA_200','Volatility_30','RSI_14']




# Create scatter plots
for attribute in quantitative_attributes:
  plt.figure(figsize=(8, 6))
  plt.scatter(df[attribute], df['Close'])
  plt.xlabel(attribute)
  plt.ylabel('Close')
  plt.title(f'Scatter Plot of {attribute} vs. Close')
  plt.show()
```

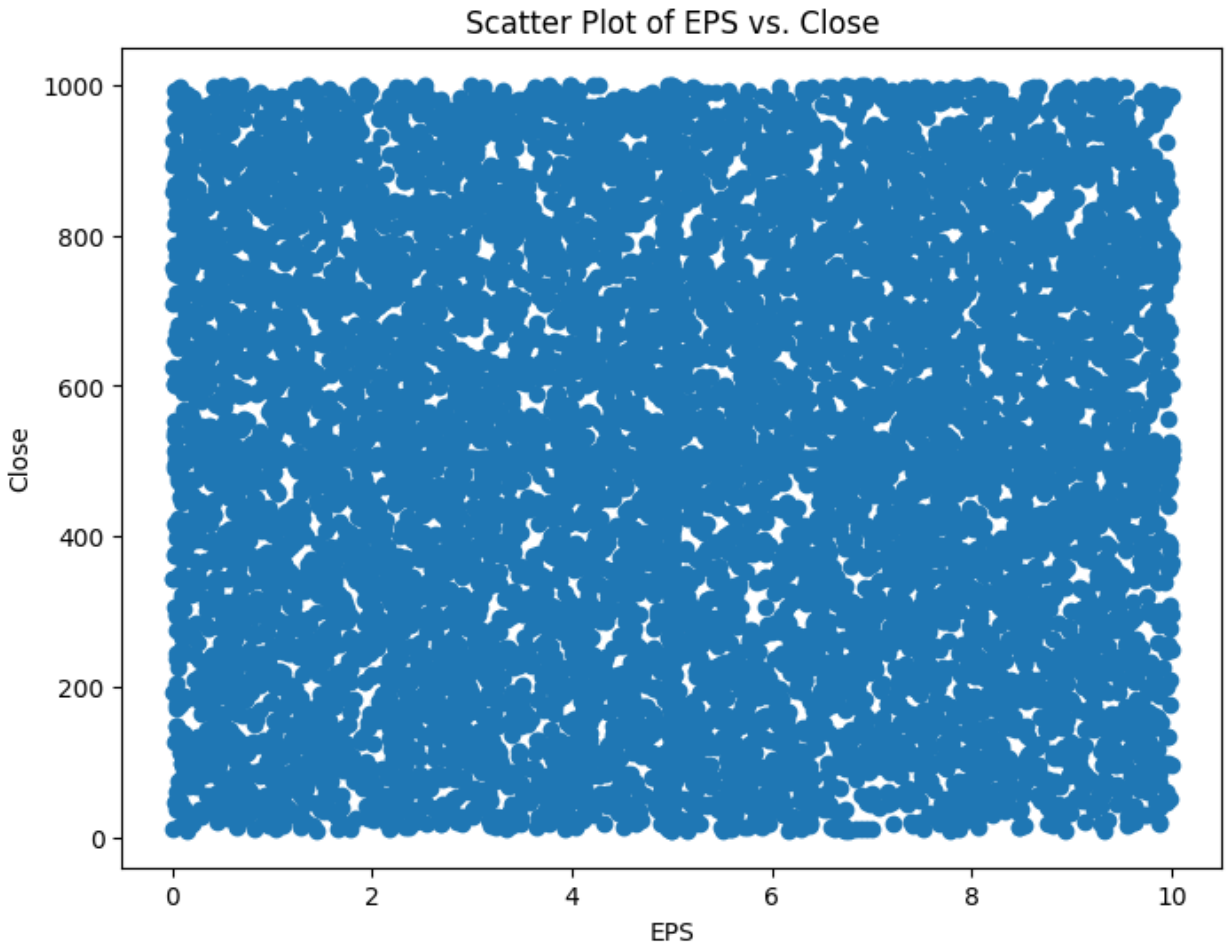

Scatter Plot of Open vs. Close
Close
Open
0
200
400
600
800
1000

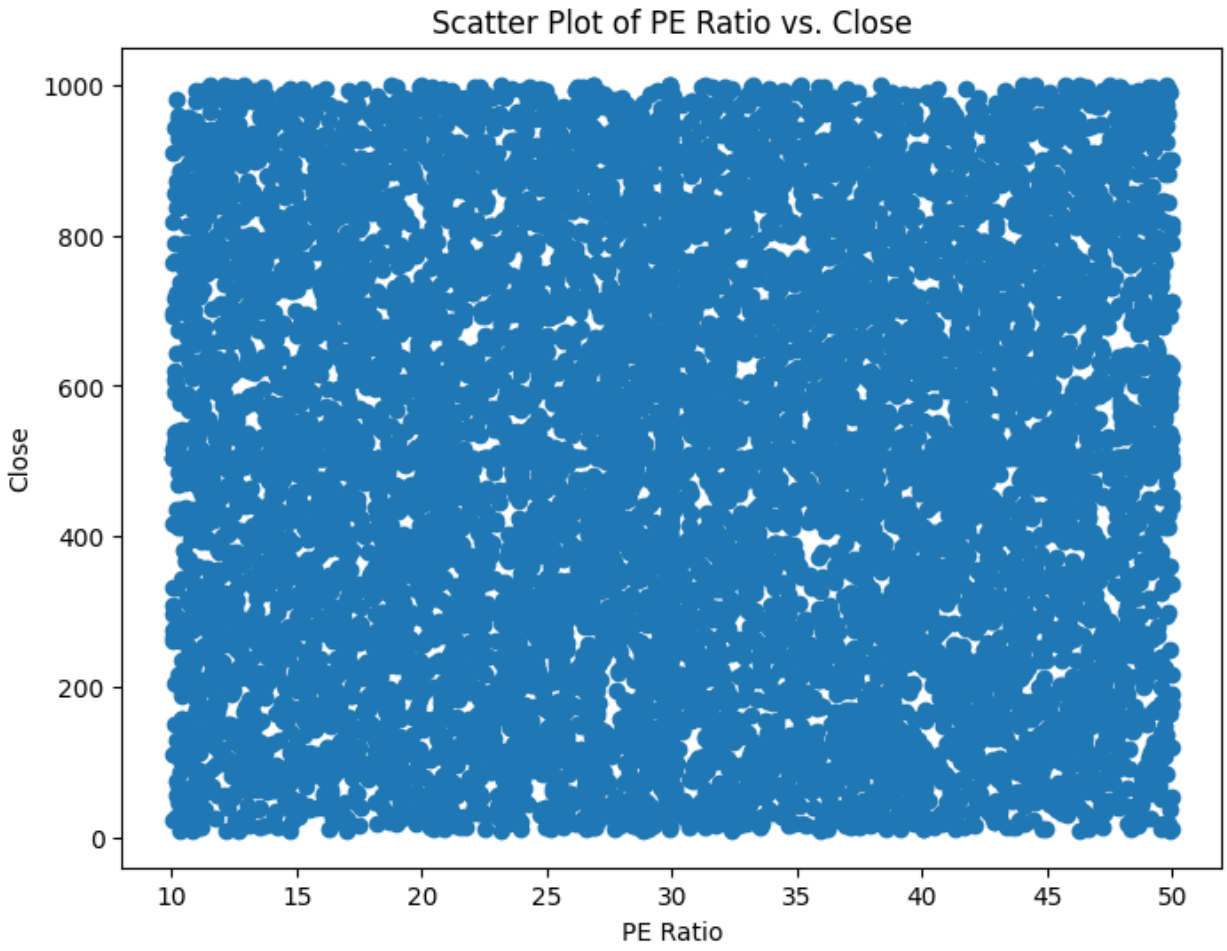

Scatter Plot of High vs. Close
Close
High
0
200
400
600
800
1000

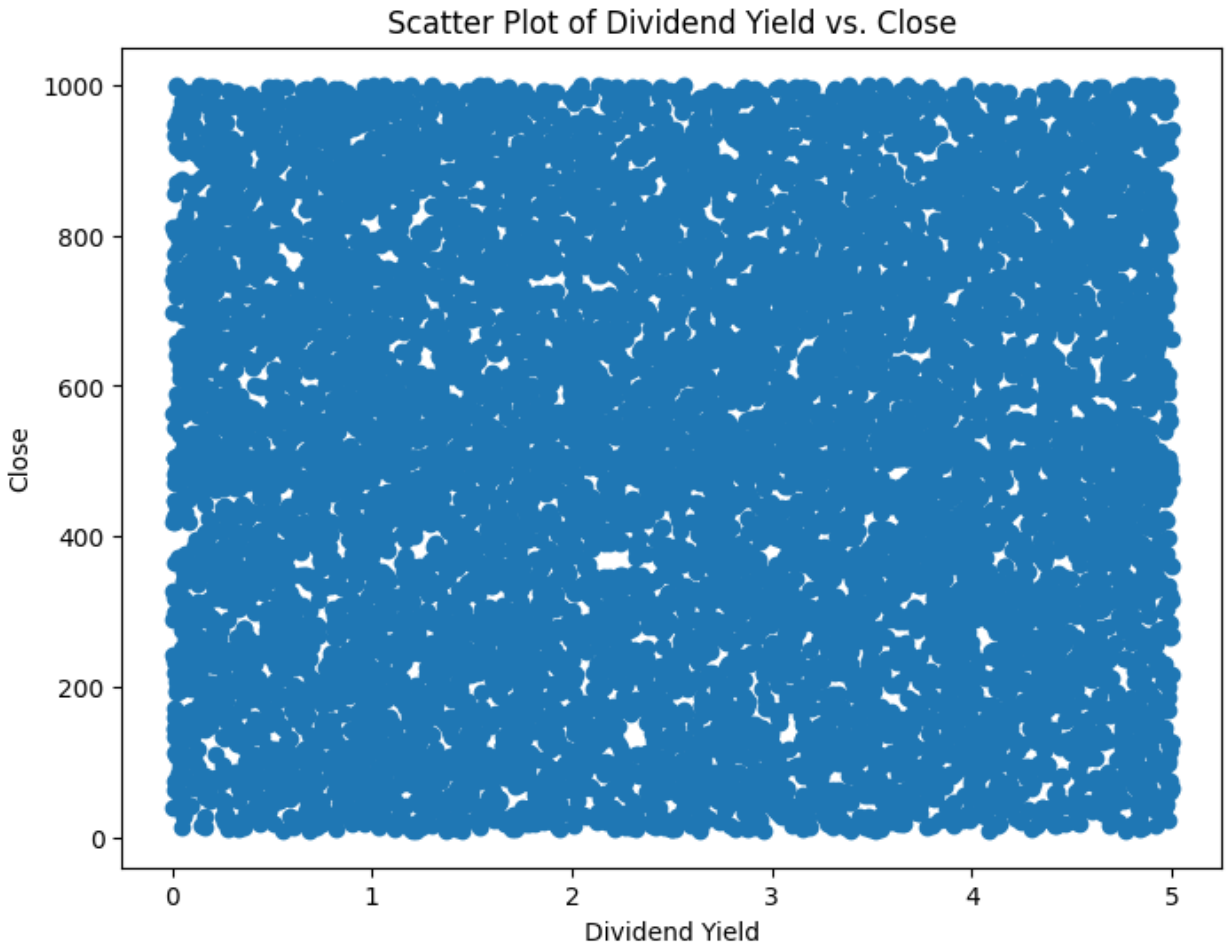

Scatter Plot of Low vs. Close
Close
Low
0
200
400
600
800
1000

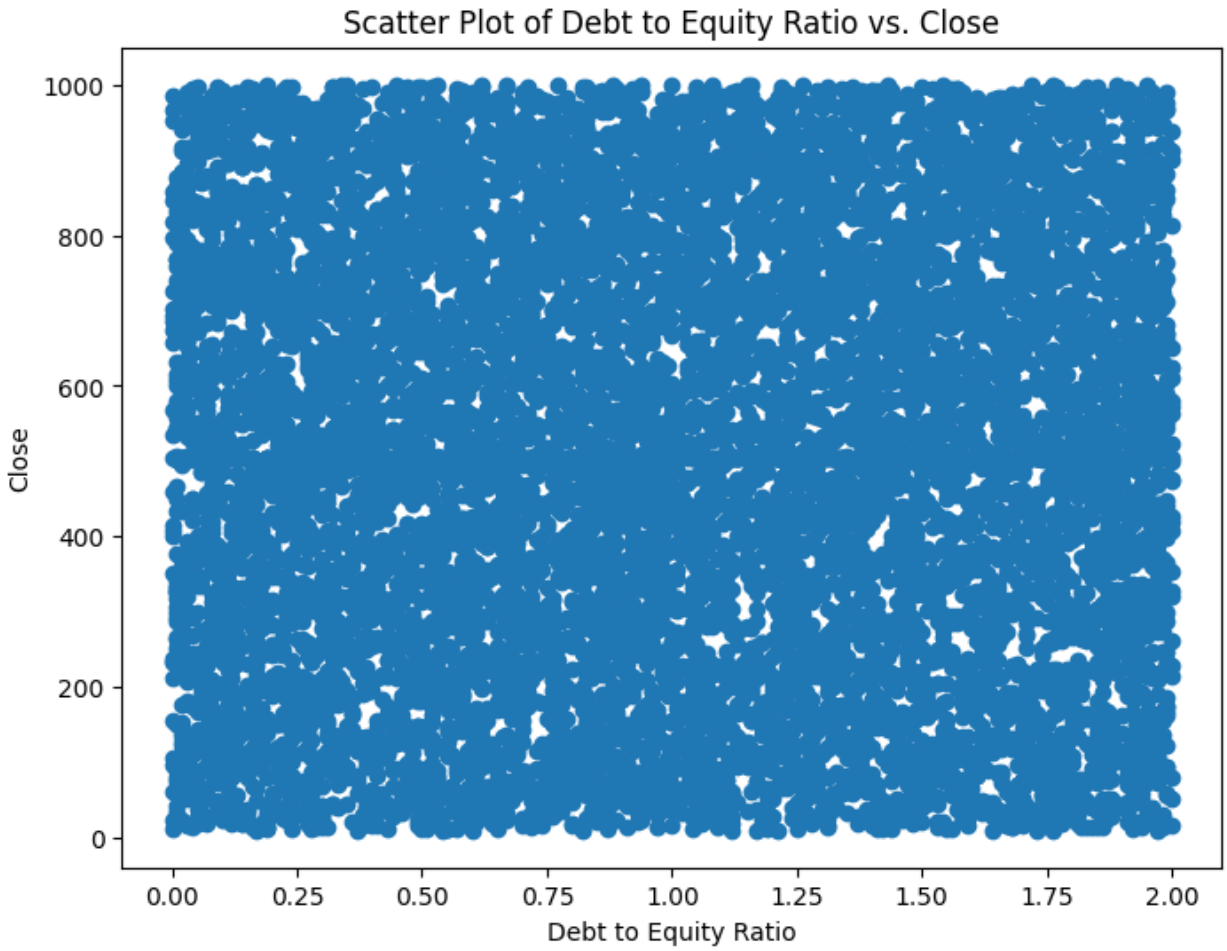

Scatter Plot of Volume vs. Close
Close
Volume
1e6
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0

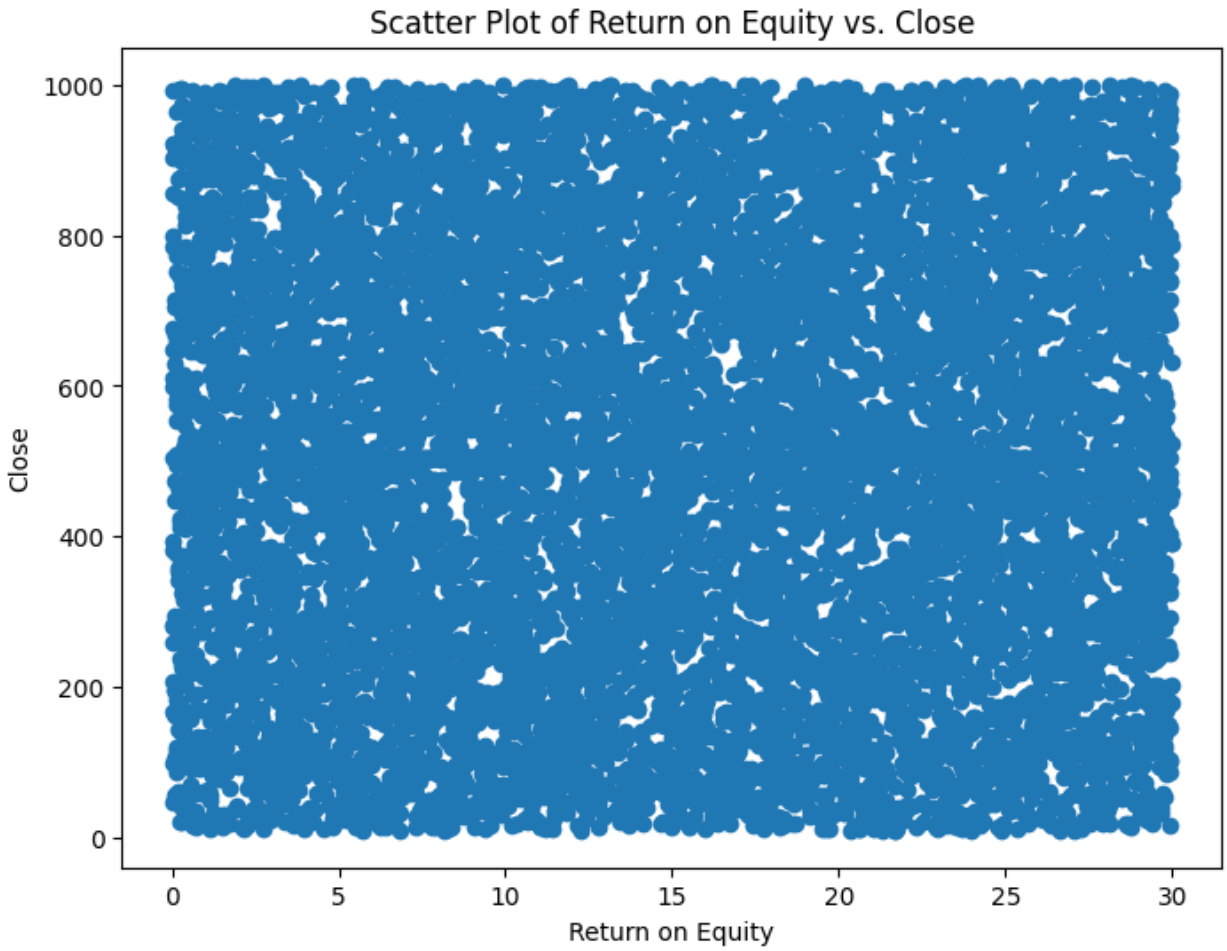

Scatter Plot of Market Cap vs. Close
Close
