Name - Samar

Suryawanshi

Uid- 2023701007

Branch - CSE-DS-BATCH-C

Aim- Exp5_Working with Python Matplotlib Library

Dataset Used - Twitter Stock Market Dataset

Dataset Link - https://www.kaggle.com/datasets/amirmotefaker/twitter-stock-market-dataset/code

In [27]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

In [28]:
```
df= pd.read_csv("/content/Twitter Stock Market Dataset.csv")
```

In [29]:
```
df.describe()
```

Out[29]:

	Open	High	Low	Close	Adj Close	Volume
count	2259.000000	2259.000000	2259.000000	2259.000000	2259.000000	2.259000e+03
mean	36.020286	36.699881	35.339465	36.003625	36.003625	2.175186e+07
std	14.118463	14.372057	13.828724	14.089989	14.089989	1.909988e+07
min	13.950000	14.220000	13.725000	14.010000	14.010000	0.000000e+00
25%	25.550000	26.215001	24.912501	25.410000	25.410000	1.233530e+07
50%	35.419998	36.099998	34.820000	35.490002	35.490002	1.691305e+07
75%	44.205000	45.015000	43.327501	44.135000	44.135000	2.428082e+07
max	78.360001	80.750000	76.050003	77.629997	77.629997	2.692131e+08

Line Graphs / Line Charts

Description: Line graphs plot data points connected by straight lines. They are commonly used to show trends over time.

Interpretation: Line charts are ideal for visualizing stock price movements and other time-series data. By plotting the Open, High, Low, Close, and Adj Close prices over time, you can observe trends, cyclical patterns, and volatility. For instance, a rising line indicates increasing stock prices, while a falling line shows declining prices. Peaks and troughs can indicate periods of high volatility or significant market events.

In [30]:
```
# Sample Data (Replace with your actual data)
dates = ['2024-01-01', '2024-01-02', '2024-01-03']  # Replace with actual dates
close_prices = [36.00, 36.50, 36.10]  # Replace with actual close prices

plt.figure(figsize=(12, 6))
plt.plot(dates, close_prices, marker='o', linestyle='-', color='b')
plt.title('Stock Closing Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
```

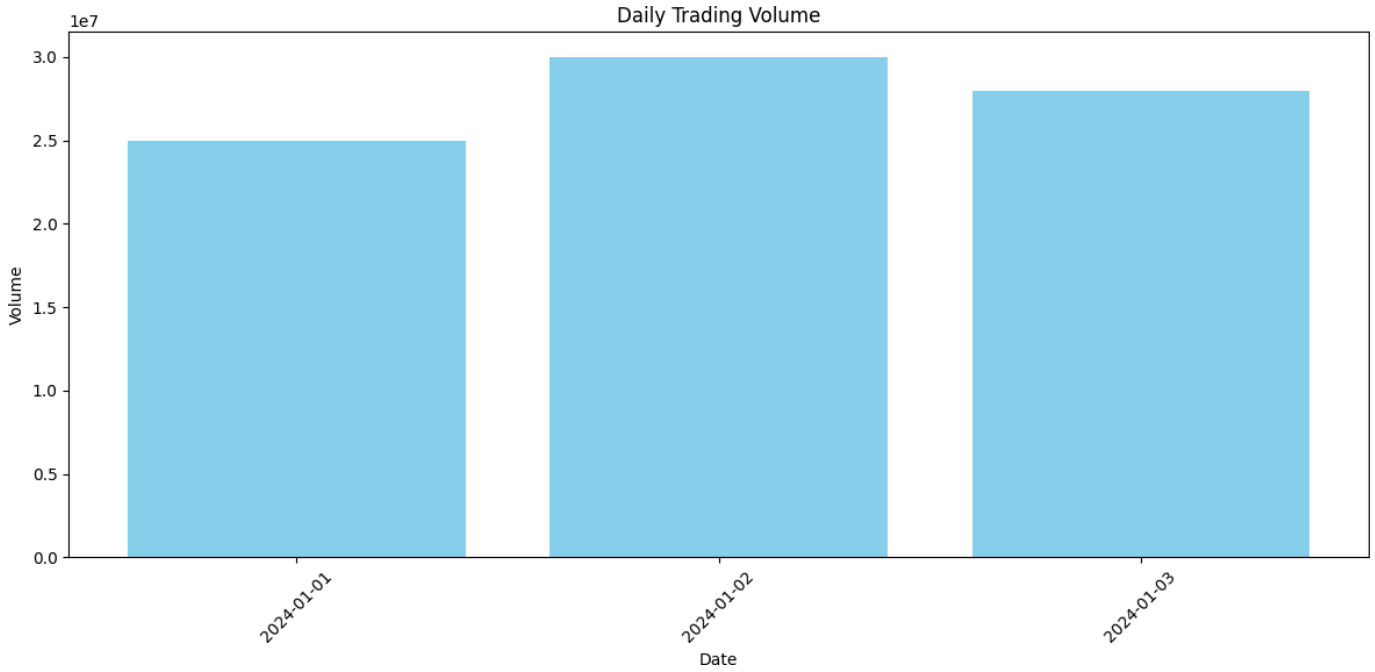
```
plt.tight_layout()
plt.show()
```

Bar Graphs / Bar Charts

Description: Bar charts use rectangular bars to represent the frequency or value of data. They are useful for comparing quantities across different categories.

Interpretation: In the context of stock data, bar charts can be used to display the trading volume for each day or month. By plotting these volumes, you can easily identify periods of high or low trading activity. Spikes in the bars might correspond to significant market events or news releases, while low bars indicate quieter periods.

