Stock Market Analysis

Assignment Tasks

Part 1: Data Cleaning and Exploration:

1. Calculate basic summary statistics for each column (mean, median, standard deviation, etc.).
2. Explore the distribution of the 'Close' prices over time.
3. Identify and analyze any outliers (if any) in the dataset.

Part 2: Time Series Analysis / Rolling Window / Moving Averages :

1. Create a line chart to visualize the 'Close' prices over time.
2. Calculate and plot the daily percentage change in closing prices.
3. Investigate the presence of any trends or seasonality in the stock prices.
4. Apply moving averages to smooth the time series data in 15/30 day intervals against the original graph.
5. Calculate the average closing price for each stock.
6. Identify the top 5 and bottom 5 stocks based on average closing price.

Part 3: Volatility Analysis:

1. Calculate and plot the rolling standard deviation of the 'Close' prices.
2. Create a new column for daily price change (Close - Open).
3. Analyze the distribution of daily price changes.
4. Identify days with the largest price increases and decreases.
5. Identify stocks with unusually high trading volume on certain days.

Part 4: Correlation and Heatmaps:

1. Explore the relationship between trading volume and volatility.
2. Calculate the correlation matrix between the 'Open' & 'High', 'Low' &'Close' prices.
3. Create a heatmap to visualize the correlations using the seaborn package.

Bonus Task:

** Rolling Window Analysis/ Moving Averages

In [8]:
```
specific_company='RECKITTBEN'

specific_data=stock_data[stock_data['Name']==specific_company]

specific_data['7_Day_Rolling_Avg']=specific_data['Close'].rolling(window=7).mean()

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

plt.plot(specific_data['Date'],specific_data['Close'],label=f'{specific_company} Closing Price',color='blue')

plt.plot(specific_data['Date'],specific_data['7_Day_Rolling_Avg'],label=f'{specific_company} 7 Day Rolling Average o


plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title(f'{specific_company} 7 Day Rolling Average of Closing Price')
plt.grid()
plt.legend()

plt.xticks(rotation=45)
plt.show()
```

```
/var/folders/dq/p8y24h810wl0mp6zk4g49g0w0000gn/T/ipykernel_74754/3922744609.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  specific_data['7_Day_Rolling_Avg']=specific_data['Close'].rolling(window=7).mean()
```

During the rolling window analysis, we encountered a warning. Find out what's causing this & apply a fix to avoid the warning.

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib as mlb
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
```

.

```
# Read the CSV file
stock_data = pd.read_csv('Stock_Market_Data.csv')
# 1st 5 rows of dataset
stock_data.head()
```

	Date	Name	Open	High	Low	Close	Volume
0	02-01-2022	01.Bank	22.83	23.20	22.59	22.93	1842350.41
1	03-01-2022	01.Bank	23.03	23.29	22.74	22.90	1664989.63
2	04-01-2022	01.Bank	22.85	23.13	22.64	22.84	1354510.97
3	05-01-2022	01.Bank	22.91	23.20	22.70	22.98	1564334.81
4	06-01-2022	01.Bank	23.12	23.65	23.00	23.37	2586344.19

Part 1: Data Cleaning and Exploration:

1. Calculating basic summary statistics for each column (mean, median, standard deviation, etc.)