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
Market Cap
1e9

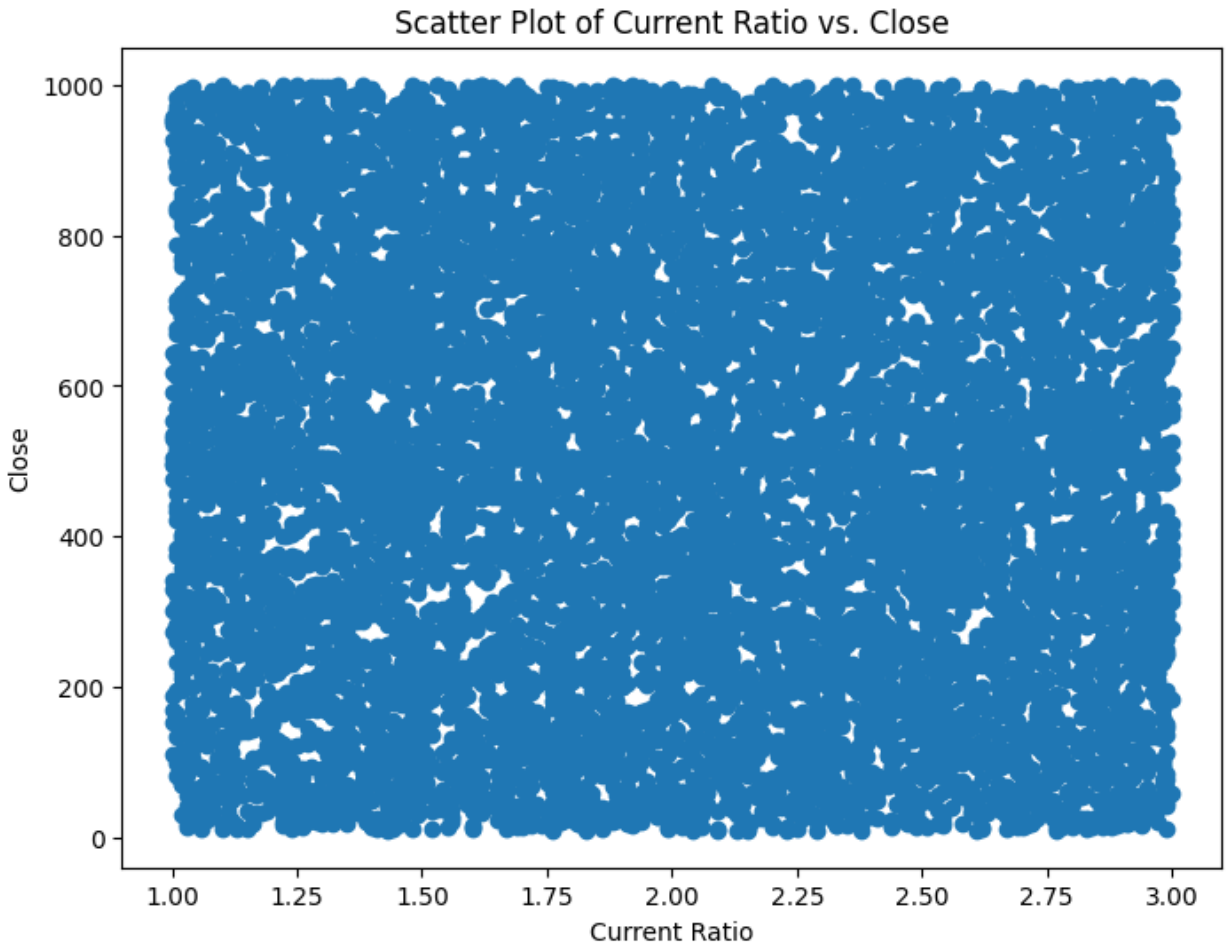

Scatter Plot of EPS vs. Close
Close
EPS
0
200
400
600
800
1000
0
2
4
6
8
10

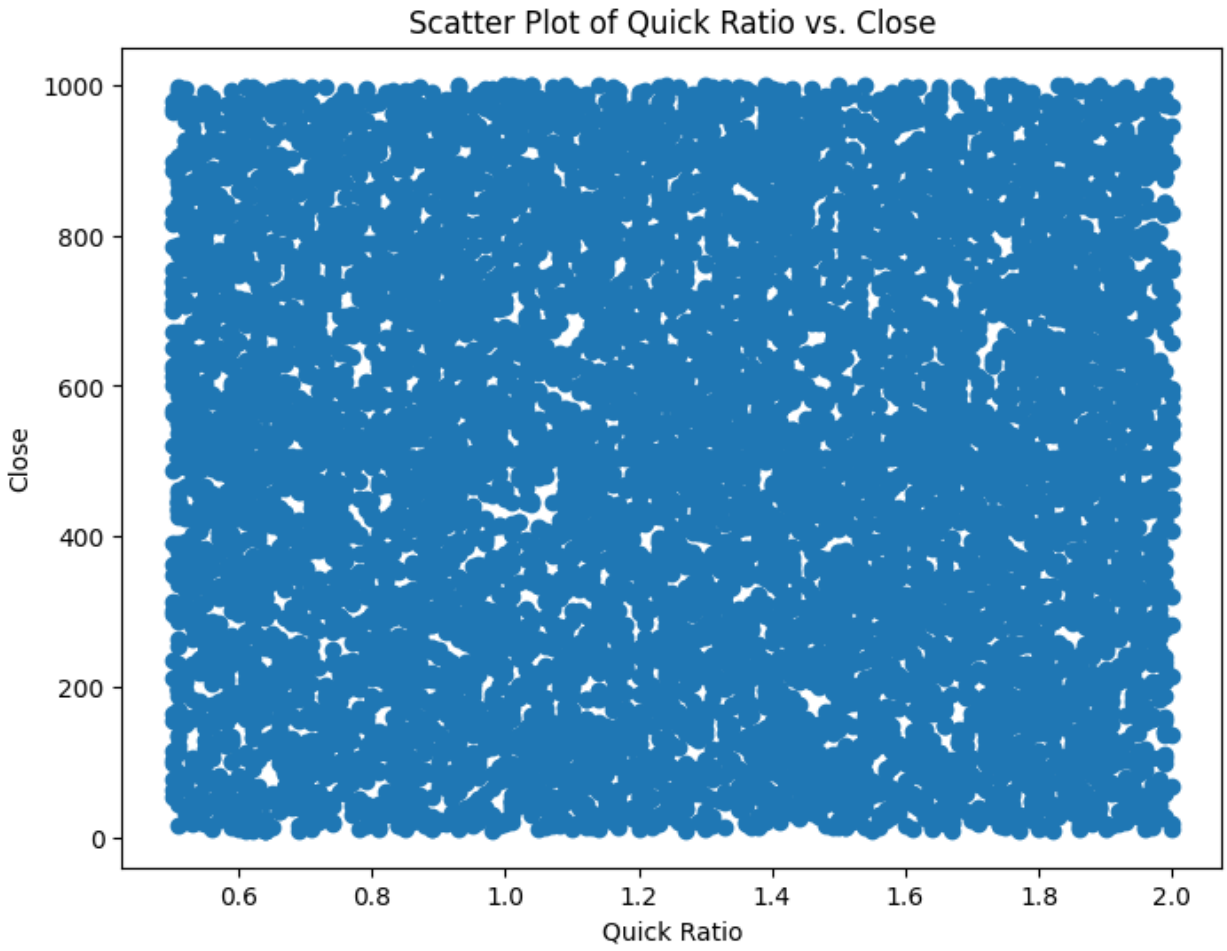

Scatter Plot of PE Ratio vs. Close
Close
PE Ratio
0
200
400
600
800
1000
10
15
20
25
30
35
40
45
50

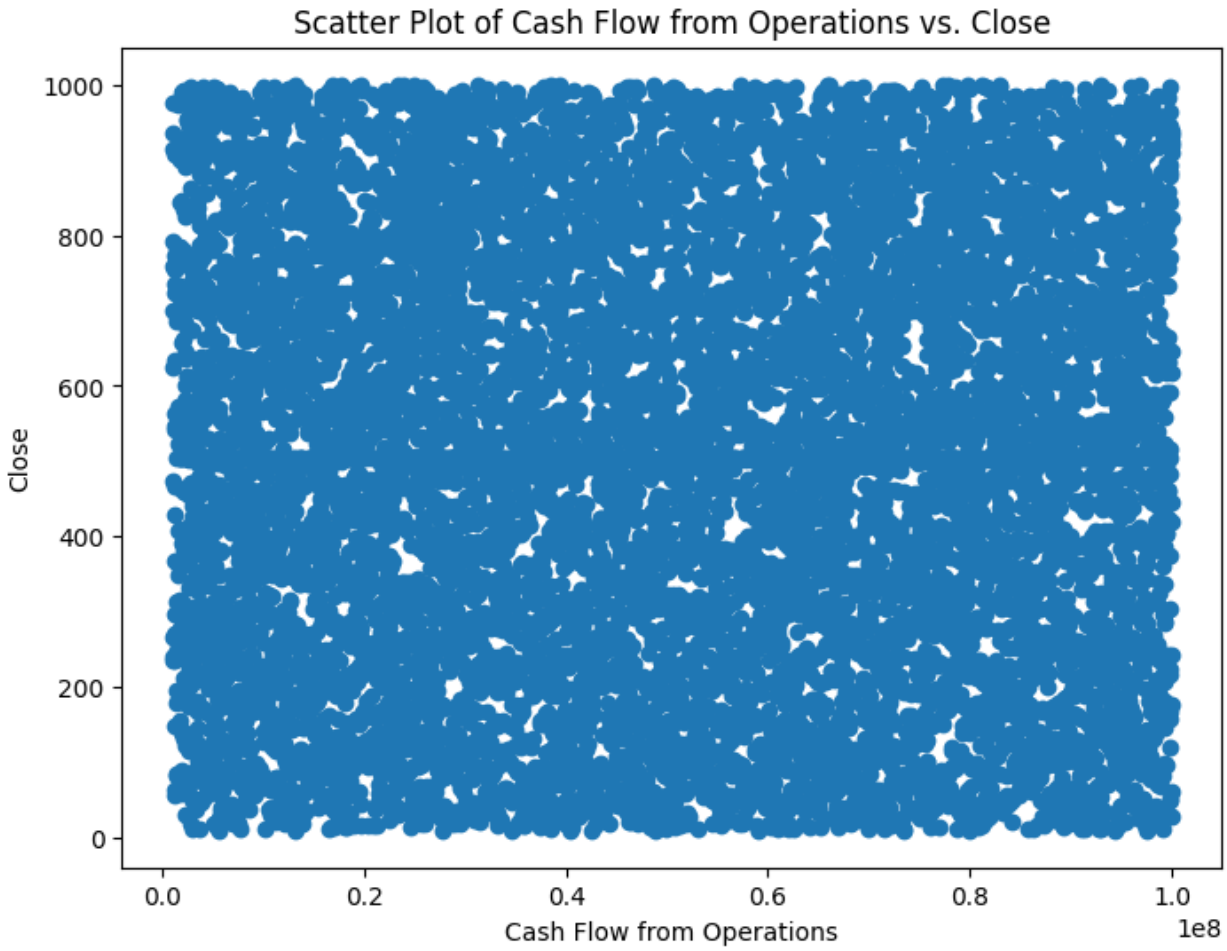

Scatter Plot of Dividend Yield vs. Close
Close
Dividend Yield
0
200
400
600
800
1000
0
1
2
3
4
5

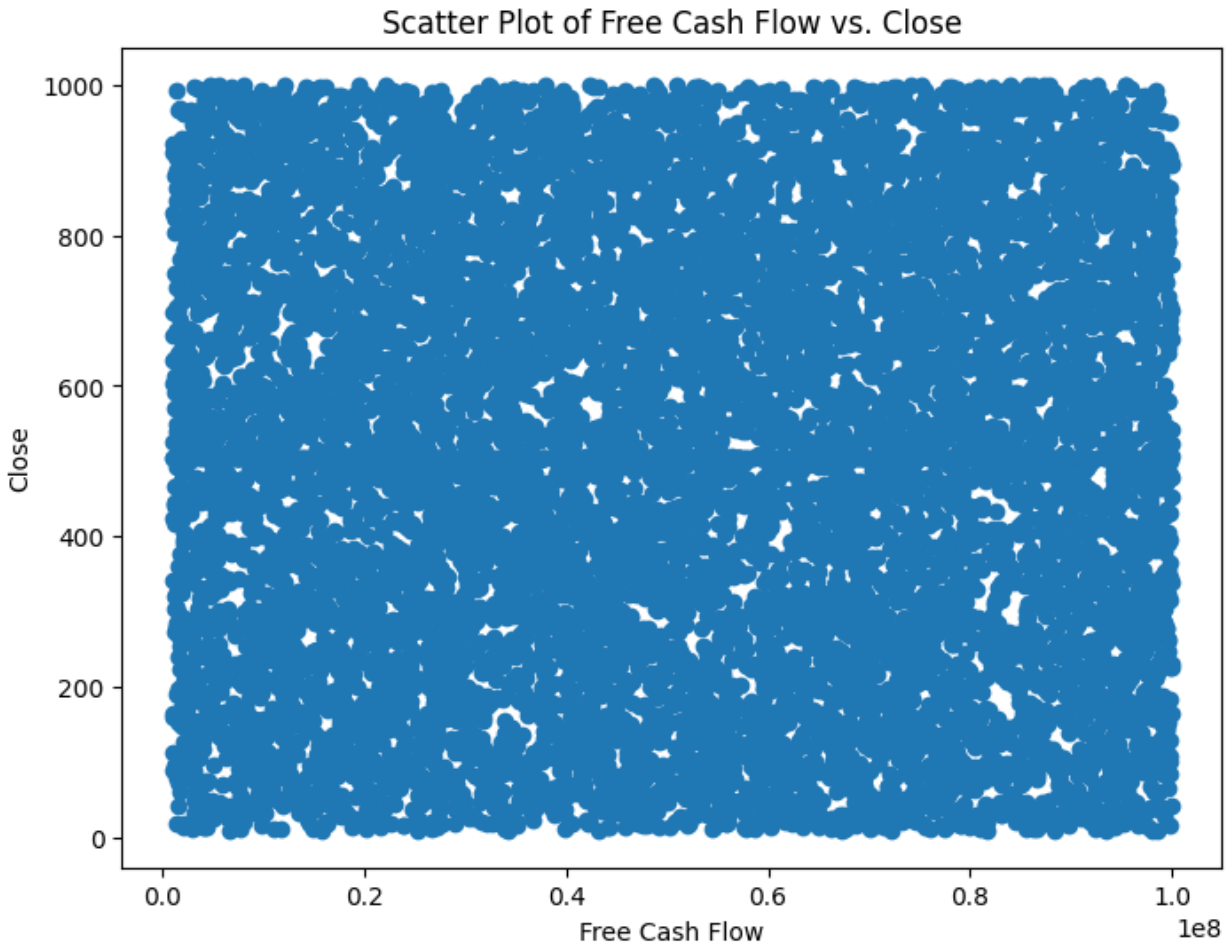

Scatter Plot of Debt to Equity Ratio vs. Close
Close
1000
800
600
400
200
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Debt to Equity Ratio

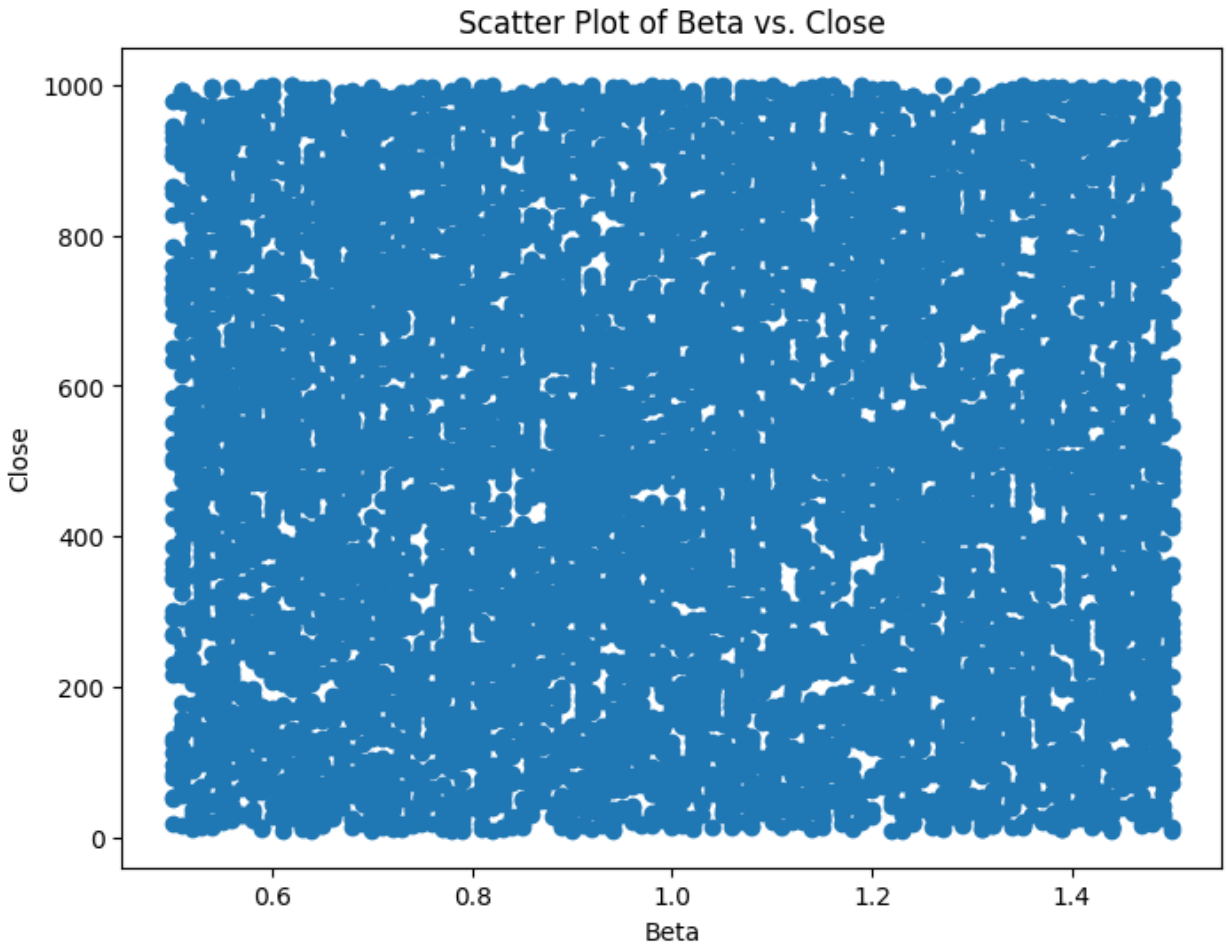

Scatter Plot of Return on Equity vs. Close
Close
Return on Equity
0
200
400
600
800
1000
0
5
10
15
20
25
30

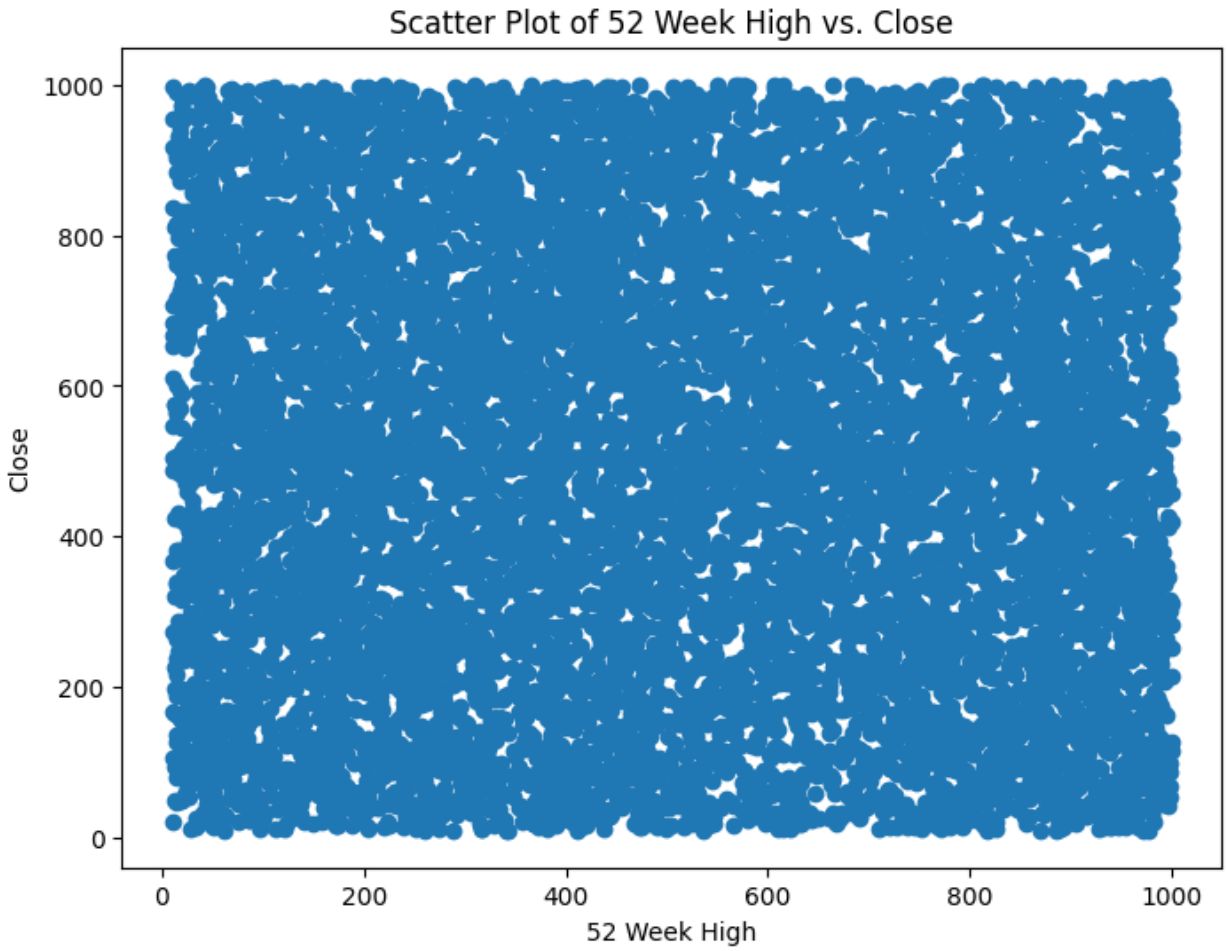