In [31]:
```
# Sample Data (Replace with your actual data)
dates = ['2024-01-01', '2024-01-02', '2024-01-03']  # Replace with actual dates
volumes = [2.5e7, 3.0e7, 2.8e7]  # Replace with actual trading volumes

plt.figure(figsize=(12, 6))
plt.bar(dates, volumes, color='skyblue')
plt.title('Daily Trading Volume')
plt.xlabel('Date')
plt.ylabel('Volume')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Scatter Plots

Description: Scatter plots display individual data points on a Cartesian plane, showing the relationship between two variables.

Interpretation: Scatter plots are useful for exploring correlations between different stock metrics. For example, plotting Close prices against Volume can reveal whether there is a relationship between trading volume and price levels. Clusters of points might suggest trends, while outliers could indicate unusual trading days.

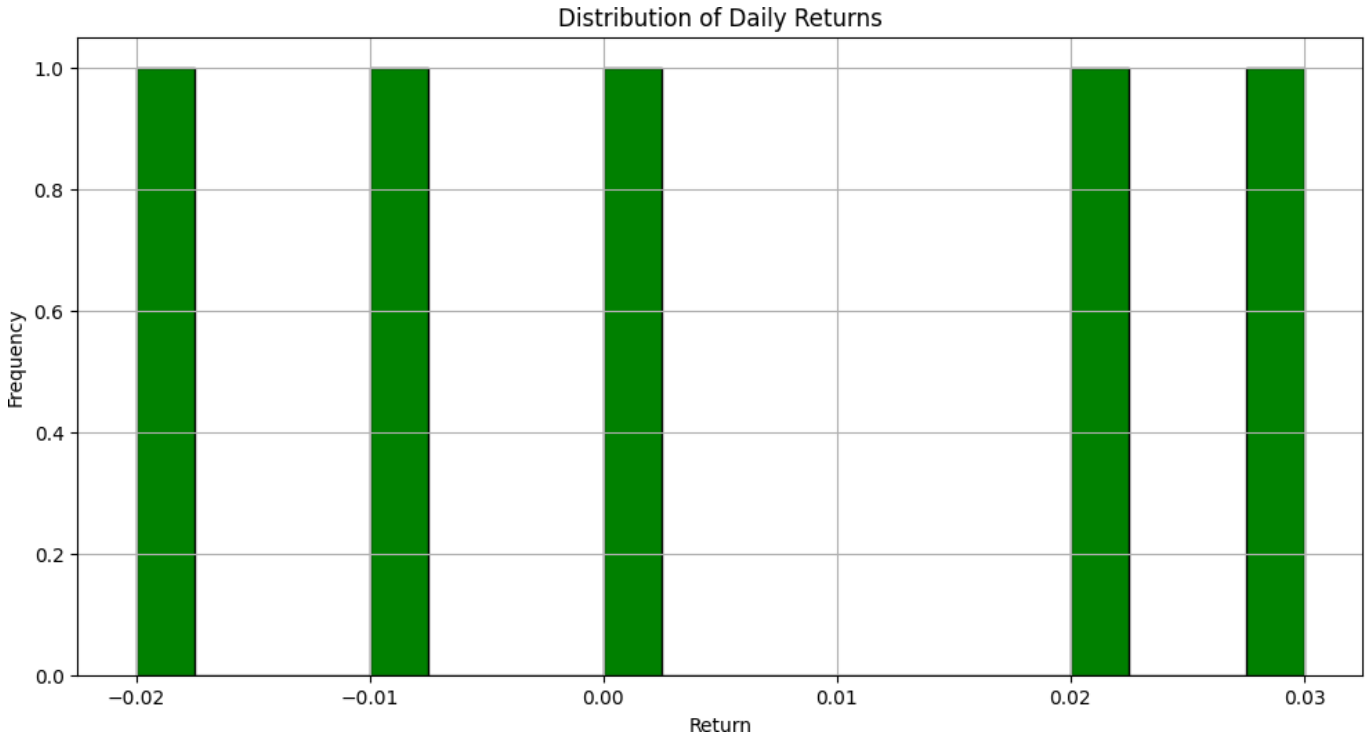
In [32]:
```
# Sample Data (Replace with your actual data)
close_prices = [36.00, 36.50, 36.10]  # Replace with actual close prices
volumes = [2.5e7, 3.0e7, 2.8e7]  # Replace with actual trading volumes

plt.figure(figsize=(12, 6))
plt.scatter(close_prices, volumes, color='r')
plt.title('Close Prices vs Trading Volume')
plt.xlabel('Close  Price')
plt.ylabel('Volume')
plt.grid(True)
plt.show()
```

Histograms

Description: Histograms show the distribution of a dataset by grouping data into bins and displaying the frequency of data points in each bin.

Interpretation: Histograms can be used to analyze the distribution of stock prices, trading volumes, or returns. For example, a histogram of daily returns can reveal whether the returns are normally distributed or skewed. Understanding this distribution helps in assessing the risk and volatility of the stock.

In [33]:
```
# Sample Data (Replace with your actual data)
returns = [0.02, -0.01, 0.03, 0.00, -0.02]  # Replace with actual daily returns

plt.figure(figsize=(12, 6))
plt.hist(returns, bins=20, color='g', edgecolor='black')
plt.title('Distribution of Daily Returns')
plt.xlabel('Return')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

