In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_excel('Downloads/Dataset - Business Metrics.xlsx')
```

In [2]:
```python
data['Date'] = pd.to_datetime(data['Date'])
```

In [3]:
```python
data.head(5)
```

Out[3]:

	Date	Close/Last	Volume	Open	High	Low
0	2024-04-15	154.86	27136470	158.860	159.2400	154.59
1	2024-04-12	157.73	25353750	157.960	160.2225	157.14
2	2024-04-11	159.41	27166430	156.910	159.6800	156.46
3	2024-04-10	156.14	22838630	156.210	156.6100	154.68
4	2024-04-09	156.60	31113010	156.085	158.5600	155.19

In [4]:
```python
data_without_date = data.drop(columns='Date')

descriptive_stats = data_without_date.describe()
mode_values = data_without_date.mode().iloc[0]

print("Descriptive Statistics for the dataset excluding 'Date':")
display(descriptive_stats)

print("\nMode values for the dataset excluding 'Date' (most frequent values):")
display(mode_values.to_frame().T)
```

```
Descriptive Statistics for the dataset excluding 'Date':
```

	Close/Last	Volume	Open	High	Low
count	251.000000	2.510000e+02	251.000000	251.000000	251.000000
mean	132.389502	3.042197e+07	132.208677	133.687718	131.007896
std	11.875165	1.110220e+07	11.869759	11.856320	11.829998
min	103.710000	1.251432e+07	103.580000	104.980000	102.630000
25%	124.270000	2.344789e+07	124.340000	125.560000	122.712500
50%	132.720000	2.712465e+07	133.120000	134.260000	131.570000
75%	139.392500	3.449830e+07	139.130000	140.905000	138.027500
max	159.410000	8.436621e+07	158.860000	160.222500	157.140000

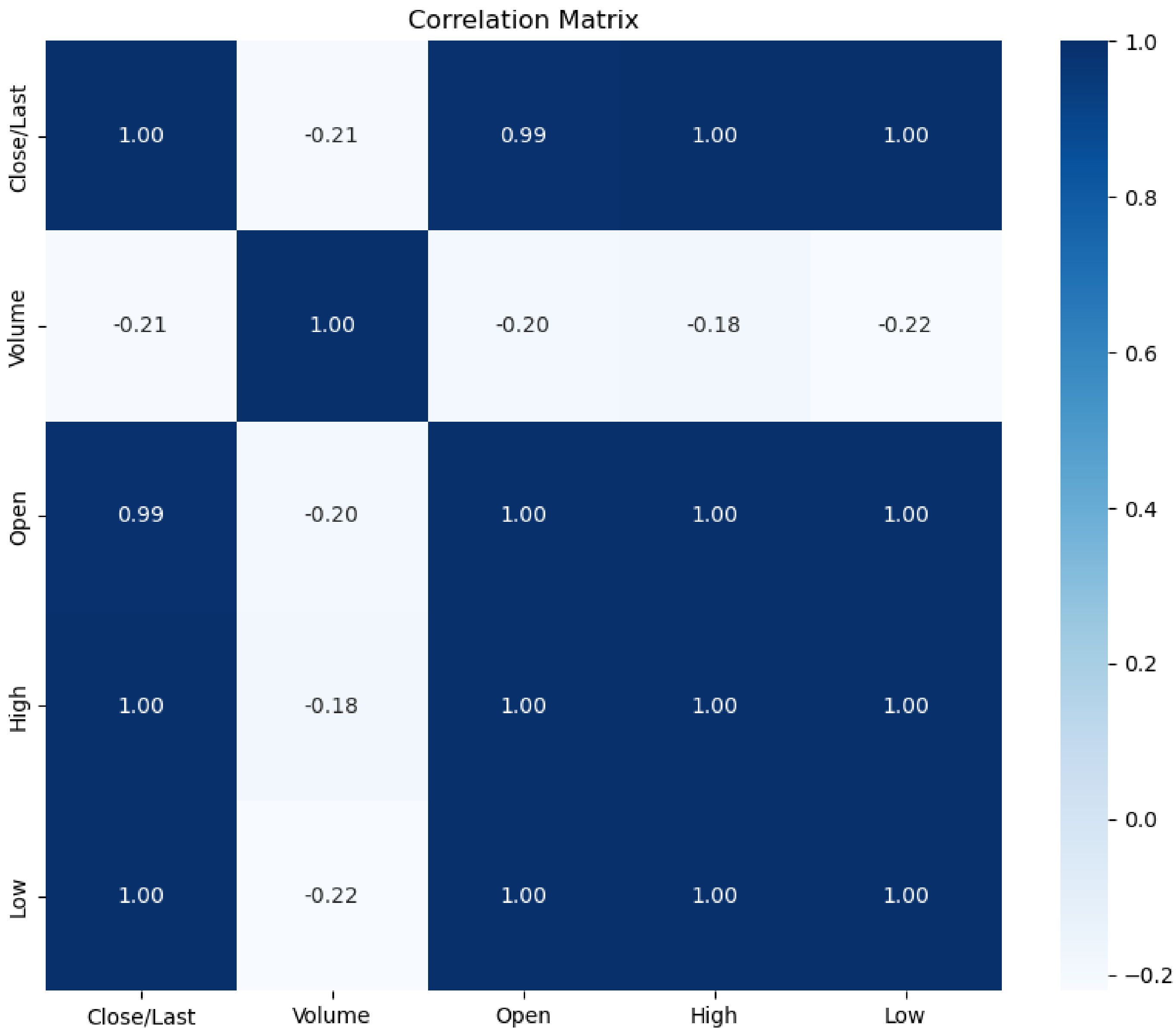
```
Mode values for the dataset excluding 'Date' (most frequent values):
```

	Close/Last	Volume	Open	High	Low
0	105.41	12514320.0	119.24	123.31	128.32

Correlation Analysis

In [5]:

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

data_without_date = data.iloc[:, 1:]

correlation_matrix = data_without_date.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='Blues')
plt.title('Correlation Matrix')
plt.show()
```



In [6]:

```
import pandas as pd

# Assuming 'Date' is the first column, you can drop it using iloc
data_without_date = data.iloc[:, 1:]

# Calculate correlation matrix
correlation_matrix = data_without_date.corr()

# Print correlation values for all pairs of variables
for i in range(len(correlation_matrix.columns)):
    for j in range(i+1, len(correlation_matrix.columns)):
        variable1 = correlation_matrix.columns[i]
        variable2 = correlation_matrix.columns[j]
```