```
stock_data.describe()
#50% is also the median
```

	Open	High	Low	Close	Volume
count	49158.000000	49158.000000	49158.000000	49158.000000	4.915800e+04
mean	157.869018	159.588214	155.906364	157.351462	5.619999e+05
std	520.191624	523.348078	517.136149	519.711667	1.276909e+06
min	3.900000	3.900000	3.000000	3.800000	1.000000e+00
25%	19.000000	19.300000	18.700000	19.000000	5.109475e+04
50%	40.300000	41.000000	39.535000	40.100000	1.824160e+05
75%	89.400000	90.500000	87.700000	88.700000	5.401398e+05
max	6000.000000	6050.000000	5975.000000	6000.500000	6.593180e+07

.

.

.

.

.

.

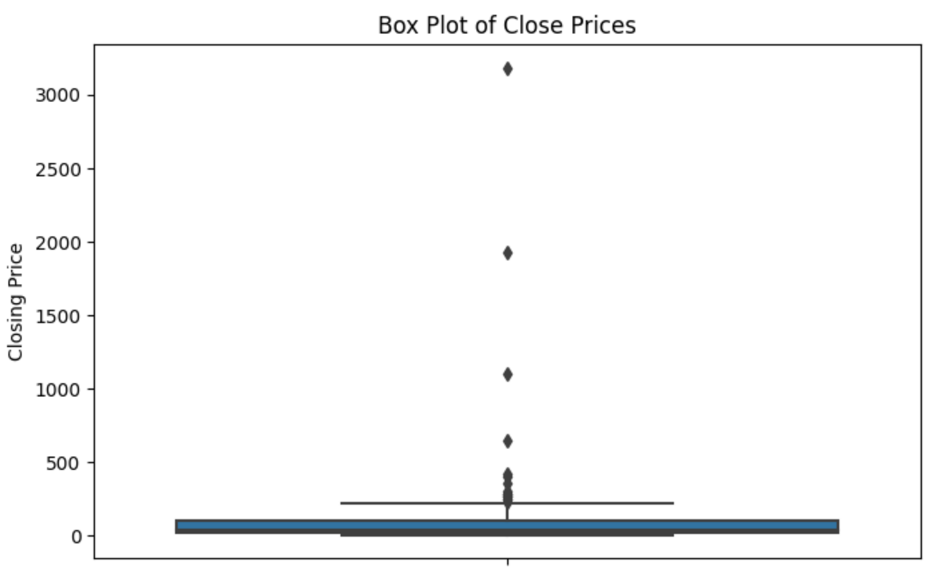
.

2. Exploring the distribution of the 'Close' prices over time.

```
plt.figure(figsize = (8, 5))
plt.hist(stock_data['Close'])
#sns.histplot(stock_data['Close'])
plt.xlabel('Closing Price')
plt.ylabel('Frequency')
plt.title('Distribution of Close Prices Over Time')
plt.show()
```

3. Identifying and analyzing any outliers (if any) in the dataset.

```
# Box plot to visualize outliers in 'Close' prices
plt.figure(figsize = (8, 5))
sns.boxplot(y = stock_data['Close'])
plt.ylabel('Closing Price')
plt.title('Box Plot of Close Prices')
plt.show()
```




```
# Calculating the Interquartile Range (IQR)
Q1 = stock_data['Close'].quantile(0.25)
Q3 = stock_data['Close'].quantile(0.75)
IQR = Q3 - Q1

# Defining a threshold for identifying outliers
threshold = 1.5 * IQR

# Identifying and analyzing outliers
outliers = stock_data[(stock_data['Close'] < Q1 - threshold) | (stock_data['Close'] > Q3 + threshold)]

# Printing information about outliers
print("Number of outliers:", len(outliers))
print("Outliers:")
print(outliers[['Date', 'Close']])
```

```
Number of outliers: 6960
Outliers:
             Date   Close
110    02-01-2022  305.16
111    03-01-2022  303.13
112    04-01-2022  303.66
113    05-01-2022  303.43
114    06-01-2022  299.72
...           ...     ...
49043  26-06-2022  207.90
49044  27-06-2022  207.10
49045  28-06-2022  205.80
49046  29-06-2022  209.70
49047  30-06-2022  214.10

[6960 rows x 2 columns]
```

Exploratory Data Analysis (EDA)

```
print(stock_data.shape)
stock_data.dtypes
```

```
(49158, 7)
Date       object
Name       object
Open      float64
High      float64
Low       float64
Close     float64
Volume    float64
dtype: object
```