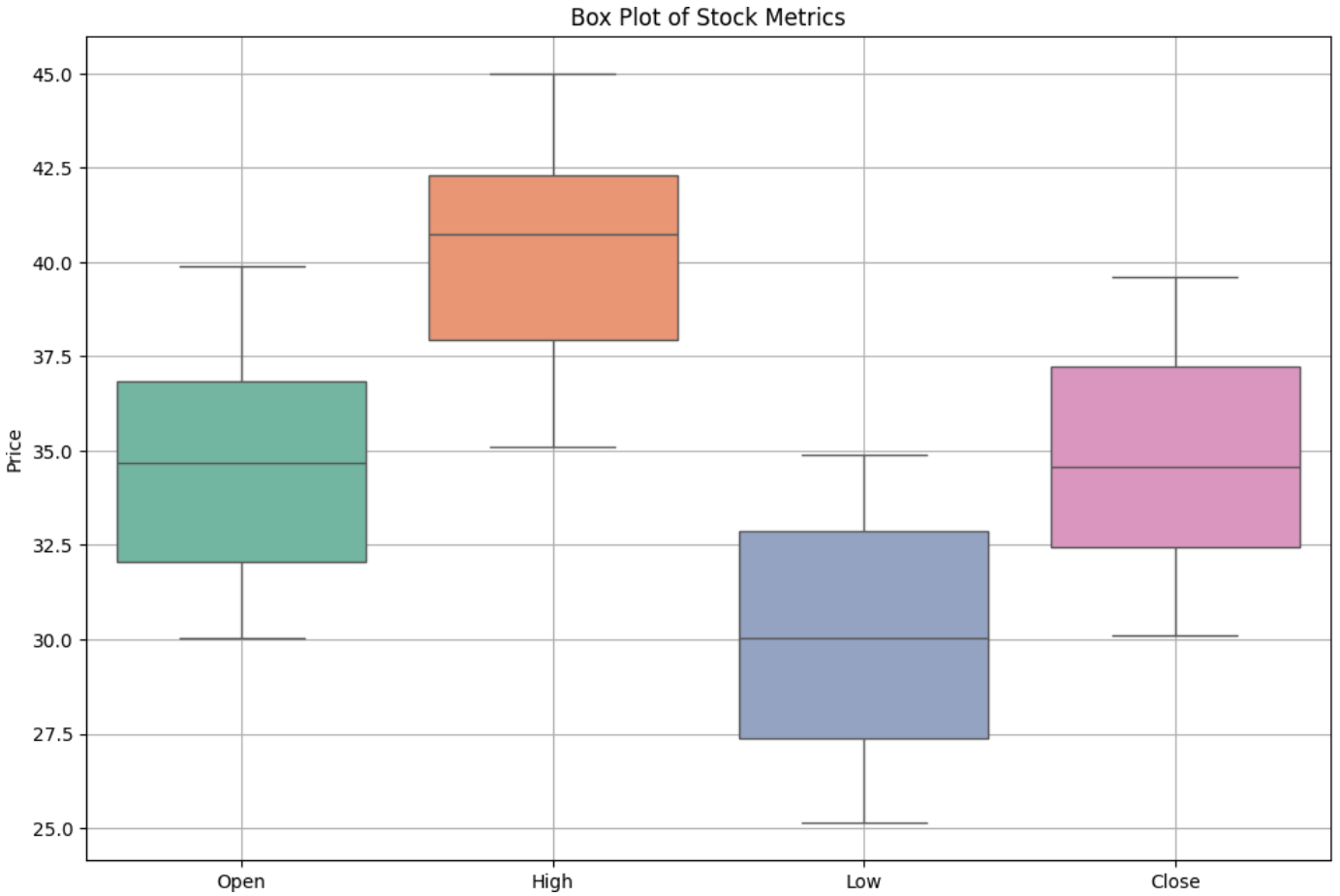


Box Plots

Description: Box plots summarize the distribution of a dataset by displaying the median, quartiles, and potential outliers.

Interpretation: Box plots are valuable for comparing the spread and central tendency of different stock price metrics. For instance, a box plot of Close prices can highlight the median price, the range within which most prices fall, and any outliers. This can help identify periods of extreme price movements.

In [34]:
```
# Sample Data (Replace with your actual data)
np.random.seed(0)
data = {
    'Open': np.random.uniform(30, 40, 100),
    'High': np.random.uniform(35, 45, 100),
    'Low': np.random.uniform(25, 35, 100),
    'Close': np.random.uniform(30, 40, 100)
}
df = pd.DataFrame(data)

plt.figure(figsize=(12, 8))
sns.boxplot(data=df, palette='Set2')
plt.title('Box Plot of Stock Metrics')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```



Violin Plots

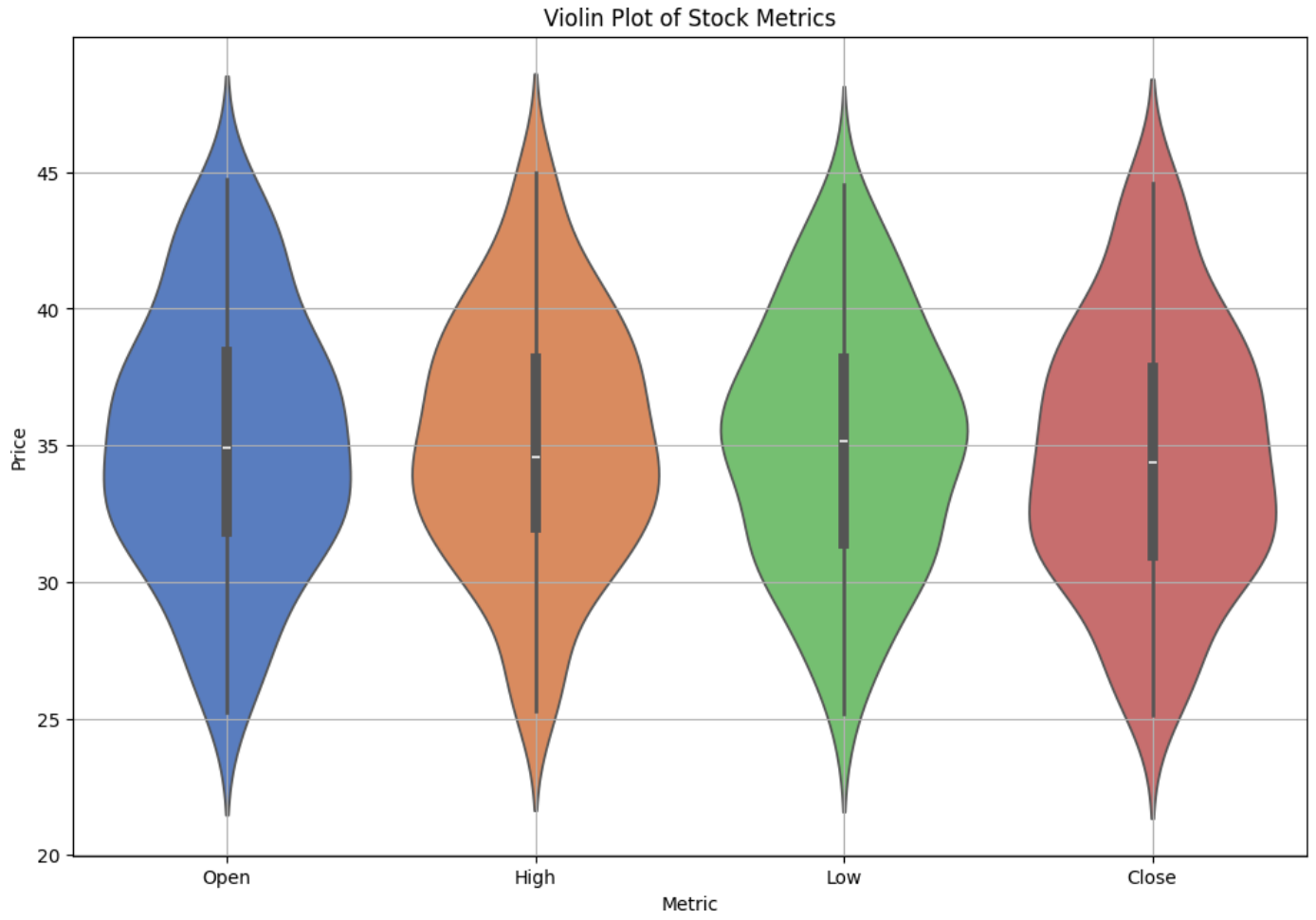
Description: Violin plots combine box plots and density plots to show the distribution of data across different categories.

Interpretation: Violin plots provide a detailed view of the distribution of stock prices or volumes, revealing the probability density of the data across different price levels. This visualization is useful for understanding how stock prices are distributed and for identifying multimodal distributions.

In [35]:
```
# Sample Data (Replace with your actual data)
np.random.seed(0)
data = {
    'Metric': np.tile(['Open', 'High', 'Low', 'Close'], 100),
    'Value': np.concatenate([
        np.random.uniform(30,  40, 100),
        np.random.uniform(35,  45, 100),
        np.random.uniform(25,  35, 100),
        np.random.uniform(30,  40, 100)
    ])
}
df = pd.DataFrame(data)