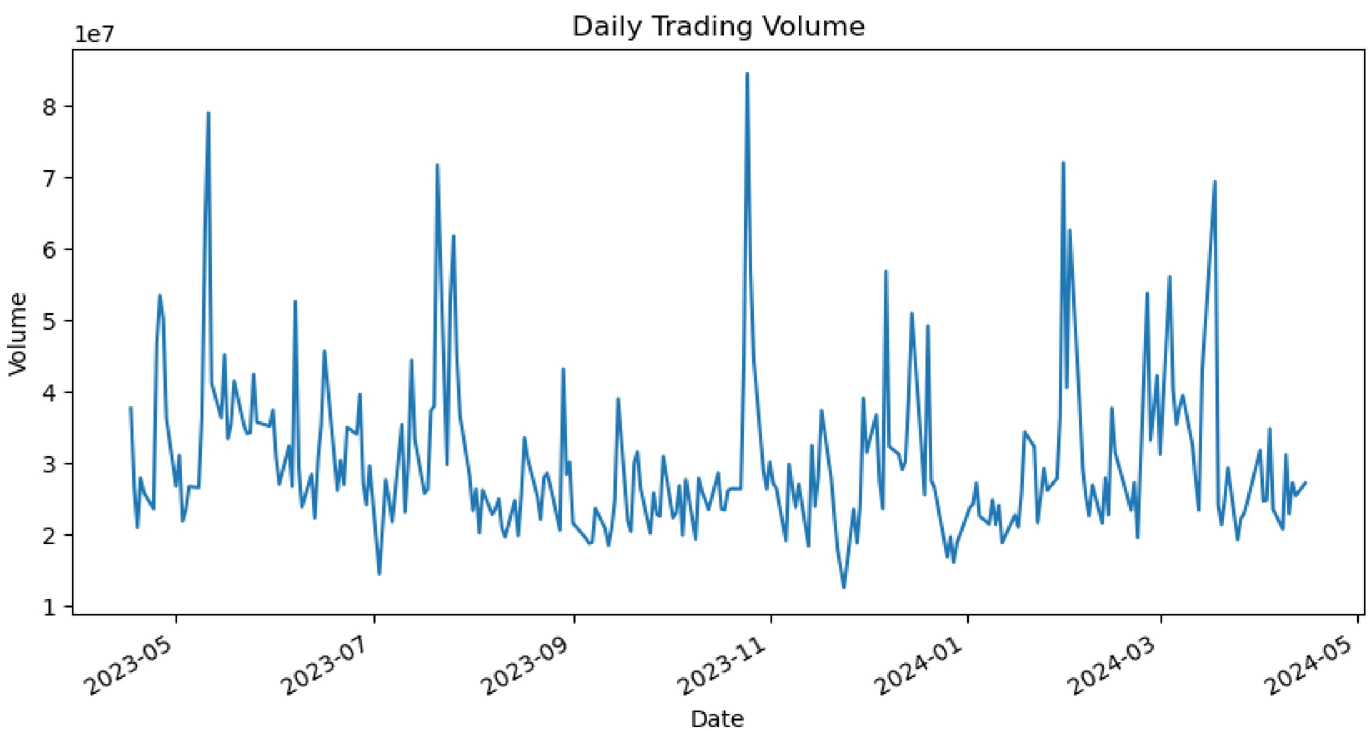
```
        correlation_value = correlation_matrix.iloc[i, j]
        print(f"Correlation between {variable1} and {variable2}: {correlation_value:.2f}
```

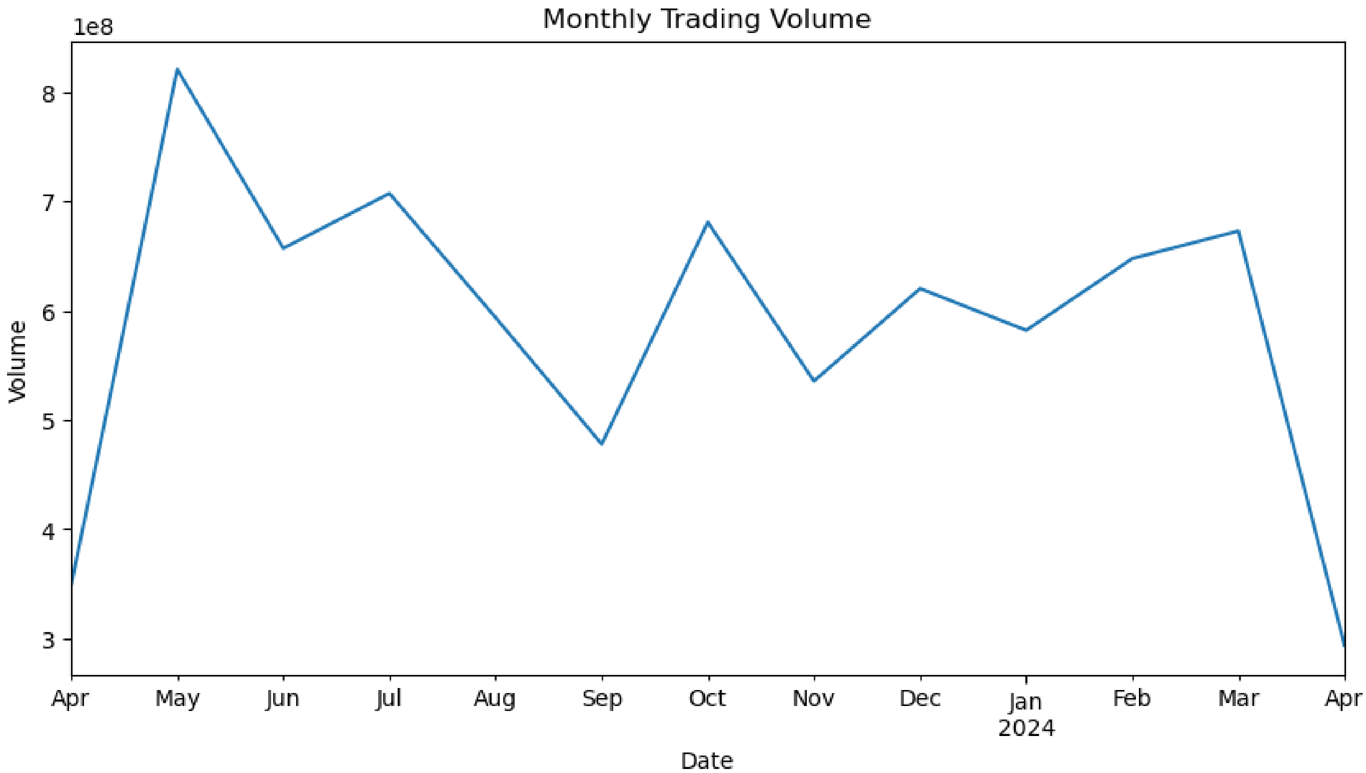
```
Correlation between Close/Last and Volume: -0.21
Correlation between Close/Last and Open: 0.99
Correlation between Close/Last and High: 1.00
Correlation between Close/Last and Low: 1.00
Correlation between Volume and Open: -0.20
Correlation between Volume and High: -0.18
Correlation between Volume and Low: -0.22
Correlation between Open and High: 1.00
Correlation between Open and Low: 1.00
Correlation between High and Low: 1.00
```