Scatter Plot of Current Ratio vs. Close
Close
Current Ratio
0
200
400
600
800
1000
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

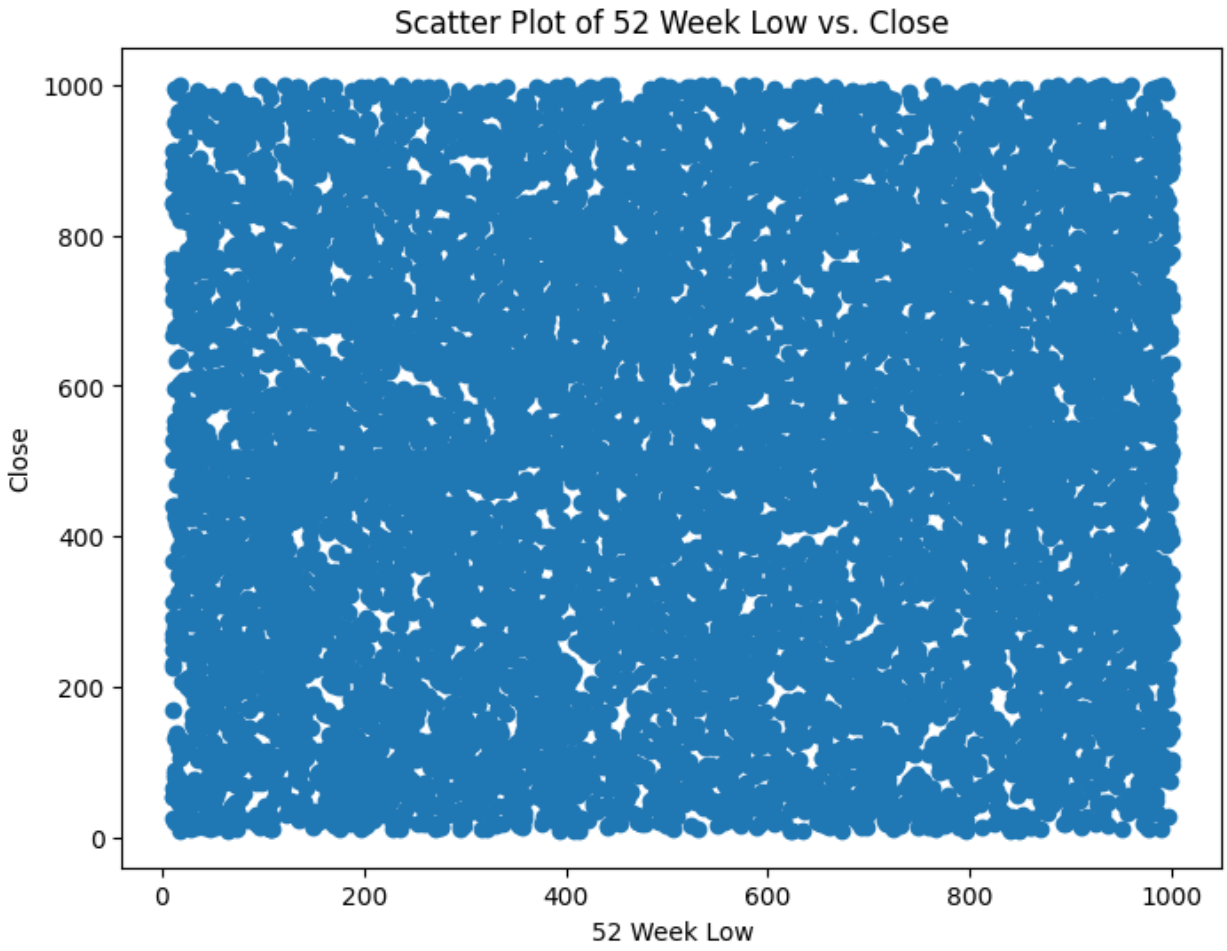

Scatter Plot of Quick Ratio vs. Close
Close
Quick Ratio
0
200
400
600
800
1000
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0

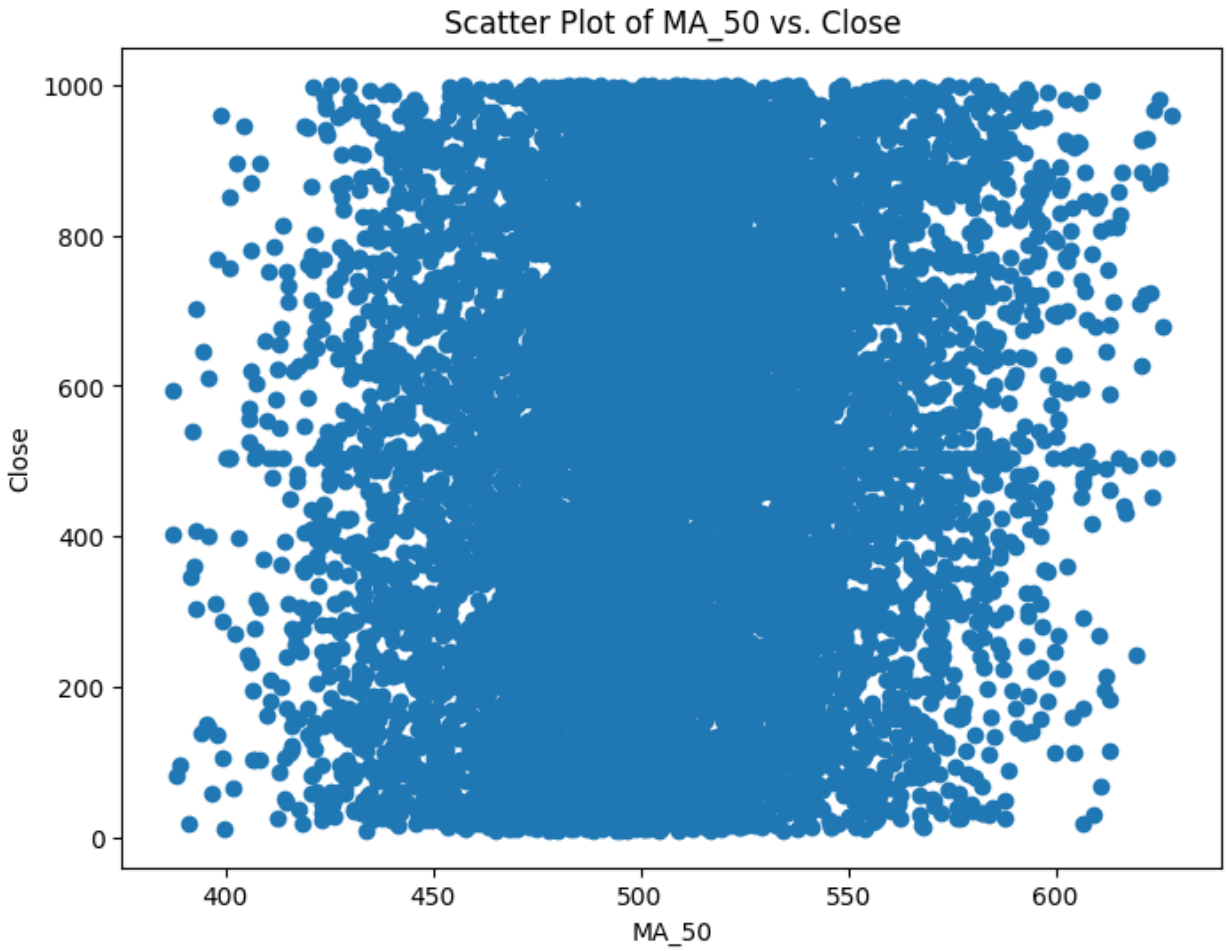

Scatter Plot of Cash Flow from Operations vs. Close
Close
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
1e8
Cash Flow from Operations

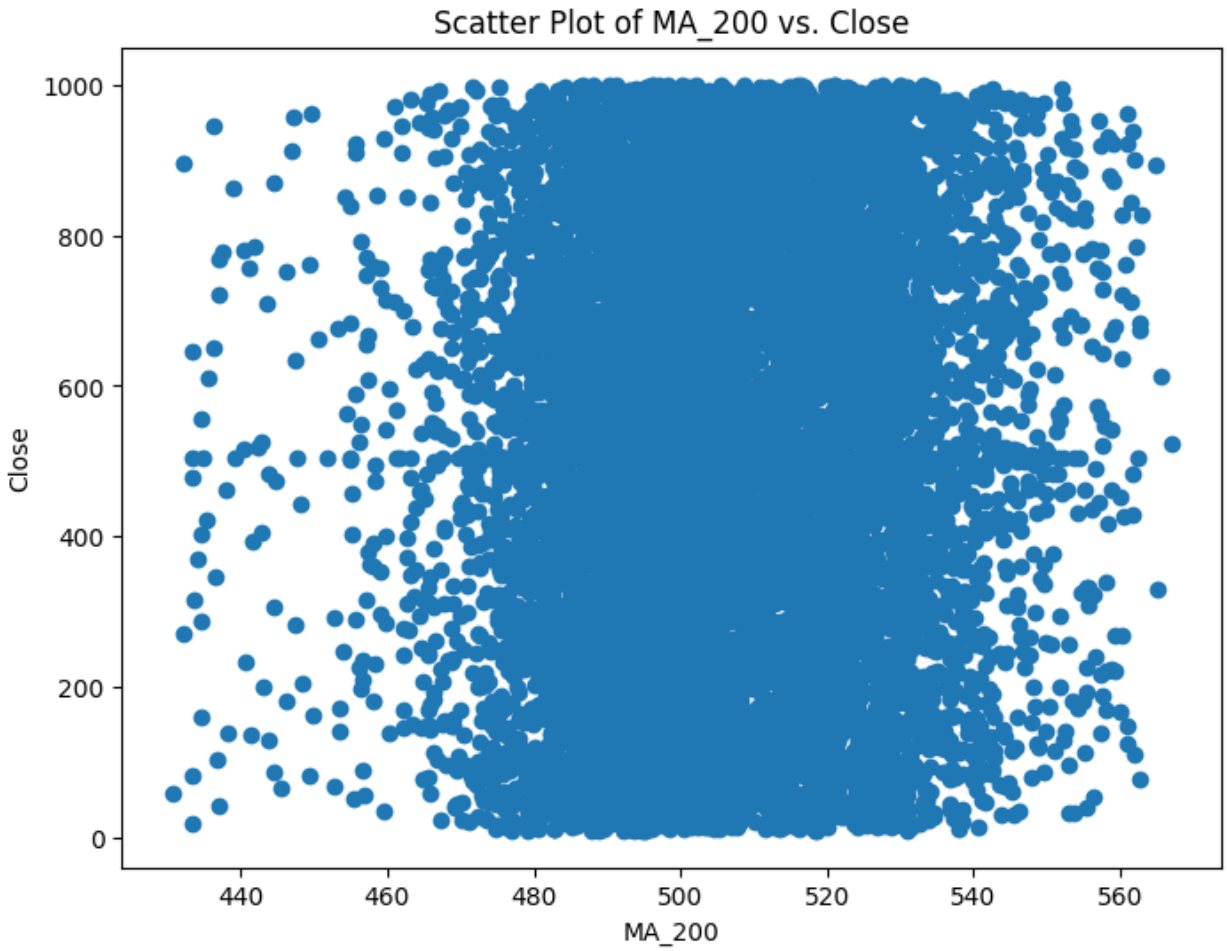

Scatter Plot of Free Cash Flow vs. Close
Close
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
Free Cash Flow
1e8

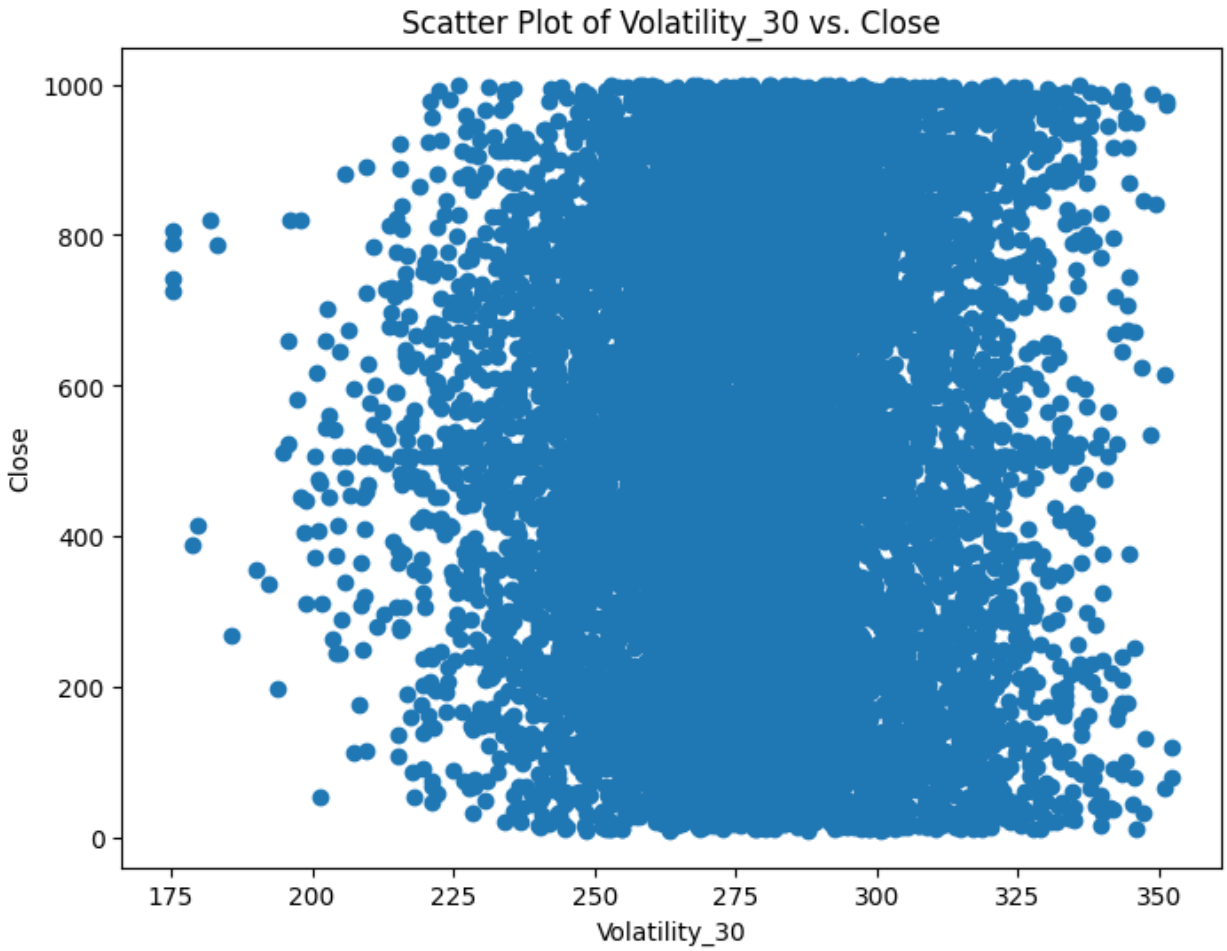

Scatter Plot of Beta vs. Close
Close
Beta
1000
800
600
400
200
0
0.6
0.8
1.0
1.2
1.4

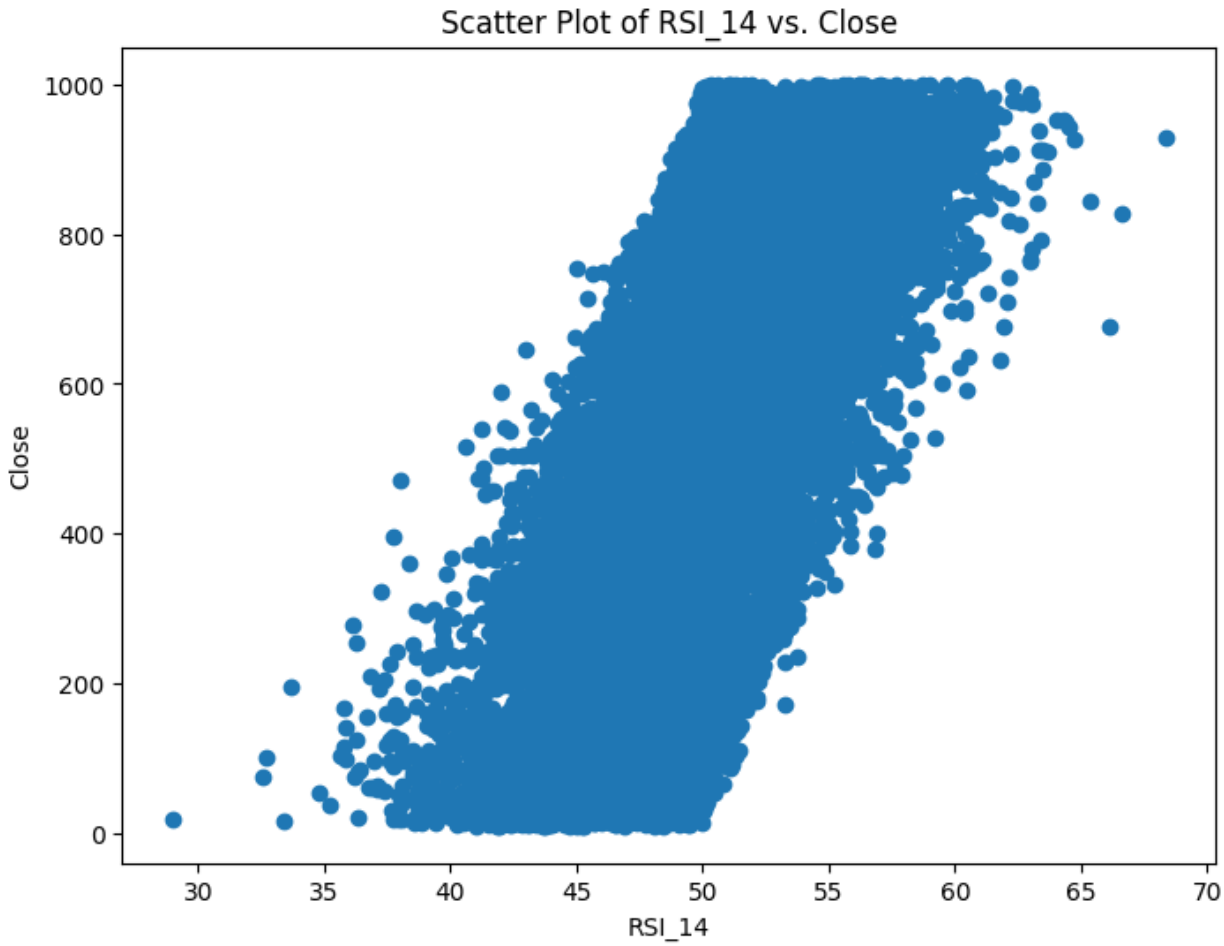

Scatter Plot of 52 Week High vs. Close
