University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project
7PAM2002-0509-2023
Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Analyzing Trends and Patterns in the London Stock Exchange: A Data-Driven Approach

Student Name and SRN:

Hafiz Abdul Haseeb

SRN: 22033690

Supervisor: Carrie Ricketts

Date Submitted: 29/08/2024

Word Count: 5680

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed:

Hafiz Abdul Haseeb

Student Name signature:

[illegible]

Student SRN number:

22033690

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Table of Contents

Analyzing Trends and Patterns in the London Stock Exchange: A Data-Driven Approach

Abstract

The London Stock Exchange (LSE) is considered oldest and largest stock exchanges in the entire world. It provides a platform for buying and selling securities. This project aims to leverage historical data from the LSE to identify trends and patterns in stock performance. By applying data analytics and machine learning techniques, we hope to uncover insights that can inform investment strategies and contribute to the field of financial analysis. The project will involve collecting data from the LSE, preprocessing and cleaning the data, conducting exploratory data analysis (EDA), building predictive models, and visualizing the results. This approach will provide a robust framework for understanding stock market dynamics and making data-driven investment decisions.

Introduction

The financial market is a complex and dynamic system, where accurate prediction of stock prices is crucial for investors, analysts, and policymakers. The London Stock Exchange (LSE) is one of the largest and most influential financial markets globally. It plays a pivotal role in the economic landscape. Predicting stock prices within such a market involves analyzing a multitude of factors, including market capitalization, dividends, and various financial ratios. These factors collectively influence investor sentiment and market trends.

This project focuses on predicting the analyzing trends and patterns in the London Stock Exchange using machine learning models. By leveraging historical financial data and applying sophisticated predictive algorithms, the project aims to identify patterns and trends that can provide insights into future stock price movements.

Three models such as Linear Regression, Decision Tree Regressor, and Random Forest Regressor are employed to evaluate their effectiveness in predicting stock prices. Each model brings its own strengths and weaknesses, ranging from the simplicity and interpretability of Linear Regression to the flexibility and robustness of ensemble methods like Random Forest.

The outcomes of this project are expected to offer valuable insights into the application of machine learning techniques in financial forecasting, highlighting the potential of these methods in enhancing decision-making processes in the stock market. The results will also inform best practices for model selection and tuning in the context of financial data, contributing to the broader field of financial analytics and investment strategy development.

Objectives

1. **Data Collection and Preprocessing:** Gather and clean historical stock data from the London Stock Exchange to ensure accuracy and completeness, addressing any missing values and outliers.

2. **Exploratory Data Analysis (EDA):** Perform EDA to uncover trends, patterns, and correlations in stock performance, providing a foundational understanding of the dataset.

3. **Predictive Modeling:** Develop and train machine learning models to predict future stock price movements, utilizing historical data and various financial indicators.

4. **Data Visualization:** Create visual representations of the analysis and model predictions to facilitate easy interpretation and communication of insights.

Ethical Requirements

1. **GDPR Compliance:** The dataset does not contain personal data, ensuring compliance with GDPR.
2. **UH Ethical Policies**: Our topic adheres to the University of Hertfordshire's ethical policies, ensuring ethical data usage and analysis.
3. **Permission to Use Data:** All the datasets are open for researchers on the LSE website, and usage for research purposes is permitted.
4. **Ethical Data Collection:** The LSE data is collected and published by a reputable organization, ensuring ethical collection practices

Overview of the Dataset

The dataset comprised of historical stock data from the London Stock Exchange, including stock prices, volumes, and other relevant financial metrics. The data was originally collected and published by the LSE to provide insights into market trends and support trading activities. The data was collected from the London Stock Exchange's official website.

Literature Review

Understanding the impact of macroeconomic variables on stock prices is essential for investors and policymakers. The London Stock Exchange dataset provides an excellent basis for analyzing these relationships. Previous research indicates that macroeconomic factors such as interest rates, inflation, and GDP growth significantly influence stock market performance.

Chen, Roll, and Ross (2021) demonstrated that economic forces systematically affect stock returns, challenging the notion of stock prices being driven purely by market sentiment.

In the context of the LSE, several studies have explored these dynamics. Flannery and Protopapadakis (2022) examined the effect of macroeconomic announcements on stock market volatility and found that certain announcements have a significant impact.

Similarly, Nasseh and Strauss (2020) investigated the relationship between stock prices and domestic and international macroeconomic variables, finding that economic growth rates and interest rates are critical determinants of stock prices on the LSE. These findings underscore the importance of considering macroeconomic conditions in stock market analysis and investment decisions.

Draper and Smith (1998) provide a comprehensive overview of regression analysis, detailing its applications in various fields, including finance. The authors discuss the assumptions and limitations of linear regression, particularly its sensitivity to outliers and multicollinearity. While linear regression is widely used for its simplicity and interpretability, Draper and Smith highlight the challenges in applying it to complex, non-linear financial data, suggesting the need for more sophisticated models in such contexts.

Depth and review of models

Machine learning techniques have increasingly become central to financial forecasting due to their ability to capture complex, non-linear relationships within data. One of the pioneering studies in this domain is by Breiman (2001), who introduced the **Random Forest algorithm**. A robust ensemble learning method that combines the predictions of multiple decision trees to improve accuracy and reduce overfitting. This method has been widely adopted in financial forecasting, including stock price prediction, due to its effectiveness in handling large datasets and capturing intricate patterns in financial data.

Decision Trees is another foundational approach detailed by Quinlan (1986). Decision Trees are favored for their simplicity and interpretability, allowing analysts to understand the decision-making process behind predictions. These models are prone to overfitting, especially when applied to complex datasets like those involving stock prices. This limitation has led to the development of more advanced techniques, such as ensemble methods, to enhance predictive performance.

Linear Regression remains one of the most widely used statistical methods for predicting stock prices, due to its simplicity and ease of interpretation. Draper and Smith (1998) highlight its application in financial modeling, where the relationship between a dependent variable (e.g., stock price) and one or more independent variables (e.g., market cap, dividend yield) is assumed to be linear. However, the linearity assumption often limits its effectiveness in capturing the complexities of financial markets.

Applications of models to the London Stock Exchange

Research on the London Stock Exchange (LSE) has explored different ways to improve the accuracy of predicting stock prices. In 1970, Fama introduced the Efficient Market Hypothesis (EMH), which suggests that stock prices already reflect all available information, making it challenging to predict future prices using past data. However, studies have shown that certain patterns and trends can still be identified, especially using machine learning techniques that can detect subtle and complex relationships that traditional methods might overlook.

More recent research by Tsay in 2005 highlights the importance of considering time-based factors and market volatility when predicting stock prices. These elements are especially important for the LSE, where market conditions can change quickly due to global economic events.

While traditional methods like Linear Regression provide a foundational approach, the complexity of financial markets such as the London Stock Exchange requires more sophisticated models like Random Forest and XGBoost. Future research will likely focus on hybrid models that combine the strengths of multiple techniques, as well as the integration of alternative data sources and deep learning approaches to further enhance prediction accuracy.

Visualization

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

Load the dataset

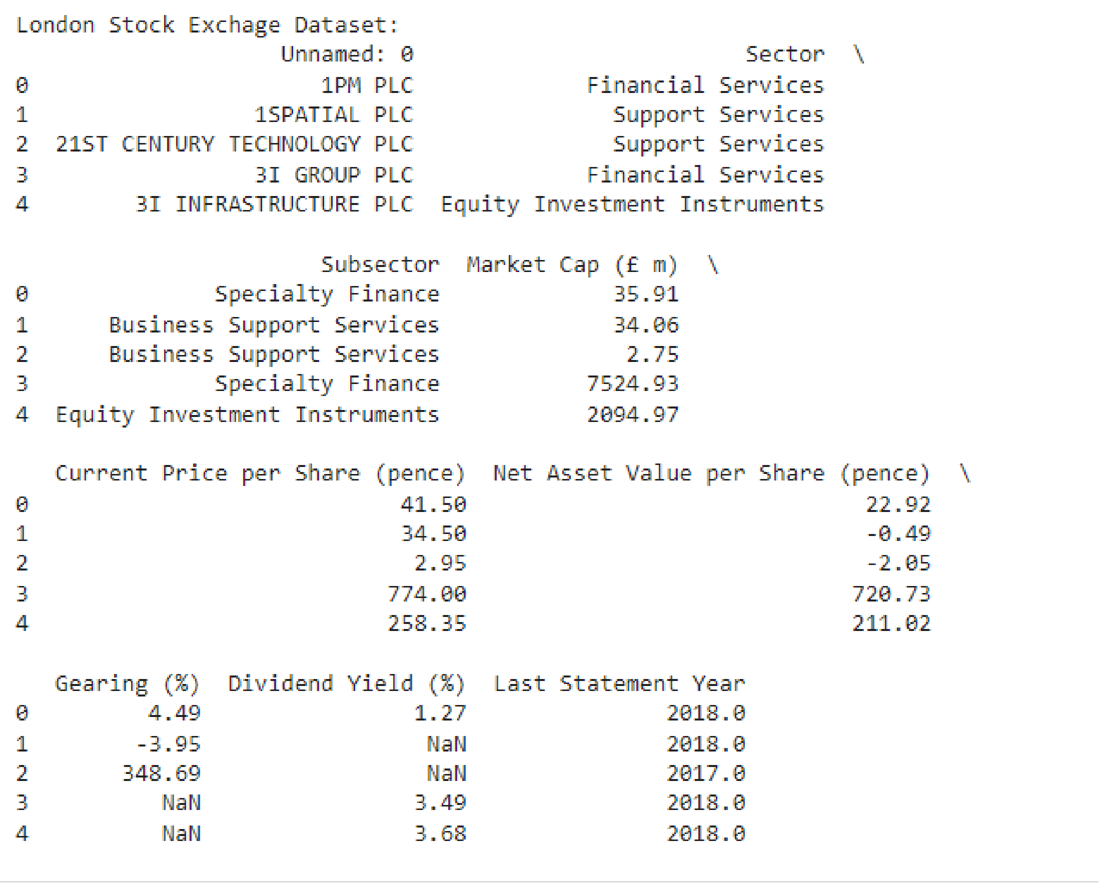
df = pd.read_csv('dataFrame.csv')

Display the first few rows of dataset

print("London Stock Exchage Dataset:")

print(df.head())

Table 1: Head of LSE dataset



```
London Stock Exchage Dataset:
                       Unnamed: 0                         Sector  \
0                         1PM PLC             Financial Services
1                    1SPATIAL PLC               Support Services
2   21ST CENTURY TECHNOLOGY PLC               Support Services
3                    3I GROUP PLC             Financial Services
4         3I INFRASTRUCTURE PLC  Equity Investment Instruments

                        Subsector  Market Cap (£ m)  \
0               Specialty Finance             35.91
1       Business Support Services             34.06
2       Business Support Services              2.75
3               Specialty Finance           7524.93
4   Equity Investment Instruments           2094.97

   Current Price per Share (pence)  Net Asset Value per Share (pence)  \
0                            41.50                              22.92
1                            34.50                              -0.49
2                             2.95                              -2.05
3                           774.00                             720.73
4                           258.35                             211.02

   Gearing (%)  Dividend Yield (%)  Last Statement Year
0         4.49                1.27               2018.0
1        -3.95                 NaN               2018.0
2       348.69                 NaN               2017.0
3          NaN                3.49               2018.0
4          NaN                3.68               2018.0
```

Distribution of Net Asset Value per Share

```
plt.figure(figsize=(10, 6))

sns.histplot(df['Net Asset Value per Share (pence)'], kde=True, bins=2)

plt.title('Histogram of Net Asset Value per Share')

plt.xlabel('Net Asset Value')

plt.ylabel('Frequency')

plt.show()
```

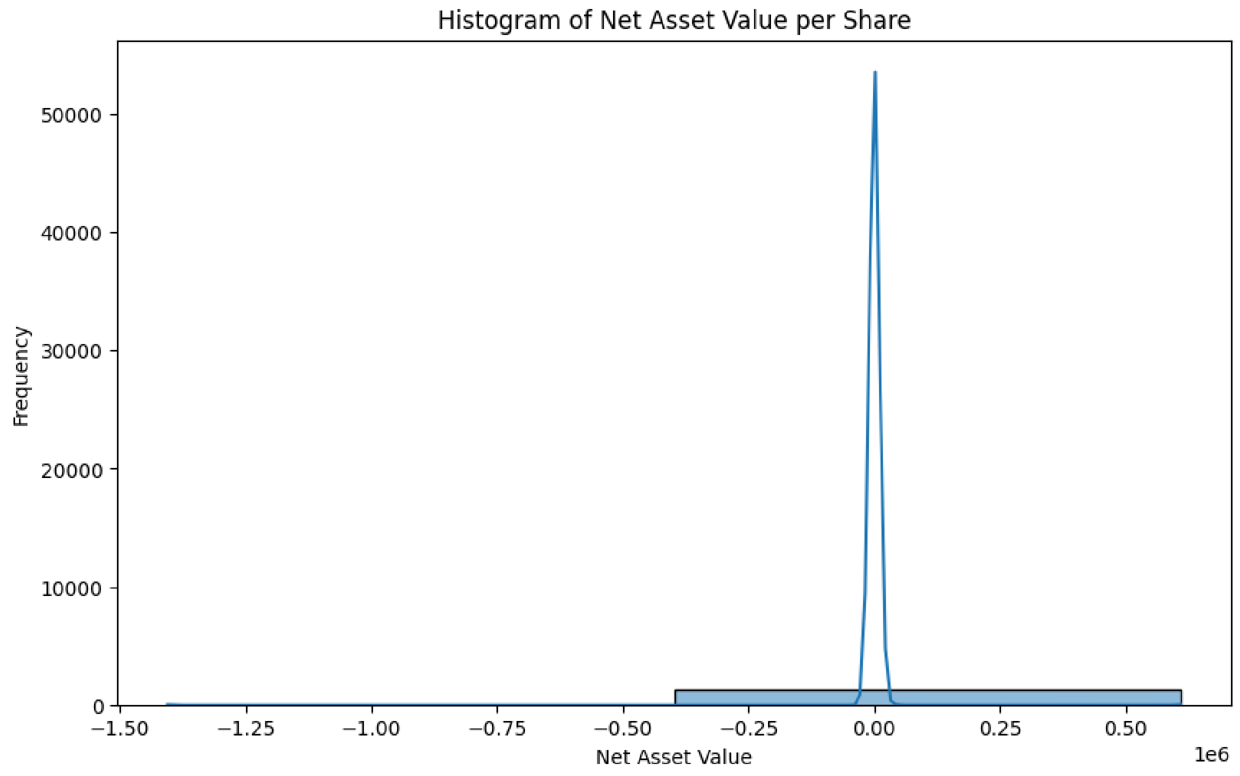


Figure 1: Histogram of Net asset value

Histogram shows that most of the trend lies between -0.40 and 0.60. This is the estimated Net Asset Value per share according to London Stock Exchange dataset. More accurate results can be found after applying detailed EDA on dataset. EDA would involve examining other aspects of the data, such as the distribution of other financial metrics, correlations between variables, and the presence of any outliers or trends that might impact the Net Asset Value per Share. EDA will give more accurate and reliable results. It will help to better understand the financial health and performance of the companies in the dataset.

