project-1

April 19, 2024

[2]:
```
!pip install -q yfinance
```

[3]:
```
# Required libraries
import pandas as pd
import numpy as np
import yfinance as yf
import math
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
```

```
D:\Python\lib\site-packages\pandas\core\computation\expressions.py:21:
UserWarning: Pandas requires version '2.8.0' or newer of 'numexpr' (version
'2.7.3' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
D:\Python\lib\site-packages\pandas\core\arrays\masked.py:62: UserWarning: Pandas
requires version '1.3.4' or newer of 'bottleneck' (version '1.3.2' currently
installed).
  from pandas.core import (
```

[4]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline

# For reading stock data from yahoo
from pandas_datareader.data import DataReader
import yfinance as yf
from pandas_datareader import data as pdr

yf.pdr_override()

# For time stamps
from datetime import datetime
```

```python
# The tech stocks we'll use for this analysis
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

# Set up End and Start times for data grab
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)

for stock in tech_list:
    globals()[stock] = yf.download(stock, start, end)


company_list = [AAPL, GOOG, MSFT, AMZN]
company_name = ["APPLE", "GOOGLE", "MICROSOFT", "AMAZON"]

for company, com_name in zip(company_list, company_name):
    company["company_name"] = com_name

df = pd.concat(company_list, axis=0)
df.tail(10)
```

```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```

[4]:

```
                  Open        High         Low       Close   Adj Close  \
Date
2024-04-05  182.380005  186.270004  181.970001  185.070007  185.070007
2024-04-08  186.899994  187.289993  184.809998  185.190002  185.190002
2024-04-09  187.240005  187.339996  184.199997  185.669998  185.669998
2024-04-10  182.770004  186.270004  182.669998  185.949997  185.949997
2024-04-11  186.740005  189.770004  185.509995  189.050003  189.050003
2024-04-12  187.720001  188.380005  185.080002  186.130005  186.130005
2024-04-15  187.429993  188.690002  183.000000  183.619995  183.619995
2024-04-16  183.270004  184.830002  182.259995  183.320007  183.320007
2024-04-17  184.309998  184.570007  179.820007  181.279999  181.279999
2024-04-18  181.380005  182.384995  178.649994  179.220001  179.220001

              Volume company_name
Date
2024-04-05  42335200       AMAZON
2024-04-08  39221300       AMAZON
2024-04-09  36546900       AMAZON
2024-04-10  35879200       AMAZON
```

```
2024-04-11  40020700       AMAZON
2024-04-12  38554300       AMAZON
2024-04-15  48052400       AMAZON
2024-04-16  32891300       AMAZON
2024-04-17  31359700       AMAZON
2024-04-18  30128186       AMAZON
```

[5]:
```python
import pandas as pd

print("Dataset loaded and displayed the first few rows.")
print(df.head(10))

print("Dataset loaded and displayed the last rows.")
df.tail(10)
```

```
Dataset loaded and displayed the first few rows.
                  Open        High         Low       Close   Adj Close  \
Date
2023-04-18  166.100006  167.410004  165.649994  166.470001  165.586151
2023-04-19  165.800003  168.160004  165.539993  167.630005  166.740005
2023-04-20  166.089996  167.869995  165.559998  166.649994  165.765182
2023-04-21  165.050003  166.449997  164.490005  165.020004  164.143860
2023-04-24  165.000000  165.600006  163.889999  165.330002  164.452194
2023-04-25  165.190002  166.309998  163.729996  163.770004  162.900497
2023-04-26  163.059998  165.279999  162.800003  163.759995  162.890533
2023-04-27  165.190002  168.559998  165.190002  168.410004  167.515854
2023-04-28  168.490005  169.850006  167.880005  169.679993  168.779099
2023-05-01  169.279999  170.449997  168.639999  169.589996  168.689575

              Volume company_name
Date
2023-04-18  49923000        APPLE
2023-04-19  47720200        APPLE
2023-04-20  52456400        APPLE
2023-04-21  58337300        APPLE
2023-04-24  41949600        APPLE
2023-04-25  48714100        APPLE
2023-04-26  45498800        APPLE
2023-04-27  64902300        APPLE
2023-04-28  55209200        APPLE
2023-05-01  52472900        APPLE
Dataset loaded and displayed the last rows.
```

[5]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2024-04-05  182.380005  186.270004  181.970001  185.070007  185.070007
2024-04-08  186.899994  187.289993  184.809998  185.190002  185.190002
```

```
2024-04-09  187.240005  187.339996  184.199997  185.669998  185.669998
2024-04-10  182.770004  186.270004  182.669998  185.949997  185.949997
2024-04-11  186.740005  189.770004  185.509995  189.050003  189.050003
2024-04-12  187.720001  188.380005  185.080002  186.130005  186.130005
2024-04-15  187.429993  188.690002  183.000000  183.619995  183.619995
2024-04-16  183.270004  184.830002  182.259995  183.320007  183.320007
2024-04-17  184.309998  184.570007  179.820007  181.279999  181.279999
2024-04-18  181.380005  182.384995  178.649994  179.220001  179.220001

              Volume company_name
Date
2024-04-05  42335200       AMAZON
2024-04-08  39221300       AMAZON
2024-04-09  36546900       AMAZON
2024-04-10  35879200       AMAZON
2024-04-11  40020700       AMAZON
2024-04-12  38554300       AMAZON
2024-04-15  48052400       AMAZON
2024-04-16  32891300       AMAZON
2024-04-17  31359700       AMAZON
2024-04-18  30128186       AMAZON
```

1 # Descriptive statistical About Data

[7]:
```
AAPL.describe()
```

[7]:
```
             Open        High         Low       Close   Adj Close  \
count  253.000000  253.000000  253.000000  253.000000  253.000000
mean   180.996087  182.464585  179.650593  181.118973  180.702876
std      8.891719    8.738406    8.831570    8.816747    8.823686
min    163.059998  165.279999  162.800003  163.759995  162.890533
25%    173.149994  174.589996  171.960007  173.570007  173.210495
50%    180.089996  181.929993  178.330002  180.570007  179.859741
75%    189.259995  189.990005  187.610001  189.300003  188.850006
max    198.020004  199.619995  197.000000  198.110001  197.857529