```
stock_data['Date'] = pd.to_datetime(stock_data['Date'], dayfirst = True)
```

.

.

.

.

.

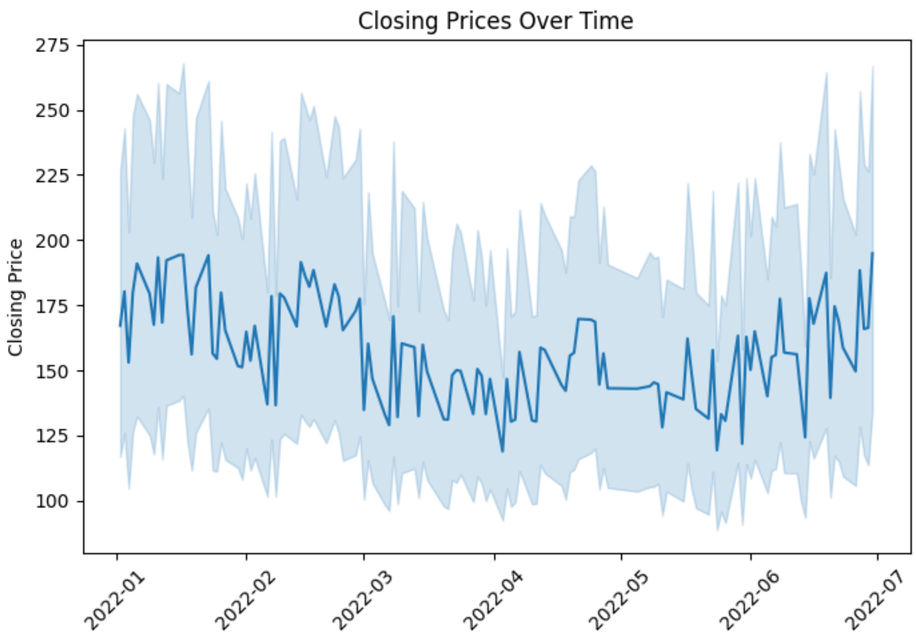
.

.

Part 2: Time Series Analysis / Rolling Window / Moving Averages :

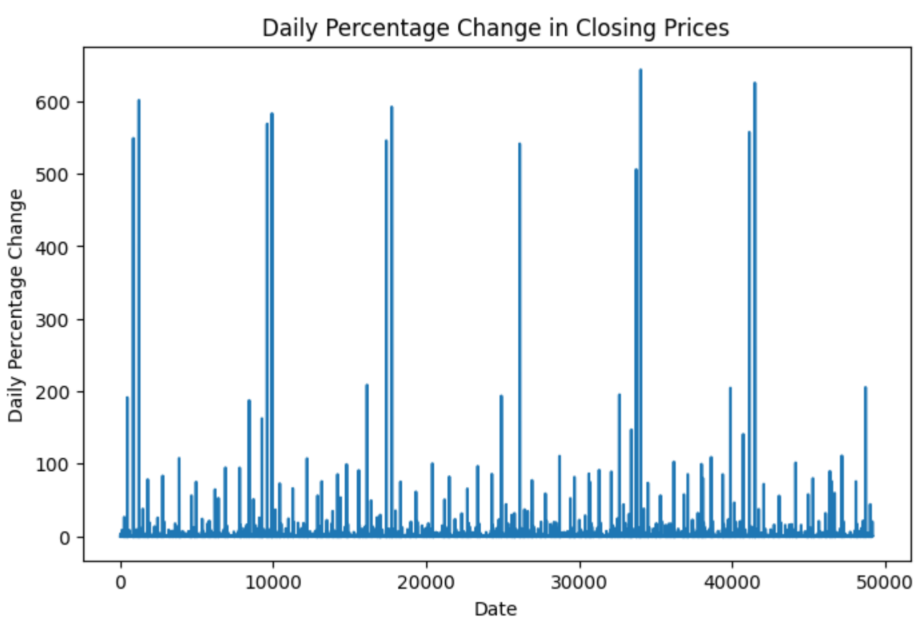
1. Creating a line chart to visualize the 'Close' prices over time.

```
# Line chart of closing stock price over time
plt.figure(figsize = (8, 5))
sns.lineplot(x = stock_data['Date'], y = stock_data['Close'])
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title('Closing Prices Over Time')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

plt.show()
```



2. Calculating and plotting the daily percentage change in closing prices.