plt.figure(figsize=(12,  8))
sns.violinplot(x='Metric', y='Value', data=df, palette='muted')
plt.title('Violin Plot of Stock Metrics')
plt.xlabel('Metric')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```

```
<ipython-input-35-72ec38adde52>:15:  FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
```

```
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.violinplot(x='Metric', y='Value', data=df, palette='muted')
```

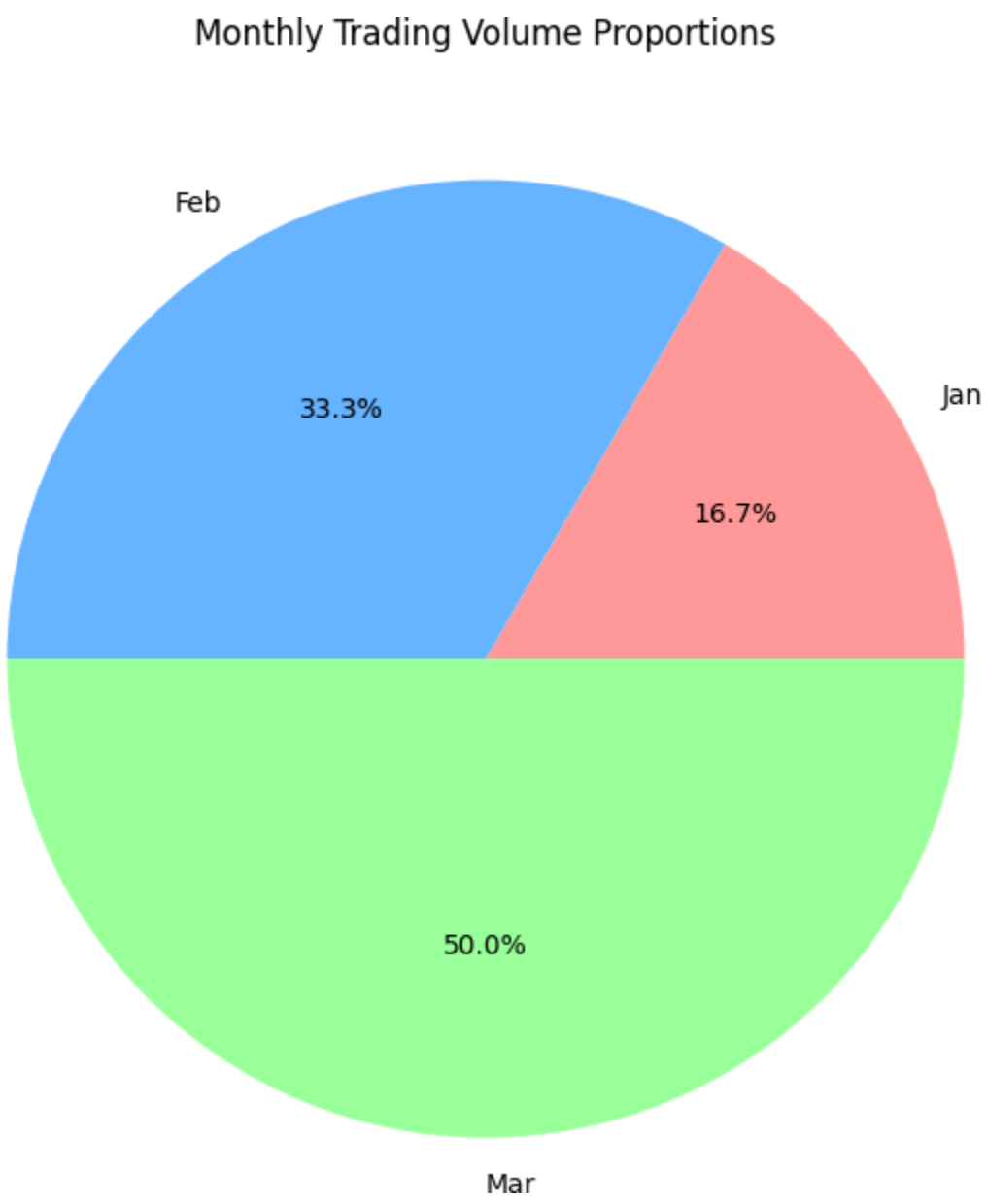


Pie Charts

Description: Pie charts show the proportion of each category as a slice of a circle.

Interpretation: Although less common for time-series data, pie charts can illustrate the proportion of total trading volume contributed by different months or years. This can help identify periods of high or low trading activity relative to the total trading volume.

In [36]:
```
# Sample Data (Replace with your actual data)
labels = ['Jan', 'Feb', 'Mar']  # Replace with actual months
sizes = [15, 30, 45]  # Replace with actual trading volumes or any other metric

plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['#ff9999','#66b3ff','#99ff99'])
plt.title('Monthly Trading Volume Proportions')
plt.show()
```