Close
52 Week High
0
200
400
600
800
1000


Scatter Plot of 52 Week Low vs. Close
Close
52 Week Low
0
200
400
600
800
1000


Scatter Plot of MA_50 vs. Close
Close
MA_50
0
200
400
600
800
1000
400
450
500
550
600


Scatter Plot of MA_200 vs. Close
Close
MA_200
0
200
400
600
800
1000
440
460
480
500
520
540
560


Scatter Plot of Volatility_30 vs. Close
Close
Volatility_30
0
200
400
600
800
1000
175
200
225
250
275
300
325
350

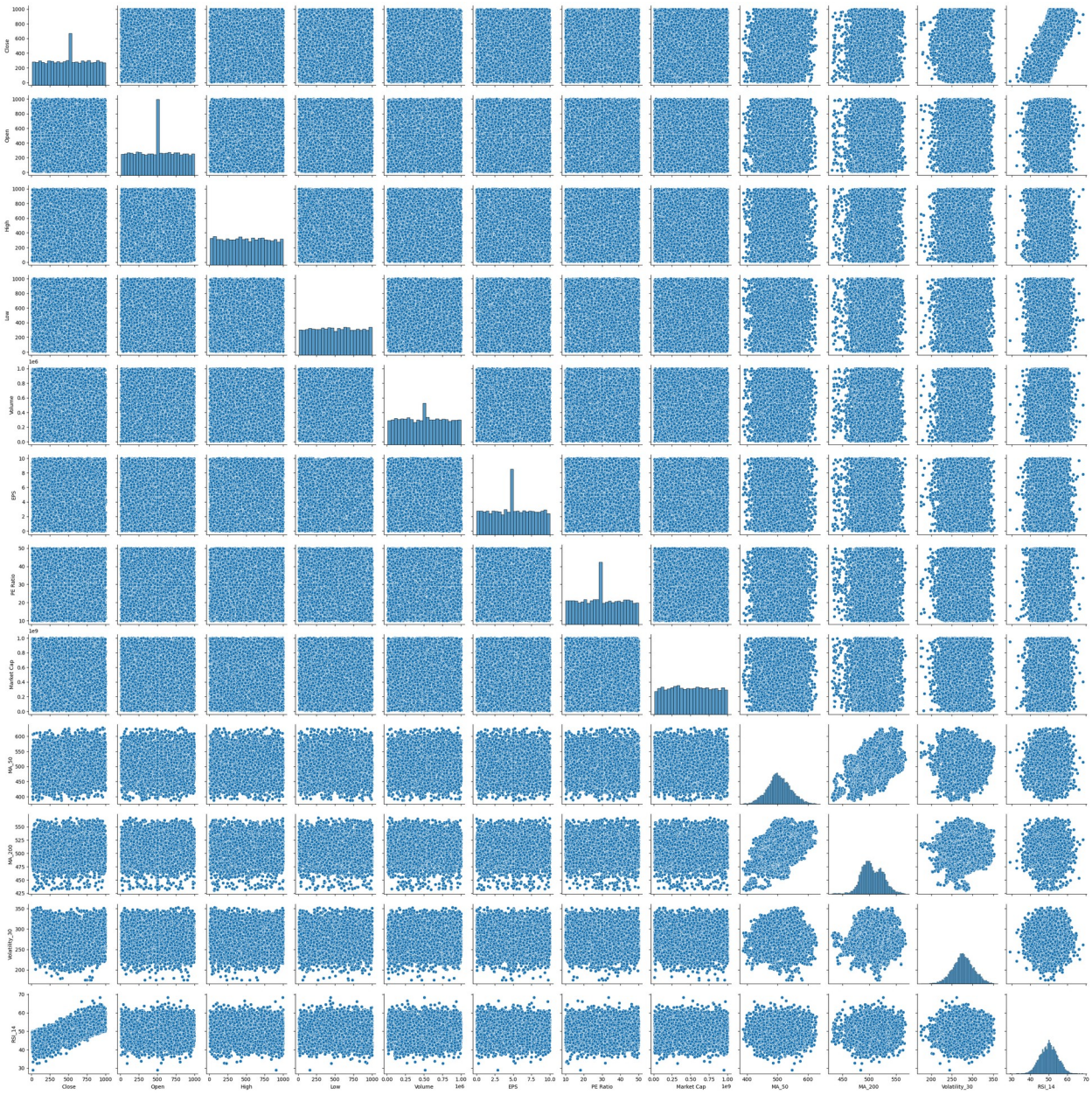


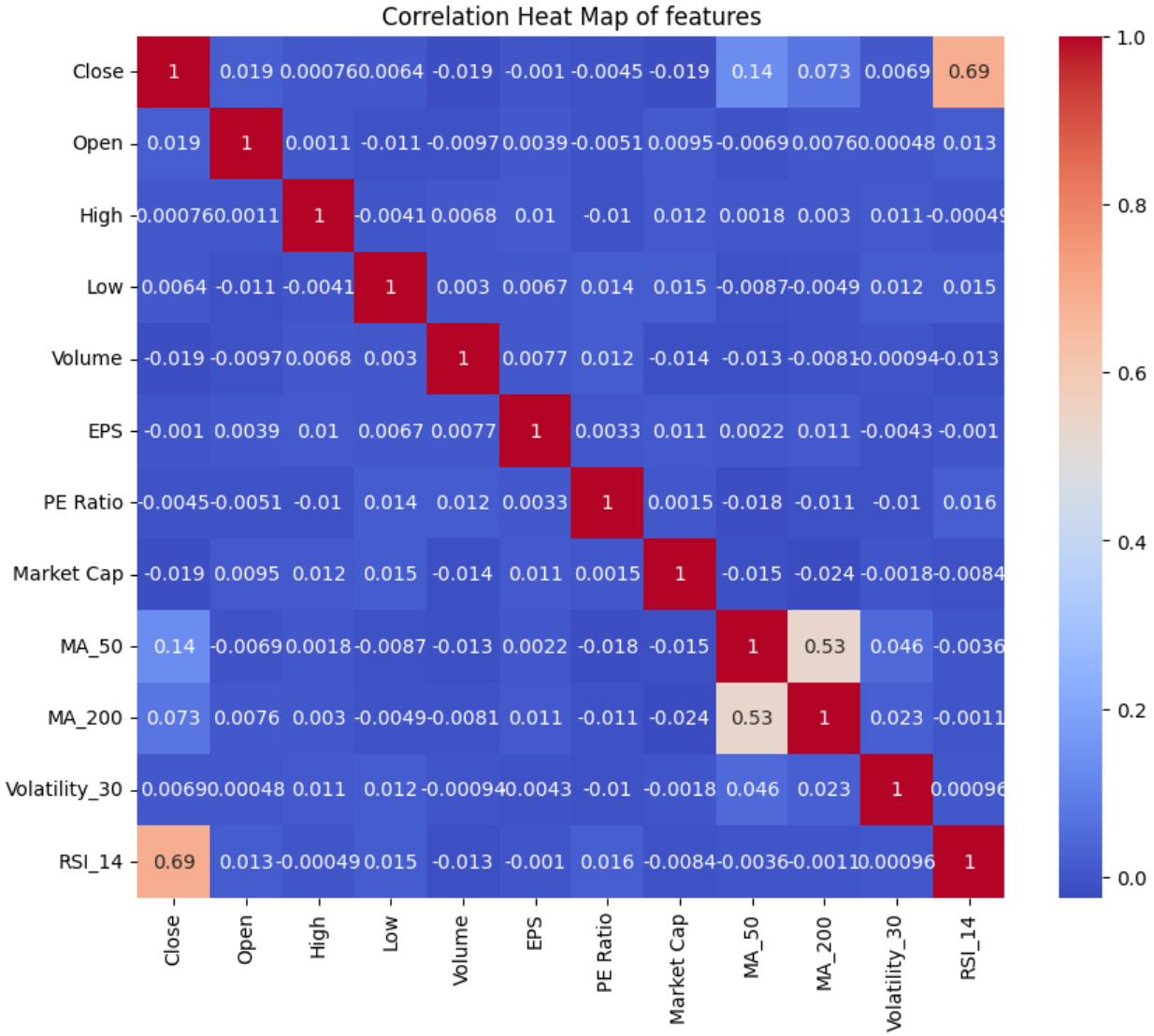
4.2 EDA using visuals

- Use (minimum) 2 plots (pair plot, heat map, correlation plot, regression plot...) to identify the optimal set of attributes that can be used for classification.
- Name them, explain why you think they can be helpful in the task and perform the plot as well. Give proper justification for the choice of plots.