Technical Analysis

In [13]:

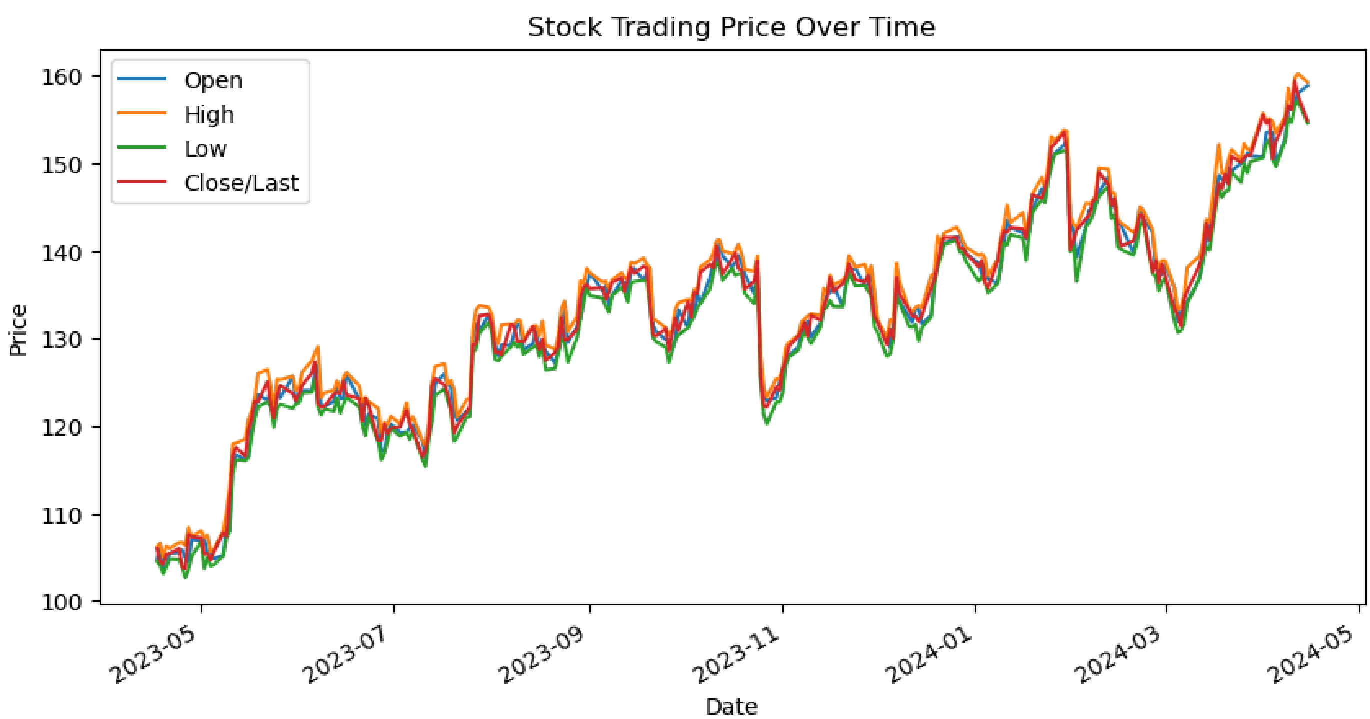
```
# Volume Graphs by Day, Month, and Year

if not pd.api.types.is_datetime64_any_dtype(data['Date']):
    data['Date'] = pd.to_datetime(data['Date'], unit='d', origin='1899-12-30')
data.set_index('Date', inplace=True)

data['Volume'].plot(title='Daily Trading Volume', figsize=(10, 5))
plt.ylabel('Volume')
plt.show()

monthly_volume = data['Volume'].resample('M').sum()
monthly_volume.plot(title='Monthly Trading Volume', figsize=(10, 5))
plt.ylabel('Volume')
plt.show()
```





In [14]:

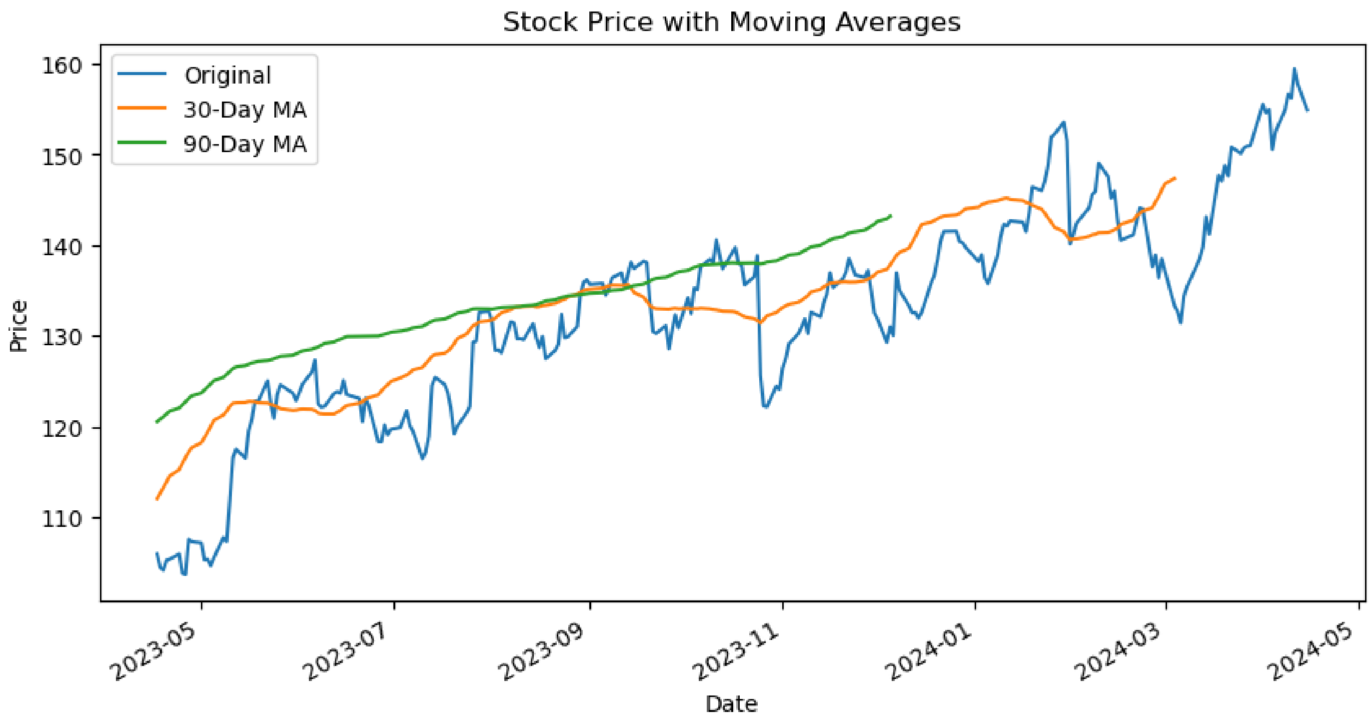
```
# Visualizing Time Series Data: Stock Trading Price Over Time

data[['Open', 'High', 'Low', 'Close/Last']].plot(title='Stock Trading Price Over Time',
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```



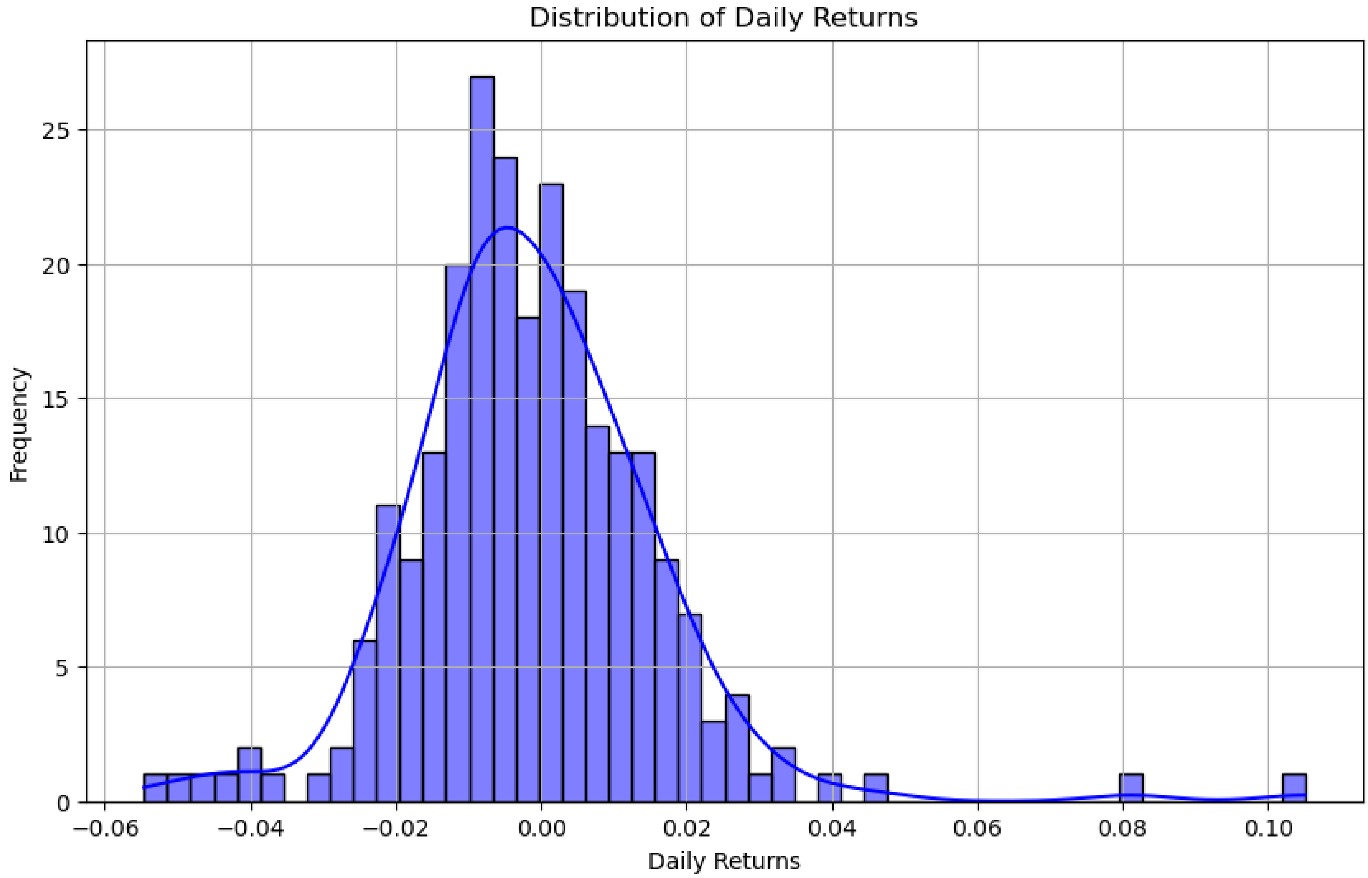
In [15]:

```
# Analyzing Stock Price Trends with Moving Averages
data['Close/Last'].plot(label='Original', figsize=(10, 5))
data['Close/Last'].rolling(window=30).mean().plot(label='30-Day MA')
data['Close/Last'].rolling(window=90).mean().plot(label='90-Day MA')
plt.title('Stock Price with Moving Averages')
plt.xlabel('Date')
plt.ylabel('Price')
```

