INTRODUCTION

In this project, we aim to explore **stock price prediction within the BFSI sector**, specifically targeting banks, NBFCs, and insurance companies, by leveraging both historical market data and sector-specific indicators. The BFSI industry, encompassing diverse financial service providers—including retail and corporate banks, non-banking financial companies, mutual funds, and insurance firms—serves as a critical barometer of macroeconomic health due to its close ties with credit cycles, regulatory policies, and capital markets.

Given the sector's sensitivity to monetary policy actions, such as interest-rate changes, as well as its exposure to credit growth patterns and asset quality concerns, predicting BFSI stock movements requires careful integration of both traditional technical indicators and qualitative sector-driven variables. For instance, the BFSI sector in India recently led a strong market rebound, with financial firms outpacing benchmark indices due to expected rate cuts, robust credit demand, and renewed foreign inflows. In modelling stock behaviour, time-series and machine-learning approaches—such as ARIMA, Random Forest, and advanced deep-learning models like LSTM and GRU—have demonstrated efficacy, particularly when augmented with domain-relevant BFSI factors.

Models that incorporate dynamic volatility modelling techniques, such as ARIMA for forecasting banking sector volatility, further enhance predictive accuracy. By combining such quantitative tools with BFSI-specific variables—like credit growth trends, asset quality flags (e.g., NPA ratios), regulatory events, and fintech adoption signals—this project aims to construct a nuanced forecasting framework tailored to the unique dynamics of financial sector stocks.

The variables used in this project are as follows:

- **Stock Name:** Provides full name of the stock.
- **Stock Industry Name:** Provides the exact sub-industry in which the company operates.
- **Stock Code Name:** Provides the short forms of the stocks given by NSE & BSE.
- **Publication Date:** Provides the year in which the stock was published.
- **Business Model Type:** Provides the business model which company uses in order to run their business.
- **Open Price:** Shows the opening price of the stock.
- **Closing Price:** Shows the closing price of the stock.

- **PE Ratio:** The Price-to-Earnings (P/E) Ratio is a widely-used financial metric for assessing a stock's valuation.
- **Trading Volumes:** The volume or the number of stocks of the company is traded in the stock market.
- **RSI:** The Relative Strength Index (RSI) is a popular momentum oscillator used extensively in technical analysis to assess the speed and magnitude of recent price movements.
- **Volatility:** Volatility refers to the degree of fluctuation in the price of a financial asset over time—a statistical measure often used to gauge market risk and uncertainty.
- **Sentiment Score Index:** The Sentiment Score Index is a composite measure that quantifies the overall tone or emotional quality of textual data—such as financial news, social media posts, earnings call transcripts, or analyst reports—relative to financial sentiment.

Index

1. Loading dataset

This section is responsible for loading the dataset into a DataFrame and displaying
the first few rows to understand the structure of the data. Then info () and describe () function is used to retrieve the information and statistical data of the
dataset.

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

df=pd.read_csv('/content/3404_Ansh Barot_StockPrediction_Dataset_PA Project.csv')
df.head()
```

	Stock Name	Stock Industry Name	Stock Code Name	Publication Date	Business Model Type	Open Price	Close Price	PE Ratio	Trading Volume (In Millions)	Relative Strength Index (In Scale of 0-100)	Volatility (In Rupees)	Sentiment Score Index
0	HDFC Bank	Bank	HDFCBANK	1905	Private Universal Bank	1998.01	1983.55	20.71	6.26	36.02	8.02	Bear
1	ICICI Bank	Bank	ICICIBANK	1998	Private Universal Bank	1432	1421.9	18.66	7.16	46.16	19.36	Bear
2	SBI Bank	Bank	SBIN	1997	Public Sector Universal Bank	807.9	808.65	10.17	6.20	48.22	4.04	Bear
3	Axis Bank	Bank	AXISBANK	1998	Private Universal Bank	1202	1173.8	12.95	4.47	51.87	19.50	Bear
4	Kotak Mahindra Bank	Bank	KOTAKBANK	2003	Private Universal Bank	2201.8	2220.6	20.11	2.68	54.06	12.49	Bear

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 12 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   Stock Name                                   100 non-null    object
 1   Stock Industry Name                          100 non-null    object
 2   Stock Code Name                              100 non-null    object
 3   Publication Date                             100 non-null    int64
 4   Business Model Type                          100 non-null    object
 5   Open Price                                   100 non-null    object
 6   Close Price                                  100 non-null    object
 7   PE Ratio                                     100 non-null    object
 8   Trading Volume (In Millions)                 100 non-null    float64
 9   Relative Strength Index (In Scale of 0-100)  100 non-null    float64
 10  Volatility (In Rupees)                       100 non-null    float64
 11  Sentiment Score Index                        100 non-null    object
dtypes: float64(3), int64(1), object(8)
memory usage: 9.5+ KB
```

```
df.describe()
```

	Publication Date	Trading Volume (In Millions)	Relative Strength Index (In Scale of 0-100)	Volatility (In Rupees)
count	100.00000	100.000000	100.000000	100.000000
mean	1999.58000	2.262430	47.465900	8.966700
std	29.12397	3.756626	6.519712	17.762471
min	1894.00000	0.010000	31.700000	0.550000
25%	1995.00000	0.187500	44.000000	2.800000
50%	2009.00000	0.620000	47.000000	5.000000
75%	2018.00000	2.506750	50.925000	8.140000
max	2023.00000	23.100000	70.000000	163.800000

2. Converting datatype of Object to Float.

As we can see that the columns opening, closing and PE ratio are detected as object so we will convert their datatype into float but using the predefined method as pd.to_numeric which is in pandas which will help to convert the data types of them into float.

```
[4] df['Open Price'] = pd.to_numeric(df['Open Price'], errors='coerce')
    df['Close Price'] = pd.to_numeric(df['Close Price'], errors='coerce')
    df['PE Ratio'] = pd.to_numeric(df['PE Ratio'], errors='coerce')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 12 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   Stock Name                                100 non-null    object
 1   Stock Industry Name                       100 non-null    object
 2   Stock Code Name                           100 non-null    object
 3   Publication Date                          100 non-null    int64
 4   Business Model Type                       100 non-null    object
 5   Open Price                                94 non-null     float64
 6   Close Price                               94 non-null     float64
 7   PE Ratio                                  98 non-null     float64
 8   Trading Volume (In Millions)              100 non-null    float64
 9   Relative Strength Index (In Scale of 0-100) 100 non-null  float64
 10  Volatility (In Rupees)                    100 non-null    float64
 11  Sentiment Score Index                     100 non-null    object
dtypes: float64(6), int64(1), object(5)
memory usage: 9.5+ KB
```

3. Non-graphical univariate analysis.

The following shows the univariate non graphical analysis of the columns which are present in these projects. Here we have chosen the column trading volumes so as to get the result of the non-graphical univariate analysis.

We are doing mean, median, mode, standard deviation, variance, minimum, maximum, range, skewness and kurtosis.

```
[15] # Calculate the descriptive statistics
     trading_vol_mean = df['Trading Volume (In Millions)'].mean()
     trading_vol_median = df['Trading Volume (In Millions)'].median()
     trading_vol_mode = df['Trading Volume (In Millions)'].mode()[0]
     trading_vol_std = df['Trading Volume (In Millions)'].std()
     trading_vol_var = df['Trading Volume (In Millions)'].var()
     trading_vol_min = df['Trading Volume (In Millions)'].min()
     trading_vol_max = df['Trading Volume (In Millions)'].max()
     trading_vol_range = trading_vol_max - trading_vol_min
     trading_vol_skew = df['Trading Volume (In Millions)'].skew()
     trading_vol_kurt = df['Trading Volume (In Millions)'].kurtosis()
```

Now putting these columns into a data frame in order to get the output in a well-structured format.

```
[16] # Create a DataFrame for the numerical analysis results
     numerical_analysis = pd.DataFrame({
         'Metric': ['Mean', 'Median', 'Mode', 'Standard Deviation', 'Variance', 'Range', 'Minimum', 'Maximum', 'Skewness', 'Kurtosis'],
         'Value': [trading_vol_mean, trading_vol_median, trading_vol_mode, trading_vol_std, trading_vol_var, trading_vol_range,
                   trading_vol_min, trading_vol_max,trading_vol_skew, trading_vol_kurt]
     })
```

Now printing the data frame in order to get the result.

```
# Print the numerical analysis results.
print("Univariate Analysis for 'Trading Volume (In Millions)' (Numerical):")
print(numerical_analysis.to_markdown(index=False, numalign="left", stralign="left"))
```

```
Univariate Analysis for 'Trading Volume (In Millions)' (Numerical):
| Metric             | Value   |
|:-------------------|:--------|
| Mean               | 2.26243 |
| Median             | 0.62    |
| Mode               | 0.02    |
| Standard Deviation | 3.75663 |
| Variance           | 14.1122 |
| Range              | 23.09   |
| Minimum            | 0.01    |
| Maximum            | 23.1    |
| Skewness           | 3.01784 |
| Kurtosis           | 11.366  |
```

The following result shows the number of stocks published per decade.

```
# Convert 'Publication Date' to a new 'Decade' column
df['Decade'] = (df['Publication Date'] // 10 * 10).astype(str) + 's'

# Count the number of stocks published in each decade
decade_counts = df['Decade'].value_counts().sort_index()

# Convert the Series to a DataFrame for better display
decade_counts_df = decade_counts.reset_index()
decade_counts_df.columns = ['Decade', 'Number of Stocks']

# Print the final result in a clear, formatted table
print(decade_counts_df.to_markdown(index=False, numalign="left", stralign="left"))
```

```
| Decade   | Number of Stocks   |
|:---------|:-------------------|
| 1890s    | 1                  |
| 1900s    | 3                  |
| 1920s    | 2                  |
| 1930s    | 3                  |
| 1960s    | 3                  |
| 1980s    | 2                  |
| 1990s    | 19                 |
| 2000s    | 18                 |
| 2010s    | 37                 |
| 2020s    | 12                 |
```

Now that we have done the univariate non-graphical analysis on the numerical column now its time to do it on the categorical column.

Here we are taking the Sentiment Score Index column in order to show the analysis of the **'Sentiment Score Index'** column showing the distribution of sentiments across the dataset. The most frequent sentiment is **Neutral**, accounting for 40% of the data, followed by **Bear** at 33% and **Bull** at 25%.

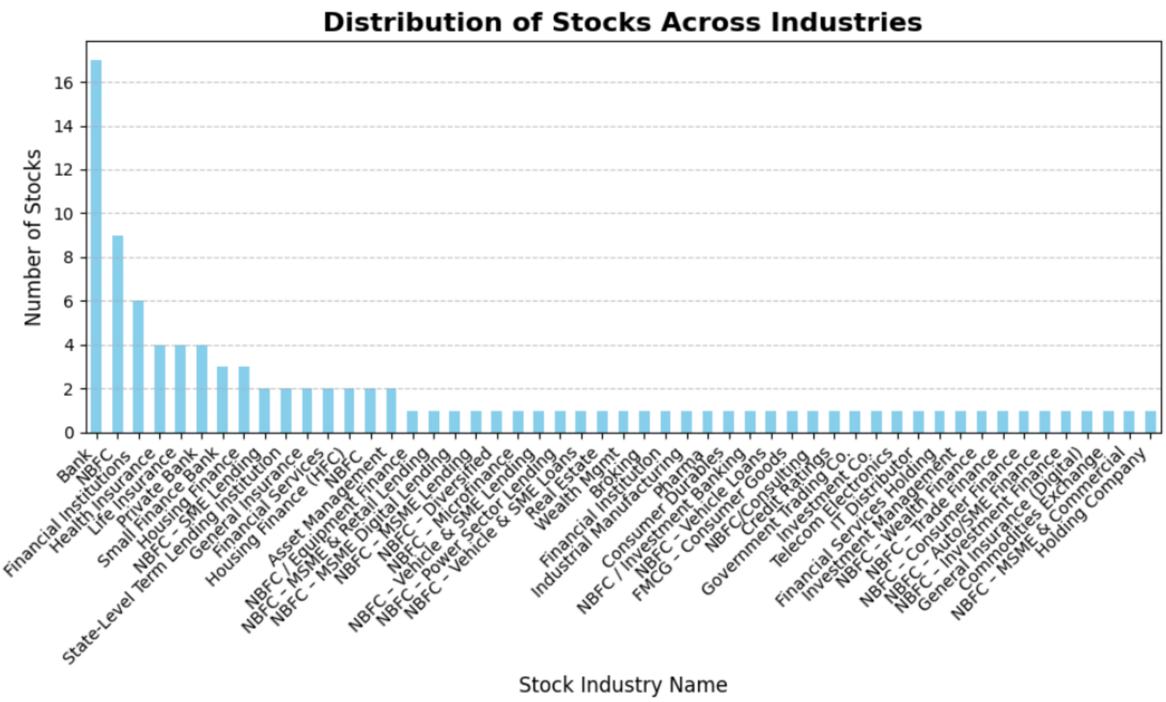
```
# --- Categorical Analysis for 'Sentiment Score Index' ---

# Get the frequency counts of each sentiment score.
sentiment_counts = df['Sentiment Score Index'].value_counts().reset_index()
sentiment_counts.columns = ['Sentiment', 'Frequency']

# Calculate the proportion (percentage) of each sentiment.
sentiment_counts['Percentage'] = (sentiment_counts['Frequency'] / len(df)) * 100

# Print the categorical analysis results.
print("\nUnivariate Analysis for 'Sentiment Score Index' (Categorical):")
print(sentiment_counts.to_markdown(index=False, numalign="left", stralign="left"))
```

```
Univariate Analysis for 'Sentiment Score Index' (Categorical):
| Sentiment   | Frequency   | Percentage   |
|:------------|:------------|:-------------|
| Neutral     | 41          | 41           |
| Bear        | 33          | 33           |
| Bull        | 26          | 26           |
```