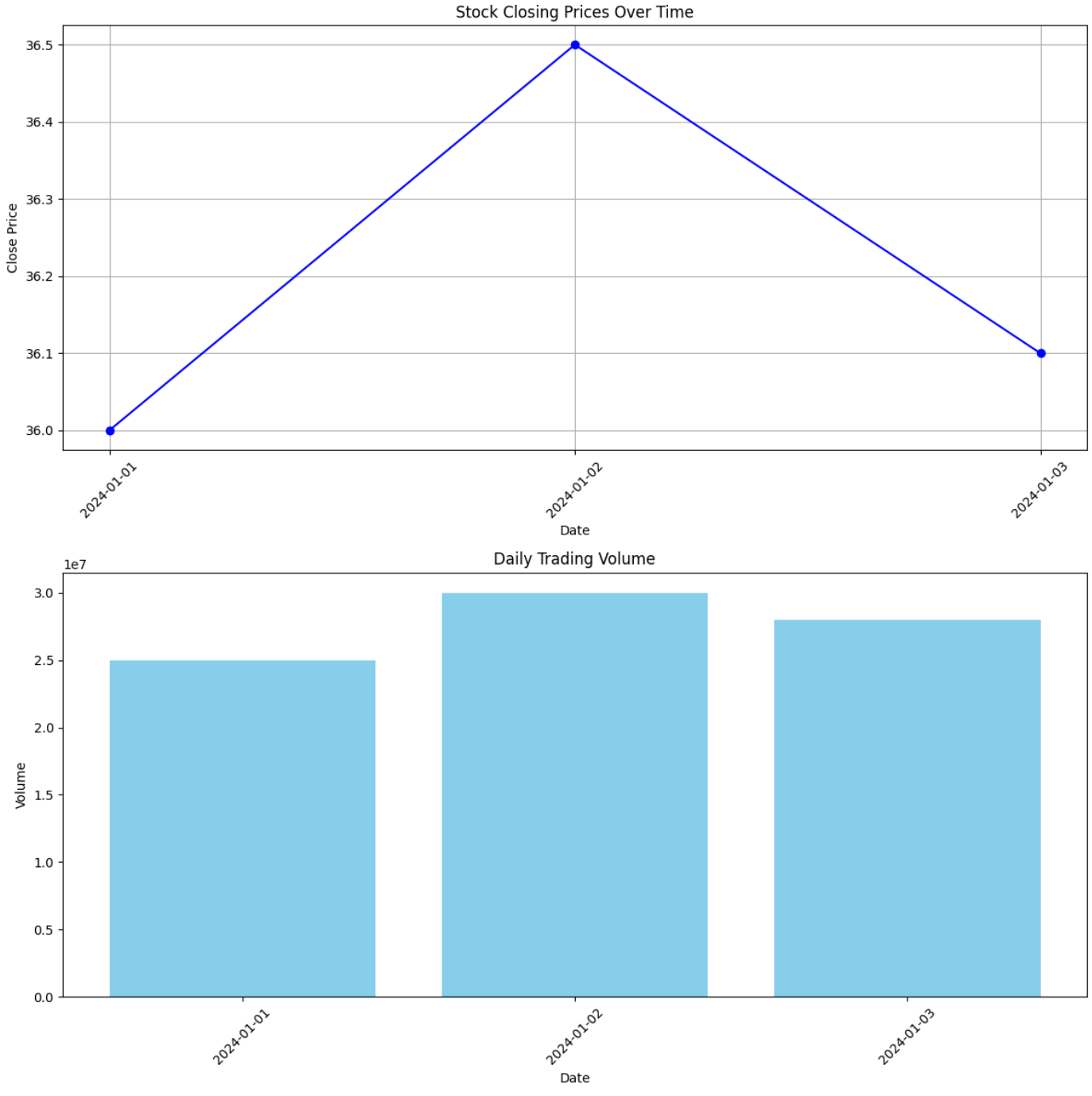
Subplots

Description: Subplots allow multiple plots to be displayed in a single figure, facilitating comparison across different datasets or metrics.

Interpretation: By arranging multiple subplots, you can compare various aspects of the stock data simultaneously. For instance, one subplot could show daily stock prices, while another displays trading volumes, making it easier to correlate price movements with trading activity.

In [37]:
```
# Sample Data (Replace with your actual data)
dates = ['2024-01-01', '2024-01-02', '2024-01-03']
close_prices = [36.00, 36.50, 36.10]
volumes = [2.5e7, 3.0e7, 2.8e7]

fig, axs = plt.subplots(2, 1, figsize=(12, 12))

# Plot 1: Line Graph
axs[0].plot(dates, close_prices, marker='o', linestyle='-', color='b')
axs[0].set_title('Stock Closing Prices Over Time')
axs[0].set_xlabel('Date')
axs[0].set_ylabel('Close  Price')
```

```
axs[0].grid(True)
axs[0].tick_params(axis='x', rotation=45)

# Plot 2: Bar Chart
axs[1].bar(dates, volumes, color='skyblue')
axs[1].set_title('Daily Trading Volume')
axs[1].set_xlabel('Date')
axs[1].set_ylabel('Volume')
axs[1].tick_params(axis='x', rotation=45)

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



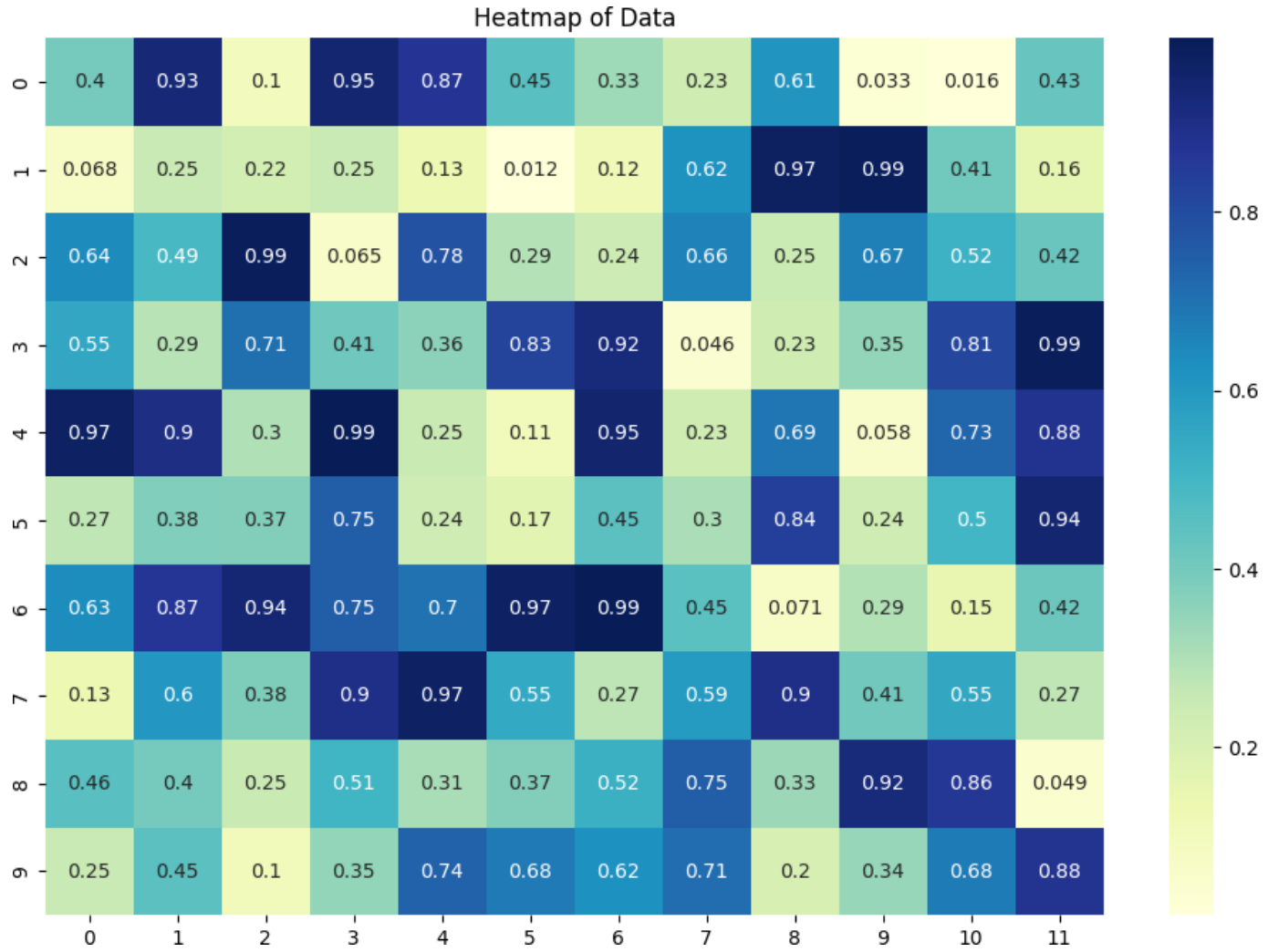
Additional Plots

Heatmaps

Description: Heatmaps use color gradients to show the intensity of values across a matrix.

Interpretation: A heatmap of daily returns or trading volumes can visually represent fluctuations and patterns. For example, higher values could be shown in warmer colors, making it easy to spot trends and anomalies.

In [38]:
```
# Sample Data (Replace with your actual data)
data = np.random.rand(10, 12)  # Replace with actual data matrix