```
plt.legend()
plt.show()
```



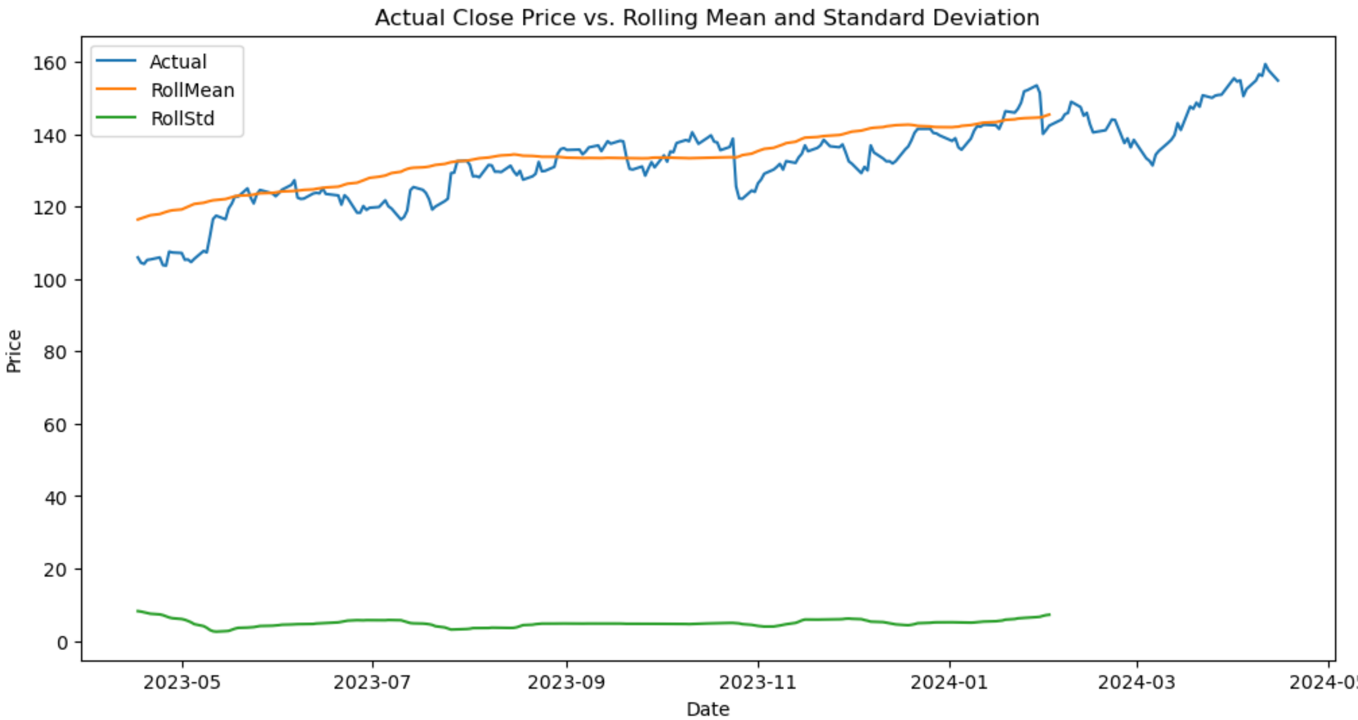
In [16]:
```
# Daily Returns Analysis of Stock

import seaborn as sns
data['Daily Returns'] = data['Close/Last'].pct_change()
desc_stats = data['Daily Returns'].describe()

plt.figure(figsize=(10, 6))
sns.histplot(data['Daily Returns'], bins=50, kde=True, color='blue')
plt.title('Distribution of Daily Returns')
plt.xlabel('Daily Returns')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Display Descriptive Statistics
print(desc_stats)
```




```
count    250.000000
mean      -0.001369
std        0.017310
min       -0.054614
25%       -0.010674
50%       -0.001869
75%        0.007566
max        0.105087
Name: Daily Returns, dtype: float64
```

In [17]:

```
# Rolling windows to calculate the rolling mean and rolling standard deviation for the '

data['RollMean'] = data['Close/Last'].rolling(window=50).mean()
data['RollStd'] = data['Close/Last'].rolling(window=50).std()

plt.figure(figsize=(12, 6))
plt.plot(data['Close/Last'], label='Actual')
plt.plot(data['RollMean'], label='RollMean')
plt.plot(data['RollStd'], label='RollStd')
plt.legend()
plt.title('Actual Close Price vs. Rolling Mean and Standard Deviation')
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```

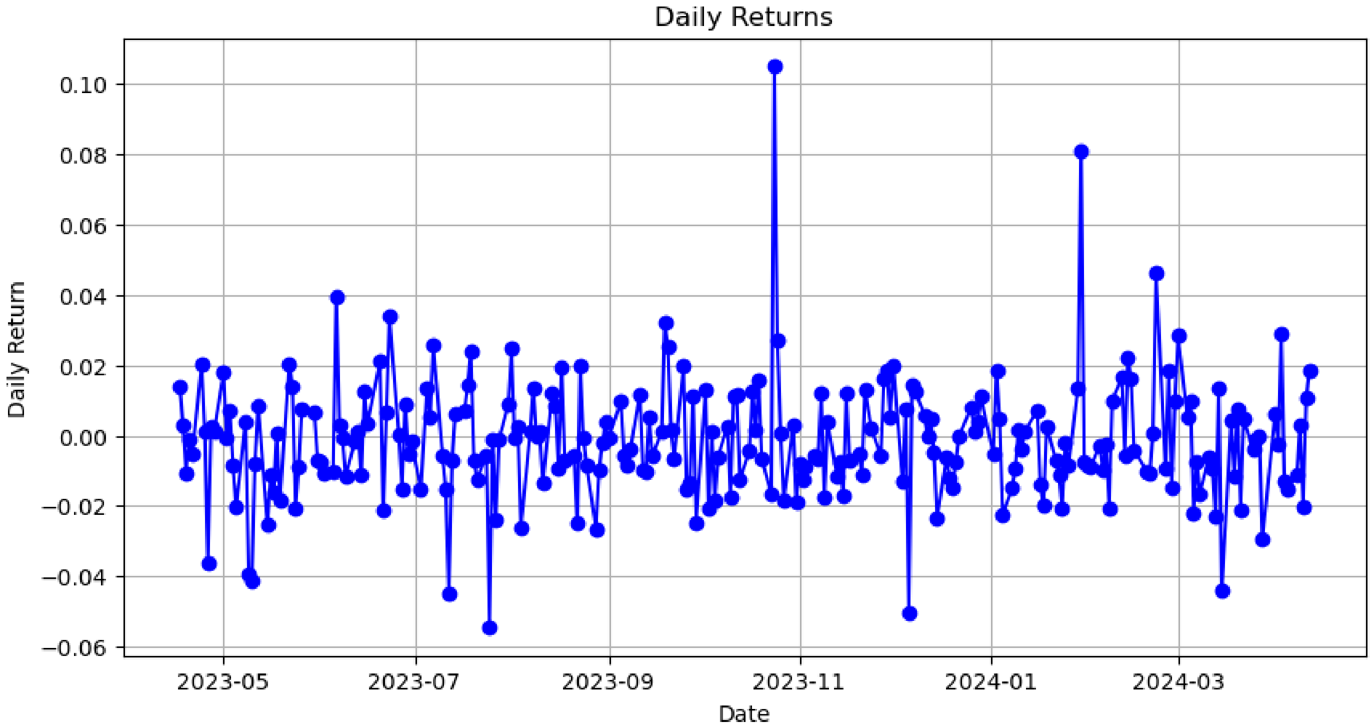


In [19]:
```
# Volatility Measurement: Calculate daily price range and standard deviation

data['Daily Range'] = data['High'] - data['Low']
volatility = data['Close/Last'].std()
print(f"Volatility (Standard Deviation of Closing Prices): {volatility:.2f}")
```

```
Volatility (Standard Deviation of Closing Prices): 11.88
```