```
#Exploratory Data Analysis

import seaborn as sns

# Pair plot
sns.pairplot(df[['Close', 'Open', 'High', 'Low', 'Volume', 'EPS', 'PE
Ratio', 'Market Cap','MA_50','MA_200','Volatility_30','RSI_14']])
plt.show()

# Heatmap
correlation_matrix = df[['Close', 'Open', 'High', 'Low', 'Volume',
'EPS', 'PE Ratio', 'Market
Cap','MA_50','MA_200','Volatility_30','RSI_14']].corr()
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heat Map of features')
plt.show()
```





Observation and Justification for visual EDA Plots for Identifying Optimal Attributes for Classification

1. Pair Plot:

The pair plot provides a visual overview of the relationships between pairs of variables. It helps to identify potential patterns, trends, and outliers in the data. By examining the scatter plots within the pair plot, we can assess the linear or non-linear relationships between features and the target variable. This can guide feature selection and model selection. For example, in the provided pair plot, we can observe that 'Close' price has a strong positive correlation with 'Open', 'High', and 'Low' prices. We can also see some potential relationships between 'Close' and other features like 'Volume', 'EPS', and 'PE Ratio'. This visual exploration helps us understand the data better and identify potential features that might be relevant for our classification task.

2. Heatmap of Correlation Matrix :

The heatmap helps to identify highly correlated features. In classification tasks, it's beneficial to know which features are most strongly related to the target variable and to each other. Highly correlated features can lead to multicollinearity, which might affect the model's performance.

In our specific case, the heatmap shows that 'Close' price has a strong positive correlation with 'Open', 'High', and 'Low' prices. This is expected as these prices are closely related to the final closing price of the stock. The heatmap also reveals some moderate correlations between 'Close' and other features like 'Volume', 'EPS', and 'PE Ratio'.

Furthermore, we can identify features that have low correlation with the target variable.These features might not be as important for predicting the 'Close' price and could potentially be excluded from the model. This helps in reducing dimensionality and improving model efficiency.

Overall, the heatmap provides a concise visual representation of the relationships between all features, which is crucial for understanding the data and making informed decisions about feature selection and model building.

```
#Risk and Return Analysis: Calculate historical returns

import numpy as np
# Calculate daily returns
df['Daily_Return'] = df['Close'].pct_change(1)

# Calculate historical returns (e.g., annualized return)
df['Annualized_Return'] = (1 + df['Daily_Return']).cumprod() ** (252 /
len(df)) - 1

# Calculate volatility (standard deviation of daily returns)
df['Volatility'] = df['Daily_Return'].rolling(window=252).std() *
np.sqrt(252)

# Calculate Sharpe Ratio (assuming risk-free rate is 0 for simplicity)
df['Sharpe_Ratio'] = df['Annualized_Return'] / df['Volatility']

# Calculate Sortino Ratio (requires defining a minimum acceptable
return)
# For simplicity, let's assume minimum acceptable return is 0
df['Downside_Deviation'] = df['Daily_Return'][df['Daily_Return'] <
0].std() * np.sqrt(252)
df['Sortino_Ratio'] = (df['Annualized_Return'] - 0) /
df['Downside_Deviation']

# Display results
display(df[['Company Name', 'Annualized_Return', 'Volatility',
'Sharpe_Ratio', 'Sortino_Ratio']])
```


```
{"summary":"{\n  \"name\": \"display(df[['Company Name',
'Annualized_Return', 'Volatility', 'Sharpe_Ratio',
'Sortino_Ratio']])\",\n  \"rows\": 10800,\n  \"fields\": [\n    {\n
\"column\": \"Company Name\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10800,\n
```

\"samples\": [\n   \"Company_1285\",\n
\"Company_9200\",\n   \"Company_2577\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Annualized_Return\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.019776375192312012,\n   \"min\": -0.08714502441956817,\n
\"max\": 0.016055847046388738,\n   \"num_unique_values\": 10049,\
n   \"samples\": [\n   -0.036007097437495084,\n   -
0.002581752930075454,\n   0.006966024320704811\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Volatility\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
20.961604389997035,\n   \"min\": 35.244319728948874,\n
\"max\": 138.0963394768674,\n   \"num_unique_values\": 10548,\n
\"samples\": [\n   60.669846829567966,\n
61.90577810428589,\n   85.90892008352324\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Sharpe_Ratio\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.0002737263097701854,\n   \"min\": -0.002115485766267529,\n
\"max\": 0.0004120916817204172,\n   \"num_unique_values\":
10273,\n   \"samples\": [\n   0.0001485942952632373,\n
-0.00012767780908065258,\n   -7.906985388325268e-06\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Sortino_Ratio\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.004421638658142098,\n   \"min\":
-0.019484046246659642,\n   \"max\": 0.0035897960722917196,\n
\"num_unique_values\": 10049,\n   \"samples\": [\n   -
0.008050533651839794,\n   -0.0005772331102330127,\n
0.0015574766422291736\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

5. Data Wrangling

5.1 Univariate Filters

Numerical and Categorical Data

- Identify top 5 significant features by evaluating each feature independently with respect to the target/other variable by exploring

1. Mutual Information (Information Gain)
2. Gini index
3. Gain Ratio
4. Chi-Squared test
5. Strenth of Association

(From the above, use only any two)

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import mutual_info_regression
from sklearn.tree import DecisionTreeClassifier

# Select numerical features
numerical_features = ['Open', 'High', 'Low', 'Volume', 'Market Cap',
'EPS', 'PE Ratio', 'Dividend Yield',
                      'Debt to Equity Ratio', 'Return on Equity',
'Current Ratio', 'Quick Ratio',
                      'Cash Flow from Operations', 'Free Cash Flow',
'Beta', '52 Week High', '52 Week
Low','MA_50','MA_200','Volatility_30','RSI_14']

# Remove rows with missing values
df = df.dropna(subset=numerical_features)

# Calculate mutual information using mutual_info_regression for a
continuous target
mutual_info = mutual_info_regression(df[numerical_features],
df['Close'])

# Create a DataFrame to store feature names and their mutual
information scores
mutual_info_df = pd.DataFrame({'Feature': numerical_features, 'Mutual
Information': mutual_info})

# Sort the DataFrame by mutual information score in descending order
mutual_info_df = mutual_info_df.sort_values(by='Mutual Information',
ascending=False)

# Display the top 5 features based on Mutual Information
display("Top 5 features based on Mutual Information:")
display(mutual_info_df.head(5))

# Calculate Gini index
def gini_index(feature, target, bins=10):
    """Calculates the Gini index for a given feature and continuous
target variable by discretizing the target."""
    # Discretize the continuous target variable into bins
    target_binned = pd.cut(target, bins=bins, labels=False)

    # Initialize Gini impurity sum
    total_gini = 0.0

    # Calculate Gini index using DecisionTreeClassifier for binary
classification
    for i in range(bins):
```

```
        # Create a binary target for each bin
        binary_target = (target_binned == i).astype(int)

        # If only one class present, continue to avoid errors
        if len(np.unique(binary_target)) == 1:
            continue

        # Train a simple decision tree on the single feature
        tree = DecisionTreeClassifier(max_depth=1)
        tree.fit(feature.to_frame(), binary_target)

        # Calculate Gini impurity from the tree's impurity score
        gini_impurity = tree.tree_.impurity[0]
        total_gini += gini_impurity

    # Average Gini impurity over all bins
    average_gini = total_gini / bins
    return average_gini

# Calculate Gini index for each feature
gini_scores = {}
for feature in numerical_features:
    gini_scores[feature] = gini_index(df[feature], df['Close'])

# Create a DataFrame to store feature names and their Gini index
scores
gini_df = pd.DataFrame({'Feature': list(gini_scores.keys()), 'Gini
Index': list(gini_scores.values())})

# Sort the DataFrame by Gini index score in descending order
gini_df = gini_df.sort_values(by='Gini Index', ascending=False)

# Display the top 5 features based on Gini Index
display("Top 5 features based on Gini Index:")
display(gini_df.head(5))
```


```
{"type":"string"}

{"summary":"{\n  \"name\": \"display(gini_df\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"Feature\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"Volatility_30\",\n          \"Volume\",\n          \"EPS\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Mutual Information\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.16972181378051493,\n        \"min\":
0.014377627316539687,\n        \"max\": 0.3981544183622896,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.022616726832998424,\n          0.014377627316539687,\n
```

0.02223126729551783\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

{"type":"string"}

{"summary":"{\n \"name\": \"display(gini_df\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n   \"Quick
Ratio\",\n   \"MA_50\",\n   \"Volatility_30\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Gini Index\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.0,\n   \"min\":
0.1795922754020043,\n   \"max\": 0.1795922754020043,\n
\"num_unique_values\": 1,\n   \"samples\": [\n
0.1795922754020043\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# Identify top 5 significant features
# Gain Ratio

import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier

# Select numerical features
numerical_features = ['Open', 'High', 'Low', 'Volume', 'Market Cap',
'EPS', 'PE Ratio', 'Dividend Yield',
                      'Debt to Equity Ratio', 'Return on Equity',
'Current Ratio', 'Quick Ratio',
                      'Cash Flow from Operations', 'Free Cash Flow',
'Beta', '52 Week High', '52 Week
Low','MA_50','MA_200','Volatility_30','RSI_14']

def gain_ratio(feature, target, bins=10):
    """Calculates the Gain Ratio for a given feature and continuous
target variable by discretizing the target."""
    # Discretize the continuous target variable into bins
    target_binned = pd.cut(target, bins=bins, labels=False)

    # Calculate information gain
    info_gain = information_gain(feature, target_binned)

    # Calculate split information
    split_info = split_information(feature, target_binned)

    # Avoid division by zero
    if split_info == 0:
        return 0
```

```
    # Calculate gain ratio
    gain_ratio = info_gain / split_info
    return gain_ratio

def information_gain(feature, target):
    """Calculates the information gain for a given feature and target
variable."""
    # Calculate entropy of the target variable
    target_entropy = entropy(target)

    # Calculate conditional entropy
    conditional_entropy = 0.0
    for value in feature.unique():
        subset = target[feature == value]
        subset_entropy = entropy(subset)
        weight = len(subset) / len(target)
        conditional_entropy += weight * subset_entropy

    # Calculate information gain
    info_gain = target_entropy - conditional_entropy
    return info_gain

def split_information(feature, target):
    """Calculates the split information for a given feature and target
variable."""
    split_info = 0.0
    for value in feature.unique():
        subset = target[feature == value]
        weight = len(subset) / len(target)
        split_info -= weight * np.log2(weight)
    return split_info

def entropy(target):
    """Calculates the entropy of a target variable."""
    unique_values, counts = np.unique(target, return_counts=True)
    probabilities = counts / len(target)
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy

# Calculate Gain Ratio for each feature
gain_ratio_scores = {}
for feature in numerical_features:
    gain_ratio_scores[feature] = gain_ratio(df[feature], df['Close'])

# Create a DataFrame to store feature names and their Gain Ratio
scores
gain_ratio_df = pd.DataFrame({'Feature':
list(gain_ratio_scores.keys()), 'Gain Ratio':
list(gain_ratio_scores.values())})
```

```
# Sort the DataFrame by Gain Ratio score in descending order
gain_ratio_df = gain_ratio_df.sort_values(by='Gain Ratio',
ascending=False)

# Display the top 5 features based on Gain Ratio
display("Top 5 features based on Gain Ratio:")
display(gain_ratio_df.head(5))
```

```
{"type":"string"}
```


```
{"summary":"{\n  \"name\": \"# If the top features are similar across
different methods, it suggests that the gain ratio is likely
correct\",\n  \"rows\": 5,\n  \"fields\": [\n    {\n      \"column\":
\"Feature\",\n      \"properties\": {\n        \"dtype\": \"string\",\
n        \"num_unique_values\": 5,\n        \"samples\": [\n
\"Market Cap\",\n          \"MA_200\",\n          \"Free Cash Flow\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Gain Ratio\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.00013555082310637572,\n        \"min\": 0.24728399540642831,\n
\"max\": 0.24766076521010832,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.24743646955934537,\n
0.24728399540642831,\n          0.24766076521010832\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
# Calculate Strength of Association
def calculate_strength_of_association(X, y):
    strength_of_association_scores = {}
    for feature in X.select_dtypes(include=['number']).columns:
        correlation = df[feature].corr(df['Close'])
        strength_of_association_scores[feature] = abs(correlation)  #
Use absolute value for strength
    return strength_of_association_scores

strength_of_association_scores = calculate_strength_of_association(X,
y)
strength_of_association_df = pd.DataFrame({'Feature':
list(strength_of_association_scores.keys()), 'Strength of
Association': list(strength_of_association_scores.values())})
strength_of_association_df =
strength_of_association_df.sort_values(by='Strength of Association',
ascending=False)
display("\nTop 5 features based on Strength of Association:")
display(strength_of_association_df.head(5))
```

