Stock Market Data Analysis And Prediction with LSTM

Time series data is a series of data points indexed in chronological order. Time series data is ubiquitous, making the ability to manipulate it essential for any data analyst or data scientist.

In this notebook, we will explore and analyze data from the stock market, focusing on some technology stocks (Apple, Amazon, Google, and Microsoft). We will demonstrate how to use yfinance to retrieve stock information and visualize various aspects of it using Seaborn and Matplotlib. Additionally, we will examine several methods for assessing the risk associated with a stock based on its historical performance. Finally, we will attempt to predict future stock prices using a Long Short-Term Memory (LSTM) model.

Throughout our exploration, we'll address the following questions:

What was the price change of the stock over time? What was the average daily return of the stock? What were the moving averages for the various stocks? What was the correlation between different stocks? How much risk is involved in investing in a particular stock? Can we predict future stock behavior, specifically the closing price of Apple Inc. using LSTM?

1,What was the change in the price of the stock over time?

In this section, we will cover how to request stock information using Pandas and analyze fundamental attributes of a stock.

In [1]:
```
!pip install -q yfinance
```

In [2]:
```
import pandas as pd
import numpy as np

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


# 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)
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is req
uired for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
```

Out[2]:

Date	Open	High	Low	Close	Adj Close	Volume	company_name
2023-10-03	128.059998	128.520004	124.250000	124.720001	124.720001	51565000	AMAZON
2023-10-04	126.059998	127.360001	125.680000	127.000000	127.000000	44203900	AMAZON
2023-10-05	126.709999	126.730003	124.330002	125.959999	125.959999	39660600	AMAZON
2023-10-06	124.160004	128.449997	124.129997	127.959999	127.959999	46795900	AMAZON
2023-10-09	126.220001	128.789993	124.760002	128.259995	128.259995	38773700	AMAZON
2023-10-10	128.820007	130.740005	128.050003	129.479996	129.479996	42178600	AMAZON
2023-10-11	129.740005	132.050003	129.610001	131.830002	131.830002	40741800	AMAZON
2023-10-12	132.169998	134.479996	131.229996	132.330002	132.330002	55528600	AMAZON
2023-10-13	132.979996	133.309998	128.949997	129.789993	129.789993	45786600	AMAZON
2023-10-16	130.690002	133.070007	130.429993	132.550003	132.550003	42790500	AMAZON

Upon reviewing our data, we observe that it consists of numeric values with dates serving as the data index. It's important to note that weekends are missing from the records.

Quick note: Utilizing globals() to set the DataFrame names is considered a less elegant approach, but it simplifies the process. With our data in hand, let's proceed with some basic data analysis to assess its quality.

Descriptive Statistics about the Data

The .describe() function generates descriptive statistics, summarizing the central tendency, dispersion, and shape of a dataset's distribution, while excluding NaN values. It can analyze both numeric and object series, as well as sets of columns in a DataFrame with mixed data types. The specific output varies depending on the provided input. For more details, please refer to the notes below.

In [3]:
```
# Summary Stats
AAPL.describe()
```

Out[3]:

	Open	High	Low	Close	Adj Close	Volume
count	251.000000	251.000000	251.000000	251.000000	251.000000	2.510000e+02
mean	163.292710	165.021992	161.782390	163.491594	163.115068	6.546845e+07
std	18.877906	18.673270	19.098570	18.856730	19.035087	2.132282e+07
min	126.010002	127.769997	124.169998	125.019997	124.488876	3.145820e+07
25%	147.919998	149.959999	146.379997	148.070000	147.498634	5.053360e+07
50%	165.000000	166.309998	164.029999	165.330002	164.878983	6.075020e+07
75%	178.275002	179.985001	177.090004	178.500000	178.499405	7.487370e+07
max	196.240005	198.229996	195.279999	196.449997	196.185074	1.647624e+08

We have only 255 records per year because weekends are not included in the data.

Information About the Data

`.info()` method prints information about a DataFrame including the index `dtype` and columns, non-null values, and memory usage.