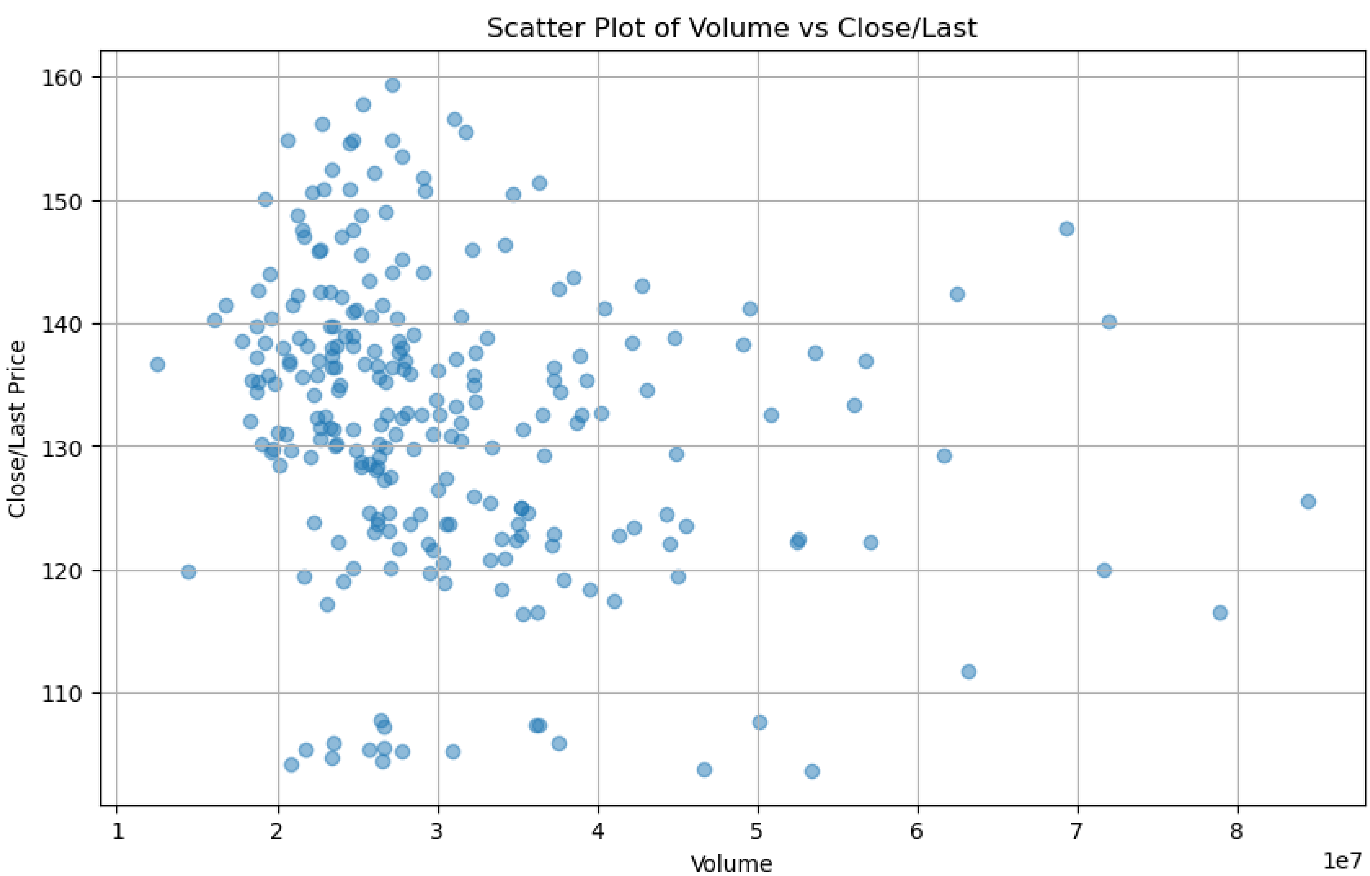
In [43]:
```
# Risk and Return: Calculate average daily return and risk

data['Daily Return'] = data['Close/Last'].pct_change()

# Calculate average daily return and risk
average_return = data['Daily Return'].mean()
risk = data['Daily Return'].std()
print(f"Average Daily Return: {average_return:.4f}")
print(f"Risk (Standard Deviation of Daily Returns): {risk:.4f}")

# Plot daily returns to visualize risk
plt.figure(figsize=(10, 5))
plt.plot(data.index, data['Daily Return'], marker='o', linestyle='-', color='blue')
plt.title('Daily Returns')
plt.xlabel('Date')
plt.ylabel('Daily Return')
plt.grid(True)
plt.show()
```

```
Average Daily Return: -0.0014
Risk (Standard Deviation of Daily Returns): 0.0173
```



In [45]:

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

plt.figure(figsize=(10, 6))
plt.scatter(data['Volume'], data['Close/Last'], alpha=0.5)
plt.title('Scatter Plot of Volume vs Close/Last')
plt.xlabel('Volume')
plt.ylabel('Close/Last Price')
plt.grid(True)
plt.show()
```



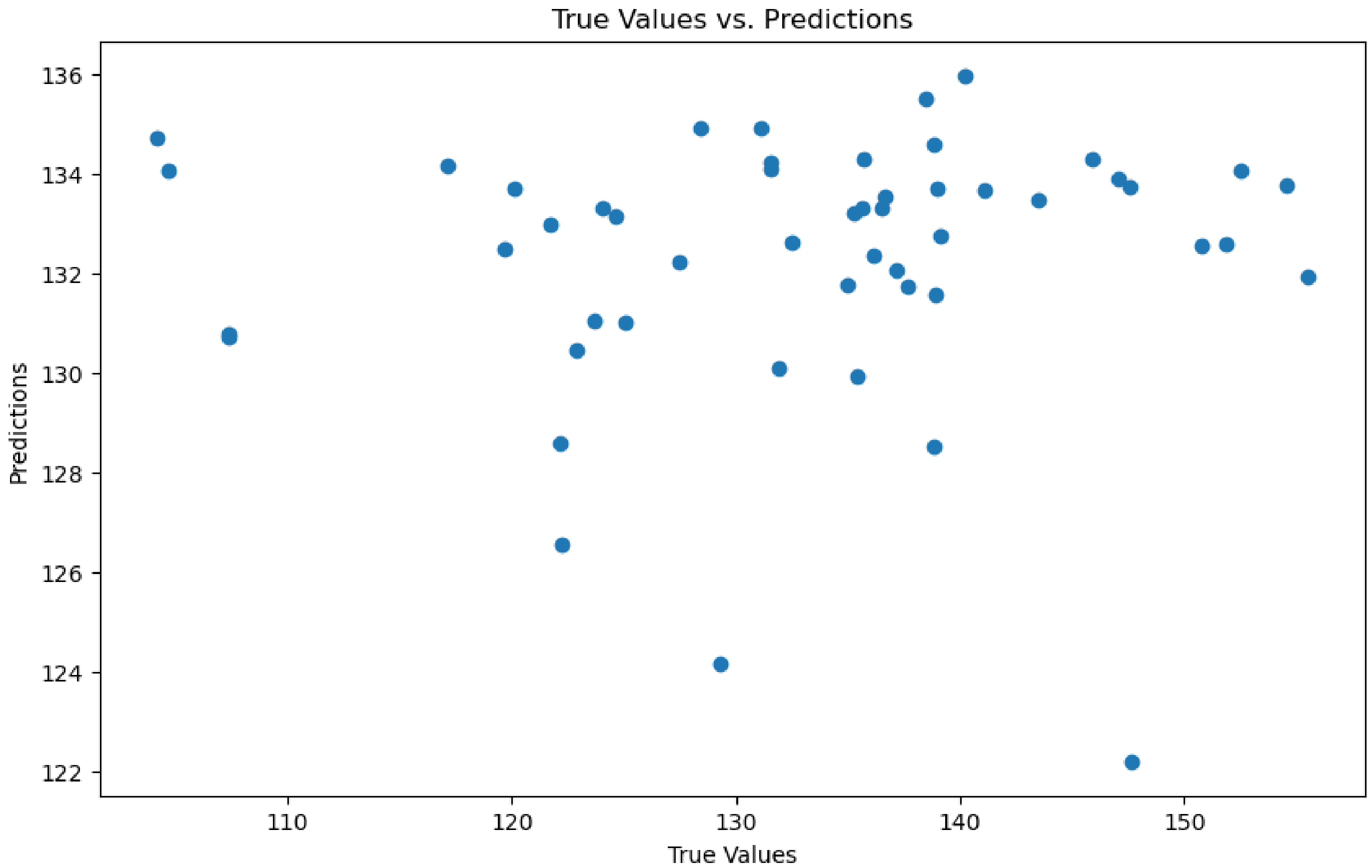
In [ ]:

```
Regression Analysis to Predict Stock Price
```

In [52]:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm

# Assuming 'data' contains your dataset with columns: 'Open', 'High', 'Low', and 'Close/

# Select features and target variable
X = data[['Volume']]  # Features
y = data['Close/Last']  # Target variable

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

# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the testing set
predictions = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error: {mse:.2f}")

# Optionally, you can visualize the predictions vs. actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True Values vs. Predictions')
plt.show()

# Fit the OLS regression model
model = sm.OLS(y, X).fit()

# Display regression summary
print(model.summary())
```

```
Mean Squared Error: 154.87
```

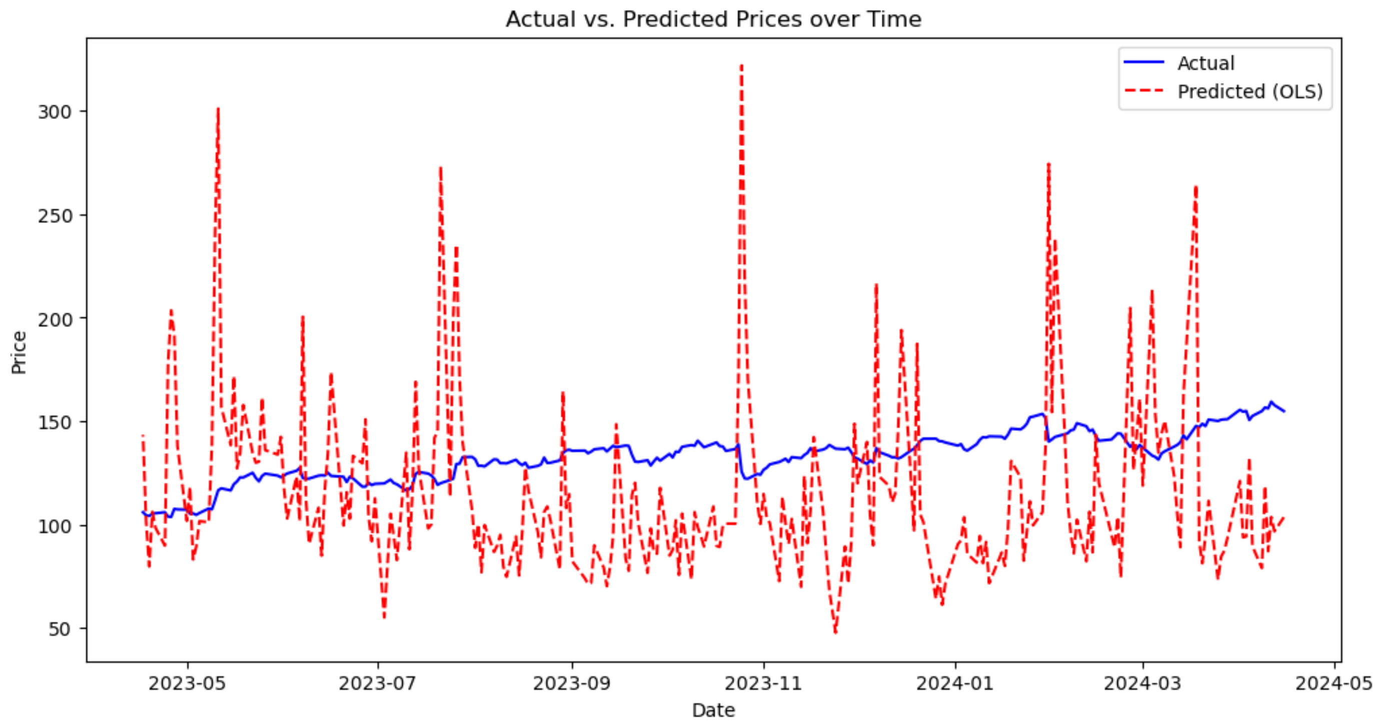



```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:             Close/Last   R-squared (uncentered):                   0.864
Model:                            OLS   Adj. R-squared (uncentered):              0.863
Method:                 Least Squares   F-statistic:                              1584.
Date:                Tue, 16 Apr 2024   Prob (F-statistic):                   3.48e-110
Time:                        02:12:04   Log-Likelihood:                         -1333.4
No. Observations:                 251   AIC:                                      2669.
Df Residuals:                     250   BIC:                                      2672.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Volume      3.815e-06   9.59e-08     39.804      0.000    3.63e-06       4e-06
==============================================================================
Omnibus:                       91.173   Durbin-Watson:                   0.800
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              249.017
Skew:                          -1.649   Prob(JB):                     8.45e-55
Kurtosis:                       6.597   Cond. No.                         1.00
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a con
stant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```

In [53]:

```
# Plot actual and predicted prices over time

model_ols = sm.OLS(y, X).fit()
predicted_values_ols = model_ols.predict(X)

plt.figure(figsize=(12, 6))
plt.plot(data.index, y, label='Actual', color='blue')
plt.plot(data.index, predicted_values_ols, label='Predicted (OLS)', linestyle='--', colo
plt.title('Actual vs. Predicted Prices over Time')
```

```
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```



Peer Analysis

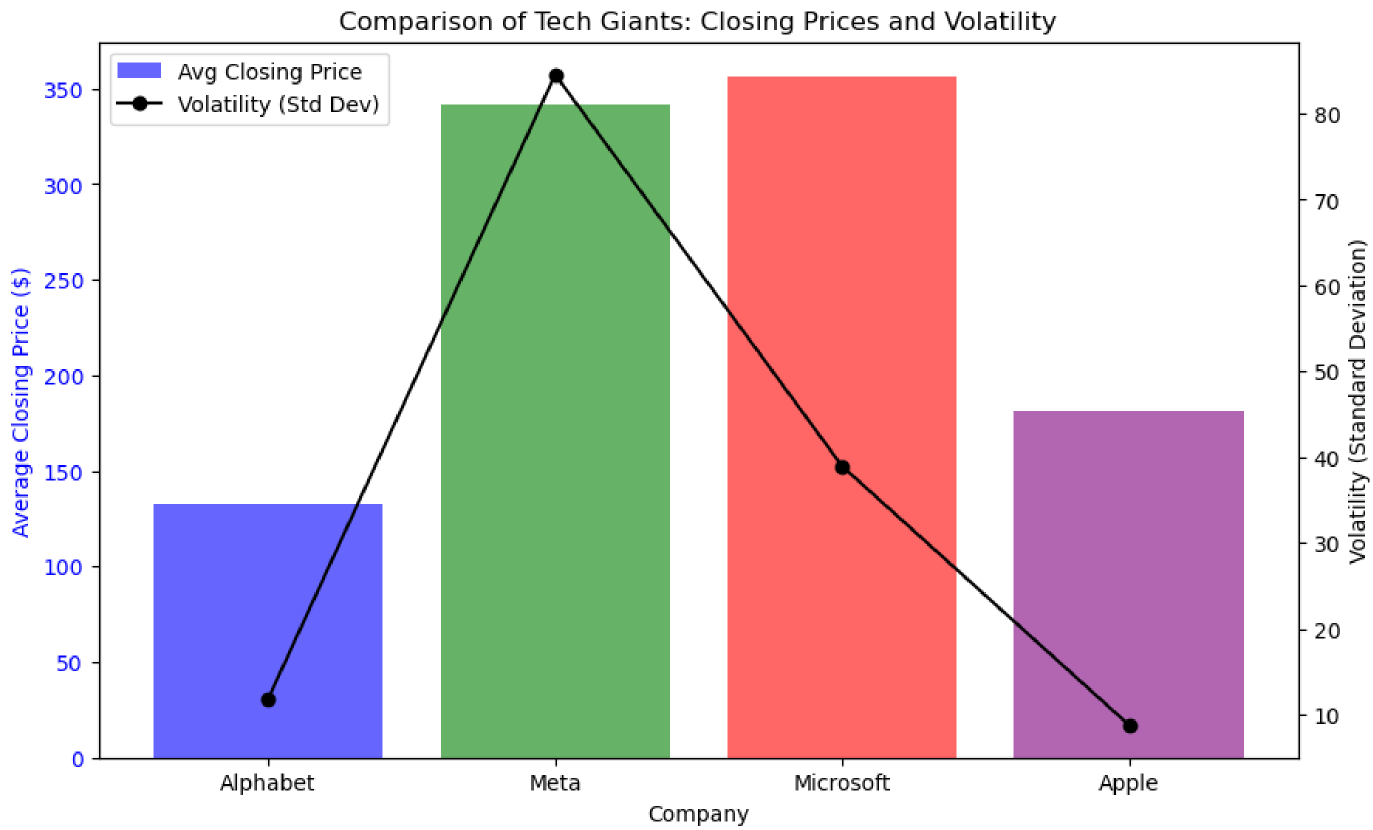
In [2]:

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

data = pd.read_excel('Downloads/Dataset - Business Metrics.xlsx')
data1 = pd.read_excel('Downloads/Meta Dataset.xlsx')
data2 = pd.read_excel('Downloads/Microsoft Dataset.xlsx')
data3 = pd.read_excel('Downloads/Apple.xlsx')
```

In [3]:

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


datasets = [data, data1, data2, data3]
for dataset in datasets:
    if 'Close/Last' in dataset.columns:
        dataset['Close/Last'] = dataset['Close/Last'].replace('[\$,]', '', regex=True).a
    elif 'Close' in dataset.columns:
        dataset['Close'] = dataset['Close'].replace('[\$,]', '', regex=True).astype(floa

metrics = pd.DataFrame({
    'Company': ['Alphabet', 'Meta', 'Microsoft', 'Apple'],
    'Avg Closing Price': [df['Close/Last'].mean() if 'Close/Last' in df.columns else df[
    'Volatility (Std Dev)': [df['Close/Last'].std() if 'Close/Last' in df.columns else d
})

fig, ax1 = plt.subplots(figsize=(10, 6))
```

```
colors = ['blue', 'green', 'red', 'purple']
ax1.bar(metrics['Company'], metrics['Avg Closing Price'], color=colors, alpha=0.6, label
ax1.set_xlabel('Company')
ax1.set_ylabel('Average Closing Price ($)', color='b')
ax1.tick_params(axis='y', labelcolor='b')

ax2 = ax1.twinx()
ax2.plot(metrics['Company'], metrics['Volatility (Std Dev)'], 'k-o', label='Volatility (
ax2.set_ylabel('Volatility (Standard Deviation)', color='k')
ax2.tick_params(axis='y', labelcolor='k')

lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, loc='upper left')

plt.title('Comparison of Tech Giants: Closing Prices and Volatility')
plt.show()
```




```
In [ ]:
```