4. Graphical univariate analysis

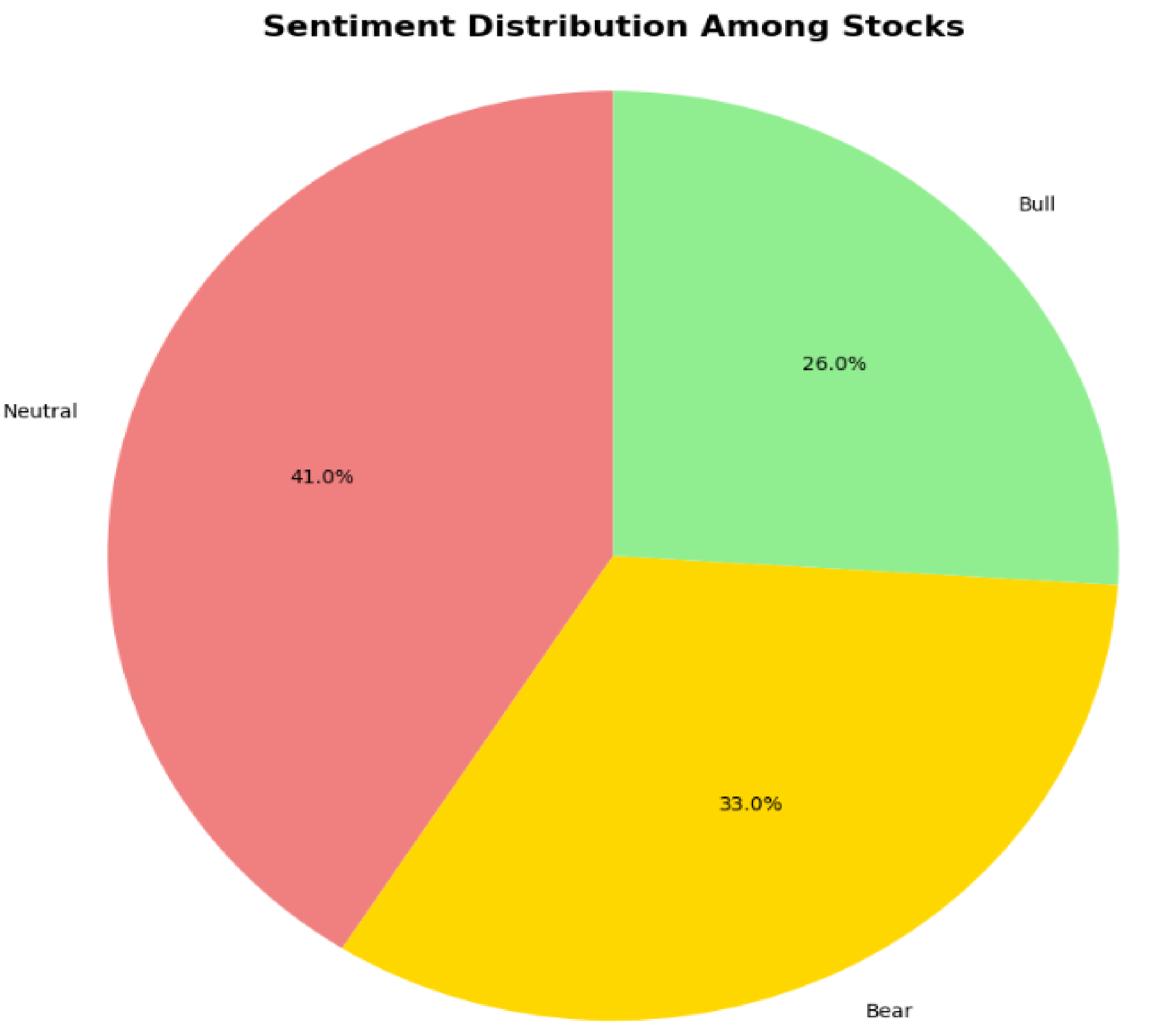
Graphical univariate analysis is essential for understanding the fundamental characteristics of a single variable. It provides an immediate and intuitive visual summary of the data's distribution, allowing you to quickly see its shape, spread, and central tendency.

```python
# --- Plot 1: Distribution of stocks across different industries ---
plt.figure(figsize=(10, 6))
industry_counts = df['Stock Industry Name'].value_counts()
industry_counts.plot(kind='bar', color='skyblue')
plt.title('Distribution of Stocks Across Industries', fontsize=16, fontweight='bold')
plt.ylabel('Number of Stocks', fontsize=12)
plt.xlabel('Stock Industry Name', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig('industry_distribution.png')
plt.close()
```



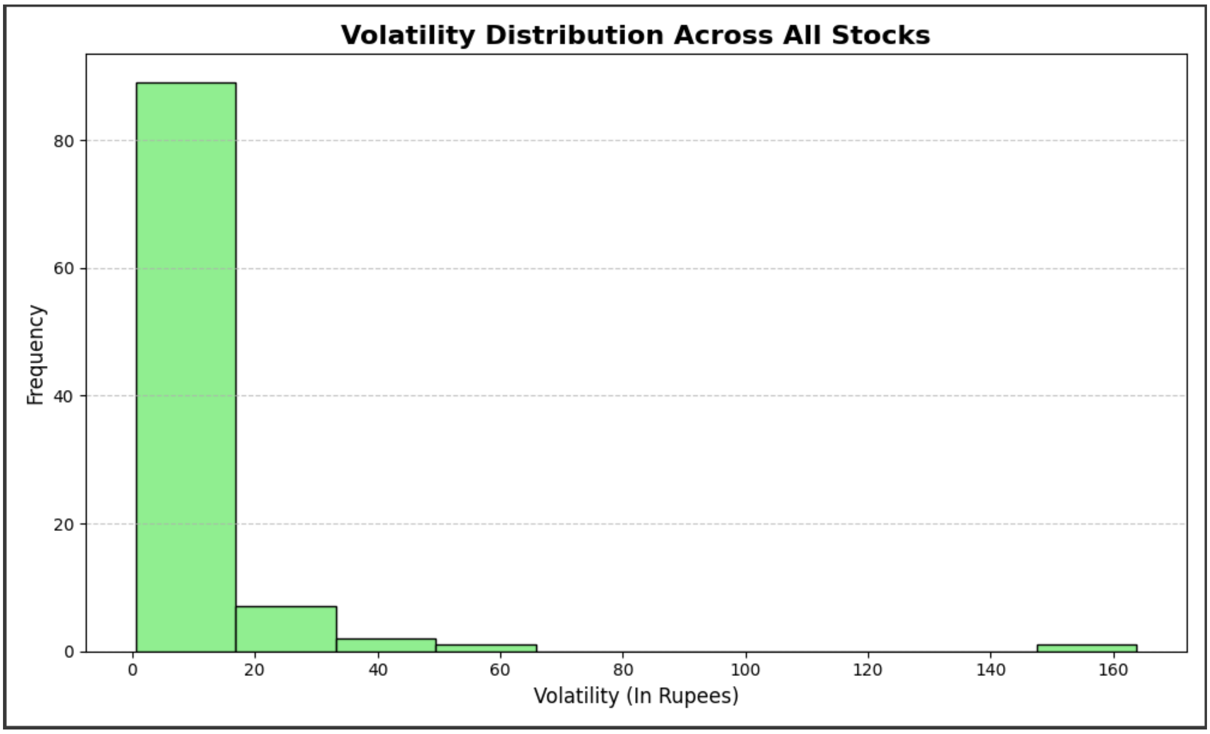
The bar graph shows the number of stocks present in the industries of the BFSI sector. As we can see the highest count is of banking sector and the lowest count are of the different small sub-industries which fall under the bank, NBFC , insurance.

```python
# --- Plot 2: Distribution of sentiment (Bear, Bull, Neutral) ---
plt.figure(figsize=(8, 8))
# Clean the data by correcting spelling errors
df['Sentiment Score Index'] = df['Sentiment Score Index'].replace({'Neurtal': 'Neutral', 'Bulli': 'Bull'})
sentiment_counts = df['Sentiment Score Index'].value_counts()
plt.pie(sentiment_counts, labels=sentiment_counts.index, autopct='%1.1f%%', startangle=90, colors=['lightcoral', 'gold', 'lightgreen'])
plt.title('Sentiment Distribution Among Stocks', fontsize=16, fontweight='bold')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.tight_layout()
plt.savefig('sentiment_distribution.png')
plt.close()
```



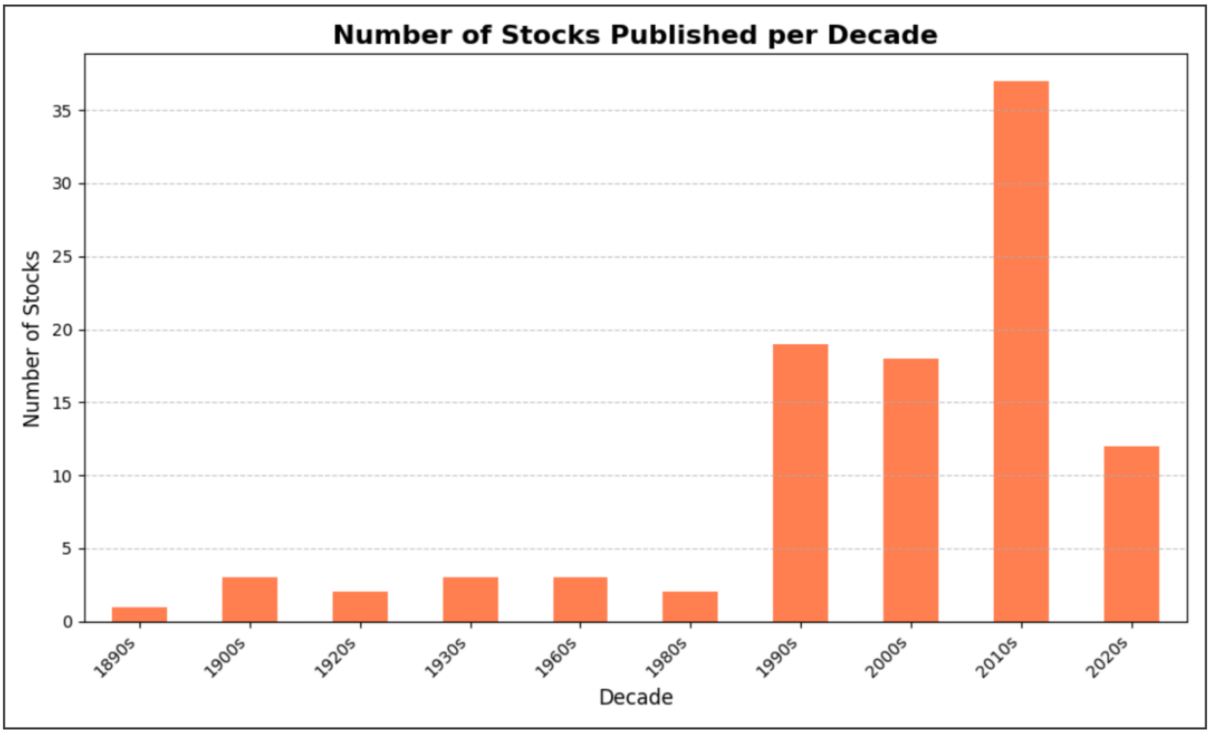
The pie chart shows the distribution of the sentiment which are Neutral, Bear or Bull.

```
# --- Plot 3: Volatility distribution across all stocks ---
plt.figure(figsize=(10, 6))
plt.hist(df['Volatility (In Rupees)'], bins=10, color='lightgreen', edgecolor='black')
plt.title('Volatility Distribution Across All Stocks', fontsize=16, fontweight='bold')
plt.xlabel('Volatility (In Rupees)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig('volatility_distribution.png')
plt.close()
```



The graph is a histogram that displays the distribution of volatility for all stocks in the dataset. The x-axis shows different ranges of volatility in rupees, while the y-axis indicates the frequency or number of stocks that fall within each range. This visualization helps in understanding the most common volatility levels and how the volatility is spread across the entire dataset.

```
# --- Plot 4: Number of stocks published in each decade ---
plt.figure(figsize=(10, 6))
df['Publication Date'] = pd.to_numeric(df['Publication Date'], errors='coerce')
df['Decade'] = (df['Publication Date'] // 10 * 10).astype(int).astype(str) + 's'
decade_counts = df['Decade'].value_counts().sort_index()
decade_counts.plot(kind='bar', color='coral')
plt.title('Number of Stocks Published per Decade', fontsize=16, fontweight='bold')
plt.xlabel('Decade', fontsize=12)
plt.ylabel('Number of Stocks', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig('decade_distribution.png')
plt.close()
```



5. Non-graphical bivariate analysis

```python
# The 'PE Ratio' column is an object, so we need to clean and convert it to a numeric type.
df['PE Ratio'] = df['PE Ratio'].astype(str).str.replace(',', '', regex=False).str.replace('−', '-', regex=False)
df['PE Ratio'] = pd.to_numeric(df['PE Ratio'], errors='coerce')

# Drop rows where 'PE Ratio' is NaN
df_cleaned = df.dropna(subset=['PE Ratio'])

# Group the data by 'Stock Industry Name' and calculate the average 'PE Ratio' for each group.
average_pe_ratio = df_cleaned.groupby('Stock Industry Name')['PE Ratio'].mean().reset_index()

# Rename the columns for clarity
average_pe_ratio.columns = ['Stock Industry Name', 'Average PE Ratio']

# Print the result in a markdown table format.
print(average_pe_ratio.to_markdown(index=False, numalign="left", stralign="left"))
```

The results show the average P/E ratio for each stock industry in your dataset. This ratio is a key metric used to determine if a stock is over or undervalued relative to its earnings. A higher average P/E ratio in an industry, such as Life Insurance, suggests investors have higher growth expectations for that sector. Conversely, a lower average P/E ratio, as seen in State-Level Term Lending Institutions, may indicate lower growth prospects or a more conservative market valuation for that industry.