             Volume
count  2.530000e+02
mean   5.774982e+07
std    1.760988e+07
min    2.404830e+07
25%    4.677800e+07
50%    5.366560e+07
75%    6.384130e+07
max    1.366826e+08
```

[8]:
```python
print("Total records in the dataset:", AAPL.shape[0])
```

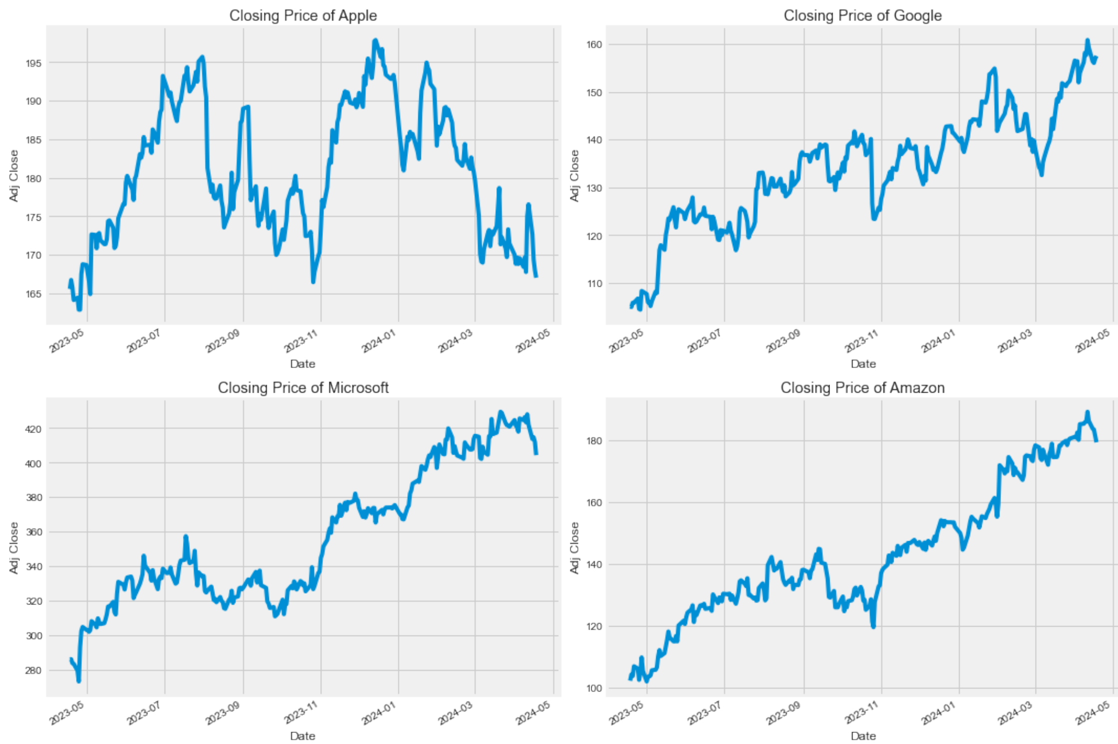
```
Total records in the dataset: 253
```

[9]:
```python
AAPL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 253 entries, 2023-04-18 to 2024-04-18
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          253 non-null    float64
 1   High          253 non-null    float64
 2   Low           253 non-null    float64
 3   Close         253 non-null    float64
 4   Adj Close     253 non-null    float64
 5   Volume        253 non-null    int64
 6   company_name  253 non-null    object
dtypes: float64(5), int64(1), object(1)
memory usage: 15.8+ KB
```

2 Close Price

[10]:
```python
import matplotlib.pyplot as plt
tech_list = ['Apple', 'Google', 'Microsoft', 'Amazon']  # List of company names
# Set up the figure
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.1, bottom=0.1)  # Adjusted for more typical use

# Plot each company's data
for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Adj Close'].plot()
    plt.ylabel('Adj Close')
    plt.xlabel('Date')  # Optionally add a label for the x-axis
    plt.title(f"Closing Price of {tech_list[i - 1]}")

plt.tight_layout()
plt.show()  # Display the plot
```



Top Left - Closing Price of Apple:

This graph shows significant fluctuations in the closing price of Apple's stock. The price peaks and troughs are visible, indicating volatility over the observed period. Top Right - Closing Price of Google:

Google's stock shows a general upward trend in the closing price with some fluctuations. The trend is less volatile compared to Apple's, with a steady increase. Bottom Left - Closing Price of Microsoft:

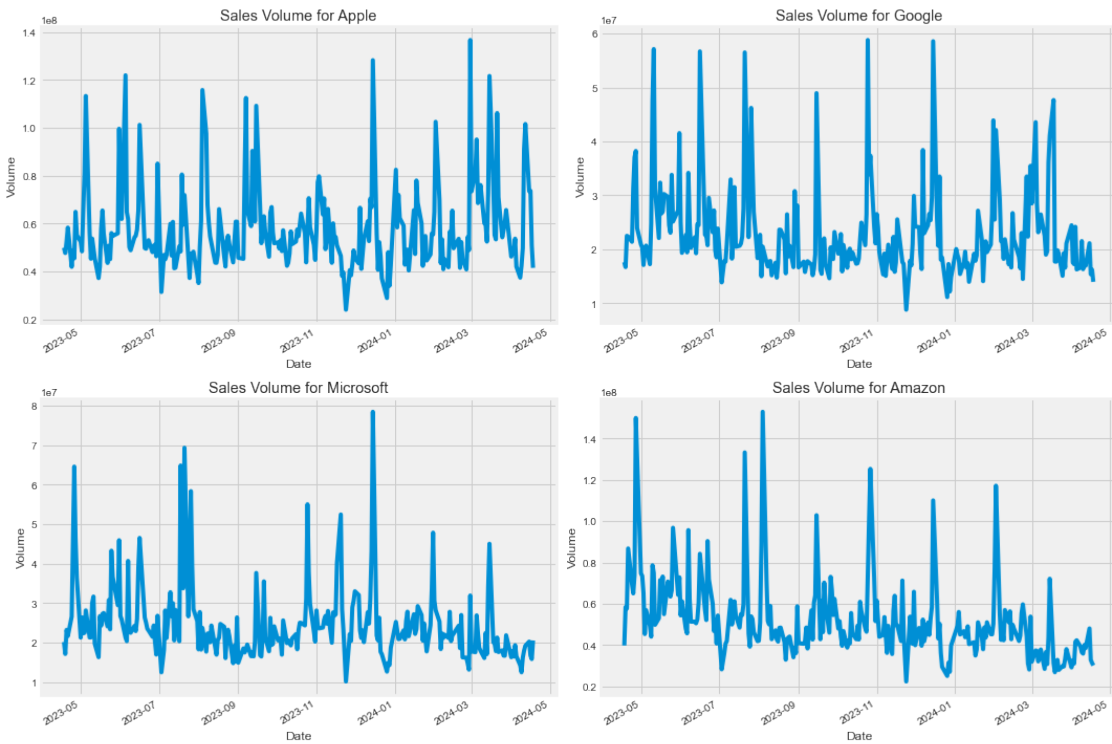
Microsoft's stock graph displays a strong upward trend, indicating a significant increase in the closing price over time. The graph shows fewer fluctuations and a more consistent growth pattern. Bottom Right - Closing Price of Amazon:

Amazon's stock also exhibits an upward trend in its closing price. The graph shows gradual growth with some periods of faster increases.

Volume of sales

[11]:
```
# Create a grid of subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))
axes = axes.flatten()  # Flatten the array to make indexing easier