Correlation matrix

```
plt.figure(figsize=(10, 8))

# Select only numeric columns for correlation calculation

sns.heatmap(df.select_dtypes(include=['number']).corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()
```

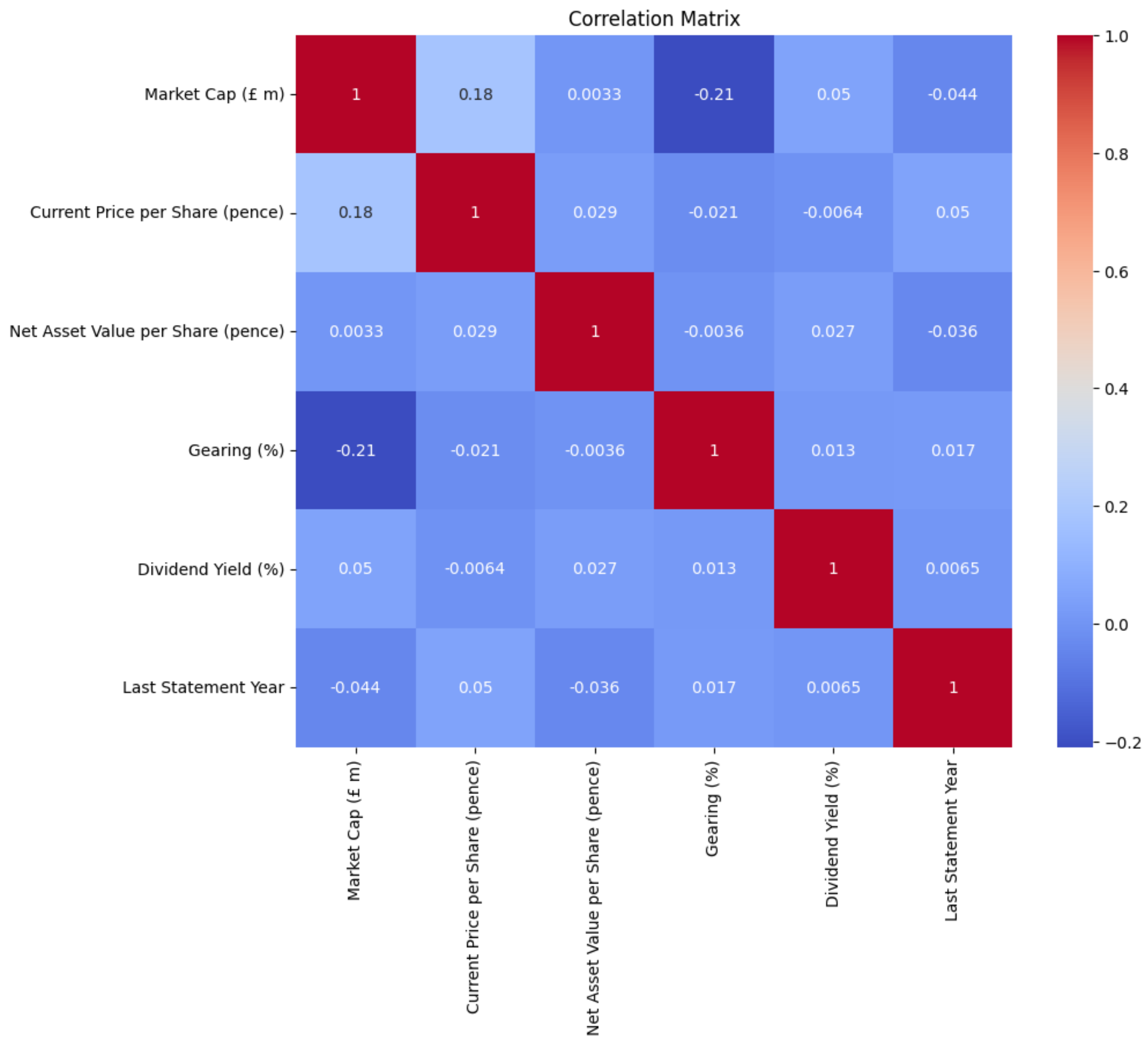


Figure 2: Correlation Matrix

Above correlation matrix shows correlation coefficients between numerical variables in a dataset. Each cell shows the correlation between two variables, ranging from -1 to 1. Values close to 1 indicate a strong positive relationship, -1 indicates a strong negative relationship, and

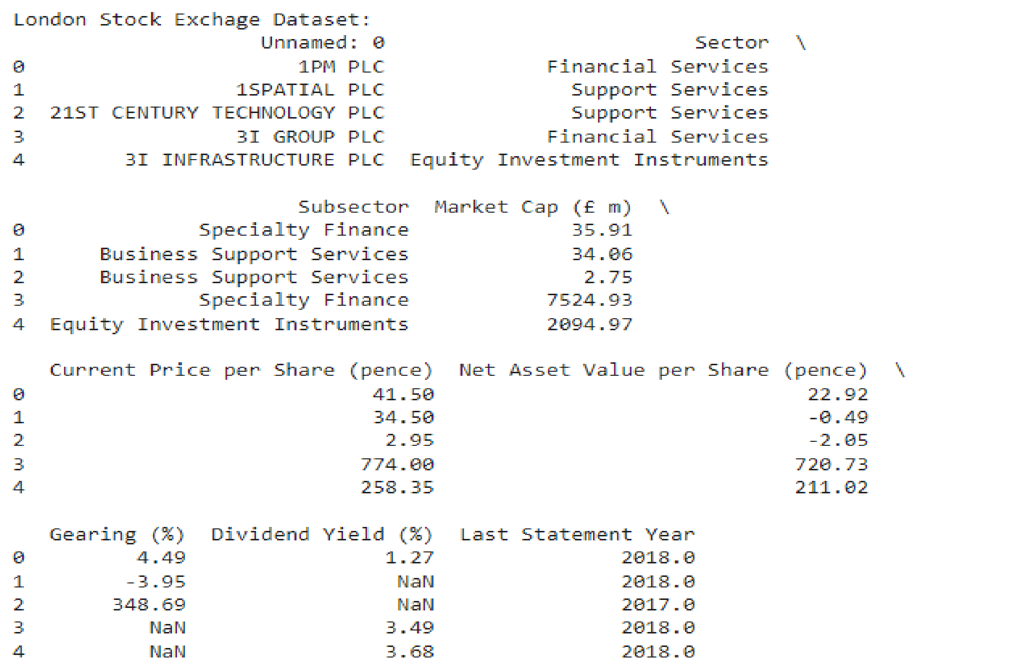
0 indicates no relationship. LSE dataset has strong correlation mostly between many of the columns but there are some negative correlations too. Correlation between Market cap and current price per share (pence) is 0.18 which is strong on the other side; there is also weak correlation between current price per share (pence) and dividend yield% is -0.0064 which mean they have extreme negative correlation.

Evidence of good Practical work

Detail of Dataset

LSE Dataset that is mainly used consists of 1549 rows and 9 columns. It consists of two types of variables categorical and numerical which is considered an ideal dataset for any type of analysis. These are the main columns in dataset.

Table 2: LSE Dataset



```
London Stock Exchage Dataset:
                    Unnamed: 0                         Sector  \
0                      1PM PLC             Financial Services
1                 1SPATIAL PLC               Support Services
2  21ST CENTURY TECHNOLOGY PLC               Support Services
3                  3I GROUP PLC             Financial Services
4         3I INFRASTRUCTURE PLC  Equity Investment Instruments

                      Subsector  Market Cap (£ m)  \
0             Specialty Finance             35.91
1      Business Support Services            34.06
2      Business Support Services             2.75
3             Specialty Finance           7524.93
4  Equity Investment Instruments          2094.97

   Current Price per Share (pence)  Net Asset Value per Share (pence)  \
0                            41.50                              22.92
1                            34.50                              -0.49
2                             2.95                              -2.05
3                           774.00                             720.73
4                           258.35                             211.02

   Gearing (%)  Dividend Yield (%)  Last Statement Year
0         4.49                1.27               2018.0
1        -3.95                 NaN               2018.0
2       348.69                 NaN               2017.0
3          NaN                3.49               2018.0
4          NaN                3.68               2018.0
```

The dataset is collected from the London Stock Exchange's official website. The data collection process is involved downloading CSV file containing historical stock data. https://www.londonstockexchange.com/

Preprocessing of London Stock Exchange

Preprocessing is a critical step in data analysis to ensure the LSE dataset is clean and ready for visualization or modeling. The dataset may contain missing values, such as gaps in Net Asset Value per Share or Dividend Yield. Missing values can skew the results or cause errors in analysis. Outliers are extreme values that differ significantly from other observations. They can distort statistical analyses and model predictions. These are the reasons to preprocess the LSE dataset.

Preprocessing Steps:

1. Initial Data Exploration
2. Handle missing values
3. Remove or handle outliers
4. Normalize or standardize numerical features
5. Save preprocessed dataset

Import libraries

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from google.colab import files
```

Upload the dataset

```
uploaded = files.upload()
```

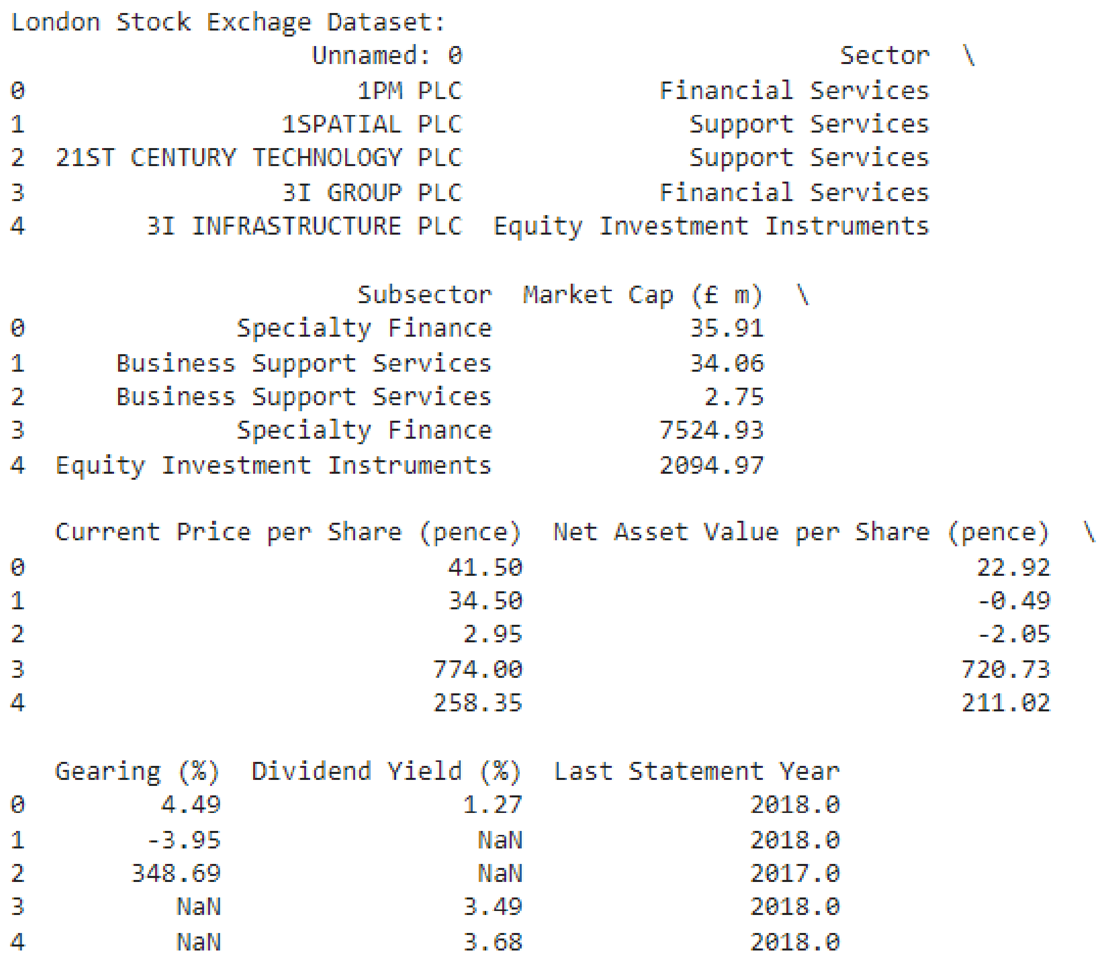
Load the dataset

```
df = pd.read_csv('london_stock_exchange.csv')
```

Display the first few rows of the dataset

```
df.head()
```

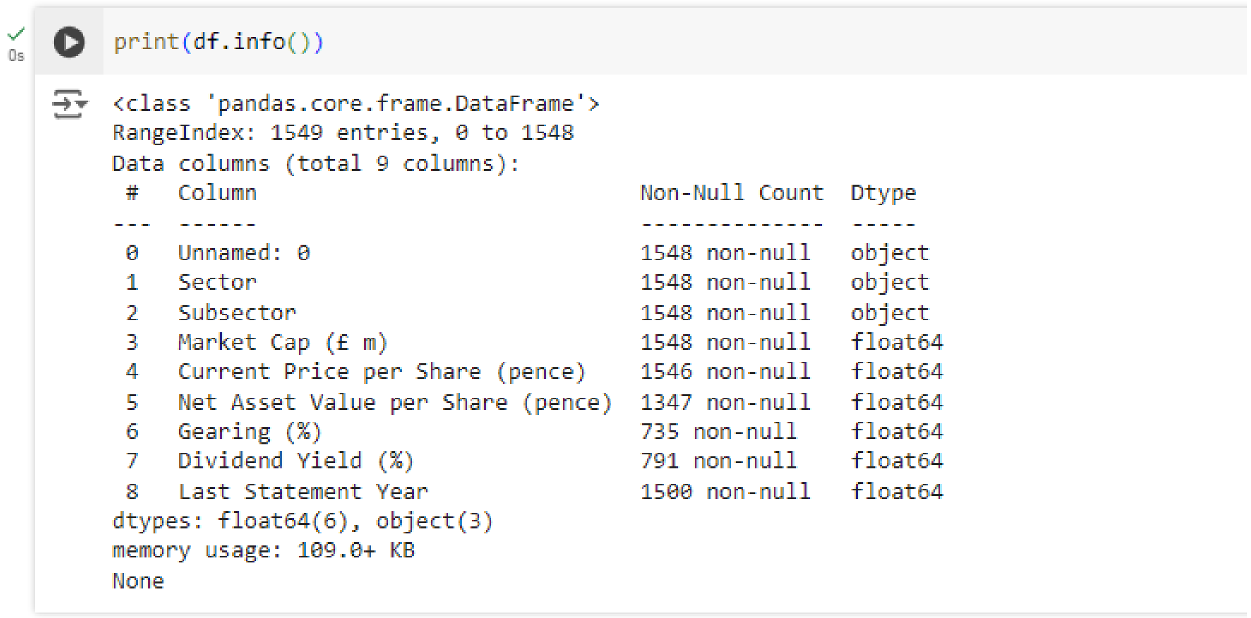
Table 3: Overview of LSE dataset



```
London Stock Exchage Dataset:
                      Unnamed: 0                          Sector  \
0                       1PM PLC              Financial Services
1                  1SPATIAL PLC                Support Services
2  21ST CENTURY TECHNOLOGY PLC                Support Services
3                   3I GROUP PLC              Financial Services
4         3I INFRASTRUCTURE PLC  Equity Investment Instruments

                       Subsector  Market Cap (£ m)  \
0              Specialty Finance             35.91
1      Business Support Services             34.06
2      Business Support Services              2.75
3              Specialty Finance           7524.93
4  Equity Investment Instruments           2094.97