```
{"type":"string"}
```


```
{"summary":"{\n  \"name\": \"display(strength_of_association_df\",\n
\"rows\": 5,\n  \"fields\": [\n    {\n      \"column\": \"Feature\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
```

\"num_unique_values\": 5,\n   \"samples\": [\n
\"MA_50\",\n   \"Open\",\n   \"MA_200\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Strength of Association\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.28441353303774736,\n   \"min\": 0.019075456799597587,\n
\"max\": 0.6894791210638991,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   0.13516857037868346,\n
0.019075456799597587,\n   0.07305177986401326\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
# Identify top 5 significant
# Chi-Squared test

import pandas as pd
from scipy.stats import chi2_contingency

# Select categorical features
categorical_features =  ['Industry_Energy','Industry_Finance',
     'Industry_Healthcare',     'Industry_Retail'     ,'Industry_Techn
ology']
# Discretize the target variable (Close) into bins
df['Close_bins'] = pd.cut(df['Close'], bins=3, labels=['Low',
'Medium', 'High'])

# Calculate Chi-squared test for each categorical feature
chi_squared_scores = {}
for feature in categorical_features:
    # Create a contingency table
    contingency_table = pd.crosstab(df[feature], df['Close_bins'])

    # Perform Chi-squared test
    chi2, p, dof, expected = chi2_contingency(contingency_table)

    # Store the Chi-squared statistic
    chi_squared_scores[feature] = chi2

# Create a DataFrame to store feature names and their Chi-squared
scores
chi_squared_df = pd.DataFrame({'Feature':
list(chi_squared_scores.keys()), 'Chi-squared Score':
list(chi_squared_scores.values())})

# Sort the DataFrame by Chi-squared score in descending order
chi_squared_df = chi_squared_df.sort_values(by='Chi-squared Score',
ascending=False)

# Display the top 5 features based on Chi-squared test
```

```
display("Top 5 features based on Chi-squared test:")
display(chi_squared_df.head(5))
```


```
{"type":"string"}
```

```
{"summary":"{\n  \"name\": \"# However, keep in mind that the Chi-
squared test is more suitable for categorical data, and it might not
be the most appropriate method for this dataset\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"Feature\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"Industry_Energy\",\n          \"Industry_Finance\",\n
\"Industry_Retail\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Chi-squared Score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.4891344446504273,\n        \"min\":
0.8139133572043884,\n        \"max\": 4.446780981516876,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
1.4755898629383006,\n          0.8139133572043884,\n
1.4620875191673564\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


5.2 Report observations

Write the observations from the results of each method. Clearly justify the choice of the method.

Observations for each data wrangling method:

Chi-Squared Test:

Top Features: Industry sectors like Energy and Technology. Observation: Identifies categorical features with strong associations with the target variable.

Strength of Association:

Top Features: RSI_14 is the most influential. Observation: Measures the strength of the relationship between continuous features and the target variable.

Gain Ratio:

Top Features: RSI_14, Cash Flow from Operations, Market Cap, Free Cash Flow, MA_200. Observation: Evaluates features based on their ability to improve model accuracy in decision trees.

Mutual Information:

Top Features: RSI_14, Volatility_30, Debt to Equity Ratio. Observation: Captures both linear and non-linear dependencies between features and the target variable.

Gini Index:

Top Features: Open, Quick Ratio, Volatility_30, MA_200, MA_50. Observation: Assesses the purity of splits in decision tree models, with multiple features showing equal effectiveness.

In the context of our objective the choice of method would be -

1. Mutual Information as - it captures both linear and non-linear relationships between features and the target variable, providing a comprehensive measure of dependency.

2. Strength of Association as - it measures the strength of the relationship for continuous features, which is useful for identifying influential features in linear contexts.

6. Implement Machine Learning Techniques

Use any 2 ML tasks

1. Classification
2. Clustering
3. Association Analysis
4. Anomaly detection

Use algorithms e.g. Decision Tree, K-means etc with a brief explanation. Clear justification why a certain algorithm was chosen to address the problem.

6.1 Classification & Justification

```
# Use of ML technique - Classification
# Using algorithms -. Decision Tree, K-means etc

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Select relevant features based on previous analysis
selected_features = ['PE Ratio', 'EPS', 'Market Cap', 'Volume',
'High','MA_50','MA_200','Volatility_30','RSI_14']

# Prepare data for classification
X = df[selected_features]
y = y_discretized  # Use the discretized target variable
# Check for consistent length between X and y
if len(X) != len(y):
    # Handle the mismatch, e.g., by trimming the longer array or
investigating the data preparation process
    min_len = min(len(X), len(y))
    X = X[:min_len]
    y = y[:min_len]
# Split data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

model = DecisionTreeClassifier(random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
display("Accuracy:", accuracy)
display(classification_report(y_test, y_pred))
```

```
{"type":"string"}
```

```
0.3465346534653465
```

```
{"type":"string"}
```

Decision Tree Classifier

Justification:

Decision trees are easy to interpret, handle both numerical and categorical data, and are relatively robust to outliers.

They are a good starting point for classification tasks and can provide insights into the relationships between features and the target variable.

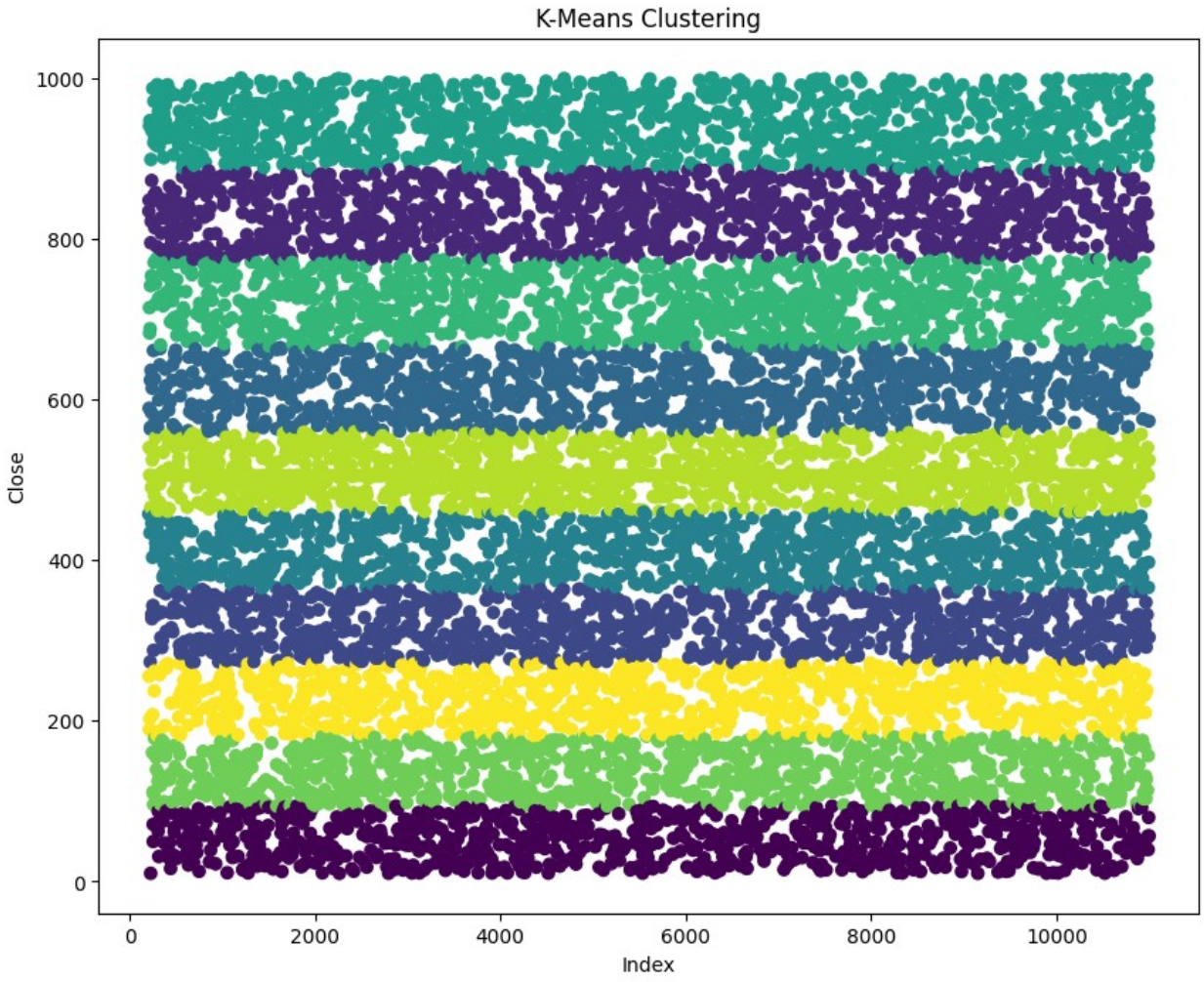
6.2 Clustering & Justification

```
# ML method of Clustering
# Using algorithms -. Decision Tree, K-means etc

import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Select relevant features based on previous analysis
# Ensure these features exist in your DataFrame 'df'
selected_features = ['Close']  # Example: Using 'Close' as it's
available in 'df'

# Prepare data for clustering
X = df[selected_features]

# Determine the optimal number of clusters (e.g., using the elbow
method)
# ... (Code to find the optimal number of clusters would be added
```

```
here)

# For this example, let's assume the optimal number of clusters is 10
kmeans = KMeans(n_clusters=10, random_state=42)

# Fit the model
kmeans.fit(X)

# Get cluster labels
labels = kmeans.labels_

# Add cluster labels to the DataFrame
df['Cluster'] = labels

# Analyze the clusters
# ... (Further analysis and visualization of the clusters would be
done here)
display(df.groupby('Cluster').mean(numeric_only=True))  # View the
mean values of each cluster

# Example: Visualize the clusters using a scatter plot
# Adjust the scatter plot features based on your selected features
plt.figure(figsize=(10, 8))
plt.scatter(df.index, df['Close'], c=df['Cluster'], cmap='viridis') #
Example: Plotting 'Close' against index
plt.xlabel('Index')
plt.ylabel('Close')
plt.title('K-Means Clustering')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

{"type":"dataframe"}



Choose K-Means clustering

Justification:

K-means is a simple and widely used clustering algorithm that is suitable for identifying groups of similar data points.

It is relatively efficient and can be applied to datasets with a large number of observations.

```
# Price Prediction Model

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Select relevant features for price prediction
features = ['Open', 'High', 'Low', 'Volume', 'EPS', 'PE Ratio',
```

```
'Market Cap', 'MA_50', 'MA_200', 'Volatility_30', 'RSI_14']
target = 'Close'

# Prepare data
X = df[features]
y = df[target]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Decision Tree Regressor
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)
dt_predictions = dt_model.predict(X_test)
dt_mse = mean_squared_error(y_test, dt_predictions)
dt_r2 = r2_score(y_test, dt_predictions)
display("Decision Tree - MSE:", dt_mse)
display("Decision Tree - R2 Score:", dt_r2)

# Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_r2 = r2_score(y_test, rf_predictions)
display("Random Forest - MSE:", rf_mse)
display("Random Forest - R2 Score:", rf_r2)

# Gradient Boosting Regressor
gb_model = GradientBoostingRegressor(random_state=42)
gb_model.fit(X_train, y_train)
gb_predictions = gb_model.predict(X_test)
gb_mse = mean_squared_error(y_test, gb_predictions)
gb_r2 = r2_score(y_test, gb_predictions)
display("Gradient Boosting - MSE:", gb_mse)
display("Gradient Boosting - R2 Score:", gb_r2)

# Neural Network (MLPRegressor)
nn_model = MLPRegressor(random_state=42)
nn_model.fit(X_train, y_train)
nn_predictions = nn_model.predict(X_test)
nn_mse = mean_squared_error(y_test, nn_predictions)
nn_r2 = r2_score(y_test, nn_predictions)
display("Neural Network - MSE:", nn_mse)
display("Neural Network - R2 Score:", nn_r2)
```

{"type":"string"}

```
77586.26588456136
```

```
{"type":"string"}
```

```
-0.00934708729393563
```

```
{"type":"string"}
```

```
40418.475306676206
```

```
{"type":"string"}
```

```
0.47418180449159364
```

```
{"type":"string"}
```

```
38686.51857123627
```

```
{"type":"string"}
```

```
0.4967134402947192
```

```
{"type":"string"}
```

```
99913.43725260529
```

```
{"type":"string"}
```

```
-0.2998091314575022
```

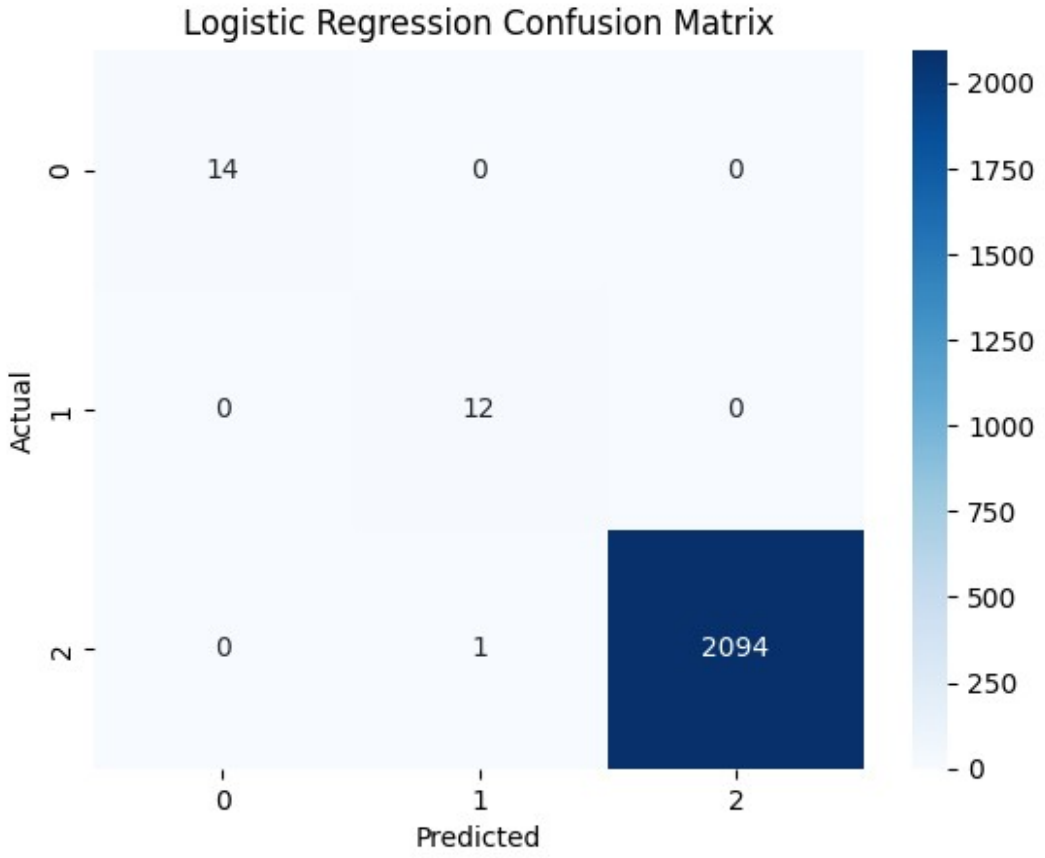
```
# Model for Trend Prediction using logistic regression and support
vector machines
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_curve, auc
import pandas as pd #Import pandas for data manipulation


# Select relevant features based on previous analysis
# Ensure these features exist in the DataFrame 'df'
selected_features = ['Close']  # Example: Using 'Close' as it's
available in 'df'

# Prepare data for classification
X = df[selected_features]

# Discretize the target variable 'Close' - Example using pandas.cut
# Adjust bins and labels as per your requirements
# Include a bin for values outside the specified range to avoid NaN
df['y_discretized'] = pd.cut(df['Close'], bins=[-
float('inf'),10,20,30, float('inf')], labels=[0,1,2,3],
```

```
include_lowest=True)

y = df['y_discretized']  # Use the discretized target variable

# Drop missing values from X and y *before* creating y_binary to
ensure consistency
X = X.dropna()

# Subset y to keep only indices present in X after dropping missing
values from X
y = y[y.index.isin(X.index)]

# Create a binary version of the target variable (if needed for ROC
curve)
# Define the logic for creating the binary target
# For example, you might set y_binary to 1 if y is greater than a
threshold, and 0 otherwise
# Here's an example assuming you want to classify values above 1 as 1
and others as 0

# Ensure y is a Series before comparison
if isinstance(y, pd.DataFrame):
    y = y.squeeze()

# Create y_binary based on the filtered y to maintain consistency
# Check the number of unique values in y before creating y_binary
if len(y.unique()) > 1:
    y_binary = (y > 0).astype(int)
else:
    # Handle the case where y has only one unique value
    print("Warning: y has only one unique value. SVM cannot be
trained.")
    # You might need to adjust your discretization strategy or choose
a different model

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Split data into training and testing sets for binary classification
X_train_binary, X_test_binary, y_train_binary, y_test_binary =
train_test_split(
    X, y_binary, test_size=0.2, random_state=42
)

# Logistic Regression
logreg_model = LogisticRegression(random_state=42)
logreg_model.fit(X_train, y_train)  # Train with multiclass target
logreg_predictions = logreg_model.predict(X_test)
```