# Plot each company's data
for i, (ax, company) in enumerate(zip(axes, company_list)):
    company['Volume'].plot(ax=ax)
```

```
    ax.set_ylabel('Volume')
    ax.set_xlabel('Date')  # Adding x-axis label for clarity
    ax.set_title(f"Sales Volume for {tech_list[i]}")

# Adjust layout to prevent overlap
fig.subplots_adjust(top=0.9, bottom=0.1, hspace=0.5, wspace=0.3)
plt.tight_layout()
plt.show()
```



Sales Volume for Apple: Located in the top left corner of the image. The graph shows fluctuations in sales volume from late March 2023 to early May 2023. The volume peaks at various points, indicating periods of high trading activity.

Sales Volume for Google: Positioned in the top right corner. Similar to Apple's graph, it displays fluctuations in sales volume over the same time period. The pattern shows several spikes, suggesting moments of increased trading.

Sales Volume for Microsoft: Found in the bottom left corner. This graph shows a different pattern with fewer but sharper spikes in volume, indicating very high sales volumes on specific days.

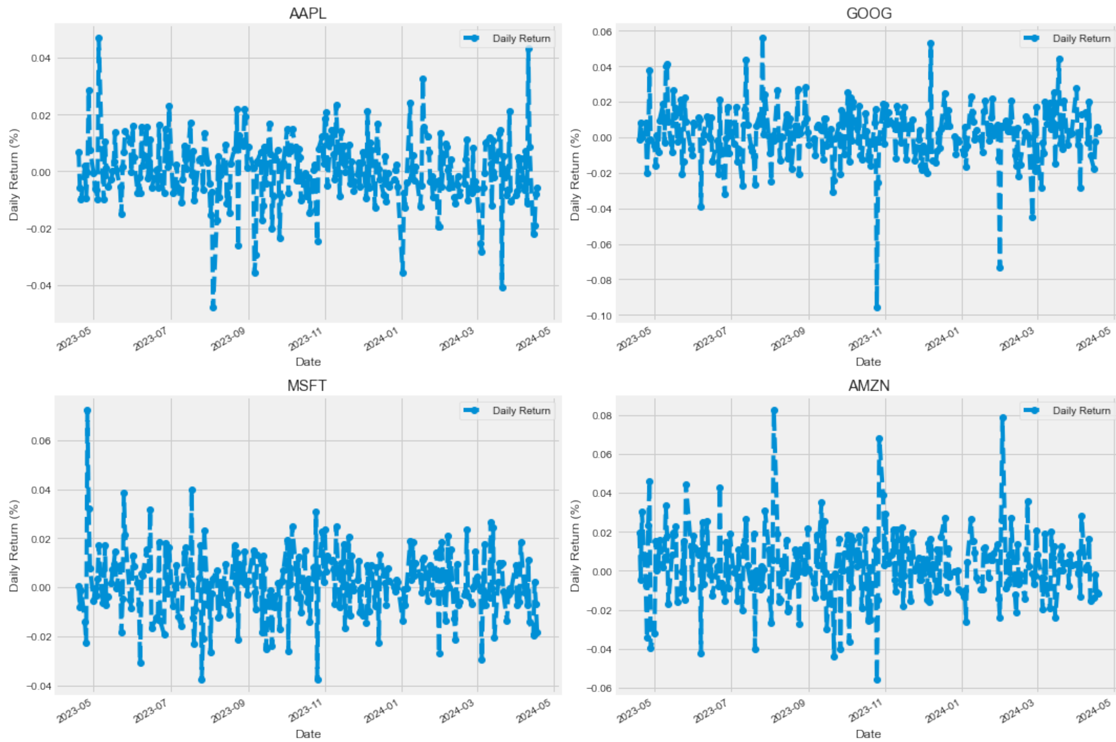
Sales Volume for Amazon: Located in the bottom right corner. The graph displays a pattern of sales volume that includes frequent and sharp peaks, which are somewhat more erratic compared to the other companies.

3 Moving Average of stocks

[14]:
```
import matplotlib.pyplot as plt

# Define company names
company_names = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

# Define the functions again to ensure they are in scope
def calculate_moving_averages(dataframes, days):
    for df in dataframes:
        for ma in days:
            column_name = f"MA for {ma} days"
            df[column_name] = df['Adj Close'].rolling(window=ma).mean()

def plot_data(dataframes, names, axes):
    for df, name, ax in zip(dataframes, names, axes.flatten()):
        columns_to_plot = ['Adj Close'] + [f"MA for {ma} days" for ma in␣
↪ma_days]
        df[columns_to_plot].plot(ax=ax)
        ax.set_title(name)
        ax.set_xlabel('Date')
        ax.set_ylabel('Adjusted Close Price')