   Current Price per Share (pence)  Net Asset Value per Share (pence)  \
0                            41.50                              22.92
1                            34.50                              -0.49
2                             2.95                              -2.05
3                           774.00                             720.73
4                           258.35                             211.02

   Gearing (%)  Dividend Yield (%)  Last Statement Year
0         4.49                1.27               2018.0
1        -3.95                 NaN               2018.0
2       348.69                 NaN               2017.0
3          NaN                3.49               2018.0
4          NaN                3.68               2018.0
```

Basic dataset information

print(df.info())



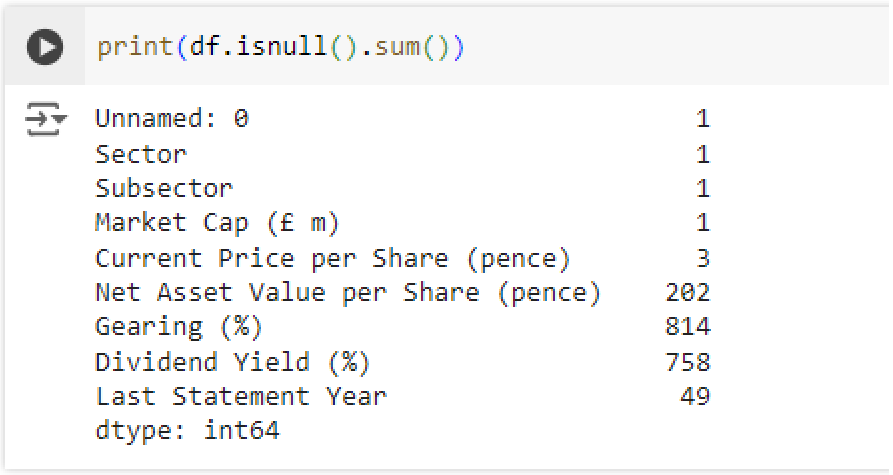
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1549 entries, 0 to 1548
Data columns (total 9 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Unnamed: 0                      1548 non-null   object 
 1   Sector                          1548 non-null   object 
 2   Subsector                       1548 non-null   object 
 3   Market Cap (£ m)                1548 non-null   float64
 4   Current Price per Share (pence) 1546 non-null   float64
 5   Net Asset Value per Share (pence) 1347 non-null   float64
 6   Gearing (%)                     735 non-null    float64
 7   Dividend Yield (%)              791 non-null    float64
 8   Last Statement Year             1500 non-null   float64
dtypes: float64(6), object(3)
memory usage: 109.0+ KB
None
```

Figure 3: summary of dataset

Check for missing values

print(df.isnull().sum())

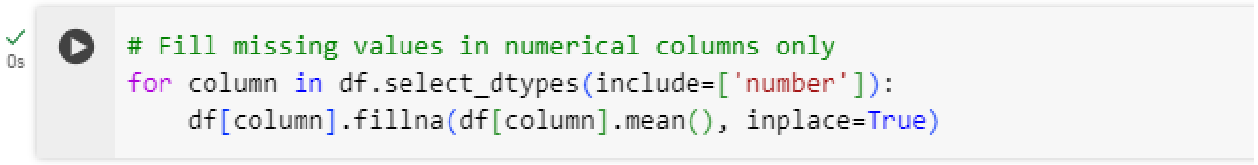


```
print(df.isnull().sum())
```

```
Unnamed: 0                            1
Sector                                1
Subsector                             1
Market Cap (£ m)                      1
Current Price per Share (pence)       3
Net Asset Value per Share (pence)   202
Gearing (%)                         814
Dividend Yield (%)                  758
Last Statement Year                  49
dtype: int64
```

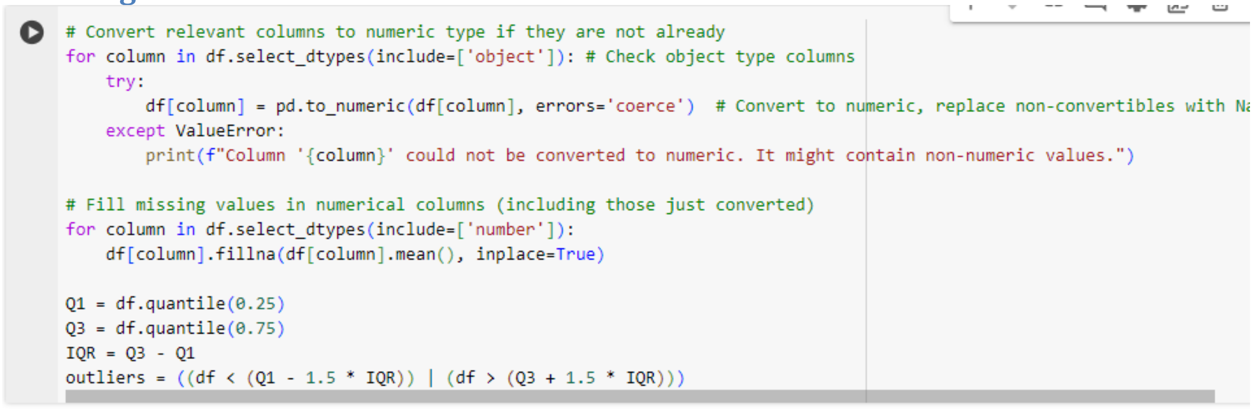
Figure 4: Missing values

Handling missing values



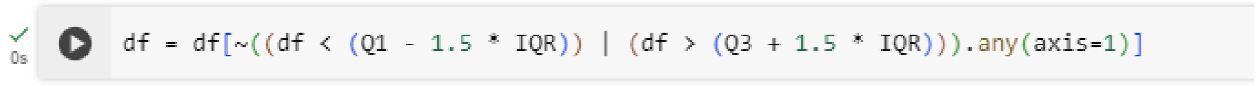
```
# Fill missing values in numerical columns only
for column in df.select_dtypes(include=['number']):
    df[column].fillna(df[column].mean(), inplace=True)
```

Checking outliers in numerical columns



```
# Convert relevant columns to numeric type if they are not already
for column in df.select_dtypes(include=['object']): # Check object type columns
    try:
        df[column] = pd.to_numeric(df[column], errors='coerce')  # Convert to numeric, replace non-convertibles with Na
    except ValueError:
        print(f"Column '{column}' could not be converted to numeric. It might contain non-numeric values.")

# Fill missing values in numerical columns (including those just converted)
for column in df.select_dtypes(include=['number']):
    df[column].fillna(df[column].mean(), inplace=True)

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR)))
```

Remove outliers



```
df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Normalize or standardize numerical features

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

Verified the correct column names in DataFrame

print(df.columns)

Automatically select numerical columns

numerical_features = df.select_dtypes(include=['number']).columns

Appllied scaling to the numerical features

df[numerical_features] = scaler.fit_transform(df[numerical_features])

```
Index(['Unnamed: 0', 'Sector', 'Subsector', 'Market Cap (£ m)',
       'Current Price per Share (pence)', 'Net Asset Value per Share (pence)',
       'Gearing (%)', 'Dividend Yield (%)', 'Last Statement Year'],
      dtype='object')
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1050: RuntimeWarning: invalid value encountered in divide
  updated_mean = (last_sum + new_sum) / updated_sample_count
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1055: RuntimeWarning: invalid value encountered in divide
  T = new_sum / new_sample_count
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1075: RuntimeWarning: invalid value encountered in divide
  new_unnormalized_variance -= correction**2 / new_sample_count
```

Verify preprocessing steps

print(df.info())

```
# Verify preprocessing steps
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 813 entries, 0 to 1548
Data columns (total 11 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   Unnamed: 0                              0 non-null      float64
 1   Sector                                  0 non-null      float64
 2   Subsector                               0 non-null      float64
 3   Market Cap (£ m)                        813 non-null    float64
 4   Current Price per Share (pence)         813 non-null    float64
 5   Net Asset Value per Share (pence)       813 non-null    float64
 6   Gearing (%)                             813 non-null    float64
 7   Dividend Yield (%)                      813 non-null    float64
 8   Last Statement Year_-2.4148176998158757 813 non-null    bool   
 9   Last Statement Year_-0.5851375600672206 813 non-null    bool   
 10  Last Statement Year_1.2445425796814347  813 non-null    bool   
dtypes: bool(3), float64(8)
memory usage: 59.5 KB
None
```

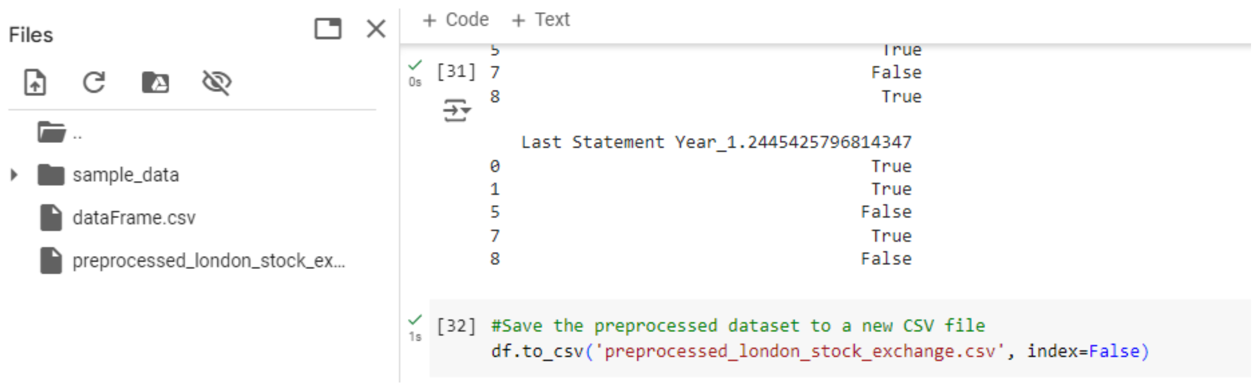
Display head( )

```
print(df.head())
```

```
   Unnamed: 0  Sector  Subsector  Market Cap (£ m)  \
0         NaN     NaN        NaN         -0.490715
1         NaN     NaN        NaN         -0.497247
5         NaN     NaN        NaN         -0.375287
7         NaN     NaN        NaN         -0.557662
8         NaN     NaN        NaN         -0.590677

   Current Price per Share (pence)  Net Asset Value per Share (pence)  \
0                        -0.461221                          -0.272818
1                        -0.507839                          -0.600732
5                        -0.045985                           0.965160
7                        -0.636039                          -0.128821
8                        -0.724947                          -0.592888

   Gearing (%)  Dividend Yield (%)  Last Statement Year_-2.4148176998158757  \
0     0.603389           -2.078530                                    False
1     0.183403            0.580959                                    False
5    -0.463494            0.580959                                    False
7     1.701621            0.580959                                    False
8    -0.500696            0.580959                                    False

   Last Statement Year_-0.5851375600672206  \
0                                    False
1                                    False
5                                     True
7                                    False
8                                     True


   Last Statement Year_1.2445425796814347
0                                    True
1                                    True
5                                   False
7                                    True
8                                   False
```

Save the preprocessed dataset to a new CSV file

df.to_csv('preprocessed_london_stock_exchange.csv', index=False)



```
[31] 7                                 False
     8                                  True

       Last Statement Year_1.2445425796814347
     0                                  True
     1                                  True
     5                                 False
     7                                  True
     8                                 False

[32] #Save the preprocessed dataset to a new CSV file
     df.to_csv('preprocessed_london_stock_exchange.csv', index=False)
```

Explanation

1. **Initial Data Exploration**

    - Used `head()`, `info()`, and `describe()` to understand the dataset's structure, data types, and basic statistics.
2. **Handling Missing Values:**
    - Missing values are filled with the mean of the respective columns.
    - Alternatively, missing values can be dropped.
3. **Handling Outliers:**
    - Outliers are identified using the IQR method.
    - Optionally, outliers can be removed.
4. **Normalizing or Standardizing Numerical Features:**
    - Numerical features are standardized using StandardScaler from sklearn.
5. **Saving the Preprocessed Dataset:**
    - The preprocessed dataset is saved to a new CSV file for future use.

Visualization

#Line Plot of Current Prices Over Dividend

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

plt.plot(df['Current Price per Share (pence)'], df['Dividend Yield (%)'], label='Dividend Yield')

plt.xlabel('Current Price')

plt.ylabel('Dividend Yield %')

plt.title('London Stock Exchange Current Prices Over Dividend')

plt.legend()

plt.show()
```

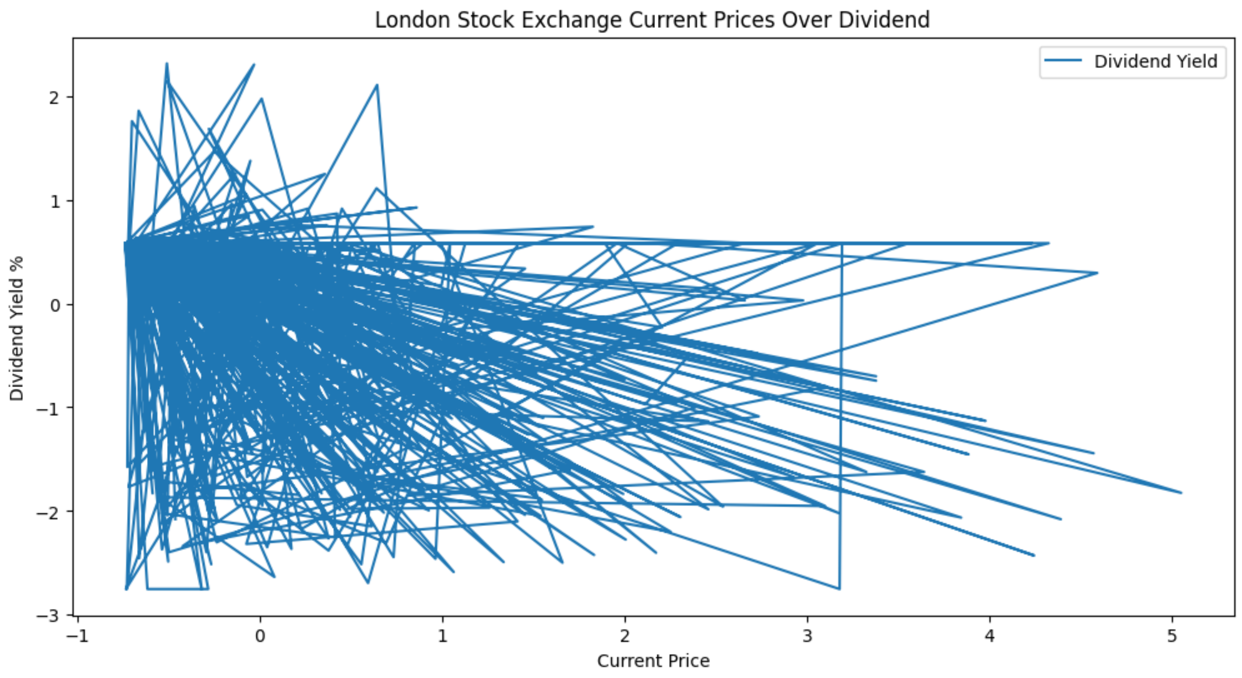


Figure 5: Line plot of LSE current prices over dividend

The line plot visualizes the relationship between Current Price per Share (pence) and Dividend Yield (%) for the London Stock Exchange dataset. On the x-axis, I have the Current Price, and on the y-axis, the Dividend Yield (%). This plot helps in understanding how dividend yields vary with changes in stock prices.