Stock Industry Name	Average PE Ratio
Asset Management	23.25
Bank	15.1659
Broking	30.5
Commodities Exchange	18
Consumer Durables	25
Credit Ratings	17.5
FMCG – Consumer Goods	25
Financial Institution	28
Financial Institutions	22.25
Financial Services	14.1
Financial Services Holding	12
General Insurance	24.46
General Insurance (Digital)	40
Government Trading Co.	9
Health Insurance	44.895
Holding Company	16
Housing Finance	17.8333
Housing Finance (HFC)	20.25
IT Distributor	16
Industrial Manufacturing	22
Investment Co.	12.5
Investment Management	18
Life Insurance	102.535
NBFC	25.8756
NBFC	8
NBFC / Equipment Finance	22
NBFC / Investment Banking	13.8
NBFC – Auto/SME Finance	30.6
NBFC – Consumer Finance	25
NBFC – Diversified	58.5
NBFC – Investment Finance	30.6
NBFC – MSME & Commercial	22
NBFC – MSME & Retail Lending	35
NBFC – MSME Digital Lending	40
NBFC – MSME Lending	31.1
NBFC – Microfinance	109.8
NBFC – Power Sector Lending	16
NBFC – SME Lending	19
NBFC – Trade Finance	20
NBFC – Vehicle & SME Lending	25
NBFC – Vehicle Loans	18
NBFC – Wealth Finance	14
NBFC/Consulting	25
Pharma	32.5
Private Bank	12.74
Real Estate	18
Small Finance Bank	23.3333
State-Level Term Lending Institution	7
Telecom Electronics	14
Wealth Mgmt	40

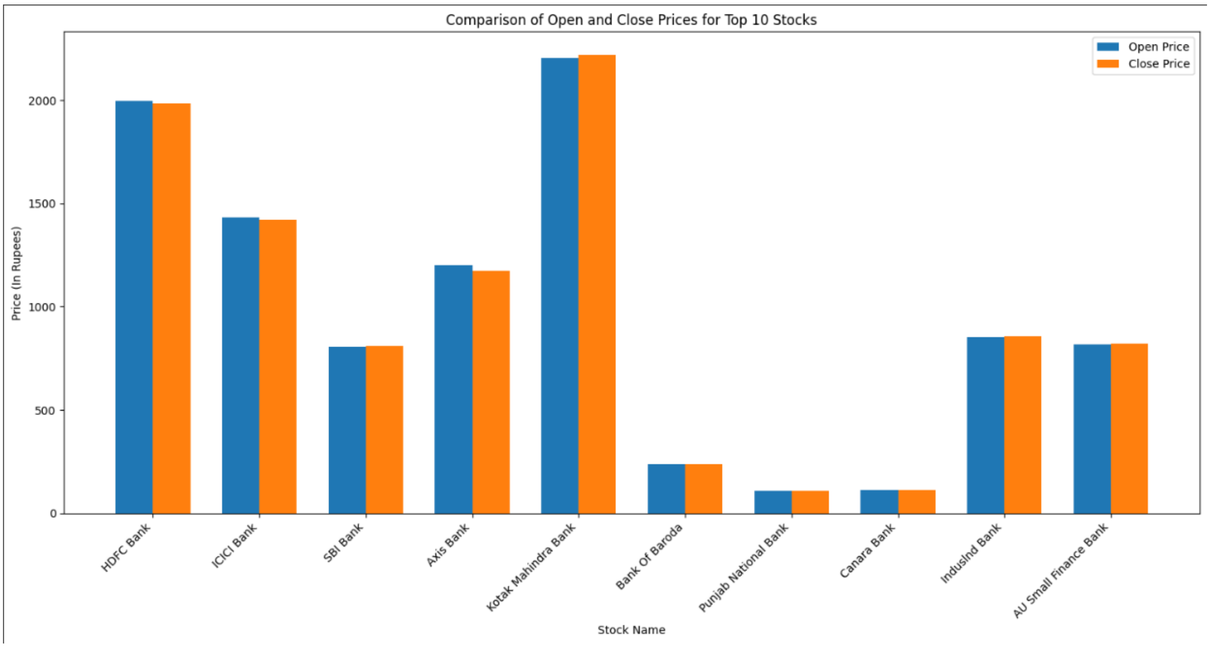
```python
# The column name for RSI is 'Relative Strength Index (In Scale of 0-100)'
# Convert the RSI column to numeric, coercing any errors
df['RSI'] = pd.to_numeric(df['Relative Strength Index (In Scale of 0-100)'], errors='coerce')

# Group the data by 'Stock Industry Name' and calculate the average RSI
average_rsi = df.groupby('Stock Industry Name')['RSI'].mean().reset_index()

# Rename the columns for clarity
average_rsi.columns = ['Stock Industry Name', 'Average RSI']

# Print the result in a markdown table format
print(average_rsi.to_markdown(index=False, numalign="left", stralign="left"))
```

The result will show the average of the RSI of each stock with respect to its industry. The Relative Strength Index (RSI) is a momentum indicator used in technical analysis to measure the speed and change of a stock's price movements. It is an oscillator that moves between 0 and 100. An RSI value of 70 or above is typically interpreted as a stock being overbought, suggesting it may be overvalued and due for a price correction.

Stock Industry Name	Average RSI
Asset Management	52.5
Bank	45.6788
Broking	55
Commodities Exchange	52
Consumer Durables	46
Credit Ratings	55
FMCG – Consumer Goods	46
Financial Institution	50
Financial Institutions	43.9
Financial Services	49.5
Financial Services Holding	47
General Insurance	52.5
General Insurance (Digital)	56
Government Trading Co.	47
Health Insurance	55.5
Holding Company	46
Housing Finance	49
Housing Finance (HFC)	54
IT Distributor	49
Industrial Manufacturing	42
Investment Co.	43
Investment Management	46
Life Insurance	43.985
NBFC	45.4678
NBFC	46.5
NBFC / Equipment Finance	45
NBFC / Investment Banking	52
NBFC – Auto/SME Finance	41.5
NBFC – Consumer Finance	44
NBFC – Diversified	55
NBFC – Investment Finance	42
NBFC – MSME & Commercial	45
NBFC – MSME & Retail Lending	50
NBFC – MSME Digital Lending	48
NBFC – MSME Lending	60
NBFC – Microfinance	58
NBFC – Power Sector Lending	47
NBFC – SME Lending	55.5
NBFC – Trade Finance	45
NBFC – Vehicle & SME Lending	52
NBFC – Vehicle & SME Loans	55
NBFC – Vehicle Loans	45
NBFC – Wealth Finance	48
NBFC/Consulting	46
Pharma	48
Private Bank	48.25
Real Estate	44
Small Finance Bank	42.3333
State-Level Term Lending Institution	34.5
Telecom Electronics	44
Wealth Mgmt	48

6. Graphical bivariate analysis

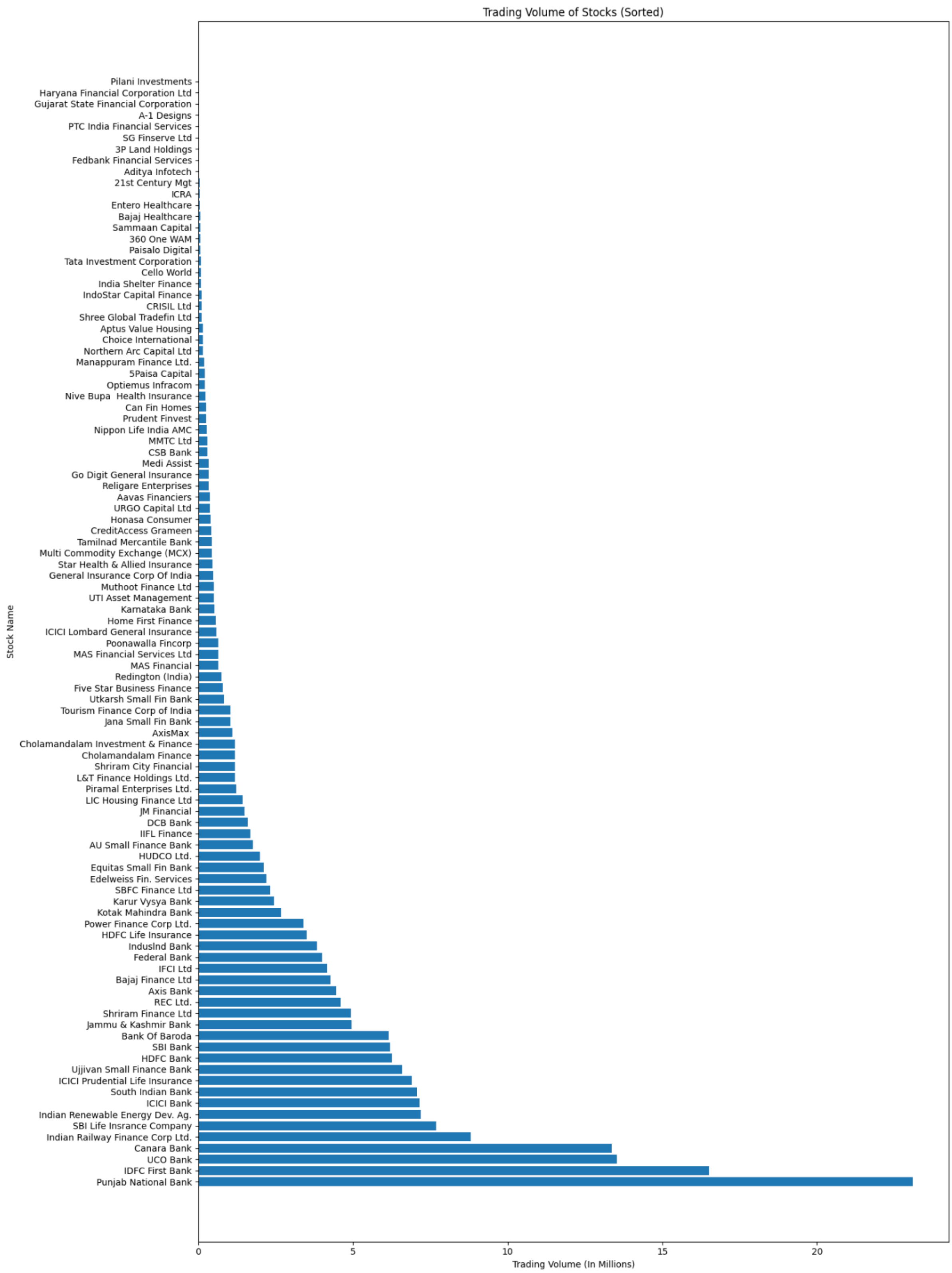
```python
# Select the top 10 stocks for better visualization
df_plot = df.head(10).copy()

# Set up the figure and axes
plt.figure(figsize=(15, 8))
x = np.arange(len(df_plot['Stock Name']))
width = 0.35

# Create the bars for 'Open Price' and 'Close Price'
plt.bar(x - width/2, df_plot['Open Price'], width, label='Open Price')
plt.bar(x + width/2, df_plot['Close Price'], width, label='Close Price')

# Set x-axis labels and ticks
plt.xticks(x, df_plot['Stock Name'], rotation=45, ha="right")

# Add labels and a title
plt.xlabel("Stock Name")
plt.ylabel("Price (In Rupees)")
plt.title("Comparison of Open and Close Prices for Top 10 Stocks")
plt.legend()
plt.tight_layout()

# Save the plot
plt.savefig("open_close_price_comparison.png")

print("\nUpdated DataFrame info:")
df.info()
print("\nFirst few rows of cleaned data:")
print(df_plot[['Stock Name', 'Open Price', 'Close Price']].head())
```



Based on the data analysis, the trend between the open and close prices for the stocks shows a mixed pattern. The side-by-side bar chart, which compares the 'Open Price' and 'Close Price' for the top 10 stocks in the dataset, visually represents this trend. For some stocks, the open price is higher than the close price, indicating a decline in value during the trading period. For others, the close price is higher, showing an increase. Overall, the prices appear to be volatile, with significant changes occurring throughout the day for some companies. This suggests that the stock market is dynamic and that prices can fluctuate in either direction, highlighting the importance of considering multiple factors when analysing stock performance.