```
# Calculating daily percentage change
daily_pct_change = stock_data['Close'].pct_change()

# Plotting daily percentage change
plt.figure(figsize = (8, 5))
plt.plot(daily_pct_change.index, daily_pct_change.values)
plt.xlabel('Date')
plt.ylabel('Daily Percentage Change')
plt.title('Daily Percentage Change in Closing Prices')
plt.show()
```



3. Investigating the presence of any trends or seasonality in the stock prices.

Time Series Components

All time series can be divided into three components:

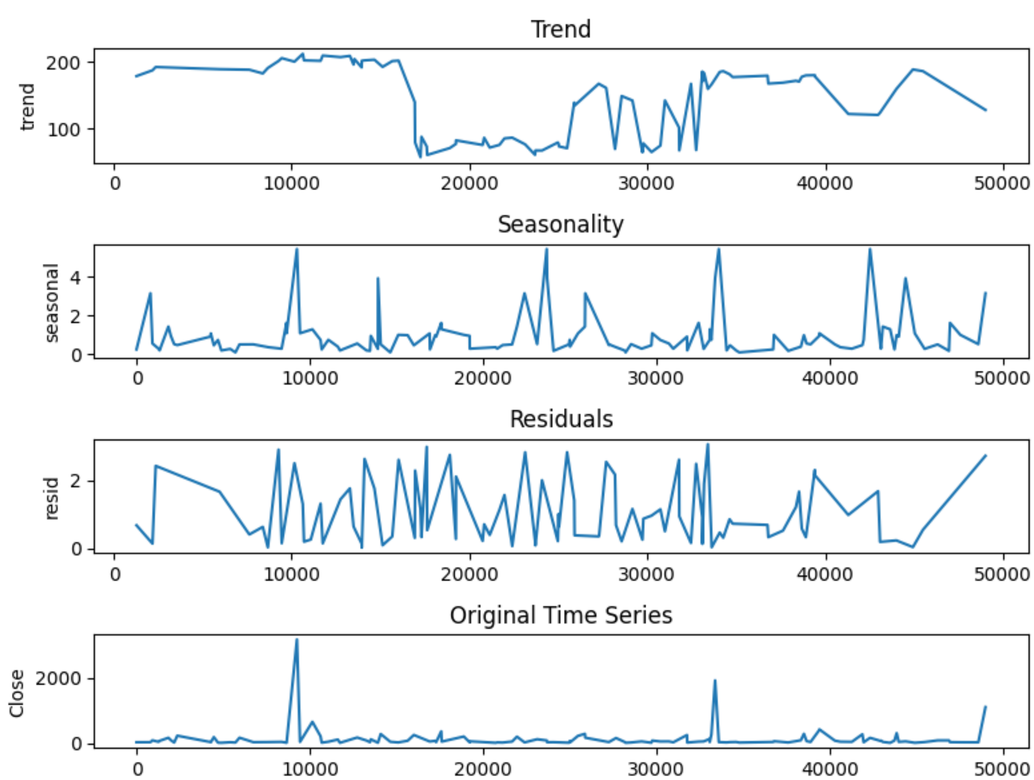
Trend: Slow-moving changes that occur in time series.

Seasonality: Patterns of variation that repeat at specific time intervals. These can be weekly, monthly, yearly, etc. Seasonal changes indicate deviations from the trend in specific directions.

Residuals: Unusual events that occur in the data, such as a sudden increase in heart rate for a person during exercise. These cause random errors and are also referred to as “white noise.”

The visualization of the above components is called “**decomposition.**”

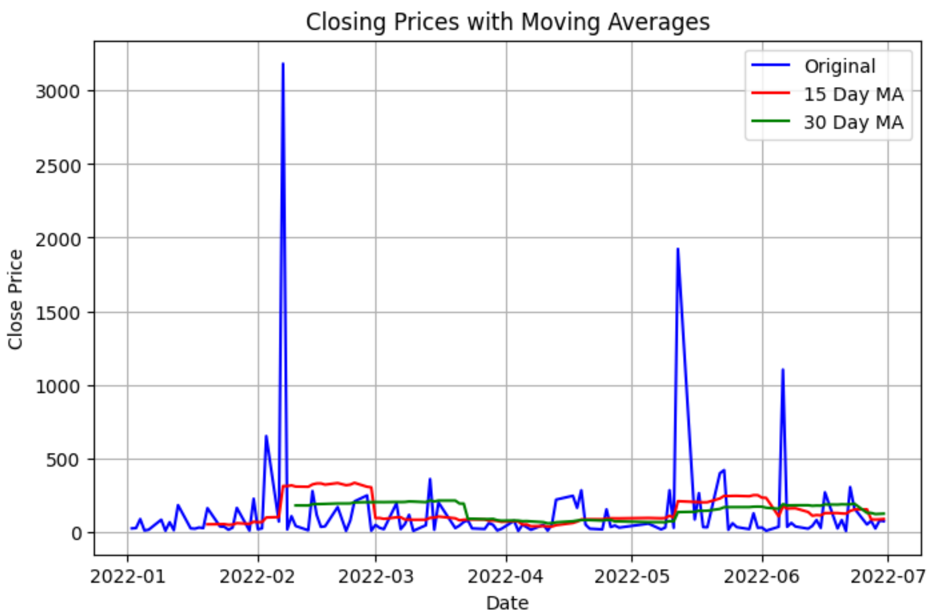
```
# Sorting DataFrame by Date
stock_data = stock_data.sort_values(by = 'Date')