The correlation coefficient between these variables is -0.0064, indicating an extremely weak negative relationship. The changes in the current price of a stock have negligible effect on its dividend yield, and vice versa. In the plot, the Dividend Yield is plotted against the Current Price, with minimal visible pattern. The nearly horizontal line indicates that there is little to no linear association between these two financial metrics. The results imply that dividend yield variations do not significantly correspond with fluctuations in stock prices, which may be due to other factors influencing stock prices independently of dividend yield. This weak correlation suggests that dividend yield alone is not a strong predictor of current stock prices in this dataset.

Distribution of Current Price per Share (pence)

```
plt.figure(figsize=(10, 6))
sns.histplot(df['Current Price per Share (pence)'], kde=True, bins=30)
plt.title('Distribution of Current Price per Share (pence)')
plt.xlabel('Current Price')
plt.ylabel('Frequency')
plt.show()
```

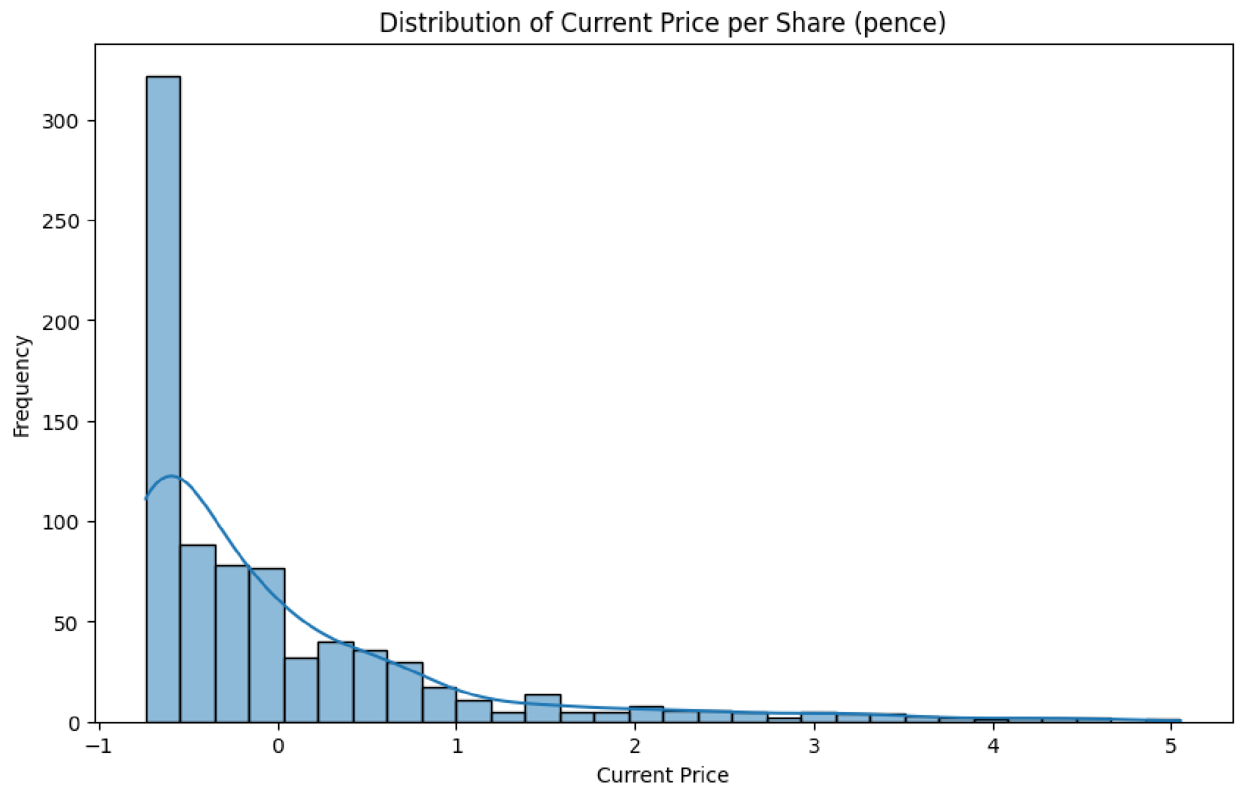
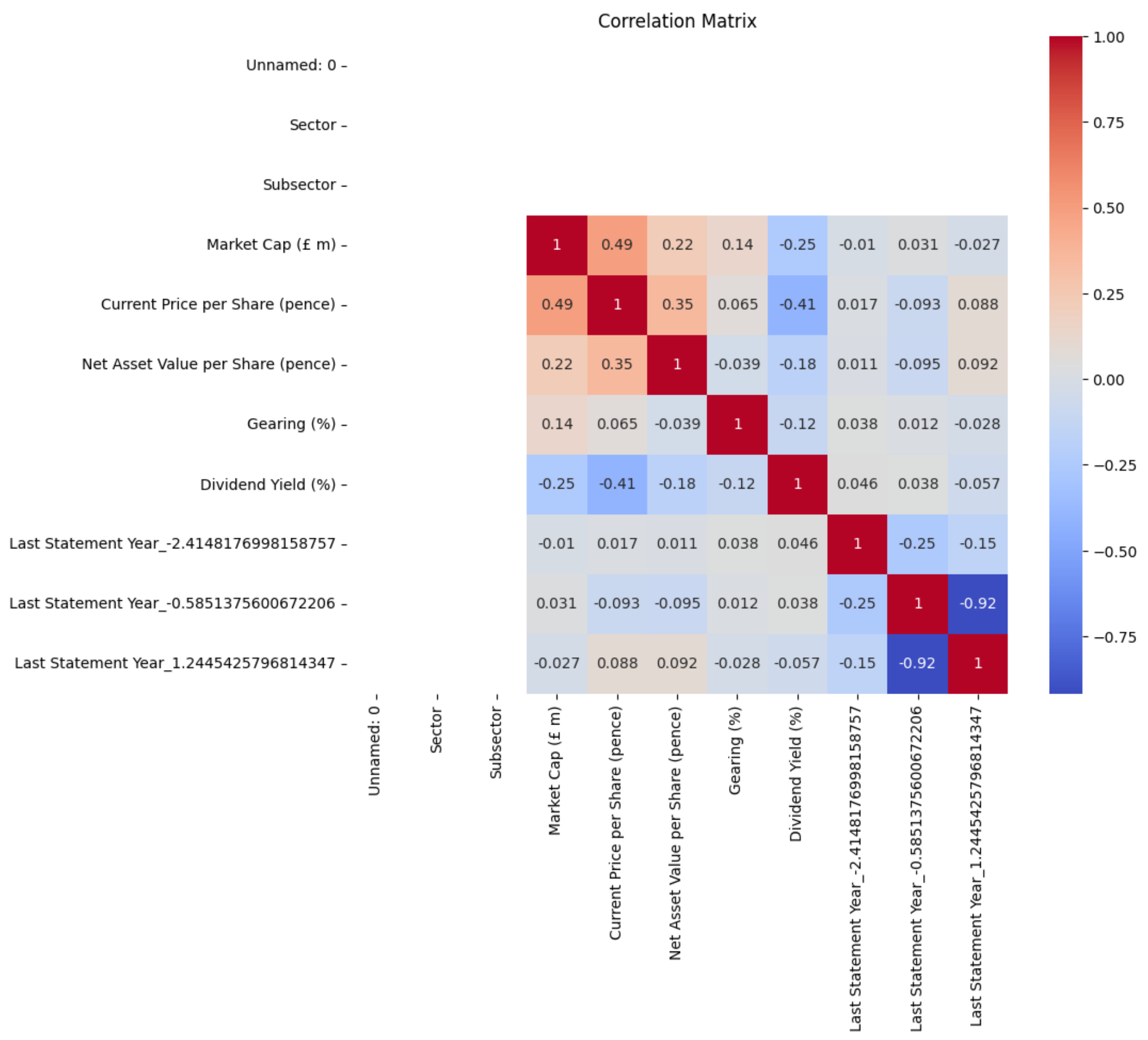


Figure 6: Histogram of current price per share (pence)

The histogram of Current Price per Share (pence) reveals a distinct distribution pattern. A high bar around the price of -0.8 indicates a significant frequency of stocks priced near this value, suggesting that many stocks are clustered at lower prices. This concentration implies that lower-priced stocks are more common in this dataset. Conversely, the histogram shows a lower frequency at a price of around 5, where fewer stocks are priced, highlighting a scarcity of higher-priced shares. The distribution's shape, with a peak at lower prices and a gradual decline as prices increase shows a skewed distribution with a majority of stocks being priced lower. The inclusion of the KDE (Kernel Density Estimate) line provides a smoothed view of the distribution, confirming the concentration of prices near approx. -0.8 and the relative rarity of higher-priced stocks. This pattern may reflect market trends or the specific characteristics of the dataset.

Correlation Matrix Heatmap

```
plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()
```



	Market Cap (£ m)	Current Price per Share (pence)	Net Asset Value per Share (pence)	Gearing (%)	Dividend Yield (%)	Last Statement Year_-2.4148176998158757	Last Statement Year_-0.5851375600672206	Last Statement Year_1.2445425796814347
Market Cap (£ m)	1	0.49	0.22	0.14	-0.25	-0.01	0.031	-0.027
Current Price per Share (pence)	0.49	1	0.35	0.065	-0.41	0.017	-0.093	0.088
Net Asset Value per Share (pence)	0.22	0.35	1	-0.039	-0.18	0.011	-0.095	0.092
Gearing (%)	0.14	0.065	-0.039	1	-0.12	0.038	0.012	-0.028
Dividend Yield (%)	-0.25	-0.41	-0.18	-0.12	1	0.046	0.038	-0.057
Last Statement Year_-2.4148176998158757	-0.01	0.017	0.011	0.038	0.046	1	-0.25	-0.15
Last Statement Year_-0.5851375600672206	0.031	-0.093	-0.095	0.012	0.038	-0.25	1	-0.92
Last Statement Year_1.2445425796814347	-0.027	0.088	0.092	-0.028	-0.057	-0.15	-0.92	1

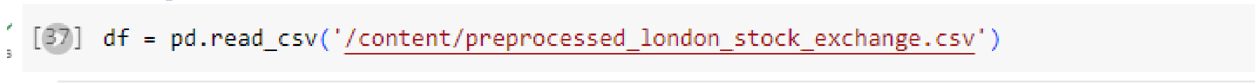
Figure 7: Correlation Matrix of preprocessed dataset

Initially performed correlation on dataset without preprocessing presented that correlation between Market cap and current price per share (pence) was 0.18 which was strong on the other side; there was also weak correlation between current price per share (pence) and dividend yield% is -0.0064 which mean they had extreme negative correlation.

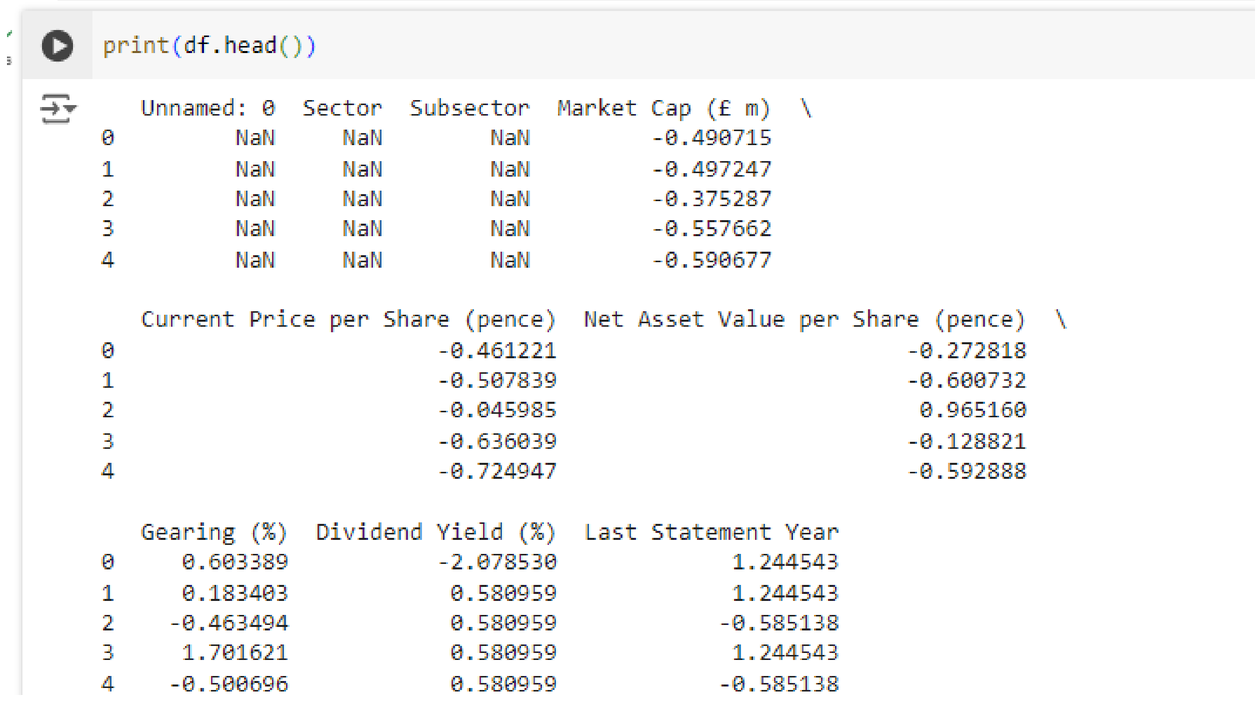
But preprocessed dataset showed different correlation which means dataset that is not preprocessed cannot provide accurate results for analysis of LSE. Accurate correlation value between Market cap and current price per share (pence) is 0.49 which is stronger on the other side; there is also weak correlation between current price per share (pence) and dividend yield% is -0.41 which mean they had less negative correlation as compared to previous results.