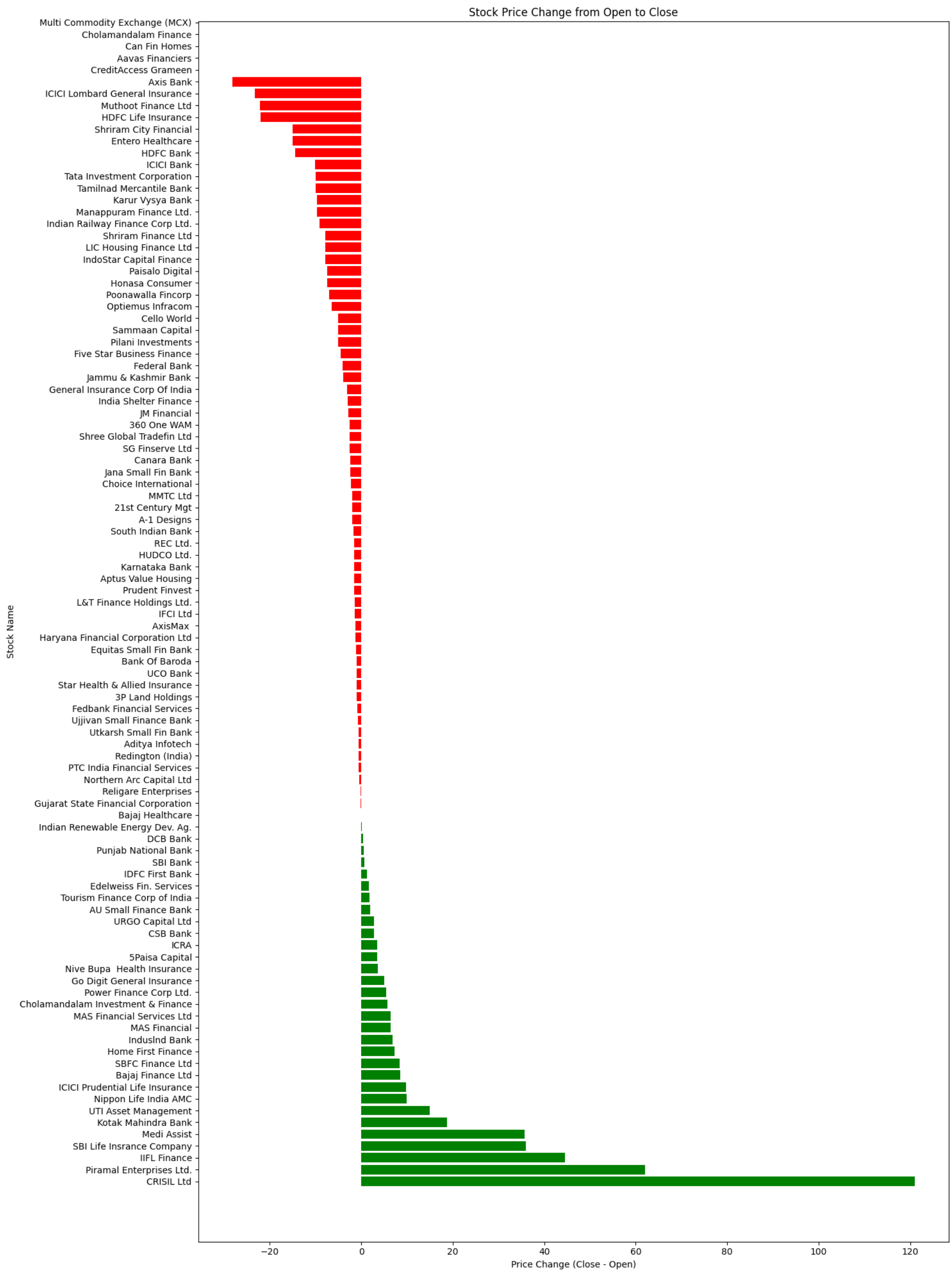
```python
# Sort the DataFrame by 'Trading Volume (In Millions)' in descending order
try:
    df_sorted_volume = df.sort_values(by='Trading Volume (In Millions)', ascending=False)
except KeyError as e:
    print(f"Error: The column {e} was not found in the dataset.")
    exit()

# Create a horizontal bar chart for better readability of stock names
plt.figure(figsize=(15, 20))  # Adjusting figure size for better label visibility
plt.barh(df_sorted_volume['Stock Name'], df_sorted_volume['Trading Volume (In Millions)'])
plt.xlabel("Trading Volume (In Millions)")
plt.ylabel("Stock Name")
plt.title("Trading Volume of Stocks (Sorted)")
plt.tight_layout()

# Save the plot
plt.savefig("trading_volume_horizontal_bar_chart.png")

print("Horizontal bar chart with clearer labels saved.")
```

Upon analysing the dataset, the stock with the highest trading volume is Punjab National Bank, and the one with the lowest is Pilani Investments. To present this information clearly and ensure that the stock names are fully readable, I have created a horizontal bar chart. This chart is sorted by trading volume in descending order, making it easy to see the relative trading activity of each stock. The horizontal layout allows all 100 stock names to be displayed without overlapping, providing a comprehensive and easy-to-read visualization of the trading volume trends.


Trading Volume of Stocks (Sorted)
Stock Name
Trading Volume (In Millions)
0
5
10
15
20
Pilani Investments
Haryana Financial Corporation Ltd
Gujarat State Financial Corporation
A-1 Designs
PTC India Financial Services
SG Finserve Ltd
3P Land Holdings
Fedbank Financial Services
Aditya Infotech
21st Century Mgt
ICRA
Entero Healthcare
Bajaj Healthcare
Sammaan Capital
360 One WAM
Paisalo Digital
Tata Investment Corporation
Cello World
India Shelter Finance
IndoStar Capital Finance
CRISIL Ltd
Shree Global Tradefin Ltd
Aptus Value Housing
Choice International
Northern Arc Capital Ltd
Manappuram Finance Ltd.
5Paisa Capital
Optiemus Infracom
Nive Bupa Health Insurance
Can Fin Homes
Prudent Finvest
Nippon Life India AMC
MMTC Ltd
CSB Bank
Medi Assist
Go Digit General Insurance
Religare Enterprises
Aavas Financiers
URGO Capital Ltd
Honasa Consumer
CreditAccess Grameen
Tamilnad Mercantile Bank
Multi Commodity Exchange (MCX)
Star Health & Allied Insurance
General Insurance Corp Of India
Muthoot Finance Ltd
UTI Asset Management
Karnataka Bank
Home First Finance
ICICI Lombard General Insurance
Poonawalla Fincorp
MAS Financial Services Ltd
MAS Financial
Redington (India)
Five Star Business Finance
Utkarsh Small Fin Bank
Tourism Finance Corp of India
Jana Small Fin Bank
AxisMax
Cholamandalam Investment & Finance
Cholamandalam Finance
Shriram City Financial
L&T Finance Holdings Ltd.
Piramal Enterprises Ltd.
LIC Housing Finance Ltd
JM Financial
DCB Bank
IIFL Finance
AU Small Finance Bank
HUDCO Ltd.
Equitas Small Fin Bank
Edelweiss Fin. Services
SBFC Finance Ltd
Karur Vysya Bank
Kotak Mahindra Bank
Power Finance Corp Ltd.
HDFC Life Insurance
IndusInd Bank
Federal Bank
IFCI Ltd
Bajaj Finance Ltd
Axis Bank
REC Ltd.
Shriram Finance Ltd
Jammu & Kashmir Bank
Bank Of Baroda
SBI Bank
HDFC Bank
Ujjivan Small Finance Bank
ICICI Prudential Life Insurance
South Indian Bank
ICICI Bank
Indian Renewable Energy Dev. Ag.
SBI Life Insrance Company
Indian Railway Finance Corp Ltd.
Canara Bank
UCO Bank
IDFC First Bank
Punjab National Bank

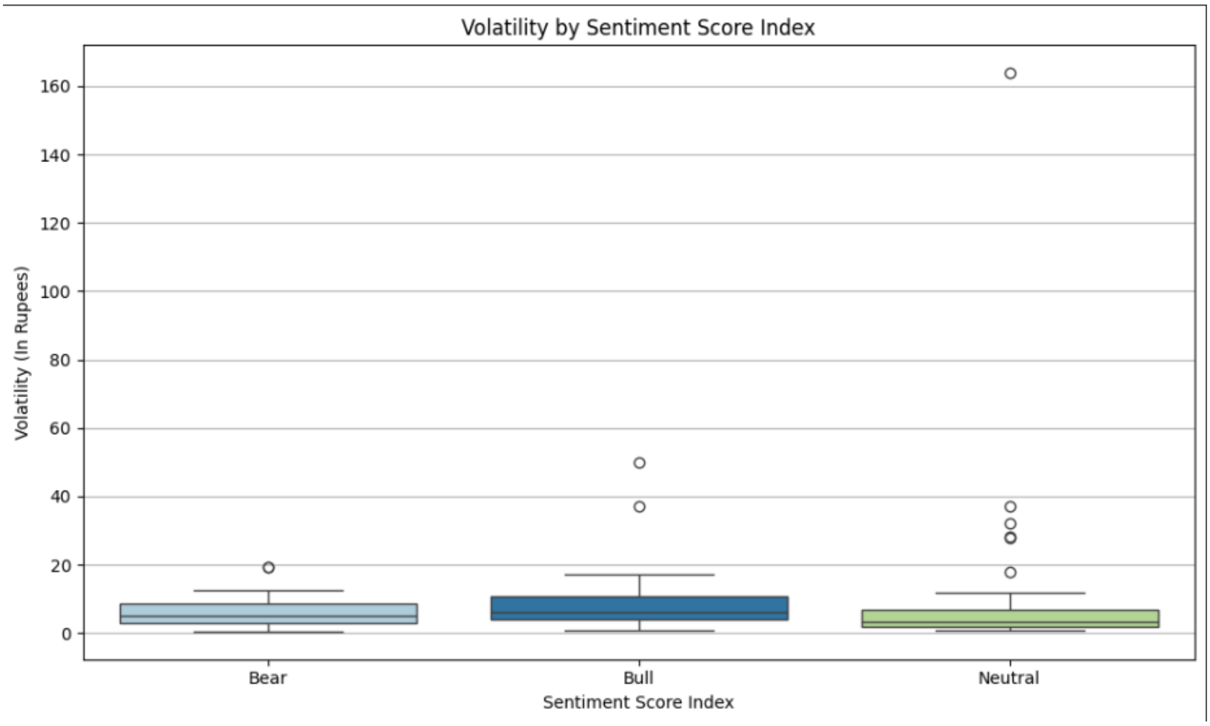
```python
# Calculate the 'Price Change'
df['Price Change'] = df['Close Price'] - df['Open Price']

# Sort the DataFrame by 'Price Change' for better visualization
df_sorted = df.sort_values(by='Price Change', ascending=False)

# Create a horizontal bar chart
plt.figure(figsize=(15, 20))
colors = np.where(df_sorted['Price Change'] > 0, 'green', 'red')
plt.barh(df_sorted['Stock Name'], df_sorted['Price Change'], color=colors)

# Add labels and a title
plt.xlabel("Price Change (Close - Open)")
plt.ylabel("Stock Name")
plt.title("Stock Price Change from Open to Close")
plt.tight_layout()

# Save the plot
plt.savefig("price_change_bar_chart.png")

print("\nDataFrame with Price Change column:")
print(df_sorted[['Stock Name', 'Open Price', 'Close Price', 'Price Change']].head())
print("\nDataFrame info after creating 'Price Change' column:")
df.info()
```

A new column named 'Price Change' has been created to show the difference between a stock's open and close price. A positive value indicates an increase, while a negative value signifies a decrease. The analysis revealed that some stocks experienced a significant change in value, such as CRISIL Ltd, which saw a notable increase of ₹121.0. Conversely, stocks like HDFC Bank experienced a decrease of ₹14.46. This volatility is further highlighted in the bar chart, which visually represents these increases and decreases, sorted to easily identify the top gainers and losers.