# Plot the data
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))
plot_data(company_list, company_names, axes)
fig.tight_layout()
plt.show()
```

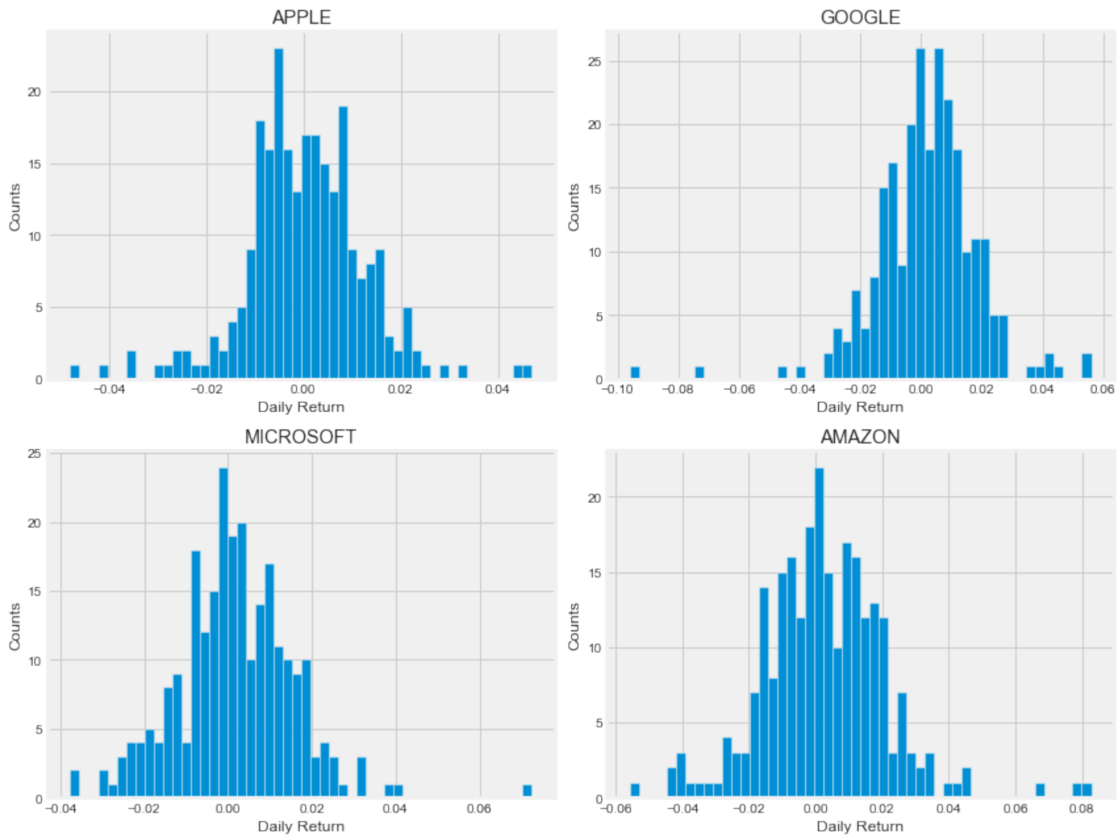
4 DAILY RETURN OF THE STOCK ON AVERAGE

[15]:
```
def calculate_daily_returns(dataframes):
    for df in dataframes:
        df['Daily Return'] = df['Adj Close'].pct_change()

calculate_daily_returns(company_list)

def plot_daily_returns(dataframes, names, axes):
    for df, name, ax in zip(dataframes, names, axes.flatten()):
        df['Daily Return'].plot(ax=ax, legend=True, linestyle='--', marker='o')
        ax.set_title(name)
        ax.set_xlabel('Date')
        ax.set_ylabel('Daily Return (%)')

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))
plot_daily_returns(company_list, company_names, axes)
fig.tight_layout()
plt.show()
```



[16]:

```
plt.figure(figsize=(12, 9))

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Daily Return'].hist(bins=50)
    plt.xlabel('Daily Return')
    plt.ylabel('Counts')
    plt.title(f'{company_name[i - 1]}')

plt.tight_layout()
```



5 CORRELATION B/W STOCKS AND CLOSING VALUE

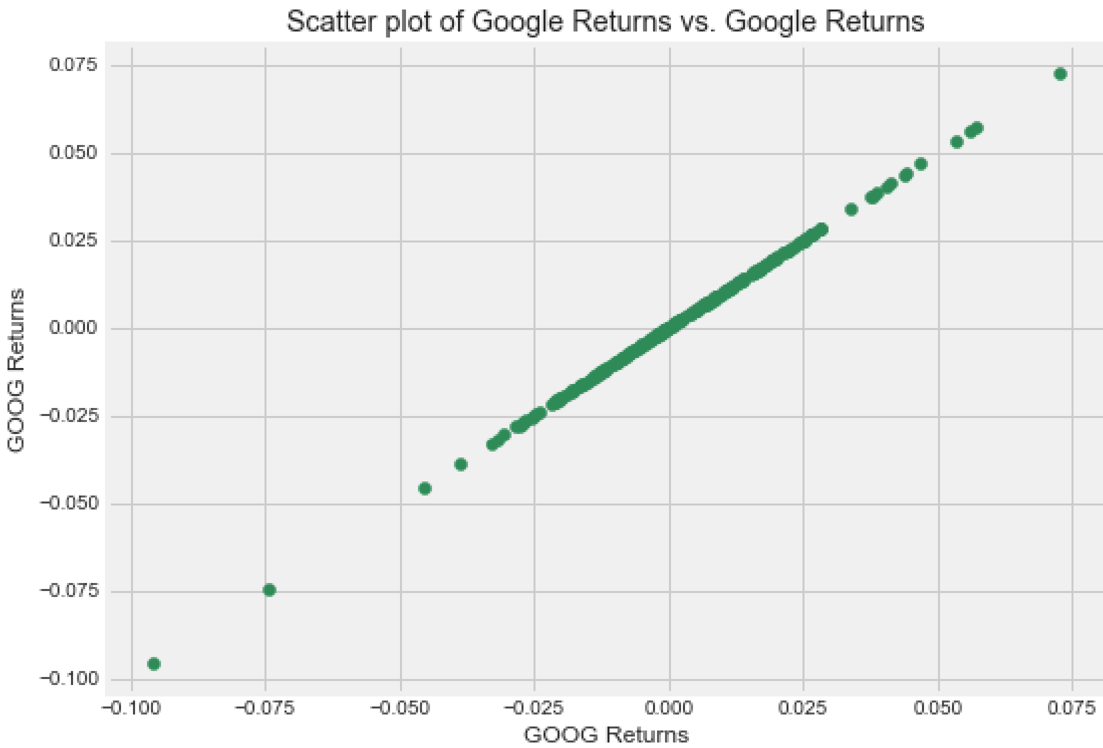
[17]:
```python
import pandas as pd
import yfinance as yf

def fetch_adjusted_close(stock_list, start_date, end_date):
    """
    Fetches the adjusted close prices for a list of stocks over a specified␣
 ↪date range using yfinance.
    """
    data = yf.download(stock_list, start=start_date, end=end_date)
    return data['Adj Close']

def calculate_returns(dataframe):
    """
    Calculates the daily percentage change for each stock in the DataFrame.
    """
    return dataframe.pct_change()
```