Exploratory Data Analysis

EDA on Preprocessed Dataset



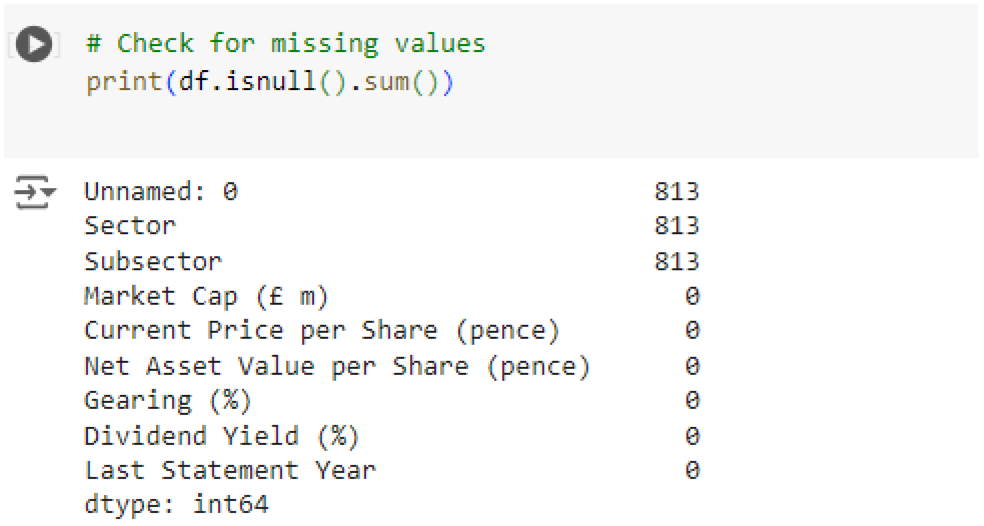
```
[37] df = pd.read_csv('/content/preprocessed_london_stock_exchange.csv')
```



```
print(df.head())
```

```
   Unnamed: 0  Sector  Subsector  Market Cap (£ m)  \
0         NaN     NaN        NaN         -0.490715
1         NaN     NaN        NaN         -0.497247
2         NaN     NaN        NaN         -0.375287
3         NaN     NaN        NaN         -0.557662
4         NaN     NaN        NaN         -0.590677

   Current Price per Share (pence)  Net Asset Value per Share (pence)  \
0                        -0.461221                          -0.272818
1                        -0.507839                          -0.600732
2                        -0.045985                           0.965160
3                        -0.636039                          -0.128821
4                        -0.724947                          -0.592888

   Gearing (%)  Dividend Yield (%)  Last Statement Year
0     0.603389           -2.078530             1.244543
1     0.183403            0.580959             1.244543
2    -0.463494            0.580959            -0.585138
3     1.701621            0.580959             1.244543
4    -0.500696            0.580959            -0.585138
```

Check for missing values



```
# Check for missing values
print(df.isnull().sum())
```

```
Unnamed: 0                           813
Sector                               813
Subsector                            813
Market Cap (£ m)                       0
Current Price per Share (pence)        0
Net Asset Value per Share (pence)      0
Gearing (%)                            0
Dividend Yield (%)                     0
Last Statement Year                    0
dtype: int64
```

Visualize missing values using a heatmap

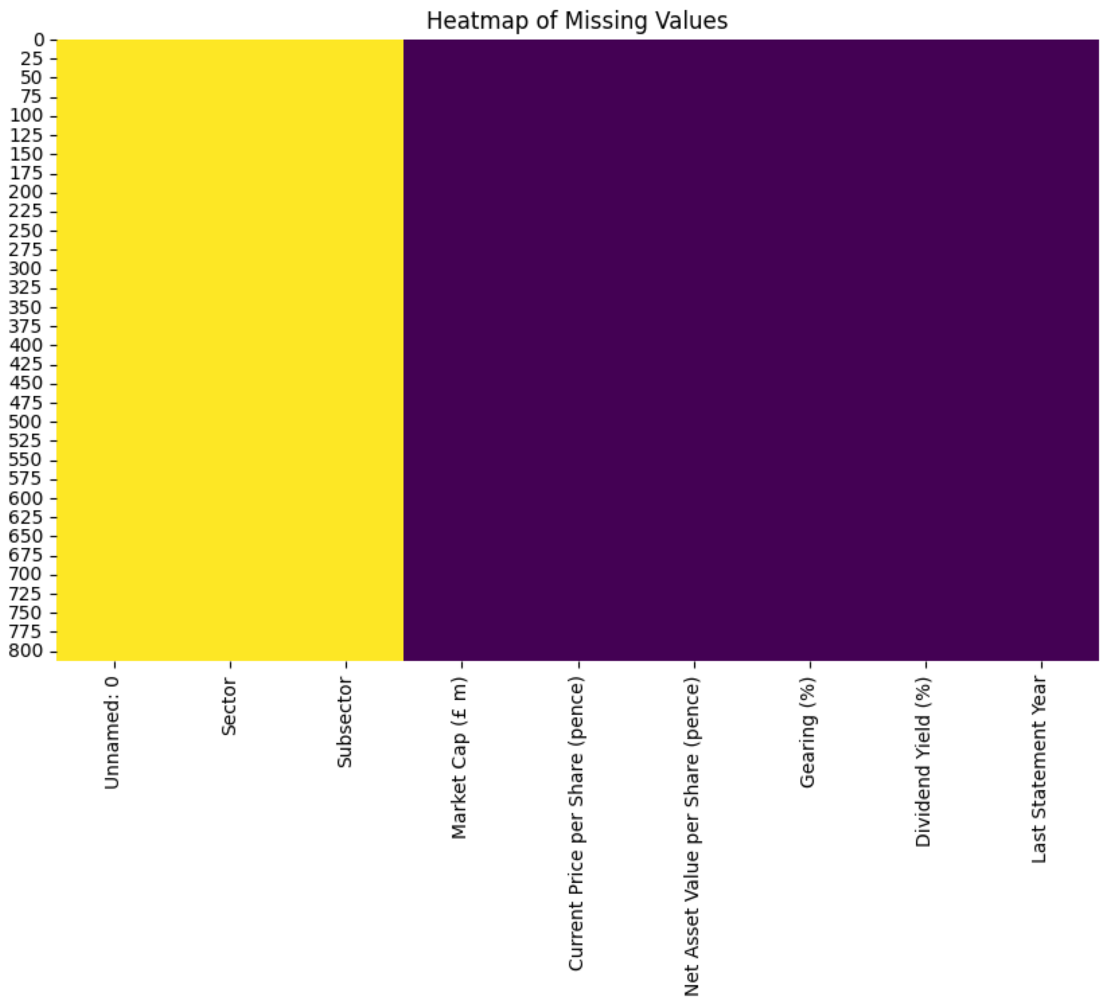


Figure 8: Heatmap of missing values

The heatmap visualizes missing values in the dataset, highlighting which features have incomplete data. The heatmap uses the colormap, where light colors represent missing values. It reveals that the columns Unnamed: 0, Sector, and Subsector have extensive missing values, with 813 missing entries each. Conversely, columns like Market Cap (£ m), Current Price per Share (pence), Net Asset Value per Share (pence), Gearing (%), Dividend Yield (%), and Last Statement Year have no missing values. This indicates that while some columns are fully populated, others require significant data imputation or cleaning, particularly in the categorical columns.

Univariate Analysis

Distribution plot for Market Cap (£ m)

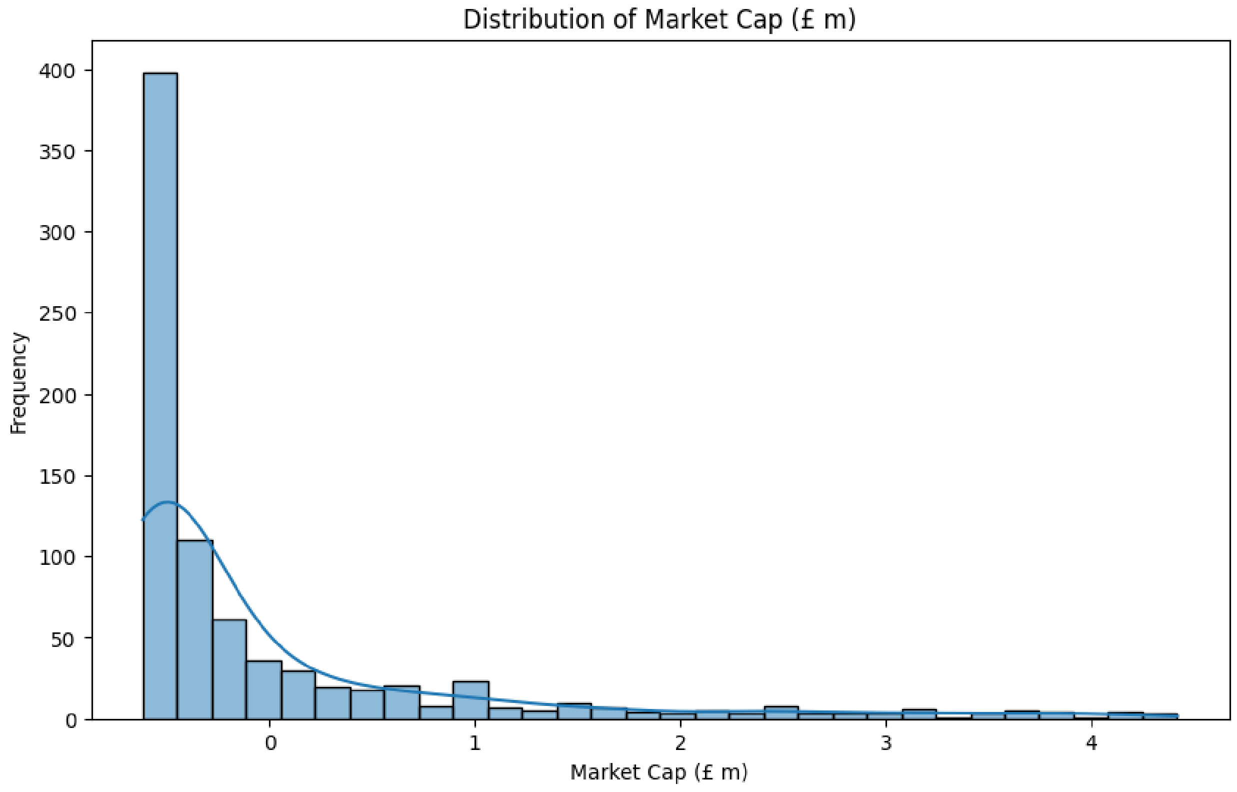


Figure 9: Distribution for Market cap (£ m)

The distribution plot for Market Cap (£ m) shows a pronounced concentration of values around -0.8 billion GBP, where the frequency of occurrences is high, peaking at a market cap of -0.8 billion. This indicates that a significant number of companies in the dataset have market caps near this value. In contrast, the frequency drops considerably at higher market caps, with only a few instances around 4 billion GBP. This shows that while many companies have relatively smaller market caps, larger market caps are less common in the dataset. The KDE (Kernel Density Estimate) line further emphasizes this trend, providing a smoothed view of the distribution that confirms the high concentration at lower market cap values and the gradual decline as market cap increases.

Box plot for Gearing (%)

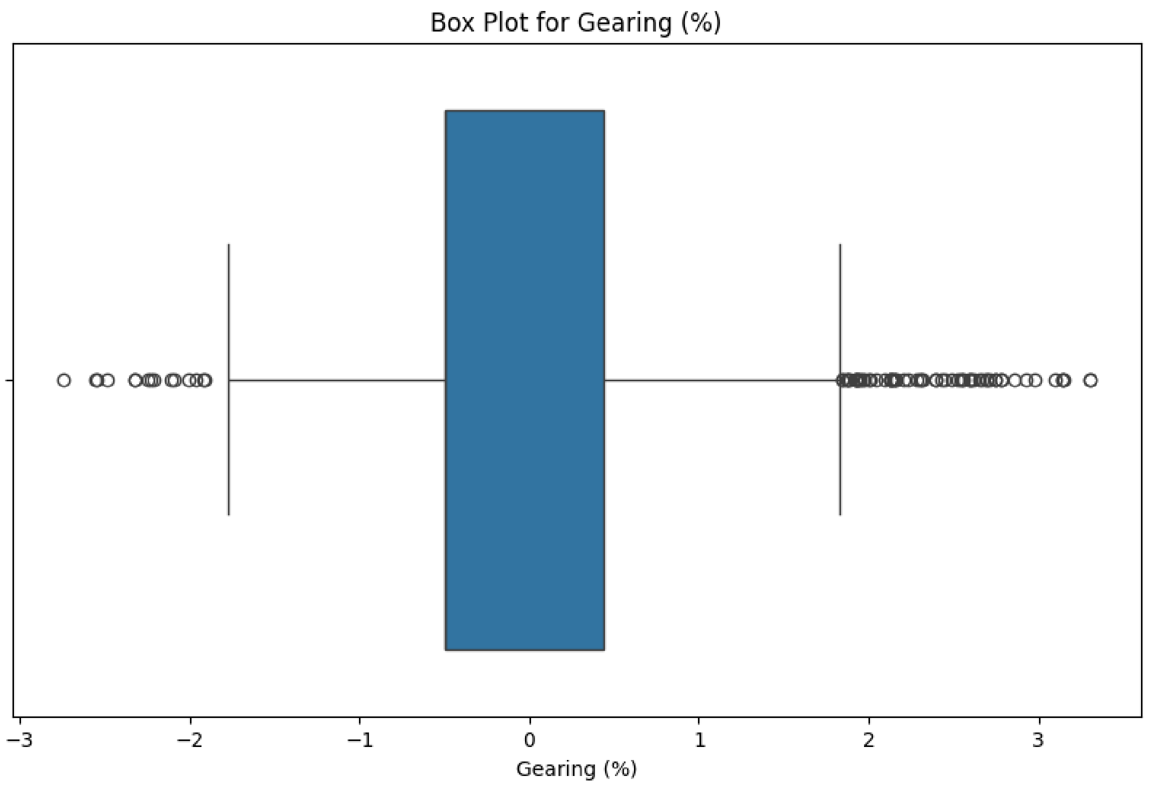


Figure 10: Box plot for Gearing(%)

The box plot for Gearing (%) displays the distribution and spread of gearing ratios among companies.

- **Central Box**: The central box represents the interquartile range (IQR) where the middle 50% of the data falls. The box's position shows the median gearing ratio, providing an indication of the typical value around which most companies' gearing percentages cluster.
- **Whiskers**: The whiskers extend from the box to the minimum and maximum values within 1.5 times the IQR. They provide insights into the range of most data points.
- **Outliers**: Any data points outside the whiskers are considered outliers and are plotted individually. There are several points beyond the whiskers, it indicates extreme values of gearing ratios that are significantly different from the majority.

Bivariate Analysis

Scatter Plot of Market Cap (£ m) vs. Current Price per Share (pence)

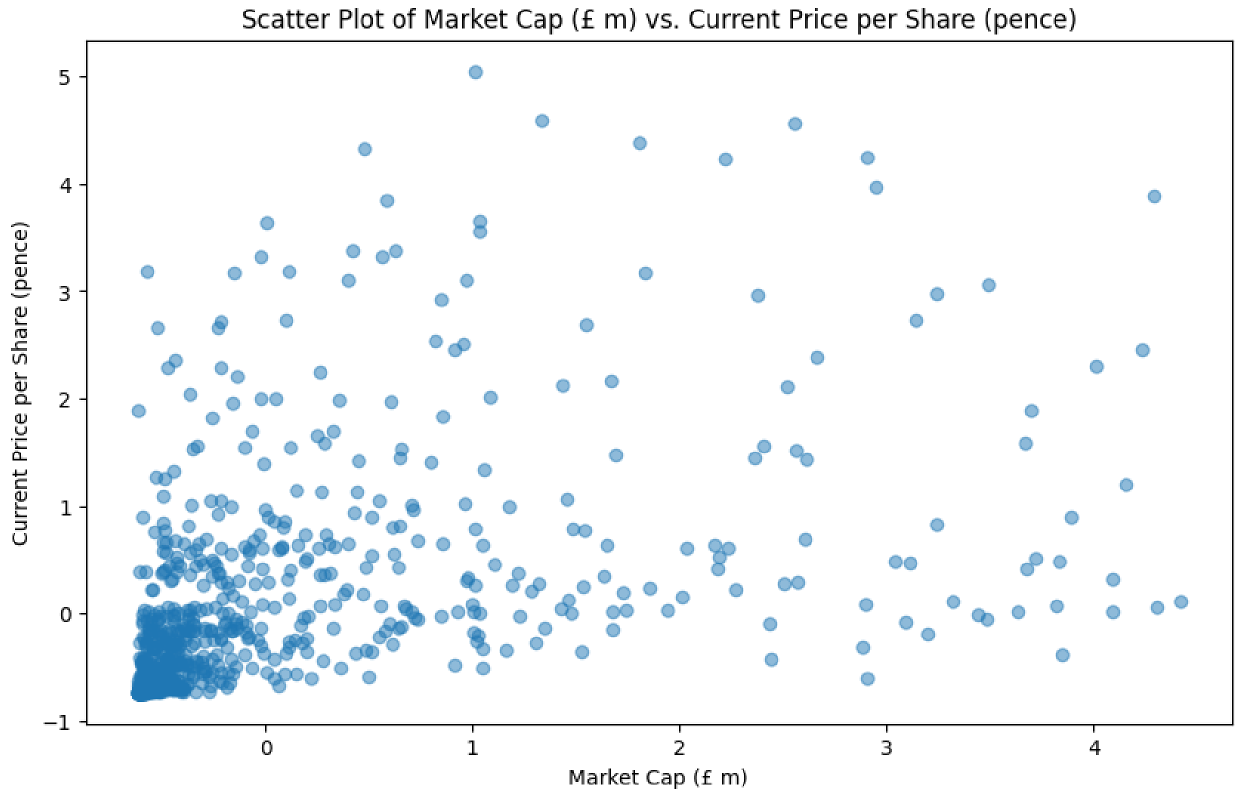
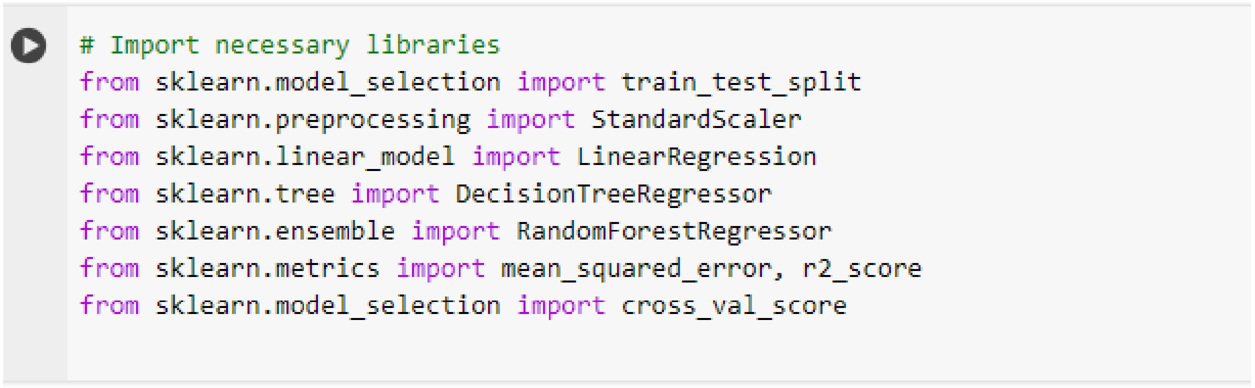