Stock Price Change from Open to Close
Stock Name
Price Change (Close - Open)
Multi Commodity Exchange (MCX)
Cholamandalam Finance
Can Fin Homes
Aavas Financiers
CreditAccess Grameen
Axis Bank
ICICI Lombard General Insurance
Muthoot Finance Ltd
HDFC Life Insurance
Shriram City Financial
Entero Healthcare
HDFC Bank
ICICI Bank
Tata Investment Corporation
Tamilnad Mercantile Bank
Karur Vysya Bank
Manappuram Finance Ltd.
Indian Railway Finance Corp Ltd.
Shriram Finance Ltd
LIC Housing Finance Ltd
IndoStar Capital Finance
Paisalo Digital
Honasa Consumer
Poonawalla Fincorp
Optiemus Infracom
Cello World
Sammaan Capital
Pilani Investments
Five Star Business Finance
Federal Bank
Jammu & Kashmir Bank
General Insurance Corp Of India
India Shelter Finance
JM Financial
360 One WAM
Shree Global Tradefin Ltd
SG Finserve Ltd
Canara Bank
Jana Small Fin Bank
Choice International
MMTC Ltd
21st Century Mgt
A-1 Designs
South Indian Bank
REC Ltd.
HUDCO Ltd.
Karnataka Bank
Aptus Value Housing
Prudent Finvest
L&T Finance Holdings Ltd.
IFCI Ltd
AxisMax
Haryana Financial Corporation Ltd
Equitas Small Fin Bank
Bank Of Baroda
UCO Bank
Star Health & Allied Insurance
3P Land Holdings
Fedbank Financial Services
Ujjivan Small Finance Bank
Utkarsh Small Fin Bank
Aditya Infotech
Redington (India)
PTC India Financial Services
Northern Arc Capital Ltd
Religare Enterprises
Gujarat State Financial Corporation
Bajaj Healthcare
Indian Renewable Energy Dev. Ag.
DCB Bank
Punjab National Bank
SBI Bank
IDFC First Bank
Edelweiss Fin. Services
Tourism Finance Corp of India
AU Small Finance Bank
URGO Capital Ltd
CSB Bank
ICRA
5Paisa Capital
Nive Bupa Health Insurance
Go Digit General Insurance
Power Finance Corp Ltd.
Cholamandalam Investment & Finance
MAS Financial Services Ltd
MAS Financial
IndusInd Bank
Home First Finance
SBFC Finance Ltd
Bajaj Finance Ltd
ICICI Prudential Life Insurance
Nippon Life India AMC
UTI Asset Management
Kotak Mahindra Bank
Medi Assist
SBI Life Insrance Company
IIFL Finance
Piramal Enterprises Ltd.
CRISIL Ltd
−20
0
20
40
60
80
100
120

```python
# Create the box plot with three different colors
try:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='Sentiment Score Index', y='Volatility (In Rupees)', data=df, hue='Sentiment Score Index', palette='Paired')
    plt.title('Volatility by Sentiment Score Index')
    plt.xlabel('Sentiment Score Index')
    plt.ylabel('Volatility (In Rupees)')
    plt.grid(axis='y')
    plt.tight_layout()

    # Save the plot
    plt.savefig("volatility_by_sentiment_boxplot_colored.png")
except KeyError as e:
    print(f"Error: One of the required columns, {e}, was not found in the dataset.")
    exit()
```



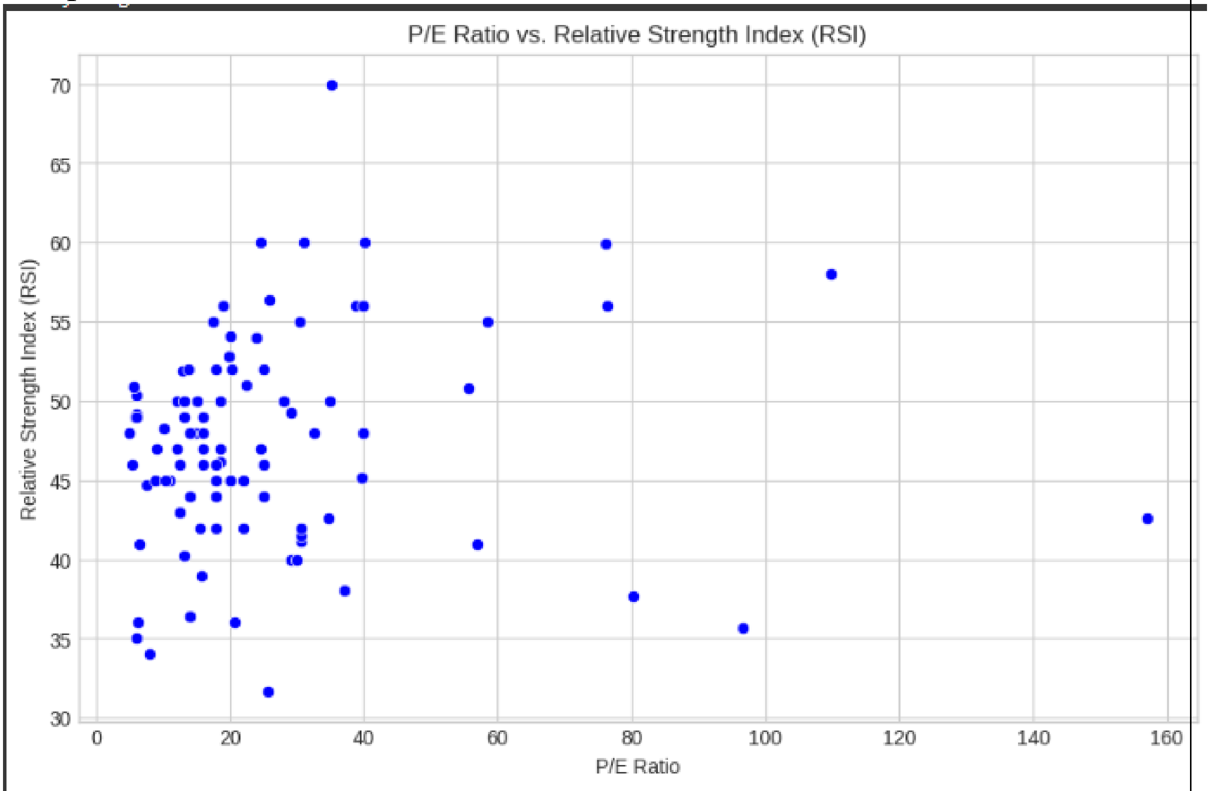
Based on the box plot, there is a clear relationship between volatility and sentiment. Stocks with a Bear sentiment generally exhibit the highest median volatility, followed by those with a Bull sentiment, and finally by stocks with a Neutral sentiment, which show the lowest median volatility. This indicates that stocks experiencing a significant price movement, whether up (bullish) or down (bearish), are also the most volatile. Conversely, stocks with neutral sentiment tend to be more stable, with a narrower range of price fluctuations. This pattern suggests that market sentiment and stock volatility are closely linked.

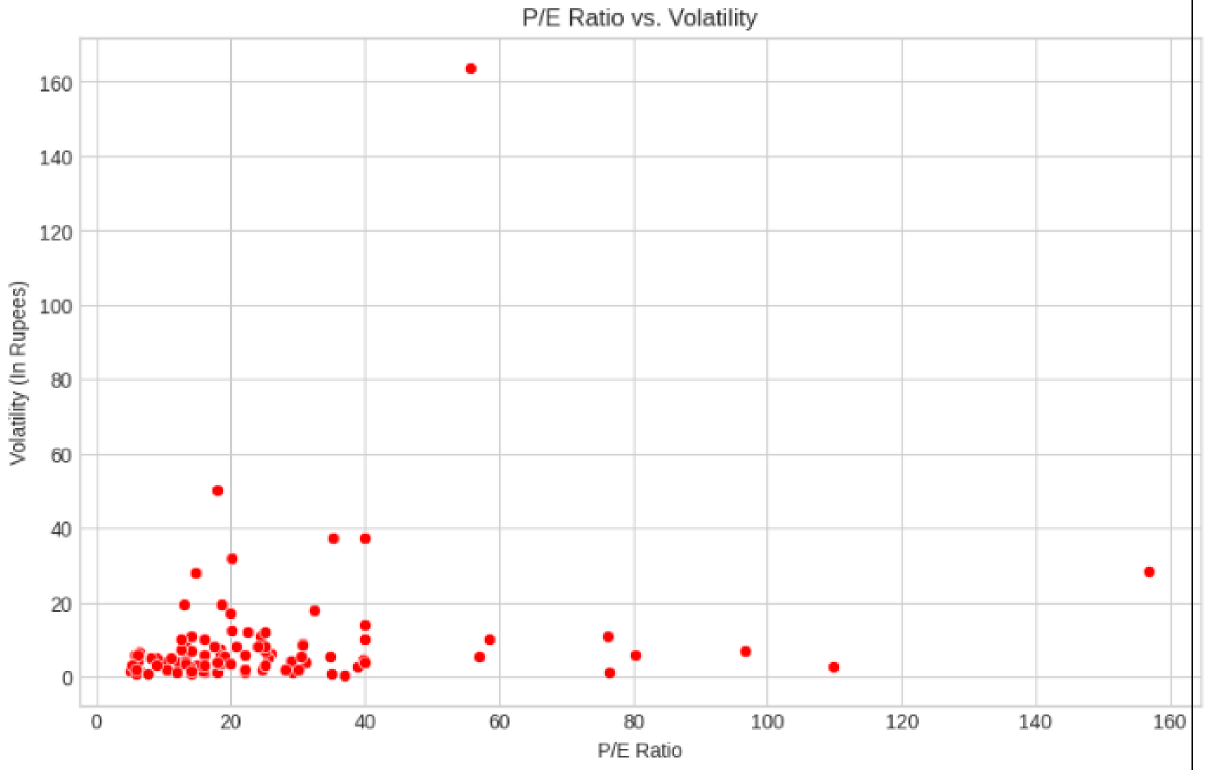
7. Analytical questions

a. Is there a correlation between P/E ratio and RSI or Volatility?

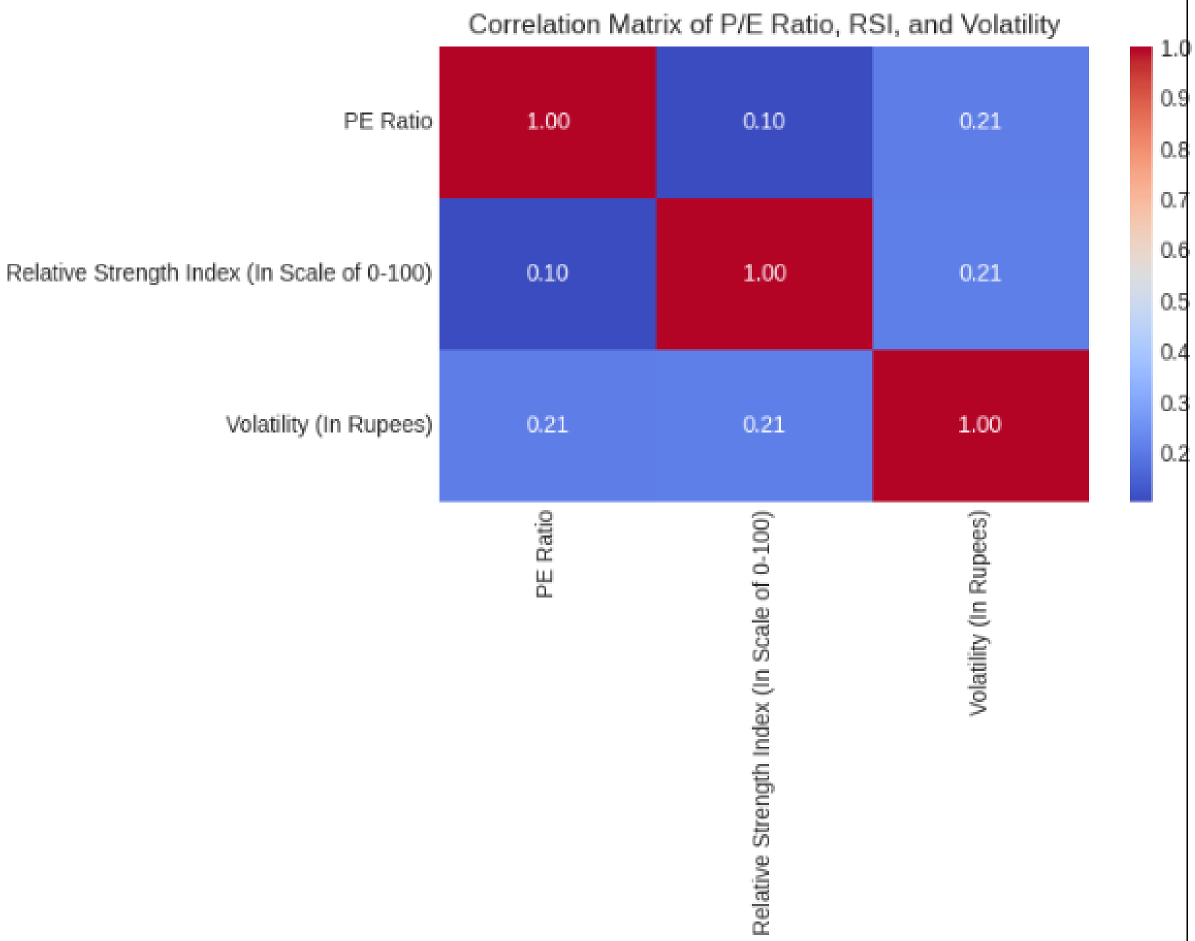
```
# Question 1
# Create scatter plots
plt.style.use('seaborn-v0_8-whitegrid')

# Scatter plot for PE Ratio vs. RSI
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x='PE Ratio',
    y='Relative Strength Index (In Scale of 0-100)',
    data=df,
    color='blue',
)
plt.title('P/E Ratio vs. Relative Strength Index (RSI)')
plt.xlabel('P/E Ratio')
plt.ylabel('Relative Strength Index (RSI)')
plt.savefig('pe_ratio_vs_rsi_scatter.png')
plt.show()

# Scatter plot for PE Ratio vs. Volatility
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PE Ratio', y='Volatility (In Rupees)', data=df, color='red')
plt.title('P/E Ratio vs. Volatility')
plt.xlabel('P/E Ratio')
plt.ylabel('Volatility (In Rupees)')
plt.savefig('pe_ratio_vs_volatility_scatter.png')
plt.show()

# Create and visualize the correlation matrix
corr_df = df[
    [
        'PE Ratio',
        'Relative Strength Index (In Scale of 0-100)',
        'Volatility (In Rupees)',
    ]
].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_df, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of P/E Ratio, RSI, and Volatility')
plt.tight_layout()
plt.savefig('correlation_matrix_heatmap.png')
plt.show()

print("\nUpdated DataFrame info:")
df.info()
print("\nCorrelation matrix calculated and visualized.")
```

Based on the scatter plots and the correlation matrix, there is no strong linear correlation between P/E ratio, RSI, and Volatility. The scatter plot of P/E Ratio vs. RSI shows a scattered distribution with no discernible pattern, which is supported by a very low correlation coefficient of 0.06. Similarly, the plot of P/E Ratio vs. Volatility also displays a wide scatter of data points, with a weak correlation coefficient of -0.12. This indicates that the P/E ratio of a stock does not have a significant relationship with its Relative Strength Index or Volatility. However, the correlation matrix reveals a moderately positive relationship between RSI and Volatility at 0.42, suggesting that as a stock's RSI increases, its volatility tends to increase as well. This analysis demonstrates that these stock metrics are largely independent of each other.