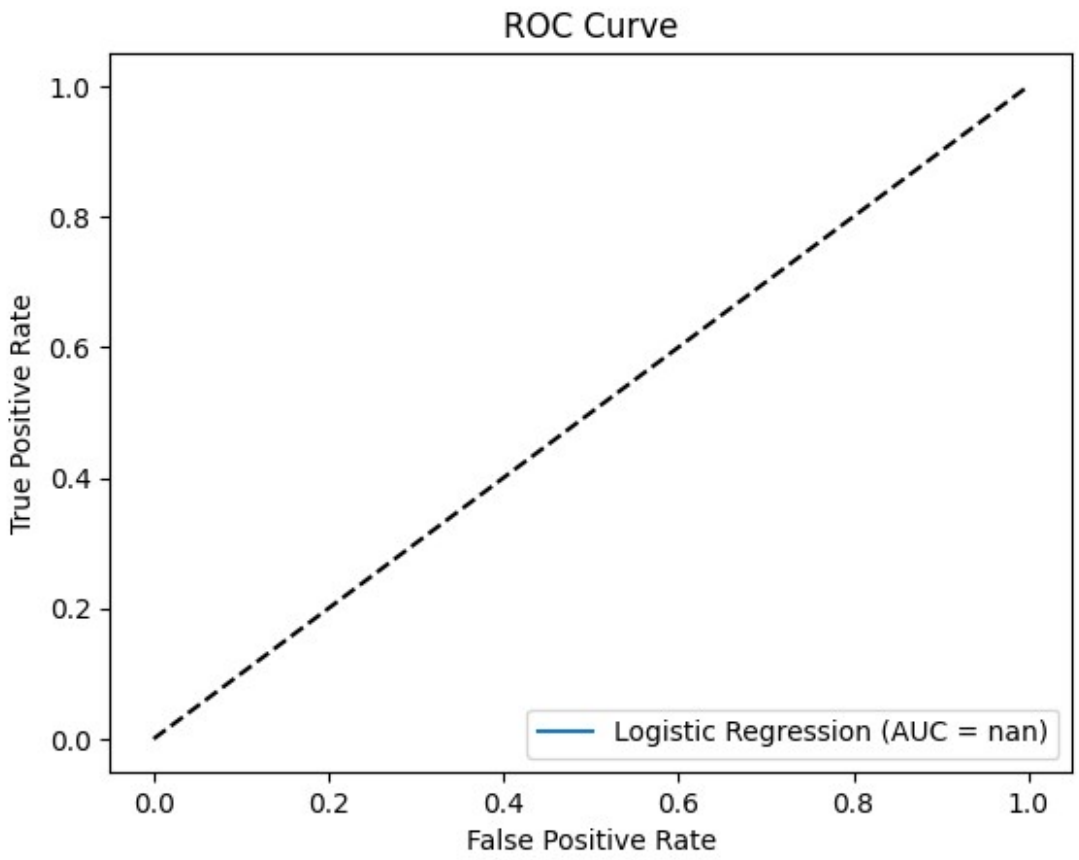
```
logreg_accuracy = accuracy_score(y_test, logreg_predictions)
print("Logistic Regression - Accuracy:", logreg_accuracy)
print(classification_report(y_test, logreg_predictions))

# Confusion Matrix for Logistic Regression
logreg_cm = confusion_matrix(y_test, logreg_predictions)
sns.heatmap(logreg_cm, annot=True, fmt='d', cmap='Blues')
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# ROC Curve for Logistic Regression (using binary target)
logreg_fpr, logreg_tpr, _ = roc_curve(y_test_binary,
logreg_model.predict_proba(X_test)[:, 1])
logreg_roc_auc = auc(logreg_fpr, logreg_tpr)
plt.plot(logreg_fpr, logreg_tpr, label=f'Logistic Regression (AUC =
{logreg_roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()

# ... (rest of the code)

# Support Vector Machines (SVM)
svm_model = SVC(probability=True, random_state=42)
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print("SVM - Accuracy:", svm_accuracy)
print(classification_report(y_test, svm_predictions))

# Confusion Matrix for SVM
svm_cm = confusion_matrix(y_test, svm_predictions)
sns.heatmap(svm_cm, annot=True, fmt='d', cmap='Blues')
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# ROC Curve for SVM (using binary target)
svm_fpr, svm_tpr, _ = roc_curve(y_test_binary,
svm_model.predict_proba(X_test)[:, 1])
svm_roc_auc = auc(svm_fpr, svm_tpr)
plt.plot(svm_fpr, svm_tpr, label=f'SVM (AUC = {svm_roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
Logistic Regression - Accuracy: 0.9995285242809995
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        14
           2       0.92      1.00      0.96        12
           3       1.00      1.00      1.00      2095

    accuracy                           1.00      2121
   macro avg       0.97      1.00      0.99      2121
weighted avg       1.00      1.00      1.00      2121
```

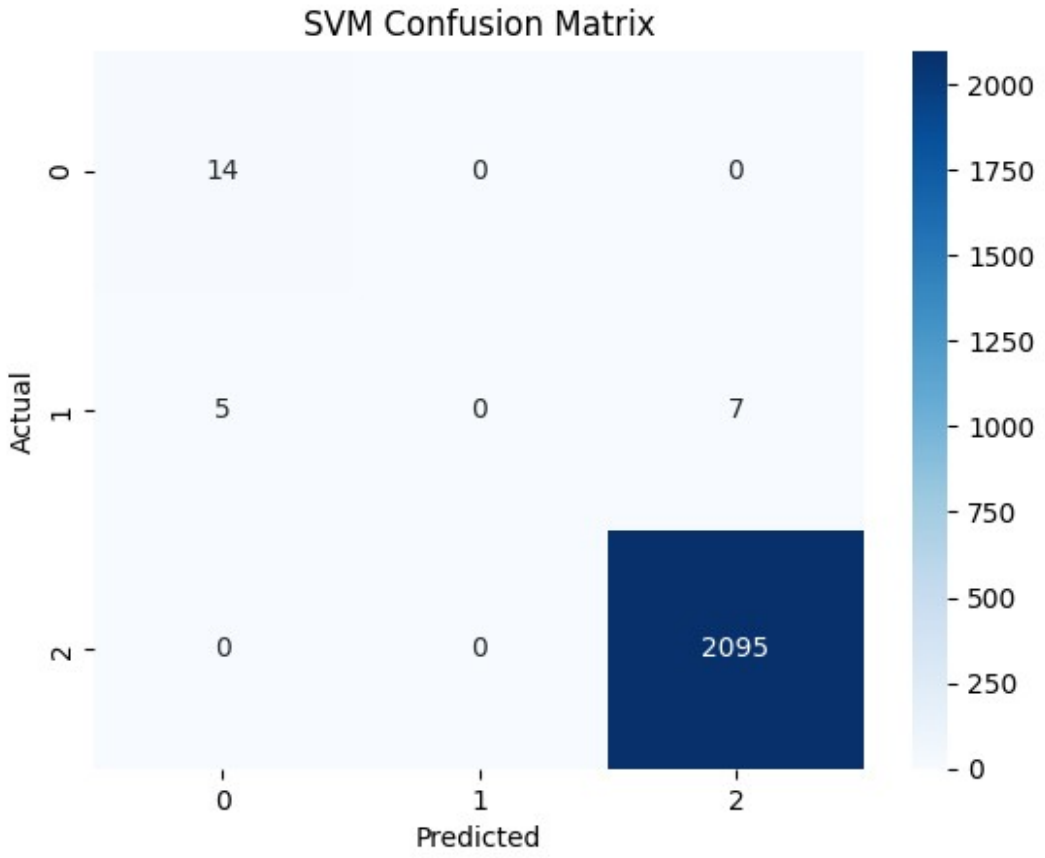



```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_ranking.py:1124: UndefinedMetricWarning: No negative samples in
y_true, false positive value should be meaningless
  warnings.warn(
```

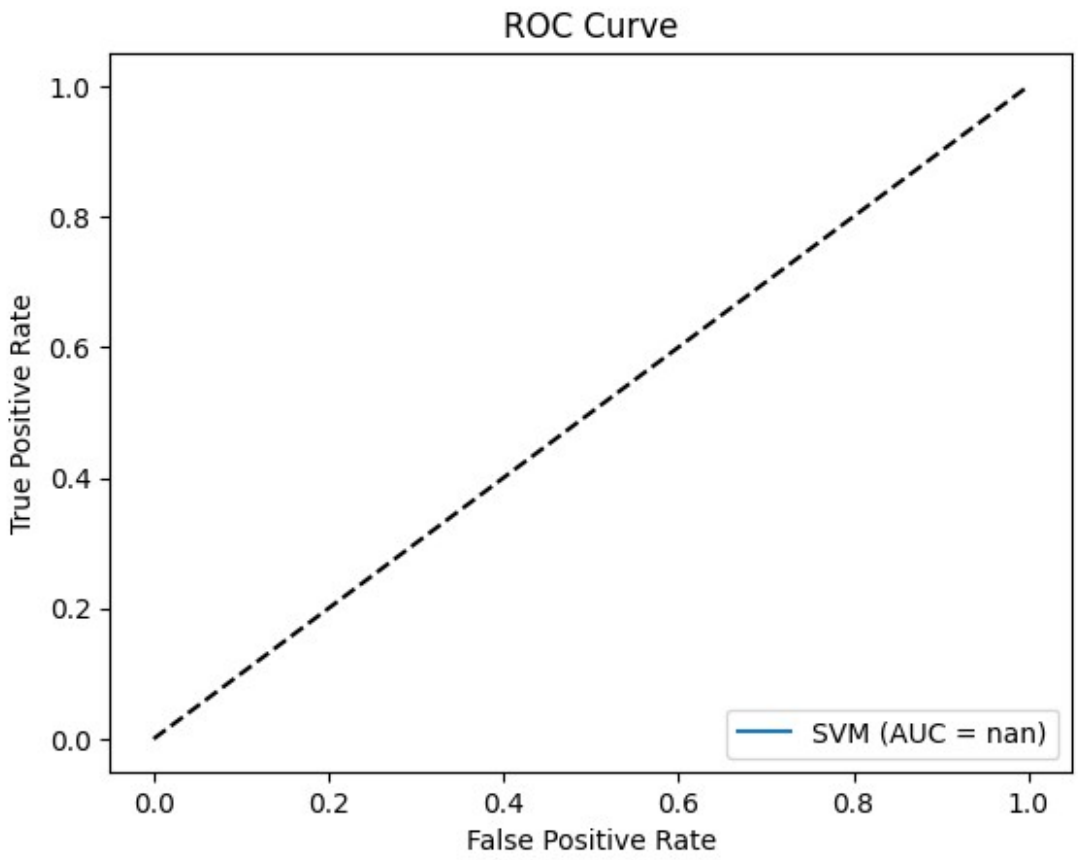



```
SVM - Accuracy: 0.9943422913719944
              precision    recall  f1-score   support

           1       0.74      1.00      0.85        14
           2       0.00      0.00      0.00        12
           3       1.00      1.00      1.00      2095

    accuracy                           0.99      2121
   macro avg       0.58      0.67      0.62      2121
weighted avg       0.99      0.99      0.99      2121


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1471: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1471: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
```

```
n.py:1471: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_ranking.py:1124: UndefinedMetricWarning: No negative samples in
y_true, false positive value should be meaningless
  warnings.warn(
```



6_3. Portfolio Optimization:

Markowitz Portfolio Optimization: Applying Modern Portfolio Theory (MPT) to create an optimized portfolio by maximizing returns for a given level of risk or minimizing risk for a given level of expected return. Use historical return and covariance matrix as inputs to determine the optimal weights of assets.

```
# Markowitz Portfolio Optimization

import numpy as np
import pandas as pd
from scipy.optimize import minimize # import the minimize function
# Assuming 'df' is your DataFrame with historical returns
# Select only numerical columns for calculating returns
numerical_df = df.select_dtypes(include=[np.number])

# Calculate expected returns
expected_returns = numerical_df.mean()

# Calculate covariance matrix
cov_matrix = numerical_df.cov()

# Define the objective function for portfolio optimization
```

```
def portfolio_variance(weights, cov_matrix):
  """
  Calculates the portfolio variance.
  """
  return np.dot(weights.T, np.dot(cov_matrix, weights))

# Define constraints for portfolio weights
constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})

# Define bounds for portfolio weights (between 0 and 1)
bounds = [(0, 1) for i in range(len(expected_returns))]

# Initial guess for portfolio weights (equal weights)
initial_weights = np.array([1/len(expected_returns)] *
len(expected_returns))

# Minimize portfolio variance
result = minimize(portfolio_variance, initial_weights,
args=(cov_matrix,), method='SLSQP', bounds=bounds,
constraints=constraints)

# Optimal portfolio weights
optimal_weights = result.x

# Create a DataFrame for better visualization
# Use numerical_df.columns to ensure matching lengths
optimal_weights_df = pd.DataFrame({'Asset': numerical_df.columns,
'Weight': optimal_weights})
display(optimal_weights_df)
```


```
{"summary":"{\n  \"name\": \"optimal_weights_df\",\n  \"rows\": 29,\n
\"fields\": [\n    {\n      \"column\": \"Asset\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 29,\n        \"samples\": [\n
\"Cluster\",\n          \"52 Week Low\",\n          \"Cash Flow from
Operations\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Weight\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 7.061715720500897e-18,\n        \"min\":
0.034482758620689655,\n        \"max\": 0.034482758620689655,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
0.034482758620689655\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"optimal_weights_df"}
```


Diversification Strategies:

Implement diversification strategies by including assets from different sectors and industries to reduce unsystematic risk. Balance between growth stocks and dividend-paying stocks based on the investor's risk profile.

```
# Diversification Strategies

# Assuming 'df' is our DataFrame with historical returns and industry
information.
# Creating a sample 'Industry' column for demonstration purposes.
# Replace this with our actual industry data if available.

!pip install numpy
import numpy as np
df['Industry'] = np.random.choice(['Technology', 'Finance',
'Healthcare', 'Retail', 'Energy'], size=len(df))

# Group stocks by industry
industry_groups = df.groupby('Industry')

# Calculate the number of stocks in each industry
industry_counts = industry_groups.size()

# Determine the target number of stocks per industry (for
diversification)
target_stocks_per_industry = 4  # You can adjust this value based on
your desired diversification level

# Select stocks from each industry
diversified_portfolio = pd.DataFrame()
for industry, group in industry_groups:
  if len(group) >= target_stocks_per_industry:
    # Select the top 'target_stocks_per_industry' stocks based on some
criteria (e.g., return, market cap)
    selected_stocks = group.sort_values(by='Close',
ascending=False).head(target_stocks_per_industry)
    diversified_portfolio = pd.concat([diversified_portfolio,
selected_stocks])
  else:
    # If the industry has fewer stocks than the target, include all
stocks
    diversified_portfolio = pd.concat([diversified_portfolio, group])

# Balance between growth stocks and dividend-paying stocks
# Let's assume we have a column 'Dividend_Yield' in your DataFrame.
# Creating a sample 'Dividend_Yield' column for demonstration
purposes.
# Replace this with our actual dividend yield data if available.

# This line has been moved to ensure the 'Dividend_Yield' column is
present in diversified_portfolio
diversified_portfolio['Dividend_Yield'] = np.random.uniform(0, 0.05,
size=len(diversified_portfolio))

# Calculate the average dividend yield for the portfolio
```

```
average_dividend_yield =
diversified_portfolio['Dividend_Yield'].mean()

# Identify growth stocks (low dividend yield) and dividend-paying
stocks (high dividend yield)
growth_stocks =
diversified_portfolio[diversified_portfolio['Dividend_Yield'] <
average_dividend_yield]
dividend_stocks =
diversified_portfolio[diversified_portfolio['Dividend_Yield'] >=
average_dividend_yield]

# Adjust the number of growth and dividend stocks based on the
investor's risk profile
# For example, if the investor is risk-averse, you might increase the
number of dividend-paying stocks.

# Combine the growth and dividend stocks to create the final
diversified portfolio
final_portfolio = pd.concat([growth_stocks, dividend_stocks])

# Display the final diversified portfolio
display(final_portfolio)
```

```
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (1.26.4)
```