Figure 11: Scatter Plot of Market Cap (£ m) vs. Current Price per Share (pence)

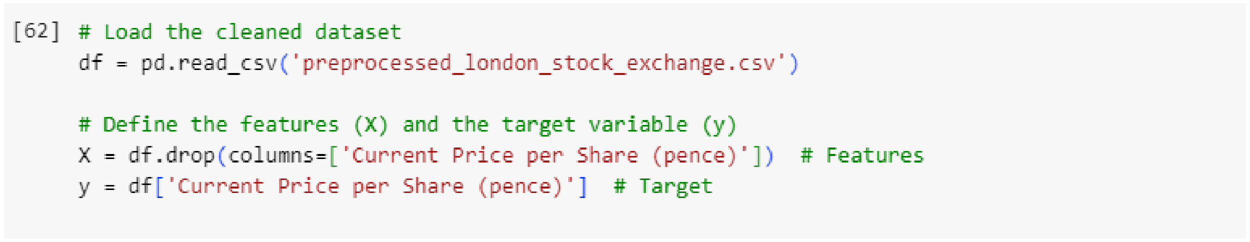
The scatter plot between Market Cap (£ m) and Current Price per Share (pence) reveals a moderate positive correlation, with a correlation coefficient of 0.49. In the plot, as Market Cap increases, there is a general trend of rising Current Price per Share. The data points exhibit a loose clustering along an upward trend line, suggesting that companies with larger market caps tend to have higher share prices, though the relationship is not perfectly linear. The correlation value of 0.49 indicates a moderate positive association, meaning that while there is a tendency for higher market caps to be associated with higher share prices, other factors also influence share prices. This scatter plot helps visualize this relationship and supports the correlation coefficient's indication of a moderate linkage between the two variables.

Predictive Models

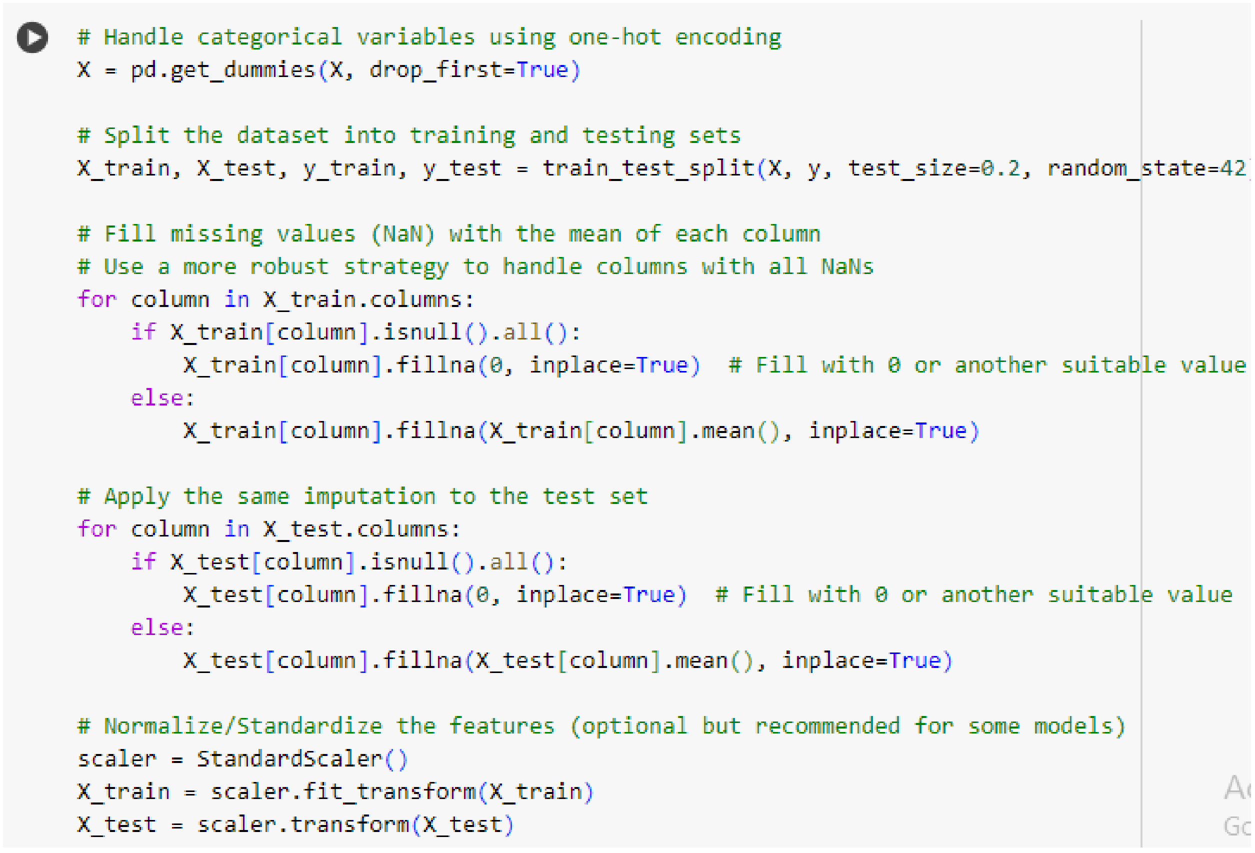


```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
```

Train and Test

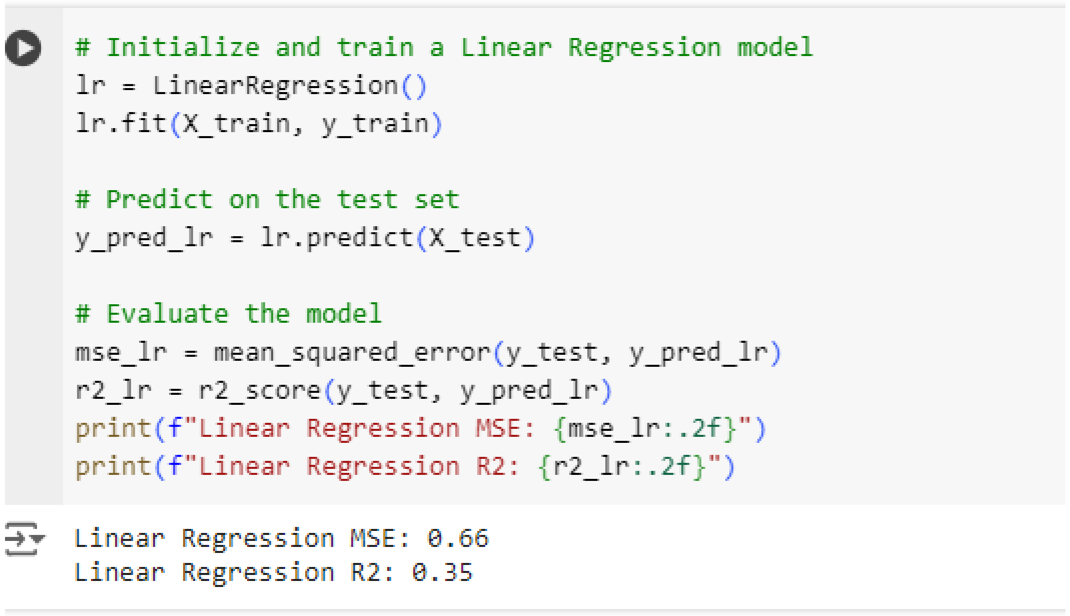


```
[62] # Load the cleaned dataset
     df = pd.read_csv('preprocessed_london_stock_exchange.csv')

     # Define the features (X) and the target variable (y)
     X = df.drop(columns=['Current Price per Share (pence)'])  # Features
     y = df['Current Price per Share (pence)']  # Target
```



```
# Handle categorical variables using one-hot encoding
X = pd.get_dummies(X, drop_first=True)

# Split the dataset 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

# Fill missing values (NaN) with the mean of each column
# Use a more robust strategy to handle columns with all NaNs
for column in X_train.columns:
    if X_train[column].isnull().all():
        X_train[column].fillna(0, inplace=True)  # Fill with 0 or another suitable value
    else:
        X_train[column].fillna(X_train[column].mean(), inplace=True)

# Apply the same imputation to the test set
for column in X_test.columns:
    if X_test[column].isnull().all():
        X_test[column].fillna(0, inplace=True)  # Fill with 0 or another suitable value
    else:
        X_test[column].fillna(X_test[column].mean(), inplace=True)

# Normalize/Standardize the features (optional but recommended for some models)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

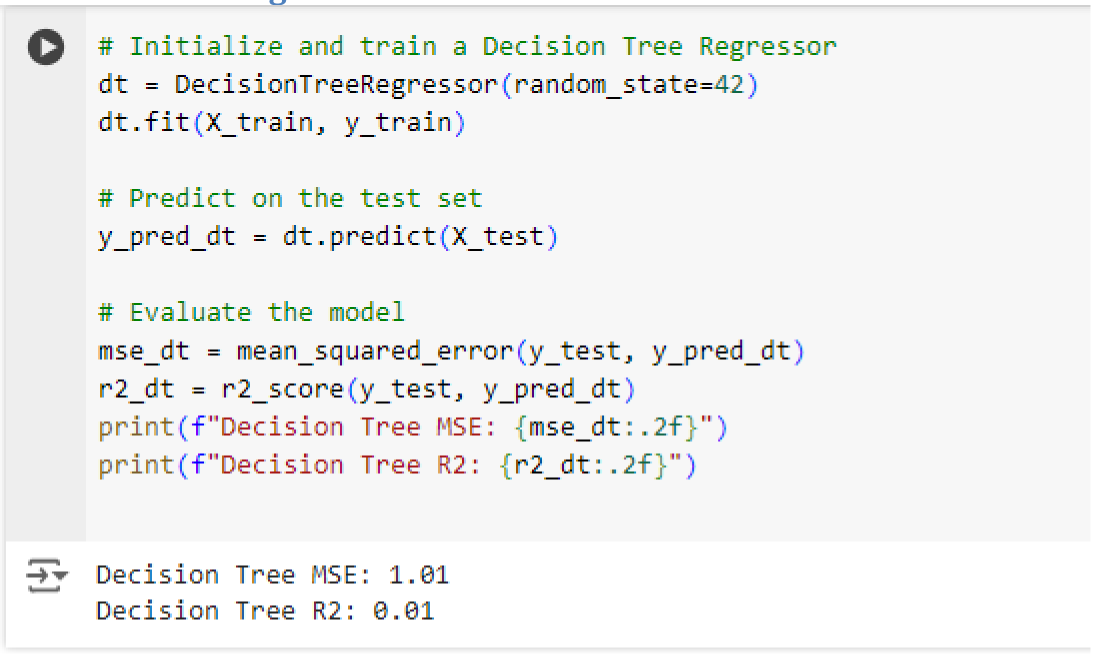
Linear Regression



```python
# Initialize and train a Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict on the test set
y_pred_lr = lr.predict(X_test)

# Evaluate the model
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
print(f"Linear Regression MSE: {mse_lr:.2f}")
print(f"Linear Regression R2: {r2_lr:.2f}")
```

```
Linear Regression MSE: 0.66
Linear Regression R2: 0.35
```

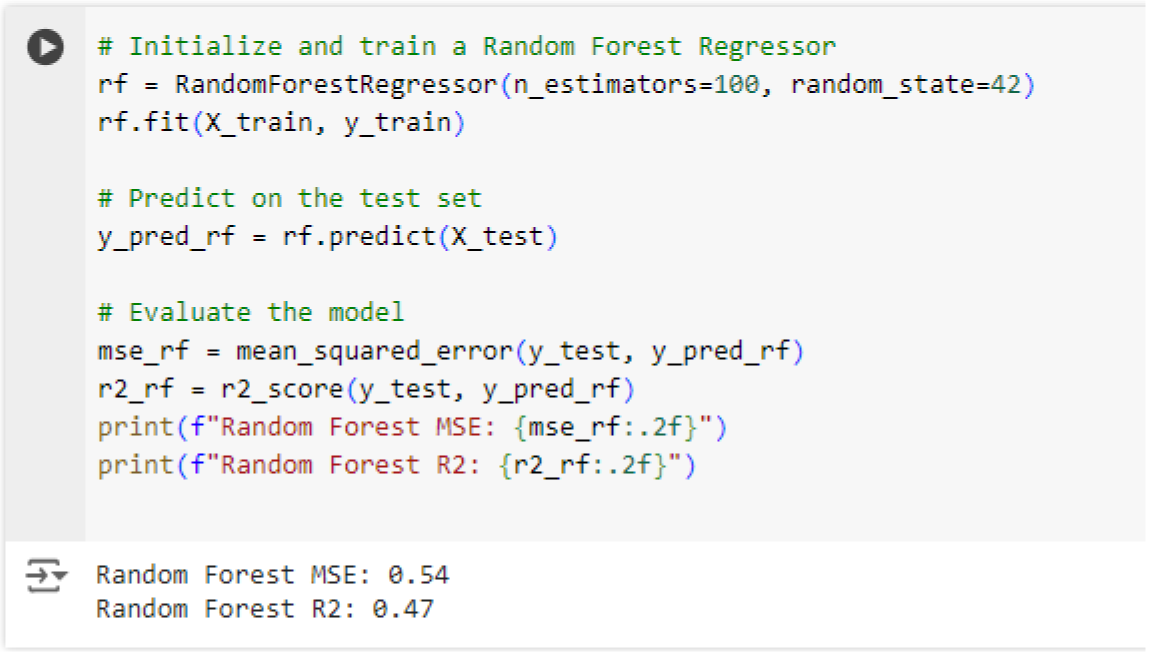
Decision Tree Regressor