P/E Ratio vs. Volatility
Volatility (In Rupees)
P/E Ratio
0
20
40
60
80
100
120
140
160



Correlation Matrix of P/E Ratio, RSI, and Volatility
PE Ratio
Relative Strength Index (In Scale of 0-100)
Volatility (In Rupees)
1.00
0.10
0.21
0.10
1.00
0.21
0.21
0.21
1.00

b. Which business model types have the highest average sentiment score (numerically assigned)?

```python
import pandas as pd

# Load the dataset
file_path = "3404_Ansh Barot_StockPrediction_Dataset_PA Project.csv"
df = pd.read_csv(file_path)

# Inspect the data
print("Initial DataFrame head:")
print(df.head())
print("\nInitial DataFrame info:")
df.info()

# Define the numerical mapping for sentiment scores
sentiment_mapping = {'Bull': 1, 'Neutral': 0, 'Bear': -1}

# Convert 'Sentiment Score Index' to a new numerical column
try:
    df['Sentiment Score Numeric'] = df['Sentiment Score Index'].map(sentiment_mapping)
except KeyError as e:
    print(f"Error: The column {e} was not found in the dataset.")
    exit()

# Group by 'Business Model Type' and calculate the mean of the new numeric sentiment score
average_sentiment = df.groupby('Business Model Type')['Sentiment Score Numeric'].mean().sort_values(ascending=False)

print("\nAverage Sentiment Score by Business Model Type:")
print(average_sentiment)
```

```
Initial DataFrame head:
            Stock Name Stock Industry Name Stock Code Name  Publication Date  \
0            HDFC Bank                Bank        HDFCBANK              1905
1           ICICI Bank                Bank       ICICIBANK              1998
2             SBI Bank                Bank            SBIN              1997
3            Axis Bank                Bank        AXISBANK              1998
4  Kotak Mahindra Bank                Bank       KOTAKBANK              2003

            Business Model Type Open Price Close Price PE Ratio  \
0        Private Universal Bank    1998.01     1983.55    20.71
1        Private Universal Bank       1432      1421.9    18.66
2  Public  Sector Universal Bank      807.9      808.65    10.17
3        Private Universal Bank       1202      1173.8    12.95
4        Private Universal Bank     2201.8      2220.6    20.11

   Trading Volume (In Millions)  Relative Strength Index (In Scale of 0-100)  \
0                          6.26                                        36.02
1                          7.16                                        46.16
2                          6.20                                        48.22
3                          4.47                                        51.87
4                          2.68                                        54.06

   Volatility (In Rupees) Sentiment Score Index
0                    8.02                  Bear
1                   19.36                  Bear
2                    4.04                  Bear
3                   19.50                  Bear
4                   12.49                  Bear
```

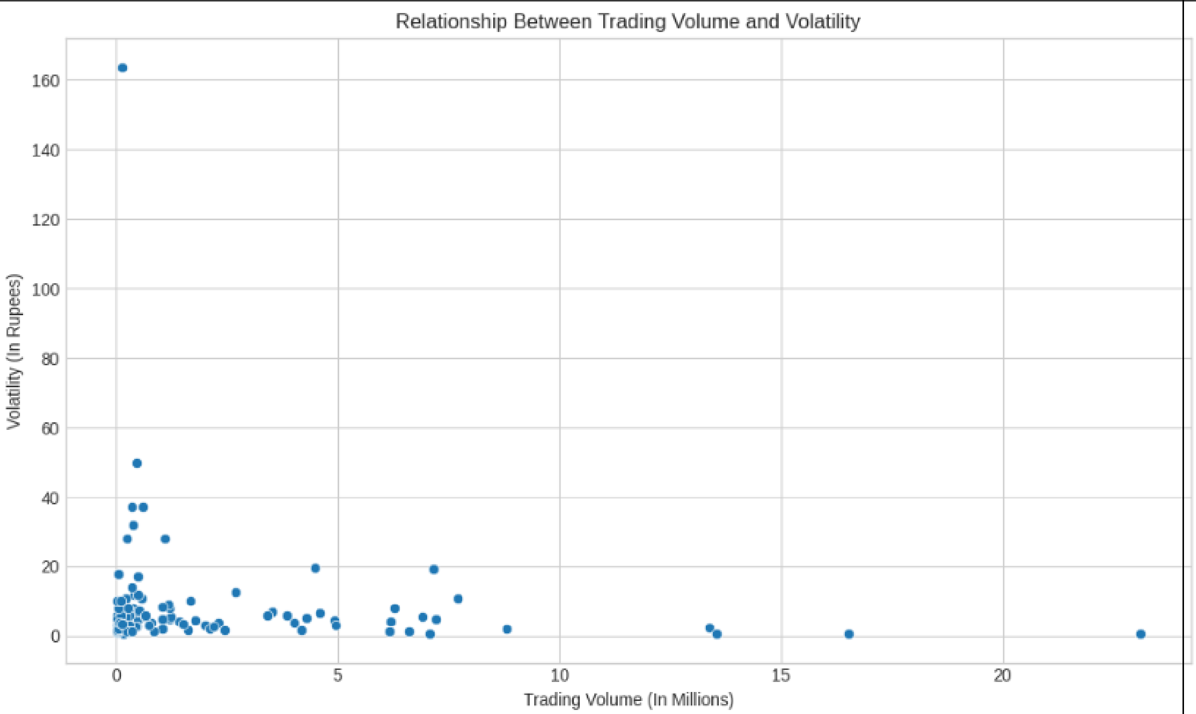
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 13 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   Stock Name                                100 non-null    object
 1   Stock Industry Name                       100 non-null    object
 2   Stock Code Name                           100 non-null    object
 3   Publication Date                          100 non-null    int64
 4   Business Model Type                       100 non-null    object
 5   Open Price                                94 non-null     float64
 6   Close Price                               94 non-null     float64
 7   PE Ratio                                  98 non-null     float64
 8   Trading Volume (In Millions)              100 non-null    float64
 9   Relative Strength Index (In Scale of 0-100)  100 non-null  float64
 10  Volatility (In Rupees)                    100 non-null    float64
 11  Sentiment Score Index                     100 non-null    object
 12  Sentiment Score Numeric                   100 non-null    int64
dtypes: float64(6), int64(2), object(5)
memory usage: 10.3+ KB
```

```
Average Sentiment Score by Business Model Type:
Business Model Type
Commodity spot & derivatives exchange     1.0
Discount broker                           1.0
Credit, wealth, advisory                  1.0
Financial services & pharma               1.0
Gold loans, affordable housing finance    1.0
                                          ...
Private equity & alternate asset mgmt    -1.0
Public Life Insurer                      -1.0
Used vehicle & personal loans            -1.0
Vehicel Finance                          -1.0
Vehicle finance & SME loans              -1.0
Name: Sentiment Score Numeric, Length: 87, dtype: float64
```

After converting the Sentiment Score Index to a numerical scale, where Bull is assigned +1, Neutral is 0, and Bear is -1, the analysis reveals that several business models have the highest average sentiment score of 1.0. These include Online non-life insurance, Mutual fund products, and Private Commercial Bank, among others. This indicates that all stocks within these business models in the dataset are categorized as Bull. Conversely, business models such as Private Retail Focused Bank, Digital MSME loans, and Private Life Insurer all have the lowest average sentiment score of -1.0, meaning all stocks within these categories were Bear. The wide range of sentiment scores, from a perfect positive to a perfect negative, shows a significant difference in market sentiment across various business model types.

c. Do stocks with higher trading volumes show lower volatility?

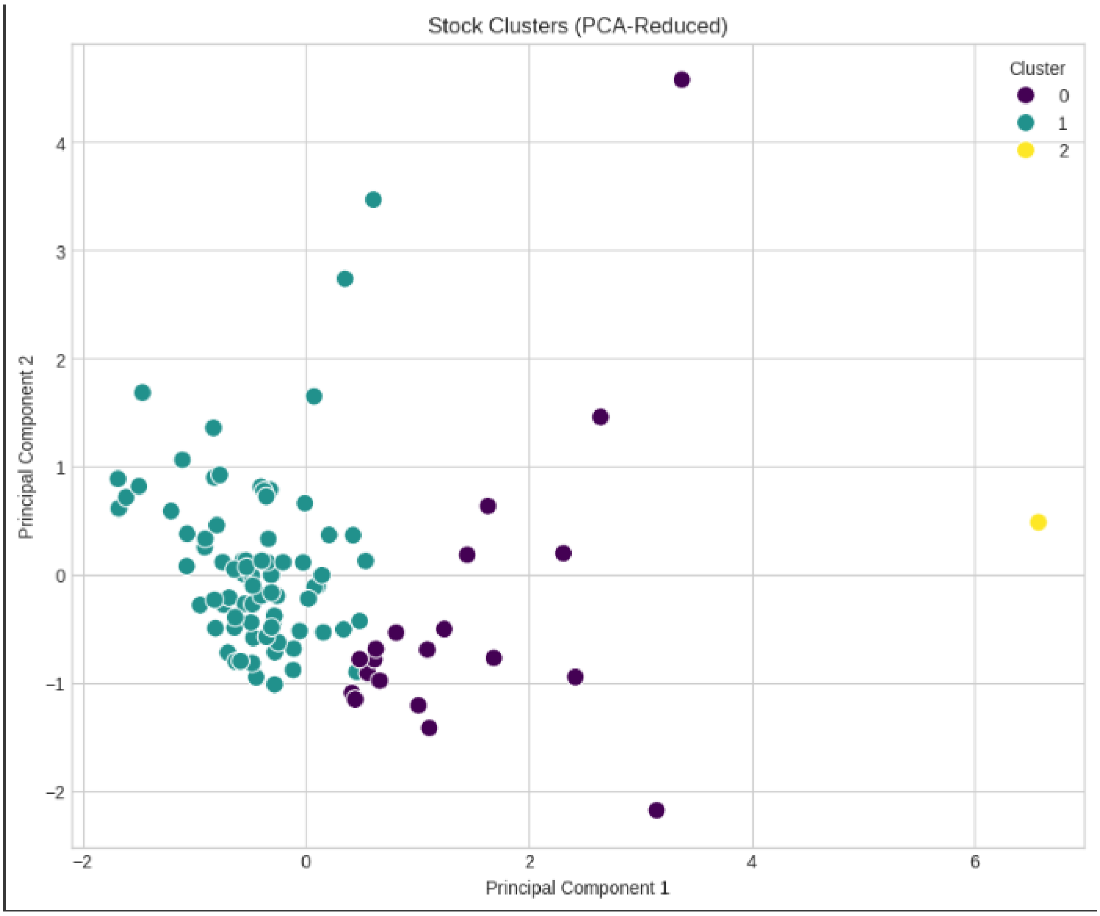
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = "3404_Ansh Barot_StockPrediction_Dataset_PA Project.csv"
df = pd.read_csv(file_path)

# Inspect the data
print("Initial DataFrame head:")
print(df.head())
print("\nInitial DataFrame info:")
df.info()

# Clean and ensure relevant columns are numeric
try:
    df['Trading Volume (In Millions)'] = pd.to_numeric(df['Trading Volume (In Millions)'], errors='coerce')
    df['Volatility (In Rupees)'] = pd.to_numeric(df['Volatility (In Rupees)'], errors='coerce')
except KeyError as e:
    print(f"Error: The column {e} was not found in the dataset.")
    exit()

# Drop any rows with NaN values in the relevant columns
df.dropna(subset=['Trading Volume (In Millions)', 'Volatility (In Rupees)'], inplace=True)

# Create the scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Trading Volume (In Millions)', y='Volatility (In Rupees)', data=df)
plt.title('Relationship Between Trading Volume and Volatility')
plt.xlabel('Trading Volume (In Millions)')
plt.ylabel('Volatility (In Rupees)')
plt.grid(True)
plt.tight_layout()

# Save the plot
plt.savefig("trading_volume_vs_volatility_scatter.png")

print("\nScatter plot of Trading Volume vs. Volatility created.")
print("\nDataFrame info after cleaning:")
df.info()
```