```python
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']
start = '2023-01-01'  #start date
end = '2023-12-31'   # End date

closing_df = fetch_adjusted_close(tech_list, start, end)
tech_rets = calculate_returns(closing_df)
print(tech_rets.head())
```

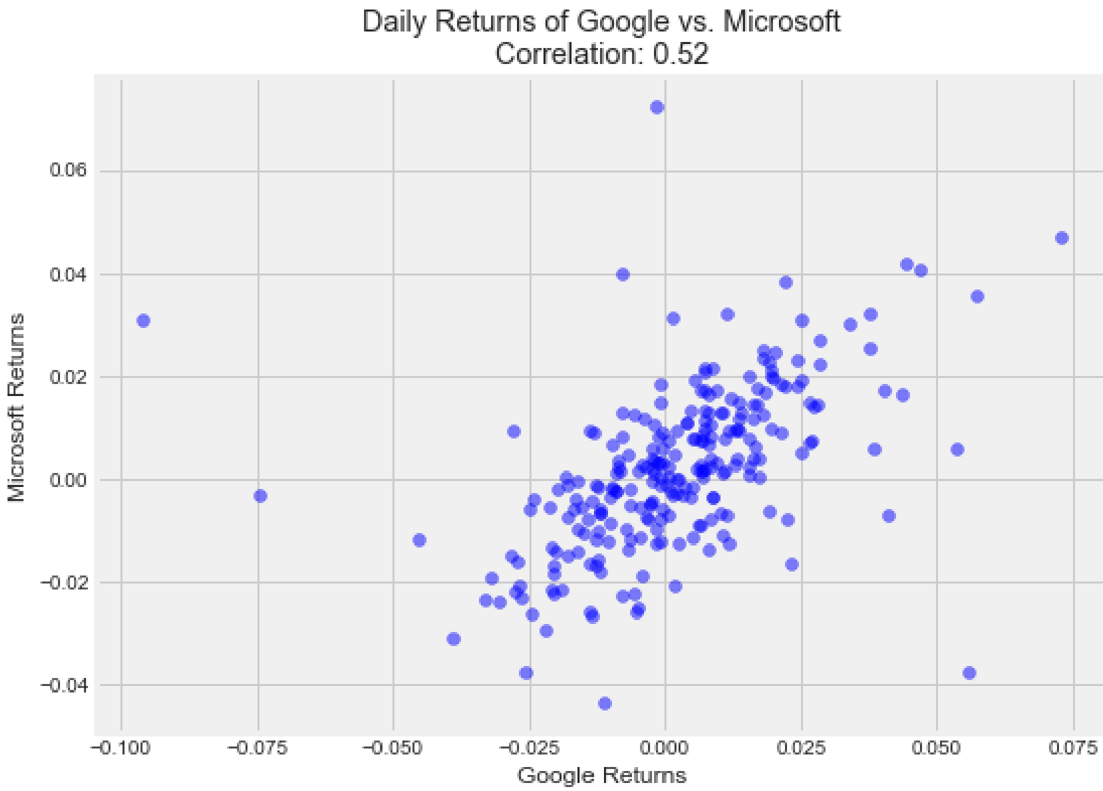
```
[*********************100%***********************]  4 of 4 completed

Ticker          AAPL      AMZN      GOOG      MSFT
Date
2023-01-03       NaN       NaN       NaN       NaN
2023-01-04  0.010314 -0.007924 -0.011037 -0.043743
2023-01-05 -0.010605 -0.023726 -0.021869 -0.029638
2023-01-06  0.036794  0.035611  0.016019  0.011785
2023-01-09  0.004089  0.014870  0.007260  0.009736
```

[18]:

```python
import matplotlib.pyplot as plt



plt.figure(figsize=(8, 6))
plt.scatter(tech_rets['GOOG'], tech_rets['GOOG'], color='seagreen')
plt.title('Scatter plot of Google Returns vs. Google Returns')
plt.xlabel('GOOG Returns')
plt.ylabel('GOOG Returns')
plt.grid(True)
plt.show()
```

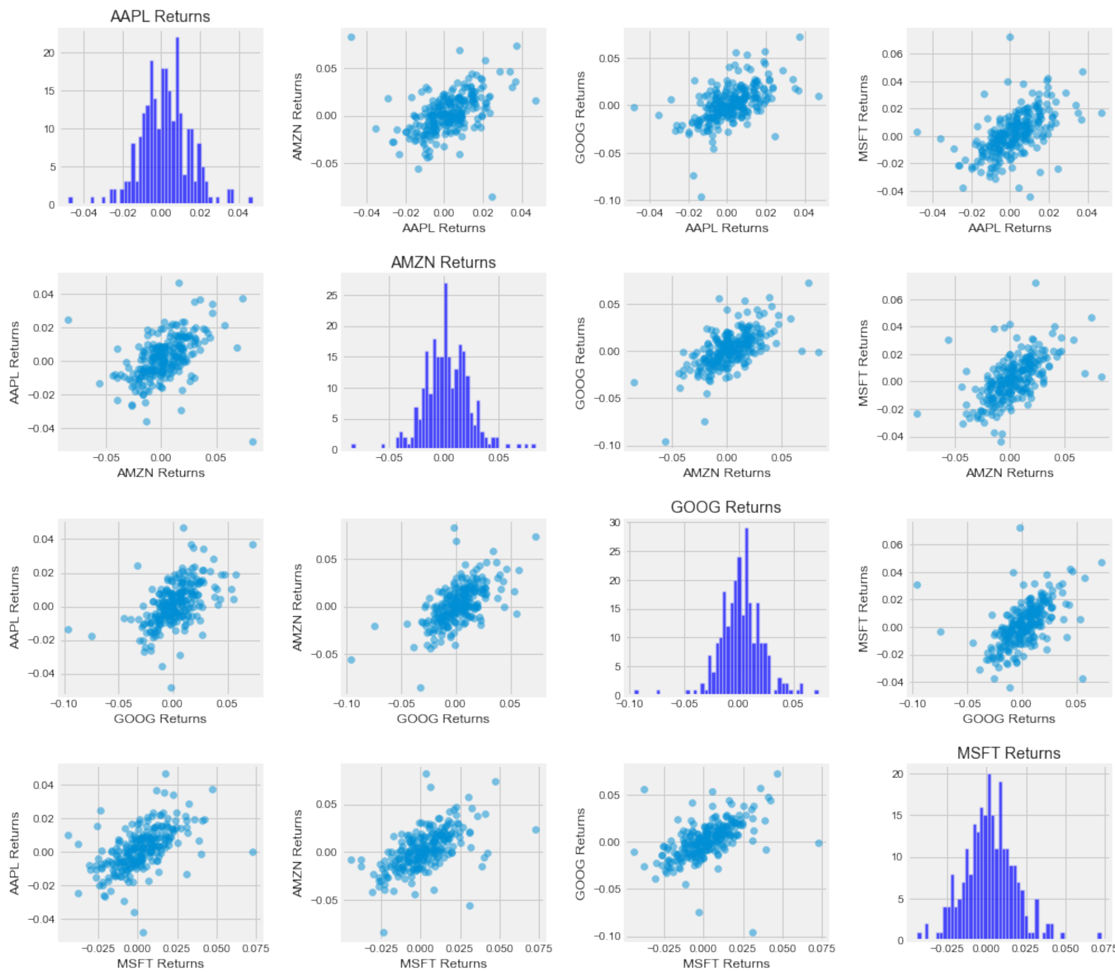


[20]:
```python
from scipy.stats import pearsonr
import matplotlib.pyplot as plt