# Droping duplicate rows based on the 'Date' column
stock_data = stock_data.drop_duplicates(subset = 'Date', keep = 'first')

# Setting 'Date' as the index
stock_data.set_index('Date')

# Decomposing the time series
result = seasonal_decompose(stock_data['Close'], model = 'multiplicative', period = 30)

# Plotting the decomposed components
plt.figure(figsize = (8, 6))

plt.subplot(4, 1, 1)
sns.lineplot(x = result.trend.index, y = result.trend)
plt.title('Trend')

plt.subplot(4, 1, 2)
sns.lineplot(x = result.seasonal.index, y = result.seasonal)
plt.title('Seasonality')

plt.subplot(4, 1, 3)
sns.lineplot(x = result.resid.index, y = result.resid)
plt.title('Residuals')

plt.subplot(4, 1, 4)
sns.lineplot(x = stock_data.index, y = stock_data['Close'])
plt.title('Original Time Series')

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



.

4. Applying moving averages to smooth the time series data in 15/30 day intervals against the original graph.

```
# Applying moving averages
stock_data['15_Day_MA'] = stock_data['Close'].rolling(window = 15).mean()
stock_data['30_Day_MA'] = stock_data['Close'].rolling(window = 30).mean()

# Plotting
plt.figure(figsize = (8, 5))
plt.plot(stock_data['Date'], stock_data['Close'], label = 'Original', color = 'blue')
plt.plot(stock_data['Date'], stock_data['15_Day_MA'], label = '15 Day MA', color = 'red')
plt.plot(stock_data['Date'], stock_data['30_Day_MA'], label = '30 Day MA', color = 'green')
plt.title('Closing Prices with Moving Averages')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.grid()
plt.show()
```



5. Calculating the average closing price for each stock.

```
average_closing_price = stock_data.groupby('Name')['Close'].mean().sort_values(ascending = False)
```

6. Identifying the top 5 and bottom 5 stocks based on average closing price.

```
top_5_stocks = average_closing_price.head(5)
bottom_5_stocks = average_closing_price.tail(5)

print("Top 5 Stocks based on Average Closing Price:")
print(top_5_stocks)

print("\nBottom 5 Stocks based on Average Closing Price:")
print(bottom_5_stocks)
```