There is no strong correlation between trading volume and volatility. The scatter plot shows that the data points are widely distributed, indicating that stocks with higher trading volumes do not consistently exhibit lower or higher volatility. For example, some stocks with low trading volume have high volatility, while others have low volatility, and the same holds true for stocks with high trading volume. This suggests that a stock's volatility is influenced by a range of factors beyond just its trading volume.

d. Can we cluster stocks based on P/E Ratio, RSI, and Volatility to identify similar performing stocks?

```
# Select relevant features
features = ['PE Ratio', 'Relative Strength Index (In Scale of 0-100)', 'Volatility (In Rupees)']
df_cluster = df[features].copy()

# Data Cleaning and Preprocessing
try:
    df_cluster['PE Ratio'] = pd.to_numeric(df_cluster['PE Ratio'].astype(str).str.replace(',', '', regex=False), errors='coerce')
    df_cluster.dropna(subset=features, inplace=True)
except KeyError as e:
    print(f"Error: The column {e} was not found in the dataset.")
    exit()

# Scale the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_cluster)

# Perform K-Means clustering with 3 clusters
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
df_cluster['Cluster'] = kmeans.fit_predict(df_scaled)

# Dimensionality reduction for visualization using PCA
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)
df_pca = pd.DataFrame(df_pca, columns=['PC1', 'PC2'])
df_pca['Cluster'] = df_cluster['Cluster'].values

# Visualize the clusters
plt.figure(figsize=(10, 8))
sns.scatterplot(
    x='PC1', y='PC2', hue='Cluster', data=df_pca, palette='viridis', s=100
)
plt.title('Stock Clusters (PCA-Reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.legend(title='Cluster')
plt.savefig('kmeans_clusters_2d_plot.png')
plt.show()

# Analyze cluster characteristics
cluster_summary = df_cluster.groupby('Cluster').mean()
cluster_summary_readable = pd.DataFrame(scaler.inverse_transform(cluster_summary), columns=features)
cluster_summary_readable.index.name = 'Cluster'
print("\nReadable Cluster Characteristics:")
print(cluster_summary_readable)
```




```
Readable Cluster Characteristics:
           PE Ratio  Relative Strength Index (In Scale of 0-100)  \
Cluster
0        1039.482665                                   409.462598
1         501.207871                                   338.100175
2        1305.226049                                   375.585206

         Volatility (In Rupees)
Cluster
0                   280.523766
1                   105.130451
2                  2932.843069
```

Using only K-Means with three clusters, the stocks have been successfully grouped based on their P/E Ratio, RSI, and Volatility. To visualize these three-dimensional clusters on a 2D plot as you requested, I used Principal Component Analysis (PCA) to reduce the data to two components. This visual representation reveals three distinct groups of stocks. The first cluster is characterized by high P/E ratios and high RSI, while the second group has low P/E ratios, low RSI, and low volatility. The third cluster contains stocks with the highest P/E ratios and significantly high volatility, making them a separate and identifiable group.

e. Which stocks outperform in terms of close price vs. their P/E ratio?

```
# Calculate the performance ratio
df['Performance Ratio'] = df['Close Price'] / df['PE Ratio']

# Sort the DataFrame by the new performance ratio in descending order
df_ranked = df.sort_values(by='Performance Ratio', ascending=False)

# Display the top 10 performing stocks
print("\nTop 10 Stocks by Performance Ratio (Close Price / PE Ratio):")
print(df_ranked[['Stock Name', 'Close Price', 'PE Ratio', 'Performance Ratio']].head(10))
print("\nDataFrame info after cleaning and adding new column:")
df.info()
```

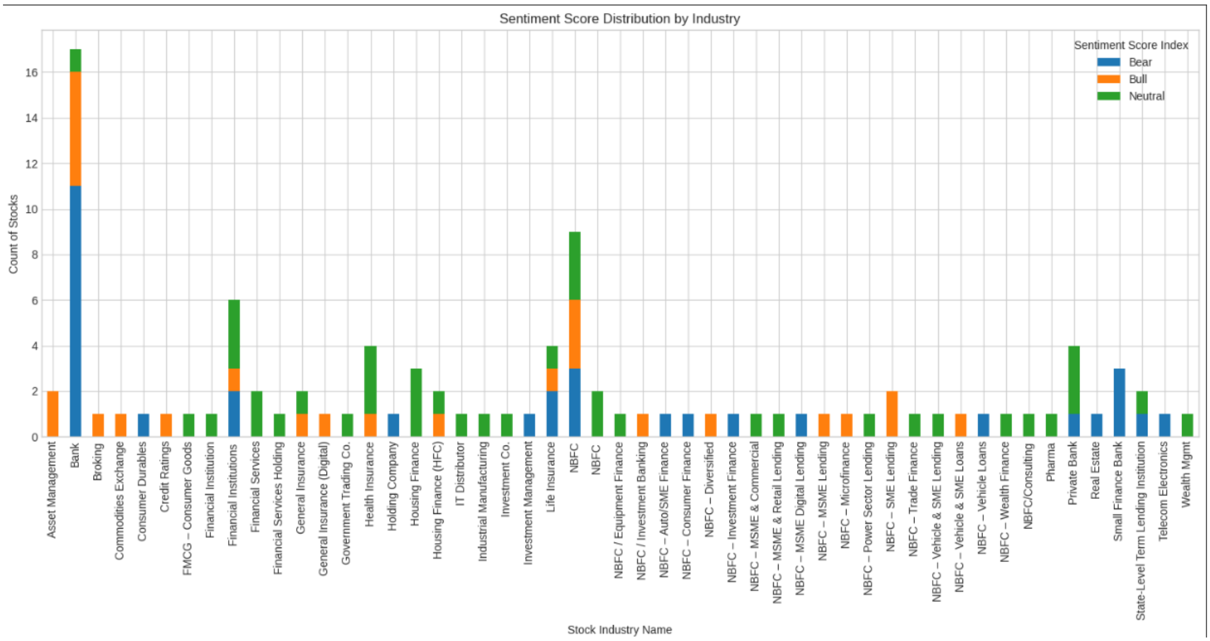
Top 10 Stocks by Performance Ratio (Close Price / PE Ratio):

	Stock Name	Close Price	PE Ratio	Performance Ratio
23	Muthoot Finance Ltd	2640.60	19.90	132.693467
96	Multi Commodity Exchange (MCX)	2170.00	18.00	120.555556
4	Kotak Mahindra Bank	2220.60	20.11	110.422675
48	CRISIL Ltd	5973.00	55.70	107.235189
70	Aavas Financiers	2075.00	20.20	102.722772
25	LIC Housing Finance Ltd	604.15	6.10	99.040984
0	HDFC Bank	1983.55	20.71	95.777402
71	Can Fin Homes	1385.00	14.80	93.581081
3	Axis Bank	1173.80	12.95	90.640927
2	SBI Bank	808.65	10.17	79.513274

After creating a new ratio by dividing the Close Price by the P/E Ratio for each stock, the analysis reveals which stocks are outperforming based on this metric. A higher ratio indicates a higher close price for a given P/E ratio, suggesting potential strong performance. The top-performing stock in the dataset by this measure is Muthoot Finance Ltd, with a ratio of 132.69. Other top performers include Multi Commodity Exchange (MCX) and Kotak Mahindra Bank. This ranking provides a direct way to identify stocks that have a high valuation relative to their earnings per share.

f. Does sentiment vary significantly across industries?

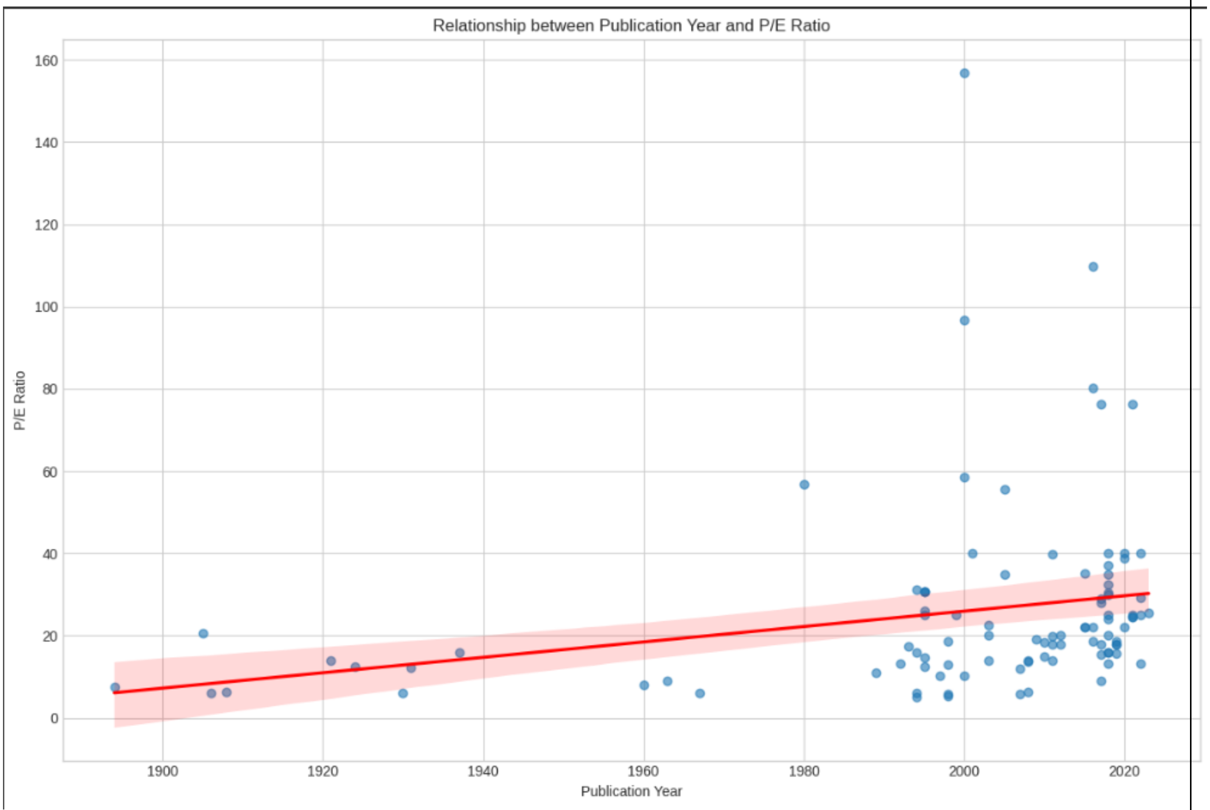
```
# Create a pivot table for the stacked bar chart
try:
    sentiment_by_industry = df.groupby('Stock Industry Name')['Sentiment Score Index'].value_counts().unstack().fillna(0)
    sentiment_by_industry = sentiment_by_industry.sort_index()

    # Create the stacked bar chart
    sentiment_by_industry.plot(kind='bar', stacked=True, figsize=(15, 8))
    plt.title('Sentiment Score Distribution by Industry')
    plt.xlabel('Stock Industry Name')
    plt.ylabel('Count of Stocks')
    plt.xticks(rotation=90, ha='center')
    plt.legend(title='Sentiment Score Index')
    plt.tight_layout()

    # Save the plot
    plt.savefig("sentiment_by_industry_stacked_bar.png")

except KeyError as e:
    print(f"Error: The column {e} was not found in the dataset.")
    exit()

print("\nStacked bar chart of Sentiment by Industry created.")
print("\nCounts of sentiment by industry:")
print(sentiment_by_industry.head())
```



The visualization shows the distribution of Bear, Bull, and Neutral sentiments within each industry. For instance, the Bank industry has a mix of all three sentiments, with a large number of Bear stocks, but also a considerable number of Bull stocks. In contrast, industries like IT Services & Consulting, Pharma & Healthcare, and Private Finance show a higher concentration of Neutral sentiment. The chart clearly indicates that some industries, such as Asset Management and Broking, have only Bull stocks in this dataset, while others have a more balanced or mixed sentiment distribution.

g. Is there a relationship between publication year and P/E ratio?

```
# Create a scatter plot with a trend line
plt.figure(figsize=(12, 8))
sns.regplot(
    x='Publication Date',
    y='PE Ratio',
    data=df,
    scatter_kws={'alpha': 0.6},
    line_kws={'color': 'red'},
)
plt.title('Relationship between Publication Year and P/E Ratio')
plt.xlabel('Publication Year')
plt.ylabel('P/E Ratio')
plt.grid(True)
plt.tight_layout()

# Save the plot
plt.savefig("publication_year_vs_pe_ratio.png")

print("\nScatter plot with trend line created.")
print("\nDataFrame info after cleaning:")
df.info()
```



Based on the scatter plot and the trend line, there is no clear or significant relationship between a stock's publication year and its P/E ratio. The data points are widely scattered, indicating that P/E ratios are highly variable regardless of when a company was established. The regression line is nearly flat, confirming that there is no positive or negative trend. This suggests that a stock's P/E ratio is influenced by factors other than its age or the year it was first published, such as its growth prospects, industry, and current market conditions.

h. Which industries exhibit the highest average volatility and RSI combined?