plt.figure(figsize=(12, 8))
sns.heatmap(data, cmap='YlGnBu', annot=True)
plt.title('Heatmap of Data')
plt.show()
```



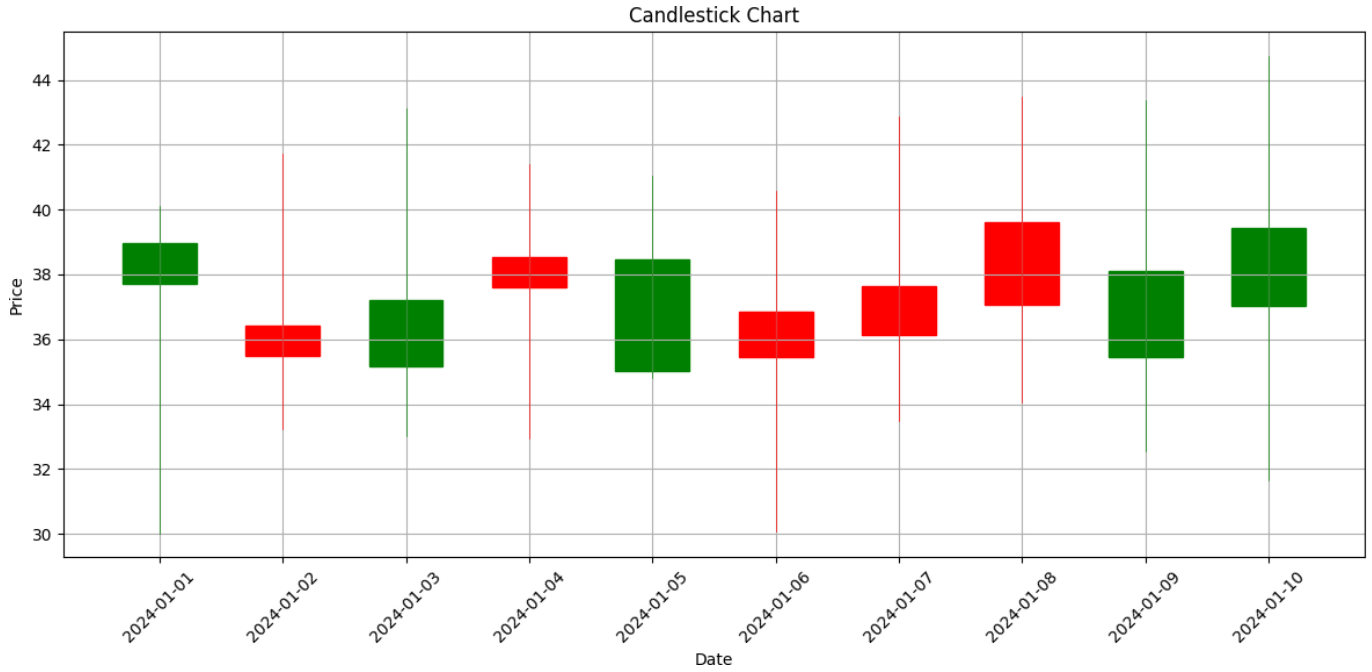
Candlestick Charts

Description: Candlestick charts show open, high, low, and close prices for each time period, providing a detailed view of price movements.

Interpretation: Candlestick charts are essential for technical analysis, showing price trends and volatility. Each candlestick represents a specific period and provides insights into price trends, potential reversals, and market sentiment.

In [39]:
```
import matplotlib.dates as mdates
from mpl_finance import candlestick_ohlc

# Sample Data (Replace with your actual data)
data = {
    'Date': pd.date_range(start='2024-01-01', periods=10),
    'Open': np.random.uniform(35, 40, 10),
```

```
    'High': np.random.uniform(40, 45, 10),
    'Low': np.random.uniform(30, 35, 10),
    'Close': np.random.uniform(35, 40, 10)
}
df = pd.DataFrame(data)
df['Date'] = mdates.date2num(df['Date'])

fig, ax = plt.subplots(figsize=(12, 6))
candlestick_ohlc(ax, df[['Date', 'Open', 'High', 'Low', 'Close']].values, width=0.6, col

ax.xaxis_date()
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.title('Candlestick Chart')
plt.xlabel('Date')
plt.ylabel('Price')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Rolling Mean and Standard Deviation