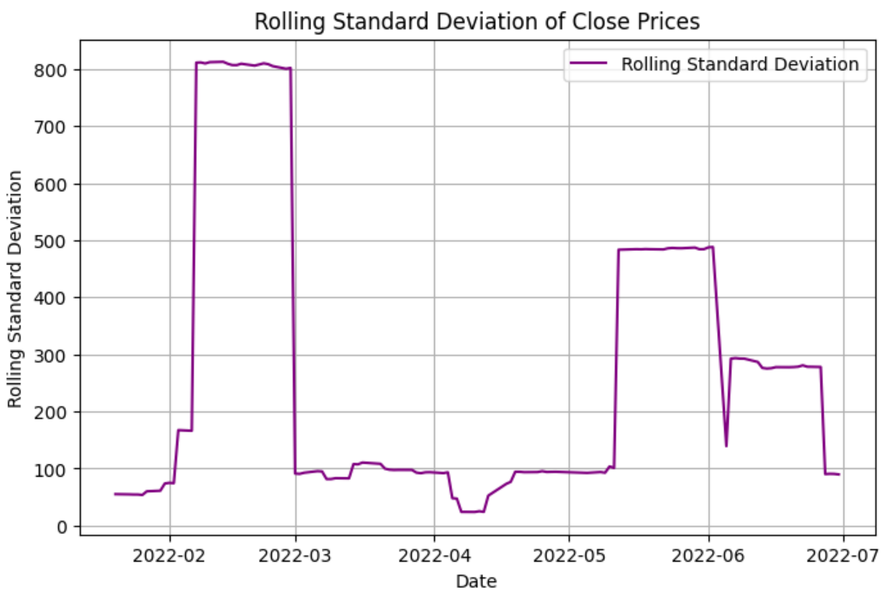
```
Top 5 Stocks based on Average Closing Price:
Name
20.Bond       2553.015
WALTONHIL     1102.600
BATBC          650.800
SONALIANSH     408.700
ANWARGALV      358.600
Name: Close, dtype: float64

Bottom 5 Stocks based on Average Closing Price:
Name
KEYACOSMET    7.0
LRGLOBMF1     6.9
ILFSL         6.1
```

```
FIRSTFIN      5.9
FBFIF         5.1
Name: Close, dtype: float64
```

Part 3: Volatility Analysis:

1. Calculating and plotting the rolling standard deviation of the 'Close' prices.

```
# Calculating the rolling standard deviation of the 'Close' prices.
stock_data['Rolling_Std'] = stock_data['Close'].rolling(window = 15).std()

# Plotting the rolling standard deviation of the 'Close' prices.
plt.figure(figsize = (8, 5))
plt.plot(stock_data['Date'], stock_data['Rolling_Std'], label = 'Rolling Standard Deviation', color = 'purple')
plt.xlabel('Date')
plt.ylabel('Rolling Standard Deviation')
plt.title('Rolling Standard Deviation of Close Prices')
plt.legend()
plt.grid()
plt.show()
```



2. Creating a new column for daily price change (Close - Open)

```
stock_data['Daily_Price_Change'] = stock_data['Close'] - stock_data['Open']
```

.

.

.

.

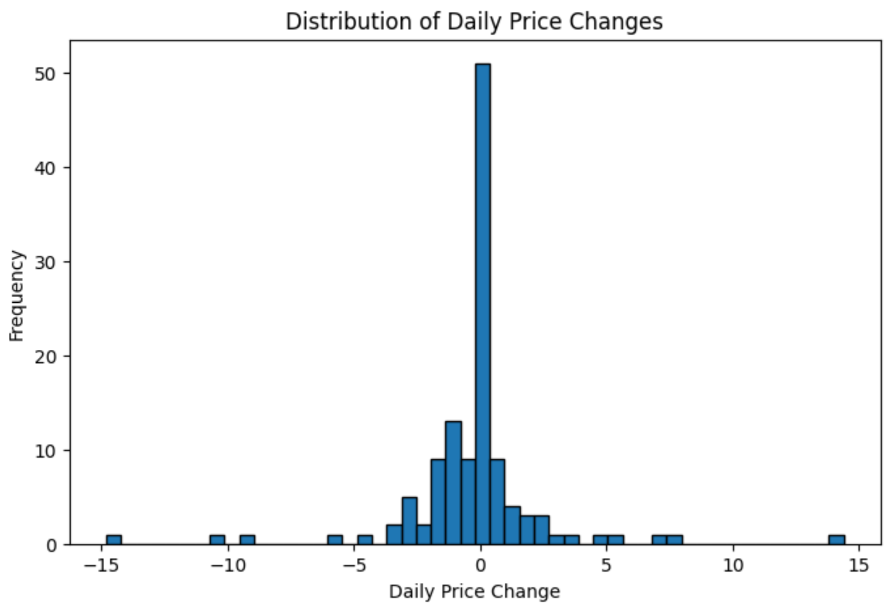
.

.

.