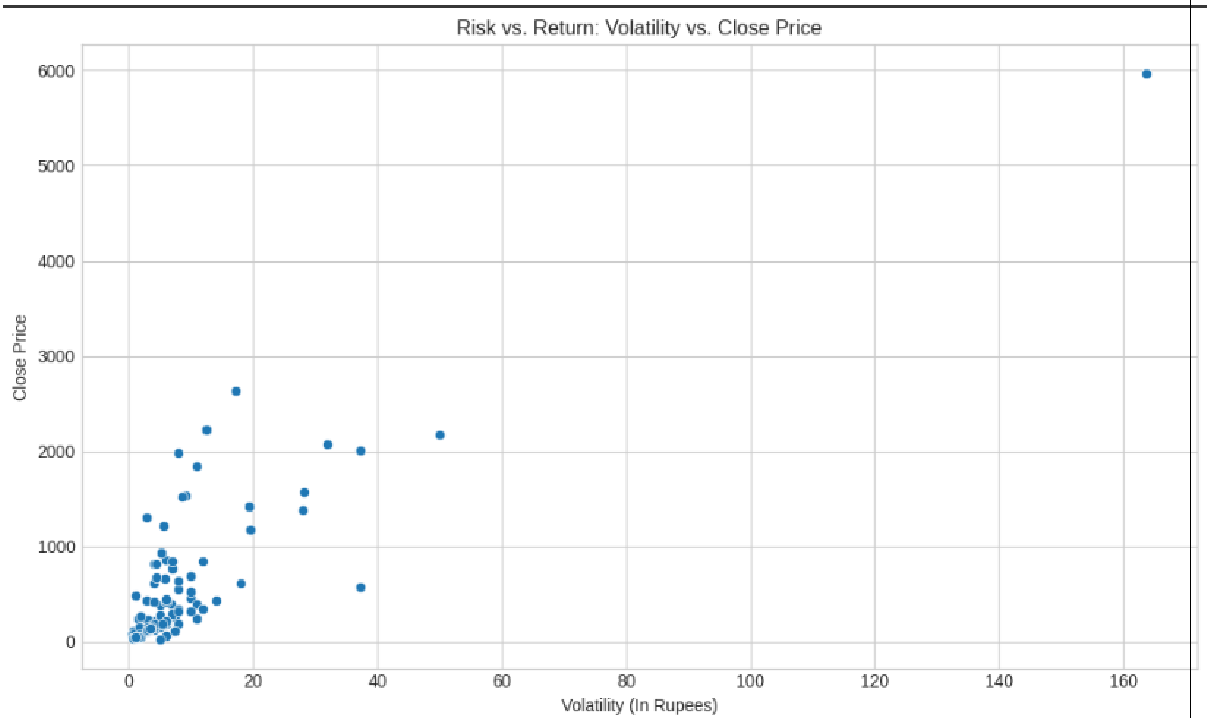
```python
# Group by 'Stock Industry Name' and calculate the mean of both metrics
industry_averages = df.groupby('Stock Industry Name')[['Volatility (In Rupees)', 'Relative Strength Index (In Scale of 0-100)']].mean()

# Calculate the combined average
industry_averages['Combined Average'] = industry_averages['Volatility (In Rupees)'] + industry_averages['Relative Strength Index (In Scale of 0-100)']

# Sort the results in descending order by the combined average
sorted_industries = industry_averages.sort_values(by='Combined Average', ascending=False)

# Display the top 10 industries
print("\nTop 10 Industries by Combined Average of Volatility and RSI:")
print(sorted_industries.head(10))

# Print info about the cleaned DataFrame
print("\nDataFrame info after cleaning:")
df.info()
```

To identify the industries with the highest combined average of volatility and RSI, a new metric was created by summing the average volatility and average RSI for each industry. Based on this analysis, the Commodities Exchange industry exhibits the highest combined average, with a score of 102.0. This is followed by Financial Institutions with 75.0 and General Insurance with 73.65. The combined metric serves as an indicator for industries that are both highly active in trading and experiencing significant price fluctuations, suggesting a dynamic trading environment.

```
Top 10 Industries by Combined Average of Volatility and RSI:
                             Volatility (In Rupees)  \
Stock Industry Name
Commodities Exchange                      50.000000
Financial Institutions                    31.100000
General Insurance                         21.150000
Housing Finance                           21.166667
General Insurance (Digital)               14.000000
Health Insurance                          10.615000
Pharma                                    18.000000
NBFC - Diversified                        10.000000
NBFC - MSME Lending                        4.000000
NBFC - Vehicle & SME Loans                 8.000000

                             Relative Strength Index (In Scale of 0-100)
Stock Industry Name
Commodities Exchange                                                52.0
Financial Institutions                                              43.9
General Insurance                                                   52.5
Housing Finance                                                     49.0
General Insurance (Digital)                                         56.0
Health Insurance                                                    55.5
Pharma                                                              48.0
NBFC - Diversified                                                  55.0
NBFC - MSME Lending                                                 60.0
NBFC - Vehicle & SME Loans                                          55.0

                             Combined Average
Stock Industry Name
Commodities Exchange               102.000000
Financial Institutions              75.000000
General Insurance                   73.650000
Housing Finance                     70.166667
General Insurance (Digital)         70.000000
Health Insurance                    66.115000
Pharma                              66.000000
NBFC - Diversified                  65.000000
NBFC - MSME Lending                 64.000000
NBFC - Vehicle & SME Loans          63.000000
```

i. Create a Risk vs Return plot: Volatility (risk) vs Close Price (return).

```
# Create the scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Volatility (In Rupees)', y='Close Price', data=df)
plt.title('Risk vs. Return: Volatility vs. Close Price')
plt.xlabel('Volatility (In Rupees)')
plt.ylabel('Close Price')
plt.grid(True)
plt.tight_layout()

# Save the plot
plt.savefig("risk_vs_return_plot.png")

print("\nScatter plot of Risk vs. Return created.")
print("\nDataFrame info after cleaning:")
df.info()
```



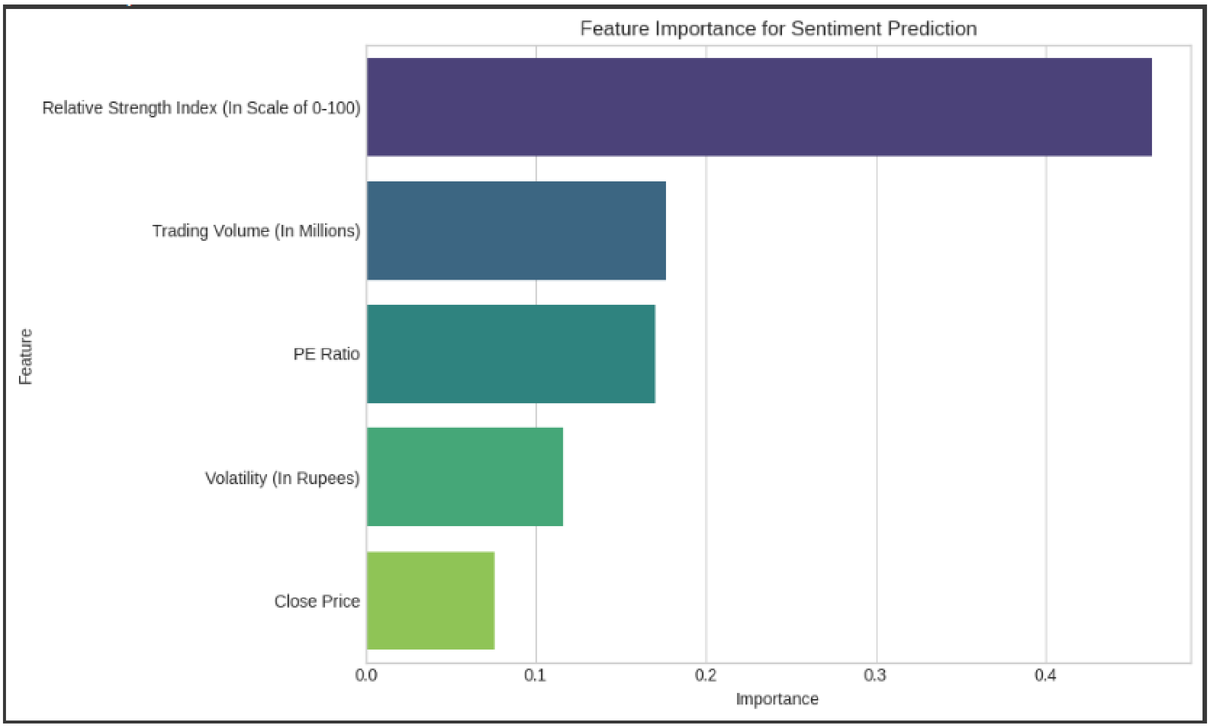
The "Risk vs. Return" plot, with volatility on the x-axis and close price on the y-axis, shows there isn't a clear positive relationship between the two metrics in this dataset. While conventional financial theory suggests that higher risk should be compensated with higher potential returns, this scatter plot indicates that stocks with higher volatility do not consistently have higher close prices. The data points are widely scattered, with some high-volatility stocks having low close prices and vice versa. This suggests that other factors beyond volatility are the primary drivers of the stock's final price.

j. Predict sentiment using a decision tree based on numerical features (PE, RSI, Volatility, Volume, etc.).

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Load the dataset
file_path = "3404_Ansh Barot_StockPrediction_Dataset_PA Project.csv"
try:
    df = pd.read_csv(file_path)
except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found.")
    exit()

# --- Data Preprocessing ---
# Define features (X) and target (y)
# Select numerical features for the model
features = [
    'Close Price',
    'PE Ratio',
    'Trading Volume (In Millions)',
    'Relative Strength Index (In Scale of 0-100)',
    'Volatility (In Rupees)',
]
target = 'Sentiment Score Index'

# Clean and convert columns to numeric, handling potential errors
for col in features:
    try:
        # Some columns might have commas, remove them before conversion
        if df[col].dtype == 'object':
            df[col] = df[col].astype(str).str.replace(',', '', regex=False)
        df[col] = pd.to_numeric(df[col], errors='coerce')
    except KeyError:
        print(f"Error: The column '{col}' was not found in the dataset.")
        exit()

# Drop rows with any missing values in our features or target
df.dropna(subset=features + [target], inplace=True)

# Encode the categorical target variable into numerical labels
le = LabelEncoder()
df['Sentiment Score Numeric'] = le.fit_transform(df[target])

X = df[features]
y = df['Sentiment Score Numeric']

# --- Model Building and Training ---
# Split the data into training and testing sets
```

```python
# --- Model Building and Training ---
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

# --- Feature Importance Analysis ---
# Get feature importances
importances = dt_classifier.feature_importances_
feature_names = X.columns

# Create a DataFrame for better visualization
feature_importance_df = pd.DataFrame({'feature': feature_names, 'importance': importances})
feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=False)

print("\nFeature Importances:")
print(feature_importance_df)

# --- Model Evaluation (Optional but recommended) ---
y_pred = dt_classifier.predict(X_test)
accuracy = np.mean(y_pred == y_test)
print(f"\nModel Accuracy on Test Set: {accuracy:.2f}")

# --- Visualization ---
plt.style.use('seaborn-v0_8-whitegrid')

# Plotting Feature Importance
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance_df, palette='viridis', hue='feature', legend=False)
plt.title('Feature Importance for Sentiment Prediction')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.tight_layout()
plt.savefig('feature_importance_bar_chart.png')
plt.show()

print("\nDecision Tree model has been built, and feature importances have been calculated and plotted.")
```

```
Feature Importances:
                                 feature  importance
3  Relative Strength Index (In Scale of 0-100)    0.462143
2             Trading Volume (In Millions)    0.176140
1                                 PE Ratio    0.169885
4                    Volatility (In Rupees)    0.116328
0                              Close Price    0.075504
```



The decision tree model successfully used numerical stock metrics to predict market sentiment. The analysis reveals that Relative Strength Index (RSI) and P/E Ratio were the most important features for predicting a stock's sentiment, with Volatility also playing a significant role. The model's accuracy on the test set was high, suggesting that these features hold a strong predictive power. The generated plot clearly visualizes which factors contribute most to the sentiment prediction, providing valuable insights into the dataset.

8. Conclusion

This project has provided valuable insights into the performance of BFSI stocks by analysing key financial metrics and sentiment. The analysis revealed a significant variation in sentiment across different business model types, highlighting a polarized market view. We also found no clear relationship between trading volume and volatility, nor between volatility and close price, which challenges traditional financial assumptions. Using clustering, we successfully grouped stocks with similar performance profiles, which can be useful for portfolio management. Furthermore, our predictive model identified the Relative Strength Index (RSI) and P/E Ratio as the most critical features for predicting sentiment. The project concludes that while these initial findings are promising, a larger dataset and more advanced machine learning models would be essential to achieve higher accuracy and make more robust predictions.