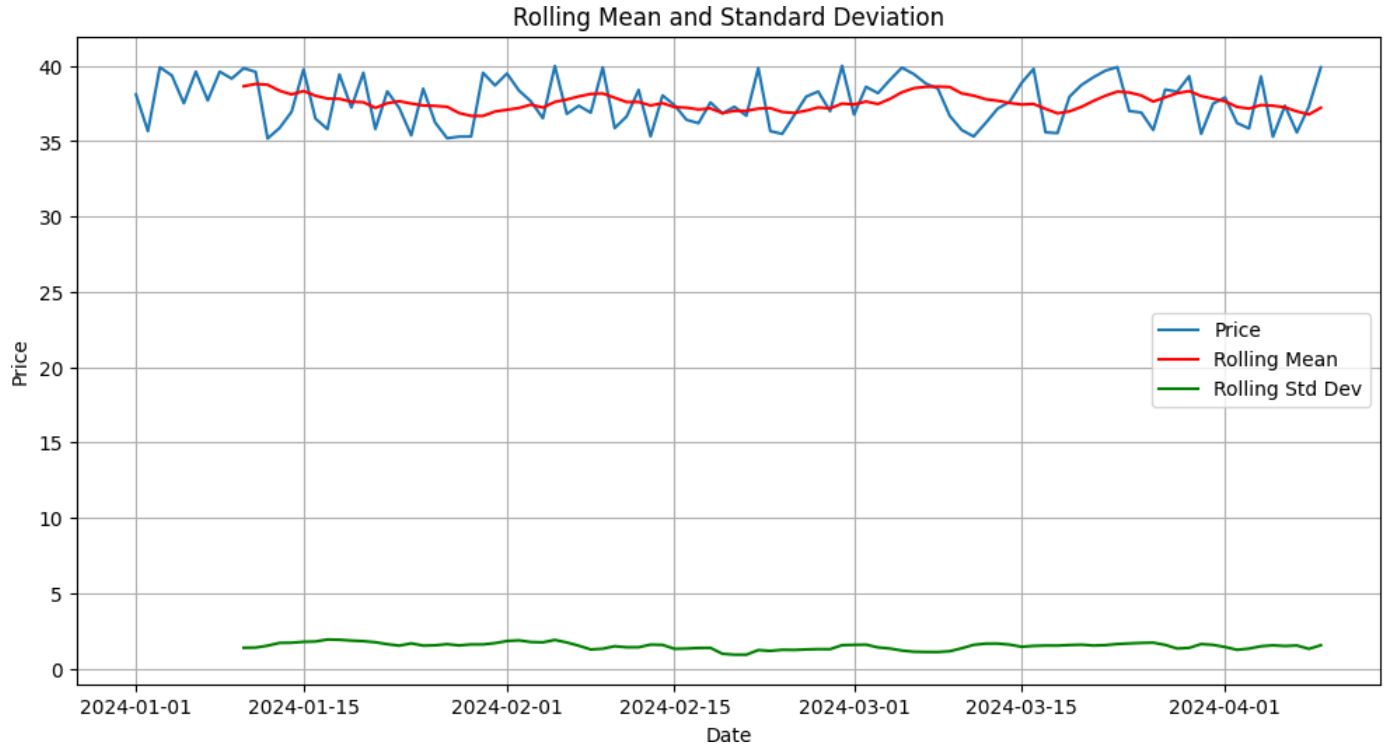
Description: These plots show the moving average and standard deviation over time, which smooth out fluctuations in the data.

Interpretation: Plotting rolling means and standard deviations helps to understand the underlying trend and volatility of stock prices. It can highlight periods of stability and volatility, providing insights into market dynamics.

In [40]:

```
# Sample Data (Replace with your actual data)
dates = pd.date_range(start='2024-01-01', periods=100)
prices = np.random.uniform(35, 40, 100)  # Replace with actual prices
df = pd.DataFrame({'Date': dates, 'Price': prices})
df.set_index('Date', inplace=True)

# Calculate rolling mean and standard deviation
rolling_mean = df['Price'].rolling(window=10).mean()
rolling_std = df['Price'].rolling(window=10).std()

plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Price'], label='Price')
```

```
plt.plot(df.index, rolling_mean, label='Rolling Mean', color='r')
plt.plot(df.index, rolling_std, label='Rolling Std Dev', color='g')
plt.title('Rolling Mean and Standard Deviation')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```

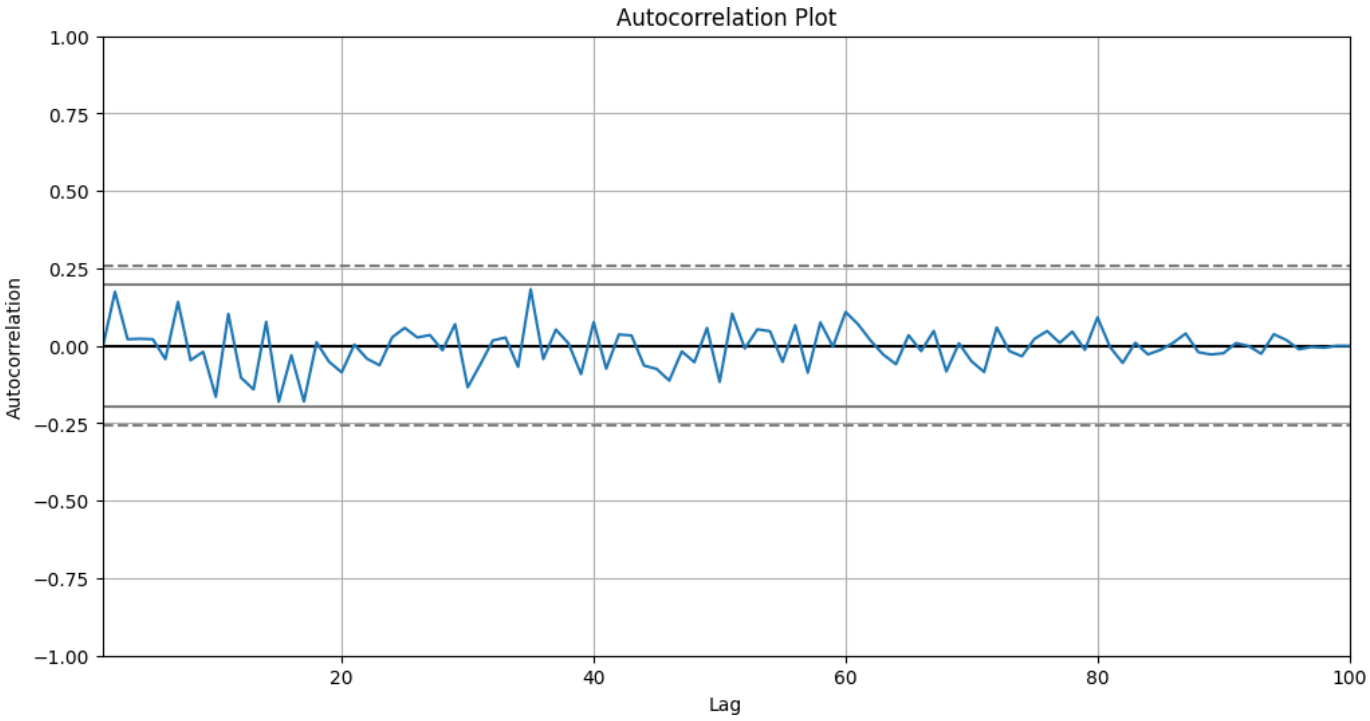


Autocorrelation Plots

Description: Autocorrelation plots show the correlation of a time series with its past values.

Interpretation: Autocorrelation plots help identify patterns and dependencies in stock prices over time. High autocorrelation at certain lags indicates repeating patterns, which can be useful for forecasting.

In [41]:
```
from pandas.plotting import autocorrelation_plot