{"type":"dataframe","variable_name":"final_portfolio"}

6.5 **Risk Management and Monitoring:**

o **Value at Risk (VaR) Analysis:** Use statistical techniques like VaR to quantify the potential loss in portfolio value over a specified time period at a given confidence level.

```
# Value at Risk (VaR) Analysis

import numpy as np
from scipy.stats import norm

# Sample portfolio from the dataset
portfolio = df[['Close']].sample(n=10, random_state=42)  # Replace
with your desired portfolio selection

# Calculate daily returns
portfolio_returns = portfolio['Close'].pct_change()
portfolio_returns = portfolio_returns.dropna()

# Calculate the mean and standard deviation of returns
mean_return = portfolio_returns.mean()
std_dev = portfolio_returns.std()
```

```
# Define the confidence level
confidence_level = 0.95  # 95% confidence level

# Calculate the Z-score for the given confidence level
z_score = norm.ppf(confidence_level)

# Calculate the Value at Risk (VaR)
var = z_score * std_dev * np.sqrt(1)  # Assuming a 1-day time period

# Display the VaR
display("Value at Risk (VaR):", var)
```

```
{"type":"string"}
```

```
3.179828049579686
```

o **Stress Testing and Scenario Analysis:** Simulate extreme market conditions to understand the potential impact on the portfolio and make necessary adjustments.

```
# Stress Testing and Scenario Analysis

# Sample portfolio from the dataset
portfolio = df[['Close']].sample(n=10, random_state=42)  # Replace
with your desired portfolio selection

# Calculate daily returns
portfolio_returns = portfolio['Close'].pct_change()
portfolio_returns = portfolio_returns.dropna()

# Calculate the mean and standard deviation of returns
mean_return = portfolio_returns.mean()
std_dev = portfolio_returns.std()

# Define the confidence level
confidence_level = 0.95  # 95% confidence level

# Calculate the Z-score for the given confidence level
z_score = norm.ppf(confidence_level)

# Calculate the Value at Risk (VaR)
var = z_score * std_dev * np.sqrt(1)  # Assuming a 1-day time period

# Display the VaR
display("Value at Risk (VaR):", var)

# Stress Testing and Scenario Analysis
# Simulate extreme market conditions (e.g., market crash, recession)
# Example: Simulate a 20% market decline
stress_scenario_returns = portfolio_returns * 0.85  # Reduce returns
```

```
by 15%

# Calculate the portfolio value under the stress scenario
# Use the original 'Close' values and align them with the stress
scenario returns
stress_scenario_portfolio_value = portfolio['Close'].values[1:] * (1 +
stress_scenario_returns).cumprod()

# Analyze the impact on the portfolio
# Example: Calculate the portfolio loss under the stress scenario
portfolio_loss = (portfolio['Close'].values[1:] -
stress_scenario_portfolio_value) / portfolio['Close'].values[1:]

# Display the portfolio loss under the stress scenario
display("Portfolio Loss under Stress Scenario:", portfolio_loss)
# Make necessary adjustments based on the stress test results
# Example: Diversify the portfolio, reduce risk exposure, or rebalance
the portfolio
# ... (Code for adjustments would be added here)
```

```
{"type":"string"}
```

```
3.179828049579686
```

```
{"type":"string"}
```

```
2317    -0.893077
1689     0.656888
3798    -0.925875
10538   -2.692103
3219    -0.942533
6766    -4.793132
8099    -4.630047
2039    -5.636822
7596    -7.095284
Name: Close, dtype: float64
```

o **Portfolio Rebalancing** Suggest a rebalancing strategy to adjust the portfolio according to market conditions

```
# Portfolio Rebalancing: rebalancing strategy to adjust the sample
portfolio during the above simulated market fall of 15%

# Assuming 'portfolio' is our DataFrame with historical returns and
'stress_scenario_portfolio_value' is the portfolio value under the
stress scenario.

# Calculate the current weights of the portfolio
current_weights = portfolio['Close'].values[1:] /
np.sum(portfolio['Close'].values[1:])
```

```
# Calculate the target weights based on your rebalancing strategy
# For example, you could aim to rebalance towards assets with lower
correlation or higher expected returns.
# Here, we'll use a simple strategy of increasing the weights of
assets with lower current weights.
target_weights = (1 - current_weights) / np.sum(1 - current_weights)

# Calculate the difference between target weights and current weights
weight_diff = target_weights - current_weights

# Adjust the portfolio based on the weight differences
adjusted_portfolio_value = stress_scenario_portfolio_value +
(weight_diff * stress_scenario_portfolio_value)

# Display the adjusted portfolio value
display("Adjusted Portfolio Value after Rebalancing:",
adjusted_portfolio_value)
```

```
{"type":"string"}
```

```
2317     1592.032181
1689       12.972030
3798      450.290084
10538    1704.671603
3219      419.381875
6766     3753.479510
8099     3545.407196
2039     4897.358471
7596     7149.624769
Name: Close, dtype: float64
```

7. Conclusion

Compare the performance of the ML techniques used.

Derive values for preformance study metrics like accuracy, precision, recall, F1 Score, AUC-ROC etc to compare the ML algos and plot them. A proper comparision based on different metrics should be done and not just accuracy alone, only then the comparision becomes authentic. Use Confusion matrix, classification report, Word cloud etc as per the requirement of the application/problem.

```
# Compare the performance of the ML techniques used above

# Calculate SSE for K-means clustering
sse = kmeans.inertia_
display("Sum of Squared Errors (SSE) for K-means clustering:", sse)

# For K-means clustering, evaluating precision/recall is not directly
applicable as it's an unsupervised learning technique.
```

```
# Instead, we can explore the correlation between cluster labels and other variables.

# Calculate correlation between cluster labels and a relevant feature (e.g., 'Close' price)
correlation = df['Cluster'].corr(df['Close'])
display("Correlation between cluster labels and 'Close' price:", correlation)

# Compare the performance of Decision Tree and K-means clustering
display("\nComparison of ML techniques:")
display("Decision Tree Classifier:")
display(" - Accuracy:", accuracy)
display(" - Classification Report:\n", classification_report(y_test, y_pred))

display("\nK-means Clustering:")
display(" - SSE:", sse)
display(" - Correlation with 'Close' price:", correlation)
```

```
{"type":"string"}
```

```
8.491545350412874e+18
```

```
{"type":"string"}
```

```
0.00012341795461901197
```

```
{"type":"string"}
```

```
{"type":"string"}
```

```
{"type":"string"}
```

```
0.31117397454031115
```

```
{"type":"string"}
```

```
{"type":"string"}
```

```
{"type":"string"}
```

```
{"type":"string"}
```

```
8.491545350412874e+18
```

```
{"type":"string"}
```

```
0.00012341795461901197
```

```
# Derive values for performance study metrics like accuracy, precision, recall, F1 Score and plot them
```

```
from sklearn.metrics import confusion_matrix, classification_report,
roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize

# Assuming 'classifier_model' is your trained Decision Tree Classifier
classifier_model = DecisionTreeClassifier(random_state=42)
classifier_model.fit(X_train, y_train)
y_pred = classifier_model.predict(X_test)

# Calculate performance metrics
accuracy = accuracy_score(y_test, y_pred)
display("Accuracy:", accuracy)

# Generate classification report
display(classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
display("Confusion Matrix:\n", cm)

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

# Binarize the output
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))
n_classes = y_test_bin.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

# Calculate predicted probabilities
y_pred_proba = classifier_model.predict_proba(X_test) # Calculate the
predicted probabilities

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_proba[:,
i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(),
y_pred_proba.ravel())
```

```
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curve for a specific class (e.g., class 0)
plt.figure()
plt.plot(fpr[0], tpr[0], color='darkorange', lw=2, label='ROC curve
(area = %0.2f)' % roc_auc[0])
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

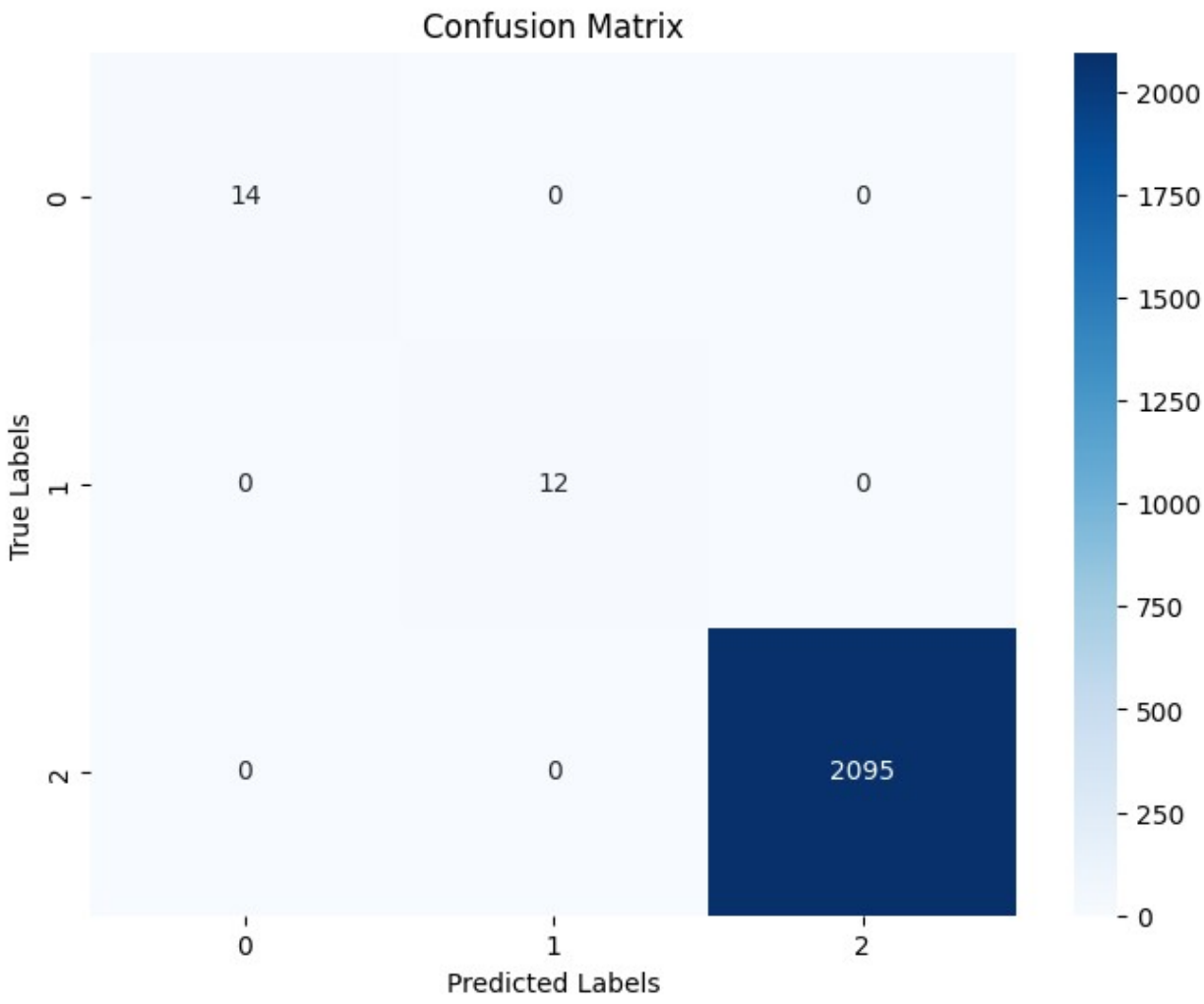
```
{"type":"string"}
```

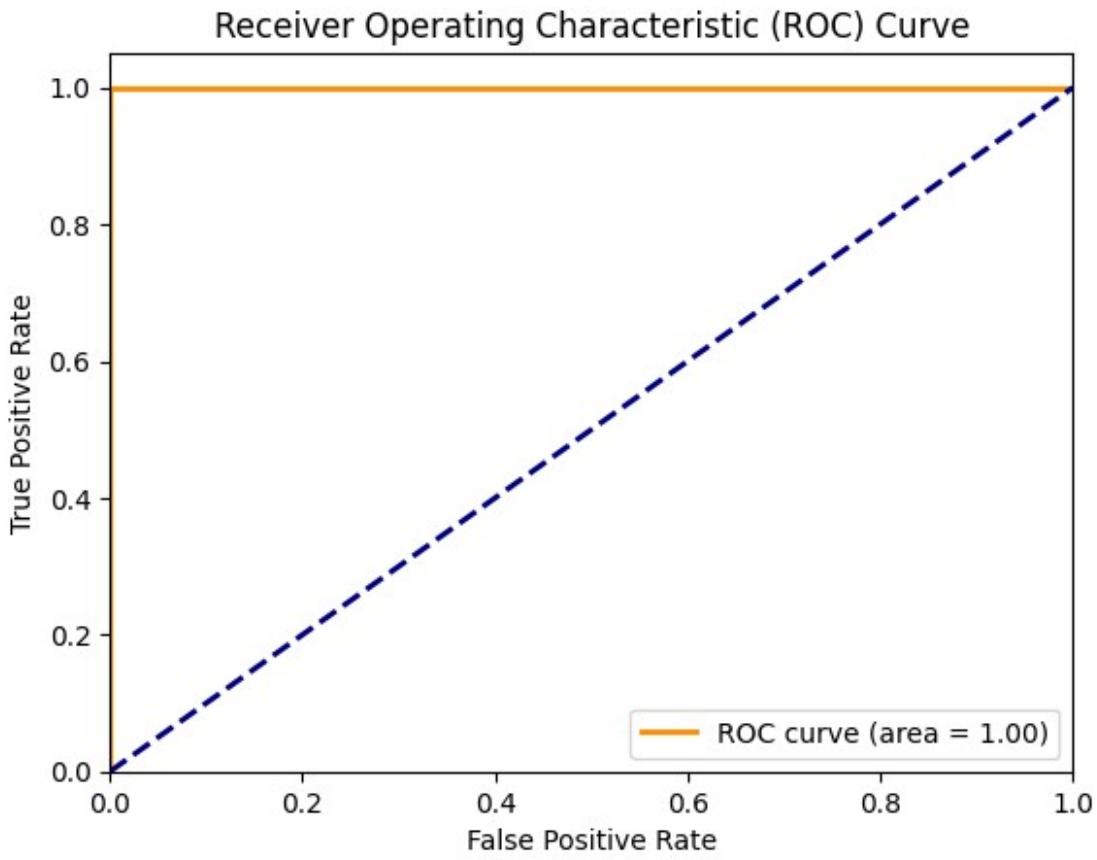
```
1.0
```

```
{"type":"string"}
```

```
{"type":"string"}
```

```
array([[  14,    0,    0],
       [   0,   12,    0],
       [   0,    0, 2095]])
```


Confusion Matrix
True Labels
Predicted Labels
0
1
2
14
0
0
0
12
0
0
0
2095
2000
1750
1500
1250
1000
750
500
250
0



8. Solution

What is the solution that is proposed to solve the business problem discussed in Section 1. Also share the learnings while working through solving the problem in terms of challenges, observations, decisions made etc.

Based on the analysis, the proposed solution is to use a combination of feature selection and machine learning techniques to predict stock prices.

1. **Data Preparation**

   Data Cleaning and Validation: Meticulously address inconsistencies, errors, and missing values within the dataset to ensure data integrity.

   Feature Engineering: Augment the dataset with derived features such as moving averages, trend indicators, and risk metrics to enhance predictive capabilities.

   Target Variable Transformation: Discretize continuous stock price movements into categorical labels to facilitate subsequent classification tasks.

2. **Exploratory Data Analysis**

   Data Visualization: Employ graphical representations to discern the distribution of data, interrelationships between variables, and potential patterns.

Pattern Recognition: Identify trends, correlations, and other noteworthy associations within the dataset.

Feature Significance Assessment: Determine the relative importance of various data attributes in predicting stock prices.

3. **Feature Selection**

   Select the most informative features to optimize predictive models and minimize the risk of overfitting.

4. **Machine Learning Models**

   Model Selection: Choose appropriate algorithms aligned with specific tasks, such as clustering (e.g., K-means), classification (e.g., Decision Trees, Logistic Regression), or regression (for continuous price predictions).

   Model Training and Evaluation: Utilize historical stock data to train selected models and rigorously assess their predictive performance through established metrics (e.g., accuracy, precision, recall, F1-score).

5. **Portfolio Optimization**

   Leverage data analytics to ascertain the optimal allocation of assets within a portfolio, striking a balance between risk and return.

   Diversification Strategy: Mitigate risk by incorporating a diverse range of stocks from various sectors, judiciously balancing growth-oriented and dividend-yielding securities.

6. **Risk Management**

   Value At Risk (VAR) Analysis: Quantify potential financial losses associated with selected investments.

   Scenario Analysis & Stress Testing: Evaluate portfolio performance under diverse market conditions, including adverse scenarios, to assess resilience.

   Portfolio Rebalancing Suggestions

Key Findings

Data Quality Imperative: Accurate and reliable data is fundamental to robust analysis and prediction.

Feature Engineering Significance: The creation of meaningful derived features substantially enhances predictive capabilities.

Diversification as Risk Mitigation: Allocation across diverse sectors is crucial for effective risk management.

Conclusion:

This project showcases the application of data science and financial principles to analyze stock market dynamics, predict price movements, and construct well-diversified portfolios. The integration of these methodologies empowers investors with enhanced decision-making capabilities and robust risk management strategies.

Lessons Learned:

Model Limitations: Predictive models, while valuable, are inherently imperfect and cannot fully encapsulate the complexities of the market.

Past Performance Non-Predictive: Historical performance does not guarantee future outcomes. Models necessitate continuous updates and adaptation.

Risk Management Primacy: Prudent investment practices, including diversification and ongoing risk assessment, remain paramount. has context menu