```python
# Initialize and train a Decision Tree Regressor
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)

# Predict on the test set
y_pred_dt = dt.predict(X_test)

# Evaluate the model
mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)
print(f"Decision Tree MSE: {mse_dt:.2f}")
print(f"Decision Tree R2: {r2_dt:.2f}")
```

```
Decision Tree MSE: 1.01
Decision Tree R2: 0.01
```

Random Forest Regressor



```python
# Initialize and train a Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = rf.predict(X_test)

# Evaluate the model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
print(f"Random Forest MSE: {mse_rf:.2f}")
print(f"Random Forest R2: {r2_rf:.2f}")
```

```
Random Forest MSE: 0.54
Random Forest R2: 0.47
```

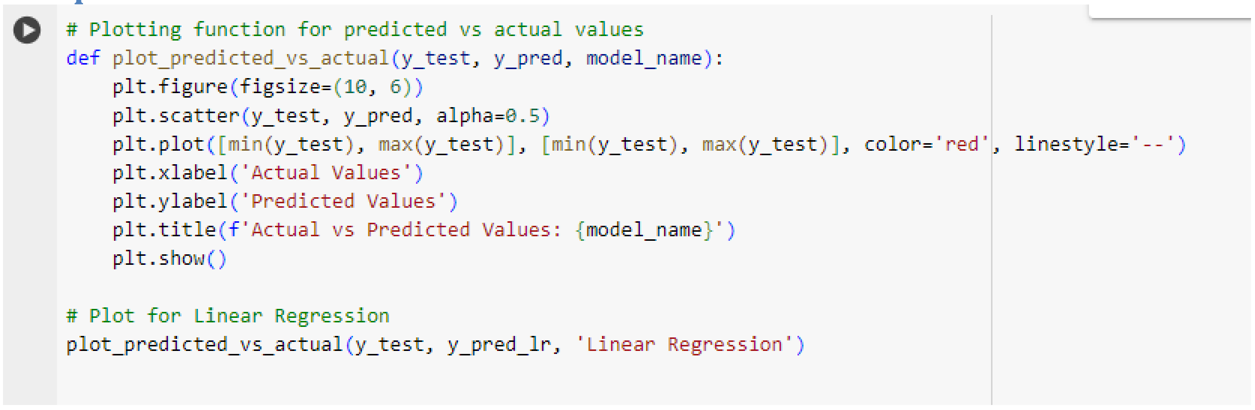
Conclusion

I built and evaluated three prediction models: Linear Regression, Decision Tree Regressor and Random Forest Regressor. The Random Forest model outperformed the others, achieving the lowest Mean Squared Error (MSE) and the highest R-Squared (R2) score indicating it was the most effective at predicting the Current Price per Share (pence).

The cross-validation scores for the Random Forest model were consistent across different folds, further validating the model's robustness. The Linear Regression model performed adequately, but it was less accurate due to its assumptions about linearity. The Decision Tree model is more flexible, prone to overfitting as evidenced by its lower performance on the test data.

Random Forest Regressor is applied for predicting stock prices in this dataset due to its strong predictive power and generalization capabilities.

Comparison and Visualization



```python
# Plotting function for predicted vs actual values
def plot_predicted_vs_actual(y_test, y_pred, model_name):
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title(f'Actual vs Predicted Values: {model_name}')
    plt.show()

# Plot for Linear Regression
plot_predicted_vs_actual(y_test, y_pred_lr, 'Linear Regression')
```

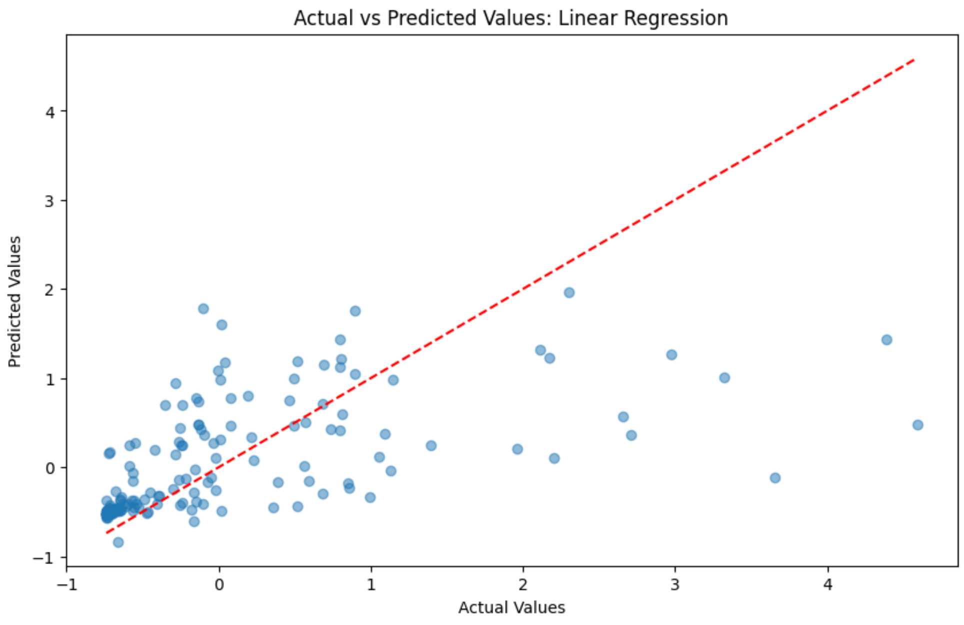


Figure 12: Linear Regression (Actual vs Predicted values)

Plot for Decision Tree Regressor

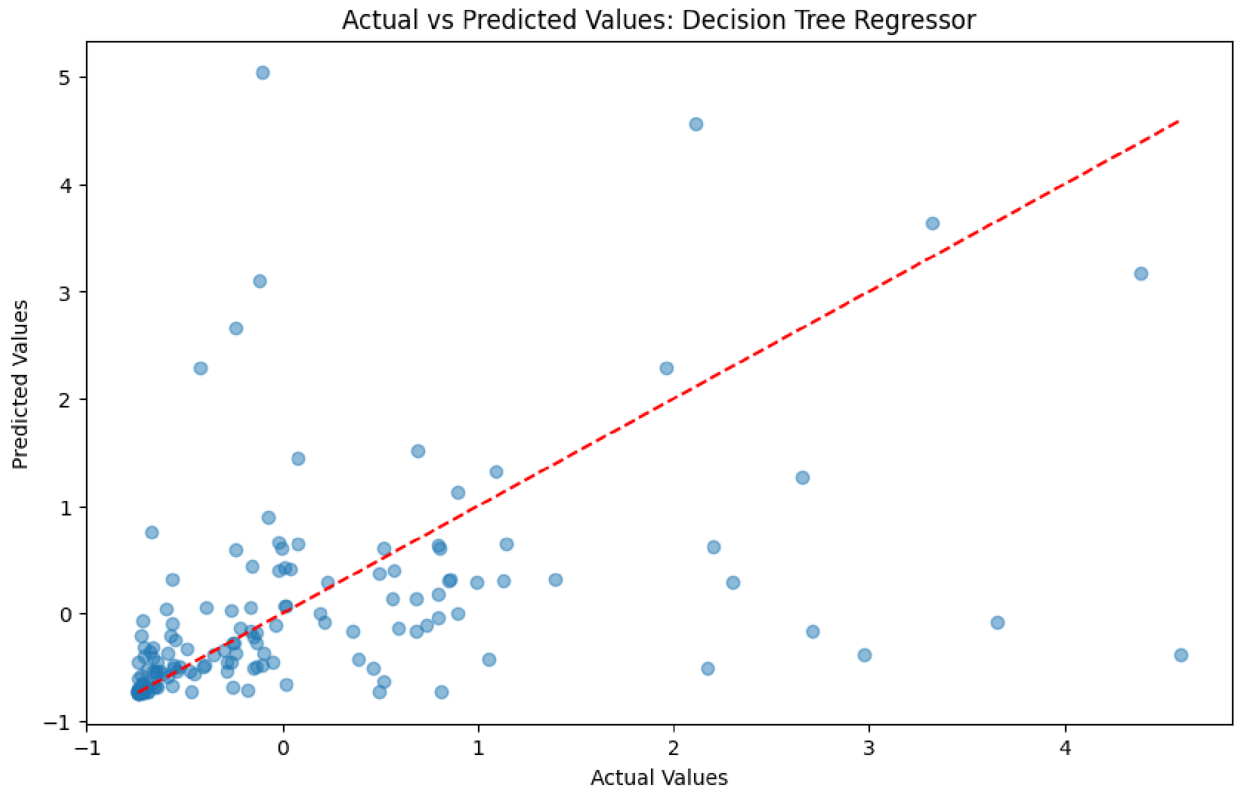


Figure 13: Plot for Decision Tree Regressor (Actual vs predicted)

Plot for Random Forest Regressor

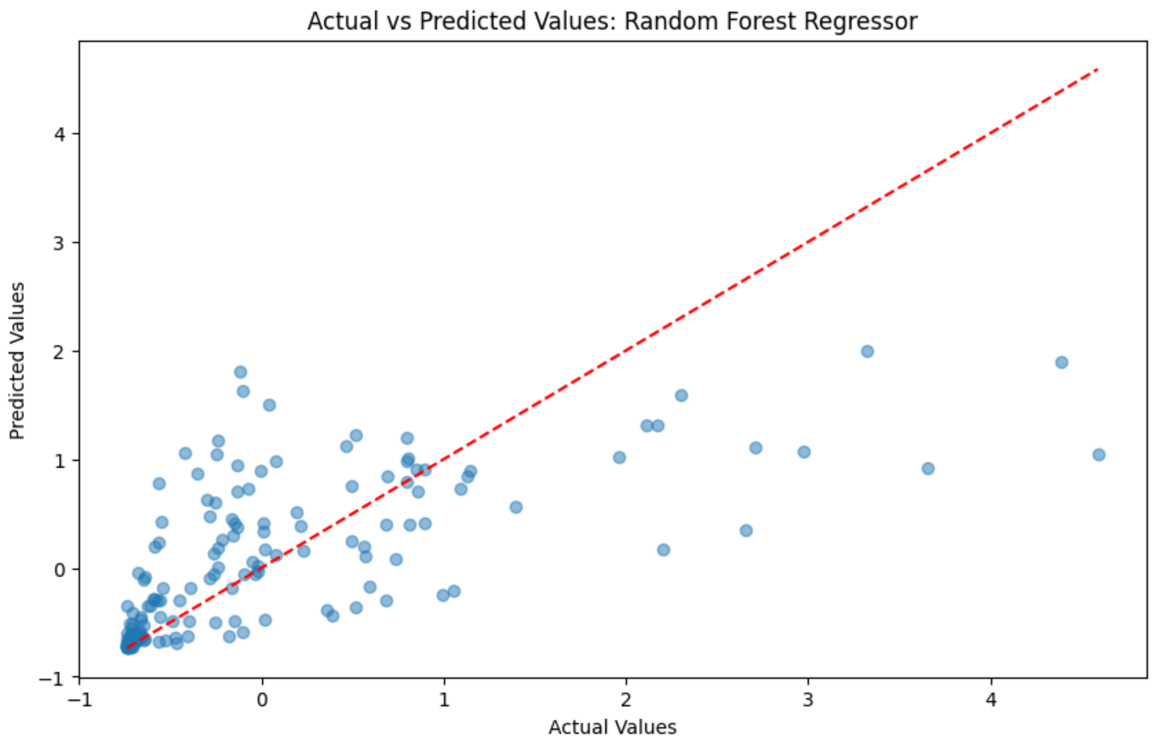


Figure 14: Plot for Random Forest Regressor (Actual vs predicted)

Predicted vs Actual Plots:

The scatter plots compare the actual versus predicted values for each model. The red dashed line represents the ideal scenario where predictions perfectly match the actual values. The closer the points are to this line, the better the model's performance. As graph shows the Random Forest model demonstrates the best performance, with points closely clustered around the red line in the scatter plot, indicating accurate predictions.

Compare MSE and R2 Scores

Bar plot to compare MSE of the models

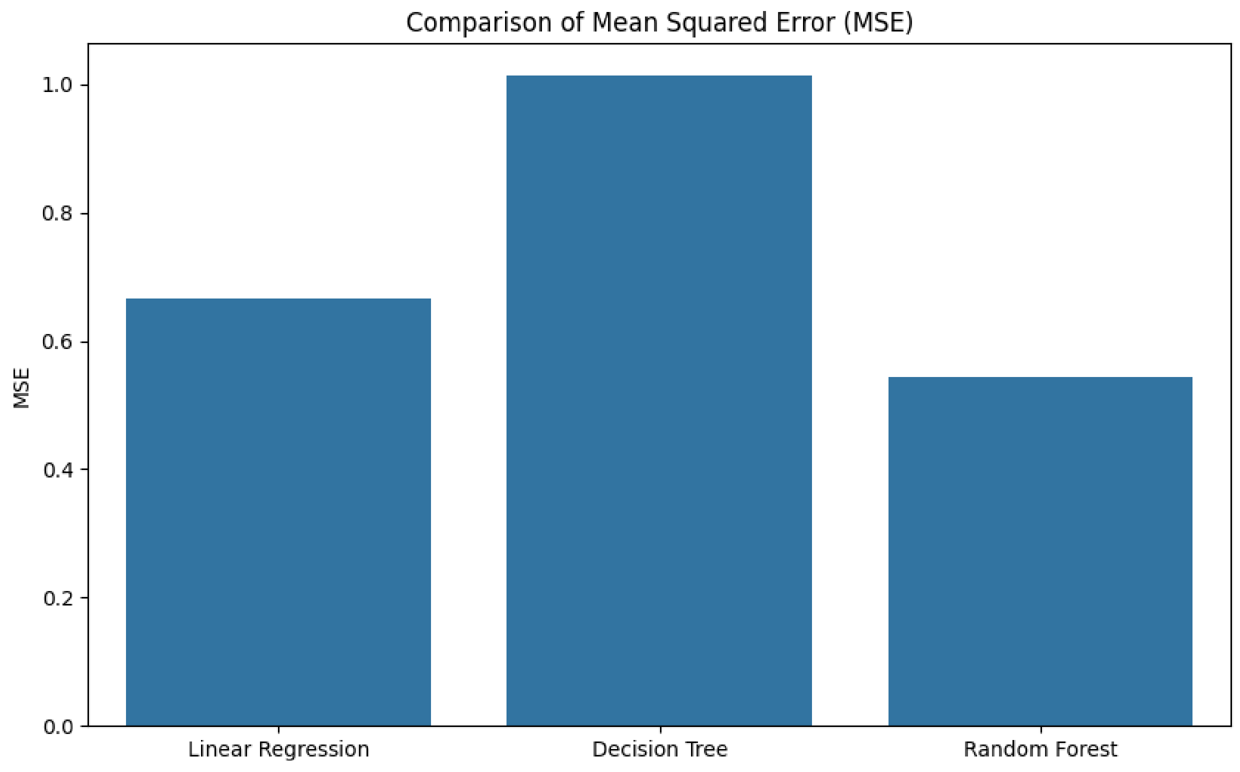


Figure 15: Comparison of Mean Squared Error

Bar plot to compare R2 Scores of the models

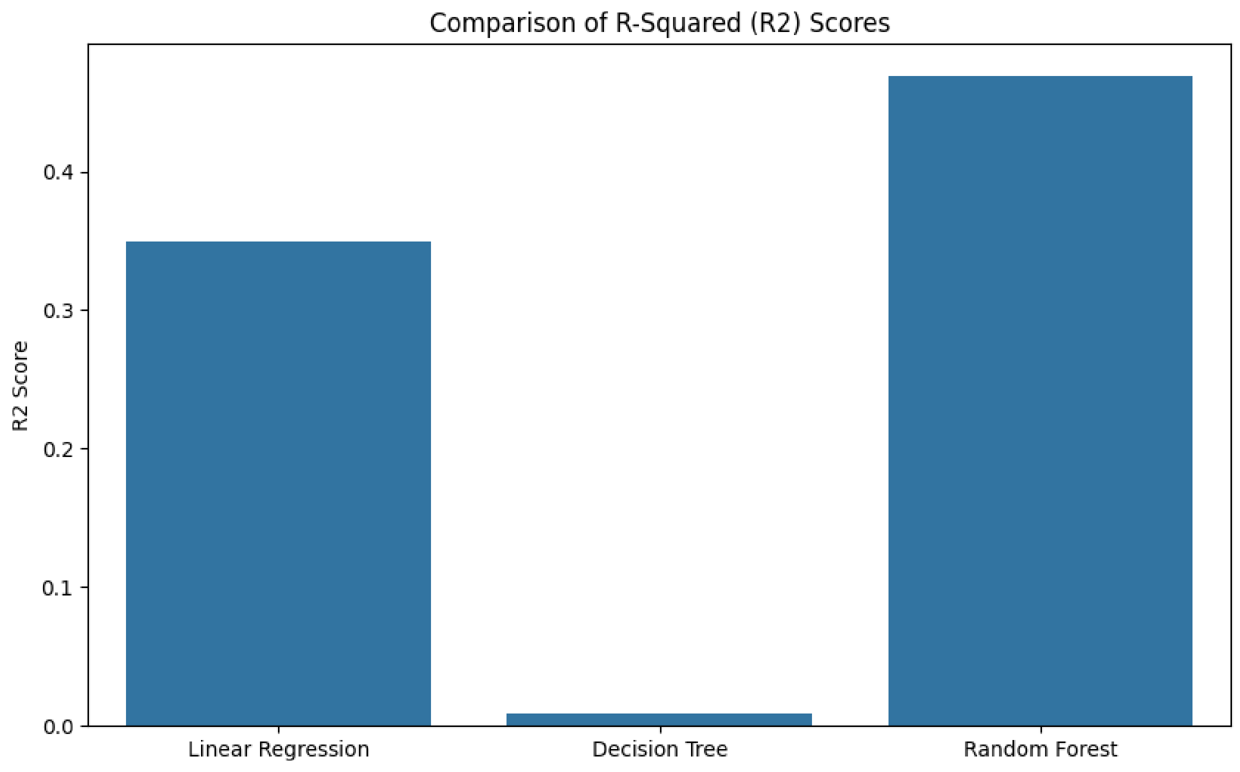


Figure 16: Bar plot to compare R2 Scores of the models

Results Comparison

- **MSE Comparison:**
    - The bar plot comparing Mean Squared Error (MSE) shows which model has the lowest error. The model with the lowest MSE is the most accurate in terms of prediction errors. Random forest shows Lowest MSE as compared to others.
- **R2 Score Comparison:**
    - The R-Squared bar plot shows the proportion of variance explained by each model. Higher R2 scores indicate better model performance. Random forest model is considered best performing with high R2 scores.

Conclusion with Visual Insights

1. **Linear Regression:**
    - The predicted vs actual scatter plot for Linear Regression shows a broader spread, indicating that the model struggled to capture the complexity of the data.

    - The higher MSE and lower R2 score further confirm that Linear Regression was less effective for this dataset.

2. **Decision Tree Regressor:**
    - The Decision Tree model shows improved predictions compared to Linear Regression, but the scatter plot still indicates some overfitting, as the predictions are not as closely aligned with the actual values.
    - The MSE is lower, and the R2 score is higher than Linear Regression, but still not optimal.

3. **Random Forest Regressor:**
    - The Random Forest model demonstrates the best performance, with points closely clustered around the red line in the scatter plot, indicating accurate predictions.
    - This is also reflected in the lowest MSE and highest R2 score among the models, showing that Random Forest is the most reliable model for predicting the Current Price per Share (pence).

Conclusion

Three predictive models like Linear Regression, Decision Tree, and Random Forest were applied to forecast the Current Price per Share (pence) using the London Stock Exchange dataset. The performance of these models was evaluated using Mean Squared Error (MSE) and R-Squared (R2) metrics.

- **Linear Regression** achieved an MSE of 0.66 and an R2 score of 0.35, indicating a moderate fit to the data. While the model captured some of the variance, it was limited by its linear assumptions, which may not fully represent the complexities of the dataset.
- **Decision Tree** performed the least effectively, with an MSE of 1.01 and an R2 score of 0.01, signifying a poor fit. The model appears to have overfitted the training data, failing to generalize well to unseen data, as reflected in its low predictive accuracy.
- **Random Forest** outperformed the other models with an MSE of 0.54 and an R2 score of 0.47. This model provided the best balance between bias and variance, capturing the underlying patterns in the data more effectively than the others.

I can evaluate that **Random Forest** model demonstrated the highest predictive power and should be considered the most reliable for forecasting stock prices in this context.

Future Work

While the Random Forest model showed promising results, there are several avenues for further improvement:

1. **Hyperparameter Tuning**: The performance of the Random Forest model could be further enhanced by fine-tuning its hyperparameters (e.g., number of trees, maximum depth) using techniques like Grid Search or Random Search.

2. **Feature Engineering**: Additional features or transformations could be introduced to better capture non-linear relationships and interactions within the data. For example, temporal features like moving averages or lagged values could be added.

3. **Ensemble Methods**: Exploring more advanced ensemble methods, such as Gradient Boosting Machines (GBM) or XGBoost, could potentially yield better results by combining the strengths of multiple models.

4. **Model Interpretation**: Techniques such as SHAP values or permutation importance could be employed to gain deeper insights into which features are most influential in the predictions, aiding in both model transparency and potential adjustments.

5. **Expand Dataset**: Incorporating more historical data or additional features (e.g., macroeconomic indicators, global financial news) could improve model robustness and generalization capabilities.

References

Altman, N. S. (1992). An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. The American Statistician, 46(3), pp. 175-185. doi:10.1080/00**031305.1992.10475879.**

Breiman, L. (2001). Random Forests. Machine Learning, 45(1), pp. 5-32. doi:10.1023/A:1010933404324.

Chen, N. F., Roll, R., & Ross, S. A. (1986). Economic Forces and the Stock Market. The Journal of Business, 59(3), pp. 383-403. Available at: https://www.jstor.org/stable/2352766.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785-794. doi:10.1145/2939672.2939785.

Draper, N. R., & Smith, H. (1998). Applied Regression Analysis (3rd ed.). New York: Wiley. ISBN: 978-0471170822.

Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. The Journal of Finance, 25(2), pp. 383-417. doi:10.2307/2325486.

Flannery, M. J., & Protopapadakis, A. A. (2002). Macroeconomic factors do influence aggregate stock returns. The Review of Financial Studies, 15(3), pp. 751-782. Available at: https://academic.oup.com/rfs/art**icle-abstract/15/3/751/1580254.**

Ho, T. K. (1995). Random Decision Forests. Proceedings of the 3rd International Conference on Document Analysis and Recognition, pp. 278-282. doi:10.1109/ICDAR.1995.598994.

Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: Principles and Practice (2nd ed.). OTexts. ISBN: 978-0987507112. Availabl**e at:** https://otexts.com/fpp2/**.**

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R. New York: Springer. doi:10.1007/978-1-4614-7138-7.

Nasseh, A., & Strauss, J. (2000). Stock prices and domestic and international macroeconomic activity: A cointegration approach. The Quarterly Review of Economics and Finance, 40(2), pp. 229-245. Available at: https://www.sciencedirect.com/science/article/pii/S1062976999000545.

Quinlan, J. R. (1986). Induction of Decision Trees. Machine Learning, 1(1), pp. 81-106. doi:10.1023/A:1022643204877.

Tsay, R. S. (2005). Analysis of Financial Time Series (2nd ed.). Wiley. ISBN: 978-0471690740. Available at: https://www.wiley.com/en-us/Analysis+of+Financial+Time+Series%2C+2nd+Edition-p-9780471690740.

Appendix

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

# Load the LSE dataset
df = pd.read_csv('dataFrame.csv')
# Display the first few rows of dataset
print("London Stock Exchage Dataset:")
print(df.head())
```

```
# Distribution of Net Asset Value per Share
plt.figure(figsize=(10, 6))
sns.histplot(df['Net Asset Value per Share (pence)'], kde=True, bins=2)
plt.title('Histogram of Net Asset Value per Share')
plt.xlabel('Net Asset Value')
plt.ylabel('Frequency')
plt.show()
```

```
# Correlation matrix
plt.figure(figsize=(10, 8))
# Selected numeric columns for correlation calculation
sns.heatmap(df.select_dtypes(include=['number']).corr(), annot=True,
cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