# Sample Data (Replace with your actual data)
dates = pd.date_range(start='2024-01-01', periods=100)
prices = np.random.uniform(35, 40, 100)  # Replace with actual prices
df = pd.DataFrame({'Date': dates, 'Price': prices})
df.set_index('Date', inplace=True)

plt.figure(figsize=(12, 6))
autocorrelation_plot(df['Price'])
plt.title('Autocorrelation Plot')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.grid(True)
plt.show()
```

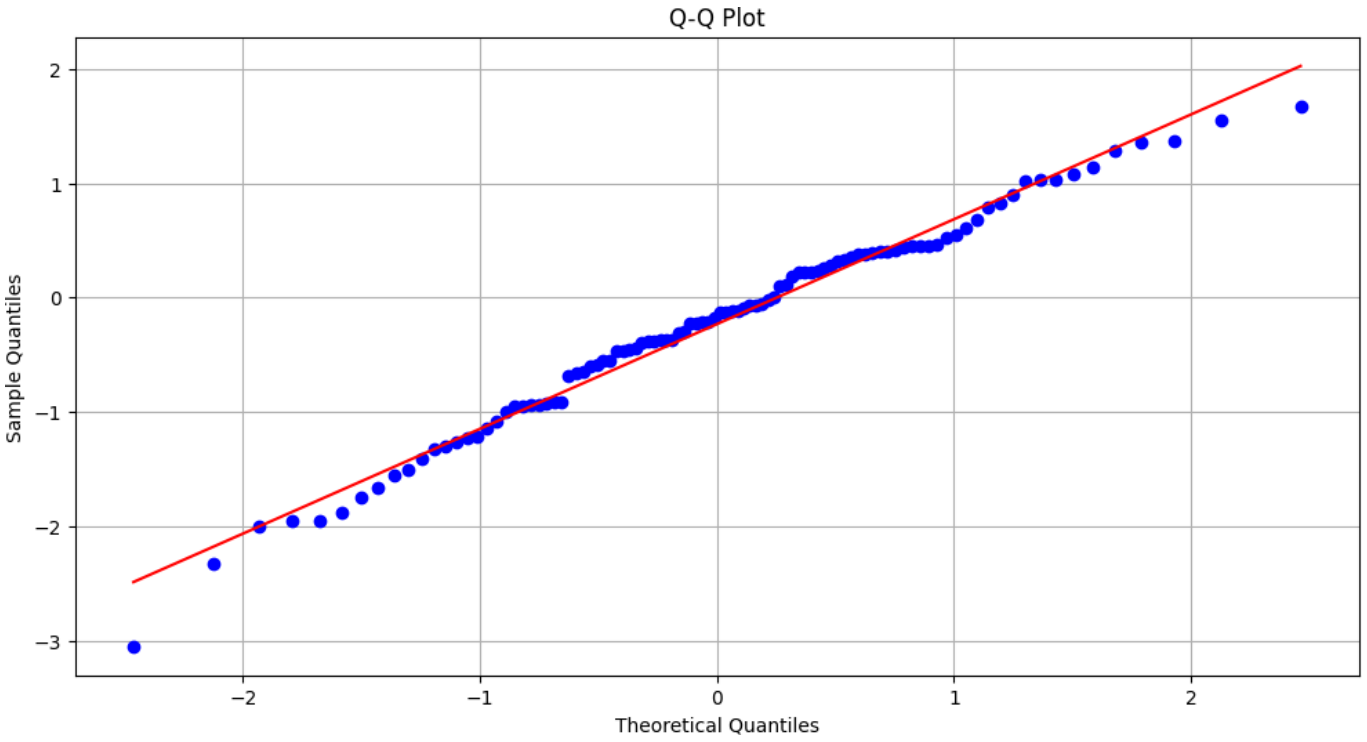


Quantile-Quantile (Q-Q Plot)

Description: Q-Q plots are graphical tools used to compare the distribution of a dataset against a theoretical distribution, such as the normal distribution. By plotting the quantiles of the dataset against the quantiles of the theoretical distribution, this plot helps assess whether the data follows the assumed distribution.

Interpretation: Q-Q plots help evaluate how well data fits a specified distribution. Points that lie on or close to the reference line suggest that the data conforms to the theoretical distribution, while deviations indicate differences. This can be useful for validating assumptions of statistical tests or modeling techniques.

In [42]:
```
import scipy.stats as stats

# Sample Data (Replace with your actual data)
data = np.random.normal(loc=0, scale=1, size=100)  # Replace with actual data

plt.figure(figsize=(12, 6))
stats.probplot(data, dist="norm", plot=plt)
plt.title('Q-Q Plot')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.grid(True)
plt.show()
```



Conclusion -

In this comprehensive analysis of stock market data, we explored various visualization techniques to uncover insights into price movements and trading volumes. Line graphs provided a clear view of trends over time, highlighting the fluctuations in closing prices and their relationship with market events. Bar charts illustrated the trading volumes, revealing periods of heightened activity. Scatter plots examined correlations between stock prices and trading volumes, while histograms showed the distribution of daily returns, helping to understand the variability of returns.