3. Analyzing the distribution of daily price changes.

```
plt.figure(figsize = (8, 5))
plt.hist(stock_data['Daily_Price_Change'].dropna(), bins = 50, edgecolor = 'black')
plt.xlabel('Daily Price Change')
plt.ylabel('Frequency')
plt.title('Distribution of Daily Price Changes')
plt.show()
```



4. Identifying days with the largest price increases and decreases.

```
largest_increase_day = stock_data.loc[stock_data['Daily_Price_Change'].idxmax()]
largest_decrease_day = stock_data.loc[stock_data['Daily_Price_Change'].idxmin()]

print("Days with the Largest Price Increases:")
print(largest_increase_day)

print("\nDays with the Largest Price Decreases:")
print(largest_decrease_day)
```

```
Days with the Largest Price Increases:
Date                  2022-01-31 00:00:00
Name                               CVOPRL
Open                                210.0
High                                229.7
Low                                 208.1
Close                               224.4
Volume                           495092.0
15_Day_MA                       68.093333
30_Day_MA                             NaN
Rolling_Std                      73.65955
Daily_Price_Change                   14.4
Name: 2354, dtype: object

Days with the Largest Price Decreases:
Date                  2022-05-22 00:00:00
Name                           SONALIANSH
Open                                413.0
High                                413.0
Low                                 397.5
Close                               398.2
Volume                             7786.0
15_Day_MA                      226.287333
30_Day_MA                      155.471667
Rolling_Std                    483.966023
Daily_Price_Change                  -14.8
Name: 39415, dtype: object
```

.

5. Identifying stocks with unusually high trading volume on certain days.

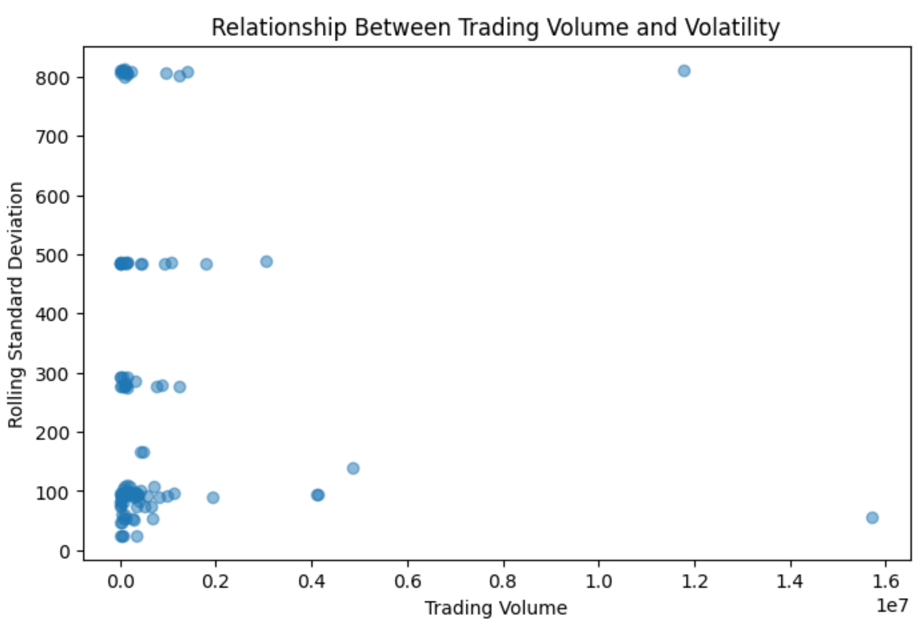
```
threshold = stock_data['Volume'].mean() + 3 * stock_data['Volume'].std()
unusual_high_volume_days = stock_data[stock_data['Volume'] > threshold]

print("Days with Unusually High Trading Volume:")
print(unusual_high_volume_days)
```

```
Days with Unusually High Trading Volume:
            Date     Name   Open   High    Low  Close      Volume   15_Day_MA  \
1984  2022-01-04      BSC   79.0   86.5   78.1   86.5   8294010.0         NaN
1842  2022-01-20  BEXIMCO  153.2  161.9  153.2  160.4  15724293.0   50.702000
13941 2022-02-08      OAL   14.4   15.5   14.2   15.5  11769388.0  310.046667

       30_Day_MA  Rolling_Std  Daily_Price_Change
1984         NaN          NaN                 7.5
1842         NaN    55.032401                 7.2
13941        NaN   811.684135                 1.1
```

Part 4: Correlation and Heatmaps:

1. Exploring the relationship between trading volume and volatility.

```
plt.figure(figsize = (8, 5))
plt.scatter(stock_data['Volume'], stock_data['Rolling_Std'], alpha=0.5)
plt.xlabel('Trading Volume')
plt.ylabel('Rolling Standard Deviation')
plt.title('Relationship Between Trading Volume and Volatility')
plt.show()
```



.