# Calculate the correlation coefficient
corr, _ = pearsonr(tech_rets['GOOG'], tech_rets['MSFT'])

# Create the plot
plt.figure(figsize=(8, 6))
plt.scatter(tech_rets['GOOG'], tech_rets['MSFT'], alpha=0.5, color='blue')
plt.title(f'Daily Returns of Google vs. Microsoft\nCorrelation: {corr:.2f}')
plt.xlabel('Google Returns')
plt.ylabel('Microsoft Returns')
plt.grid(True)
plt.show()
```



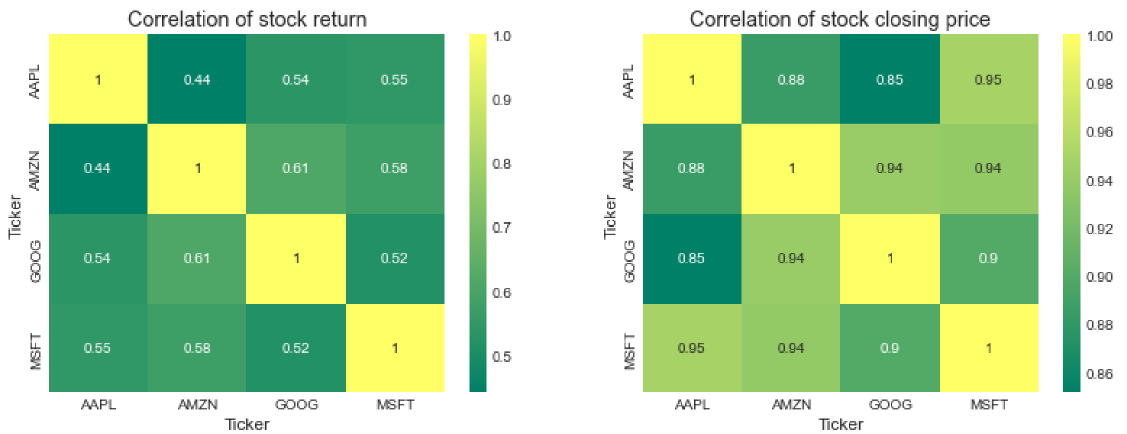
[21]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

tech_rets = tech_rets.replace([np.inf, -np.inf], np.nan).dropna()

# Get the list of stock names from the DataFrame columns
stocks = tech_rets.columns

# Create a grid of scatter plots
fig, axes = plt.subplots(nrows=len(stocks), ncols=len(stocks), figsize=(15, 15))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

for i, stock1 in enumerate(stocks):
    for j, stock2 in enumerate(stocks):
        ax = axes[i, j]
        if i == j:
            ax.hist(tech_rets[stock1], bins=50, color='blue', alpha=0.7)
            ax.set_title(stock1 + ' Returns')
        else:
            ax.scatter(tech_rets[stock1], tech_rets[stock2], alpha=0.5)
            ax.set_xlabel(stock1 + ' Returns')
```

```
            ax.set_ylabel(stock2 + ' Returns')

plt.show()
```

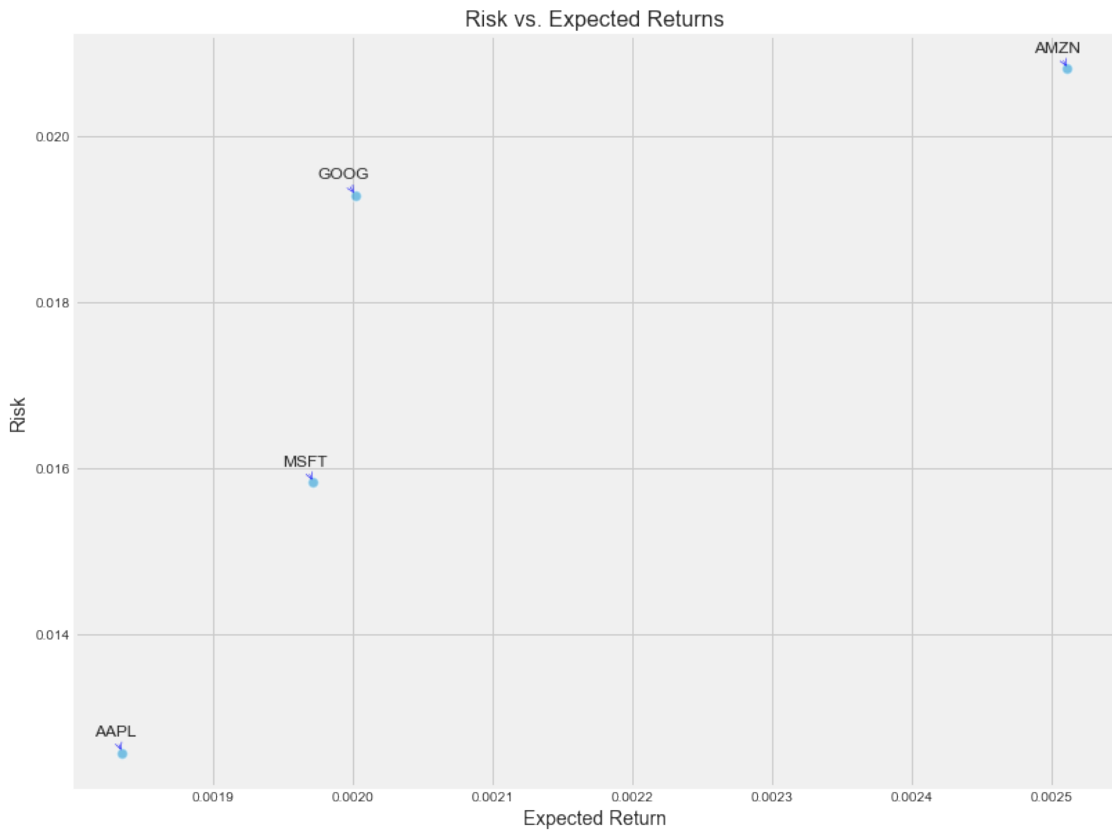


[22]:
```
plt.figure(figsize=(12, 10))

plt.subplot(2, 2, 1)
sns.heatmap(tech_rets.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock return')

plt.subplot(2, 2, 2)
sns.heatmap(closing_df.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock closing price')
```