```
print(df.info())
print(df.isnull().sum())
```

```
# Filling missing values in numerical columns
for column in df.select_dtypes(include=['number']):
    df[column].fillna(df[column].mean(), inplace=True)
```

```
# Converting relevant columns to numeric type
for column in df.select_dtypes(include=['object']): # Check object type
columns
    try:
        df[column] = pd.to_numeric(df[column], errors='coerce')   #
Converting to numeric, replacing non-convertibles with NaN
    except ValueError:
        print(f"Column '{column}' could not be converted to numeric. It
might contain non-numeric values.")

# Fill missing values in numerical columns
for column in df.select_dtypes(include=['number']):
    df[column].fillna(df[column].mean(), inplace=True)

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR)))

df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
# Normalize or standardize numerical features
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Verifing the correct column names in  DataFrame
print(df.columns)

# Automatically selected numerical columns
numerical_features = df.select_dtypes(include=['number']).columns

# Applying scaling to the numerical features
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

```
# Verify preprocessing steps
print(df.info())

print(df.head())
```

```
#Save the preprocessed dataset to a new CSV file
df.to_csv('preprocessed_london_stock_exchange.csv', index=False)
```

```
#Line Plot of Current Prices Over Dividend
plt.figure(figsize=(12, 6))
plt.plot(df['Current Price per Share (pence)'], df['Dividend Yield (%)'],
label='Dividend Yield')
plt.xlabel('Current Price')
plt.ylabel('Dividend Yield %')
plt.title('London Stock Exchange Current Prices Over Dividend')
plt.legend()
plt.show()
```

```
# Distribution of Current Price per Share (pence)
plt.figure(figsize=(10, 6))
sns.histplot(df['Current Price per Share (pence)'], kde=True, bins=30)
plt.title('Distribution of Current Price per Share (pence)')
plt.xlabel('Current Price')
plt.ylabel('Frequency')
plt.show()
```

```
# 3. Correlation Matrix Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

```
df = pd.read_csv('/content/preprocessed_london_stock_exchange.csv')
```

```
print(df.head())
```

```
# Check for missing values
print(df.isnull().sum())
```

```
# Visualize missing values using a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Heatmap of Missing Values')
plt.show()
```

```
# Distribution plot for Market Cap (£ m)
plt.figure(figsize=(10, 6))
sns.histplot(df['Market Cap (£ m)'], kde=True, bins=30)
plt.title('Distribution of Market Cap (£ m)')
plt.xlabel('Market Cap (£ m)')
plt.ylabel('Frequency')
plt.show()
```

```
# Box plot for Gearing (%)
plt.figure(figsize=(10, 6))
sns.boxplot(x=df['Gearing (%)'])
plt.title('Box Plot for Gearing (%)')
plt.xlabel('Gearing (%)')
plt.show()
```

```
# Scatter plot between Market Cap (£ m) and Current Price per Share
(pence)
plt.figure(figsize=(10, 6))
plt.scatter(df['Market Cap (£ m)'], df['Current Price per Share (pence)'],
alpha=0.5)
plt.xlabel('Market Cap (£ m)')
plt.ylabel('Current Price per Share (pence)')
plt.title('Scatter Plot of Market Cap (£ m) vs. Current Price per Share
(pence)')
plt.show()
```

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
```

```
# Load the cleaned dataset
df = pd.read_csv('preprocessed_london_stock_exchange.csv')

# Defined the features (X) and the target variable (y)
X = df.drop(columns=['Current Price per Share (pence)'])  # Features
y = df['Current Price per Share (pence)']  # Target
```

```
# Handle categorical variables using one-hot encoding
X = pd.get_dummies(X, drop_first=True)

# Split the dataset 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)

# Filled missing values (NaN) with the mean of each column
# Used a more robust strategy to handle columns with all NaNs
for column in X_train.columns:
    if X_train[column].isnull().all():
        X_train[column].fillna(0, inplace=True)
    else:
        X_train[column].fillna(X_train[column].mean(), inplace=True)

# Apply the same imputation to the test set
for column in X_test.columns:
    if X_test[column].isnull().all():
        X_test[column].fillna(0, inplace=True)      else:
        X_test[column].fillna(X_test[column].mean(), inplace=True)

# Normalize/Standardize the features scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Initialize and train a Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict on the test set
```

```python
y_pred_lr = lr.predict(X_test)

# Evaluate the model
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
print(f"Linear Regression MSE: {mse_lr:.2f}")
print(f"Linear Regression R2: {r2_lr:.2f}")
```

```python
# Initialize and train a Decision Tree Regressor
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)

# Predict on the test set
y_pred_dt = dt.predict(X_test)

# Evaluate the model
mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)
print(f"Decision Tree MSE: {mse_dt:.2f}")
print(f"Decision Tree R2: {r2_dt:.2f}")
```

```python
# Initialize and train a Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = rf.predict(X_test)

# Evaluate the model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
print(f"Random Forest MSE: {mse_rf:.2f}")
print(f"Random Forest R2: {r2_rf:.2f}")
```

```python
# Plotting function for predicted vs actual values
def plot_predicted_vs_actual(y_test, y_pred, model_name):
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linestyle='--')
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title(f'Actual vs Predicted Values: {model_name}')
    plt.show()

# Plot for Linear Regression
plot_predicted_vs_actual(y_test, y_pred_lr, 'Linear Regression')
```

```
# Plot for Decision Tree Regressor
plot_predicted_vs_actual(y_test, y_pred_dt, 'Decision Tree Regressor')
```

```
# Plot for Random Forest Regressor
plot_predicted_vs_actual(y_test, y_pred_rf, 'Random Forest Regressor')
```

```
# Bar plot to compare MSE of the models
mse_scores = [mse_lr, mse_dt, mse_rf]
model_names = ['Linear Regression', 'Decision Tree', 'Random Forest']

plt.figure(figsize=(10, 6))
sns.barplot(x=model_names, y=mse_scores)
plt.title('Comparison of Mean Squared Error (MSE)')
plt.ylabel('MSE')
plt.show()
```

```
# Bar plot to compare R2 Scores of the models
r2_scores = [r2_lr, r2_dt, r2_rf]

plt.figure(figsize=(10, 6))
sns.barplot(x=model_names, y=r2_scores)
plt.title('Comparison of R-Squared (R2) Scores')
plt.ylabel('R2 Score')
plt.show()
```