.

.

2. Calculating the correlation matrix between the 'Open' & 'High', 'Low' &'Close' prices.

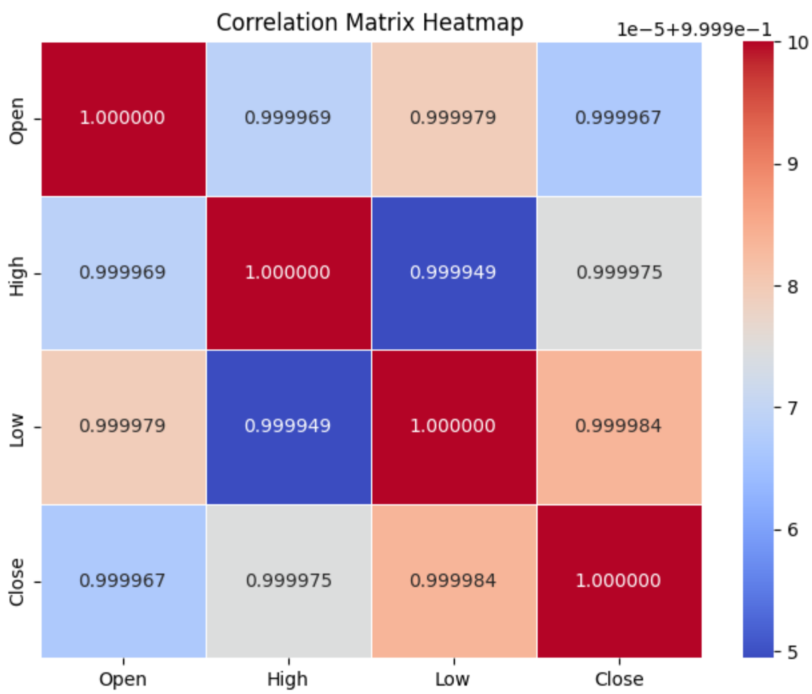
```
correlation_matrix = stock_data[['Open', 'High', 'Low', 'Close']].corr()

print("Correlation Matrix:")
correlation_matrix
```

```
Correlation Matrix:
```

	Open	High	Low	Close
Open	1.000000	0.999969	0.999979	0.999967
High	0.999969	1.000000	0.999949	0.999975
Low	0.999979	0.999949	1.000000	0.999984
Close	0.999967	0.999975	0.999984	1.000000

3. Creating a heatmap to visualize the correlations using the seaborn package.

```
plt.figure(figsize = (8, 6))
sns.heatmap(correlation_matrix, annot = True, cmap = 'coolwarm', fmt = '.6f', linewidths = .5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Bonus Task:

SettingWithCopyWarning, is a pandas warning that occurs when we try to modify a DataFrame that is a subset of another DataFrame (a view) rather than a copy of the original data. If I use,

specific_data = stock_data[stock_data['Name'] == specific_company].copy()

instead of,

specific_data = stock_data[stock_data['Name'] == specific_company]

then the warning will be fix.

Rolling Window Analysis / Moving Averages

In [10]:
```
specific_company = 'RECKITTBEN'

specific_data = stock_data[stock_data['Name'] == specific_company].copy()
specific_data['7_Day_Rolling_Avg'] = specific_data['Close'].rolling(window = 7).mean()

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

plt.plot(specific_data['Date'], specific_data['Close'], label = f'{specific_company} Closing Price', color = 'blue')

plt.plot(specific_data['Date'], specific_data['7_Day_Rolling_Avg'], label = f'{specific_company} 7 Day Rolling Average of Closing


plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title(f'{specific_company} 7 Day Rolling Average of Closing Price')
plt.grid()
plt.legend()

plt.xticks(rotation = 45)
plt.show()
```