[22]: Text(0.5, 1.0, 'Correlation of stock closing price')



[23]:
```
import matplotlib.pyplot as plt
import numpy as np

# Assuming 'rets' is your DataFrame containing the returns data for multiple␣
↪stocks
rets = tech_rets.dropna()

area = np.pi * 20  # Control the size of scatter points

plt.figure(figsize=(12, 10))  # Set the figure size for better visibility
plt.scatter(rets.mean(), rets.std(), s=area, alpha=0.5, edgecolors='w',␣
↪linewidth=1.5)
plt.xlabel('Expected Return', fontsize=14)
plt.ylabel('Risk', fontsize=14)
plt.title('Risk vs. Expected Returns', fontsize=16)
plt.grid(True)  # Add grid for better readability

# Annotate each point with the stock name
for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label,
                 xy=(x, y),
                 xytext=(10, 10),
                 textcoords='offset points',
                 ha='right',
                 va='bottom',
                 fontsize=12,
                 arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=0',␣
↪color='blue'))

plt.show()
```



6 PREDICTING CLOSING PRICE STOCK PRICE OF APPLE INC

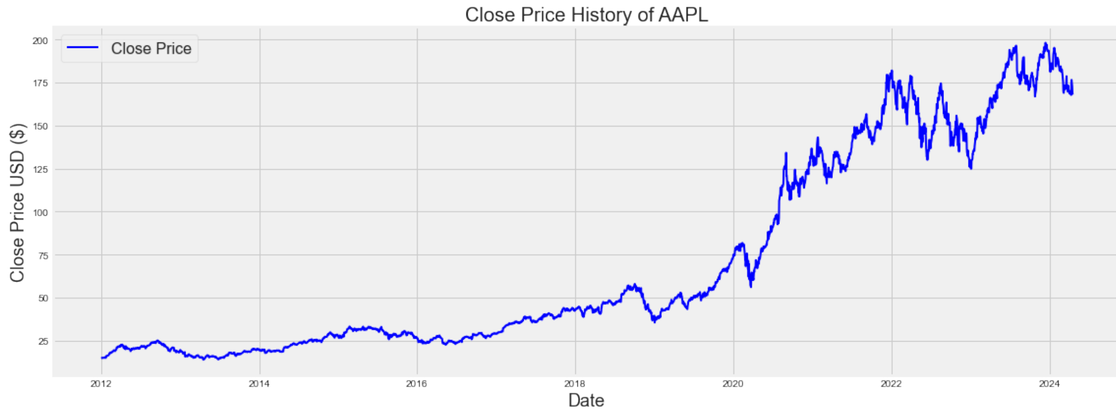
[24]:
```
# Fetch data using yfinance instead of pandas_datareader
aapl_data = yf.download('AAPL', start='2012-01-01', end=datetime.now().
 ↪strftime('%Y-%m-%d'))

# Display the head of the dataframe to confirm successful data retrieval
print(aapl_data)
```

```
[*********************100%%**********************]  1 of 1 completed

                 Open       High        Low      Close  Adj Close  \
Date
2012-01-03  14.621429  14.732143  14.607143  14.686786  12.433821
2012-01-04  14.642857  14.810000  14.617143  14.765714  12.500643
2012-01-05  14.819643  14.948214  14.738214  14.929643  12.639426
2012-01-06  14.991786  15.098214  14.972143  15.085714  12.771558
2012-01-09  15.196429  15.276786  15.048214  15.061786  12.751299
…                 …          …          …          …          …
```

```
2024-04-11  168.339996  175.460007  168.160004  175.039993  175.039993
2024-04-12  174.259995  178.360001  174.210007  176.550003  176.550003
2024-04-15  175.360001  176.630005  172.500000  172.690002  172.690002
2024-04-16  171.750000  173.759995  168.270004  169.380005  169.380005
2024-04-17  169.610001  170.649994  168.000000  168.000000  168.000000

               Volume
Date
2012-01-03  302220800
2012-01-04  260022000
2012-01-05  271269600
2012-01-06  318292800
2012-01-09  394024400
…                 …
2024-04-11   91070300
2024-04-12  101593300
2024-04-15   73531800
2024-04-16   73711200
2024-04-17   50901200

[3092 rows x 6 columns]
```

[25]:

```
plt.figure(figsize=(16, 6))  # Set the figure size for better visibility
plt.plot(aapl_data['Close'], label='Close Price', color='blue', linewidth=2)  #␣
↪Plot the 'Close' column
plt.title('Close Price History of AAPL', fontsize=20)  # Set the title of the␣
↪graph
plt.xlabel('Date', fontsize=18)  # Label for the x-axis
plt.ylabel('Close Price USD ($)', fontsize=18)  # Label for the y-axis
plt.legend(loc='upper left', fontsize=16)  # Add a legend to the plot
plt.grid(True)
plt.tight_layout()  # Adjust the layout to make sure everything fits without␣
↪overlap
plt.show()
```



[26]:
```
data = aapl_data.filter(['Close'])
# Convert the dataframe to a numpy array
dataset = data.values
# Get the number of rows to train the model on
training_data_len = int(np.ceil( len(dataset) * .95 ))

training_data_len
```

[26]: 2938

[27]:
```
# Scale the data
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

scaled_data
```

[27]:
```
array([[0.00401431],
       [0.00444289],
       [0.00533302],
       …,
       [0.86196974],
       [0.84399649],
       [0.83650309]])
```

7 Split Dataset in to Train data and Test data

[28]:
```
# Create the training data set
train_data = scaled_data[0:int(training_data_len), :]
# Split the data into x_train and y_train data sets
```

```python
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
    if i<= 61:
        print(x_train)
        print(y_train)
        print()

# Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

# Reshape the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

```
[array([0.00401431, 0.00444289, 0.00533302, 0.00618049, 0.00605056,
       0.00634339, 0.00620958, 0.00598462, 0.00567821, 0.00662652,
       0.00748175, 0.007218  , 0.00577323, 0.00715207, 0.00579457,
       0.01088518, 0.01049151, 0.01100542, 0.01211663, 0.01278955,
       0.01273332, 0.01252582, 0.01341013, 0.01424207, 0.01518457,
       0.01670691, 0.01990478, 0.01995326, 0.02173353, 0.02306387,
       0.02077746, 0.02165789, 0.02164044, 0.02410915, 0.02375813,
       0.02440779, 0.02557523, 0.0262249 , 0.02809631, 0.02945961,
       0.02985329, 0.02999098, 0.02765997, 0.02709757, 0.02718096,
       0.02937236, 0.02998905, 0.03131358, 0.03443581, 0.03860139,
       0.0378218 , 0.03782373, 0.04083544, 0.04177794, 0.04110694,
       0.04049413, 0.03985611, 0.04197573, 0.0434302 , 0.04403914])]
[0.042534249860459186]

[array([0.00401431, 0.00444289, 0.00533302, 0.00618049, 0.00605056,
       0.00634339, 0.00620958, 0.00598462, 0.00567821, 0.00662652,
       0.00748175, 0.007218  , 0.00577323, 0.00715207, 0.00579457,
       0.01088518, 0.01049151, 0.01100542, 0.01211663, 0.01278955,
       0.01273332, 0.01252582, 0.01341013, 0.01424207, 0.01518457,
       0.01670691, 0.01990478, 0.01995326, 0.02173353, 0.02306387,
       0.02077746, 0.02165789, 0.02164044, 0.02410915, 0.02375813,
       0.02440779, 0.02557523, 0.0262249 , 0.02809631, 0.02945961,
       0.02985329, 0.02999098, 0.02765997, 0.02709757, 0.02718096,
       0.02937236, 0.02998905, 0.03131358, 0.03443581, 0.03860139,
       0.0378218 , 0.03782373, 0.04083544, 0.04177794, 0.04110694,
       0.04049413, 0.03985611, 0.04197573, 0.0434302 , 0.04403914]),
array([0.00444289, 0.00533302, 0.00618049, 0.00605056, 0.00634339,
       0.00620958, 0.00598462, 0.00567821, 0.00662652, 0.00748175,
       0.007218  , 0.00577323, 0.00715207, 0.00579457, 0.01088518,
       0.01049151, 0.01100542, 0.01211663, 0.01278955, 0.01273332,
```

```
       0.01252582, 0.01341013, 0.01424207, 0.01518457, 0.01670691,
       0.01990478, 0.01995326, 0.02173353, 0.02306387, 0.02077746,
       0.02165789, 0.02164044, 0.02410915, 0.02375813, 0.02440779,
       0.02557523, 0.0262249 , 0.02809631, 0.02945961, 0.02985329,
       0.02999098, 0.02765997, 0.02709757, 0.02718096, 0.02937236,
       0.02998905, 0.03131358, 0.03443581, 0.03860139, 0.0378218 ,
       0.03782373, 0.04083544, 0.04177794, 0.04110694, 0.04049413,
       0.03985611, 0.04197573, 0.0434302 , 0.04403914, 0.04253425])]
[0.042534249860459186, 0.04053485447430975]
```

[29]:
```
# Print the shape of x_train to confirm the correct reshaping
print("Shape of x_train:", x_train.shape)
```

```
Shape of x_train: (2878, 60, 1)
```

[30]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM

# Initialize the LSTM model
model = Sequential([
    LSTM(128, return_sequences=True, input_shape=(x_train.shape[1], 1)),
    LSTM(64, return_sequences=False),
    Dense(25),
    Dense(1)
])

# Compile the model with optimizer and loss function
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model with specified batch size and number of epochs
model.fit(x_train, y_train, batch_size=1, epochs=1)
```

```
2878/2878 [==============================] - 90s 31ms/step - loss: 0.0012
```

[30]:
```
<keras.callbacks.History at 0x2c98557cac0>
```

[31]:
```
# Create the testing data set
# Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data[training_data_len - 60: , :]
# Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

# Convert the data to a numpy array
x_test = np.array(x_test)
```

```
# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))
```

[33]:
```
print("Shape of x_test: ", x_test.shape)
```

```
Shape of x_test:  (154, 60, 1)
```

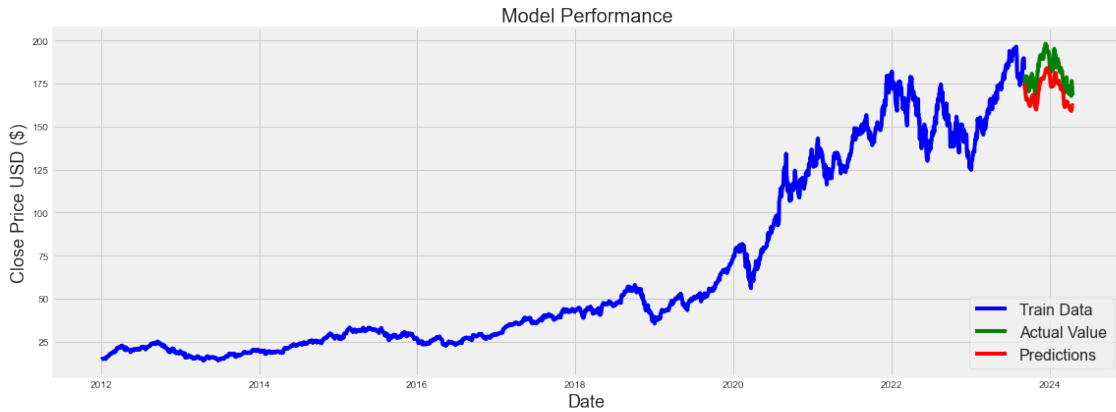
[34]:
```
# Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

# Get the root mean squared error (RMSE)
rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))
rmse
```

[34]: 11.373496542663576

[35]:
```
# Prepare the data for plotting
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions  # Add predictions to the validation set

# Set up the plot
plt.figure(figsize=(16, 6))
plt.title('Model Performance', fontsize=20)
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)

# Plot the training and validation data
plt.plot(train['Close'], label='Train Data', color='blue')
plt.plot(valid['Close'], label='Actual Value', color='green')
plt.plot(valid['Predictions'], label='Predictions', color='red')

# Add a legend to the plot
plt.legend(loc='lower right', fontsize=16)
plt.grid(True)  # Add a grid for better readability
plt.tight_layout()  # Adjust the layout to make sure everything fits without␣
 ↪overlap
plt.show()
```

```
C:\Users\SAURAV~1.THA\AppData\Local\Temp/ipykernel_3600/3055029717.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid['Predictions'] = predictions  # Add predictions to the validation set
```



[36]:
```
valid
```

[36]:
```
                 Close  Predictions
Date
2023-09-07  177.559998   175.256424
2023-09-08  178.179993   173.382156
2023-09-11  179.360001   171.399277
2023-09-12  176.300003   169.956192
2023-09-13  174.210007   168.468094
…                  …          …
2024-04-11  175.039993   159.115723
2024-04-12  176.550003   160.125015
2024-04-15  172.690002   161.669922
2024-04-16  169.380005   162.435623
2024-04-17  168.000000   162.164566

[154 rows x 2 columns]
```