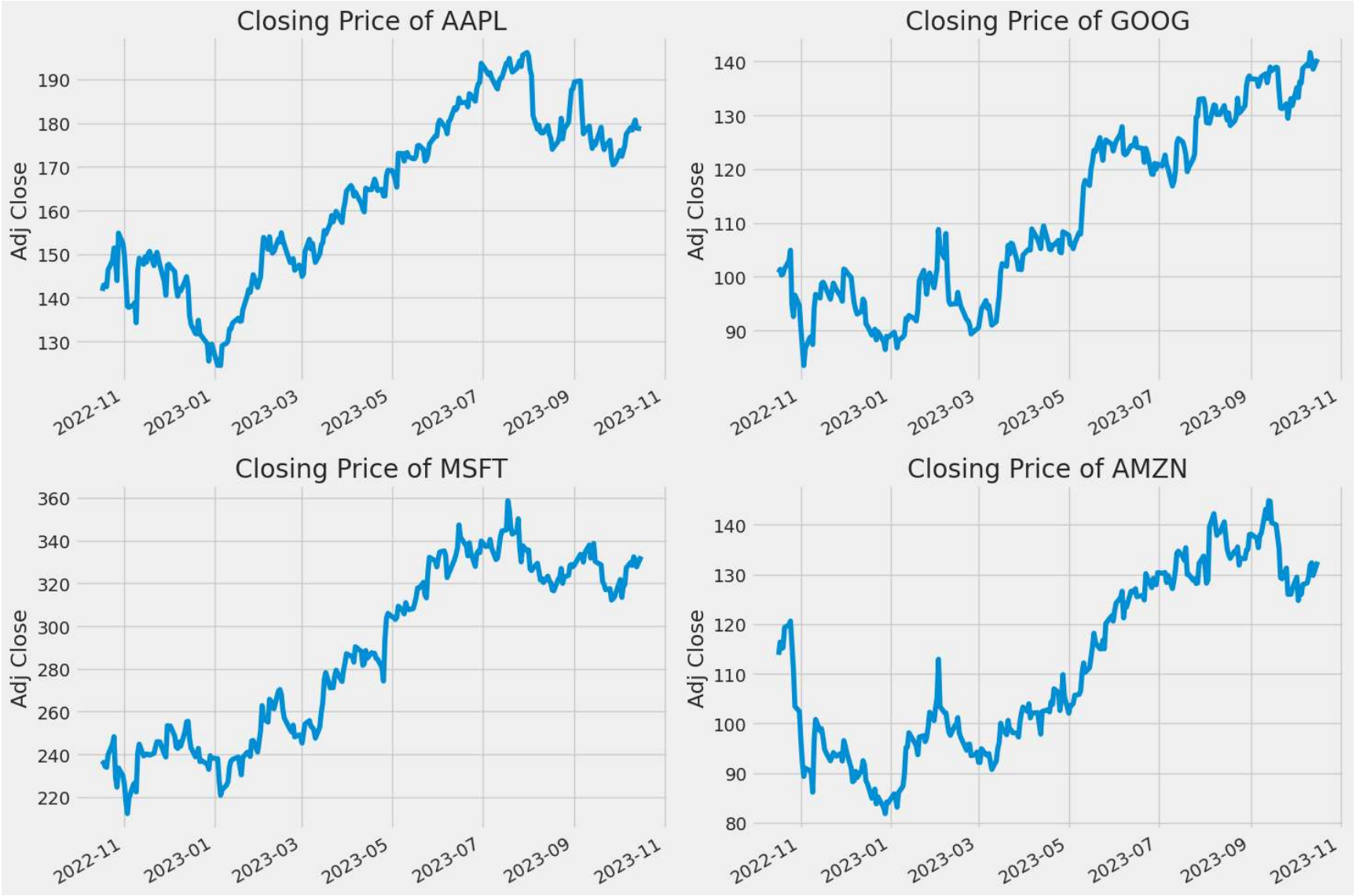
In [4]:
```
# General info
AAPL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2022-10-17 to 2023-10-16
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          251 non-null    float64
 1   High          251 non-null    float64
 2   Low           251 non-null    float64
 3   Close         251 non-null    float64
 4   Adj Close     251 non-null    float64
 5   Volume        251 non-null    int64  
 6   company_name  251 non-null    object 
dtypes: float64(5), int64(1), object(1)
memory usage: 15.7+ KB
```

Closing Price

The closing price is the last price at which a stock is traded during the regular trading day and serves as the standard benchmark for investors to track its performance over time.

In [5]:
```
# Let's see a historical view of the closing price
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Adj Close'].plot()
    plt.ylabel('Adj Close')
    plt.xlabel(None)
    plt.title(f"Closing Price of {tech_list[i - 1]}")

plt.tight_layout()
```

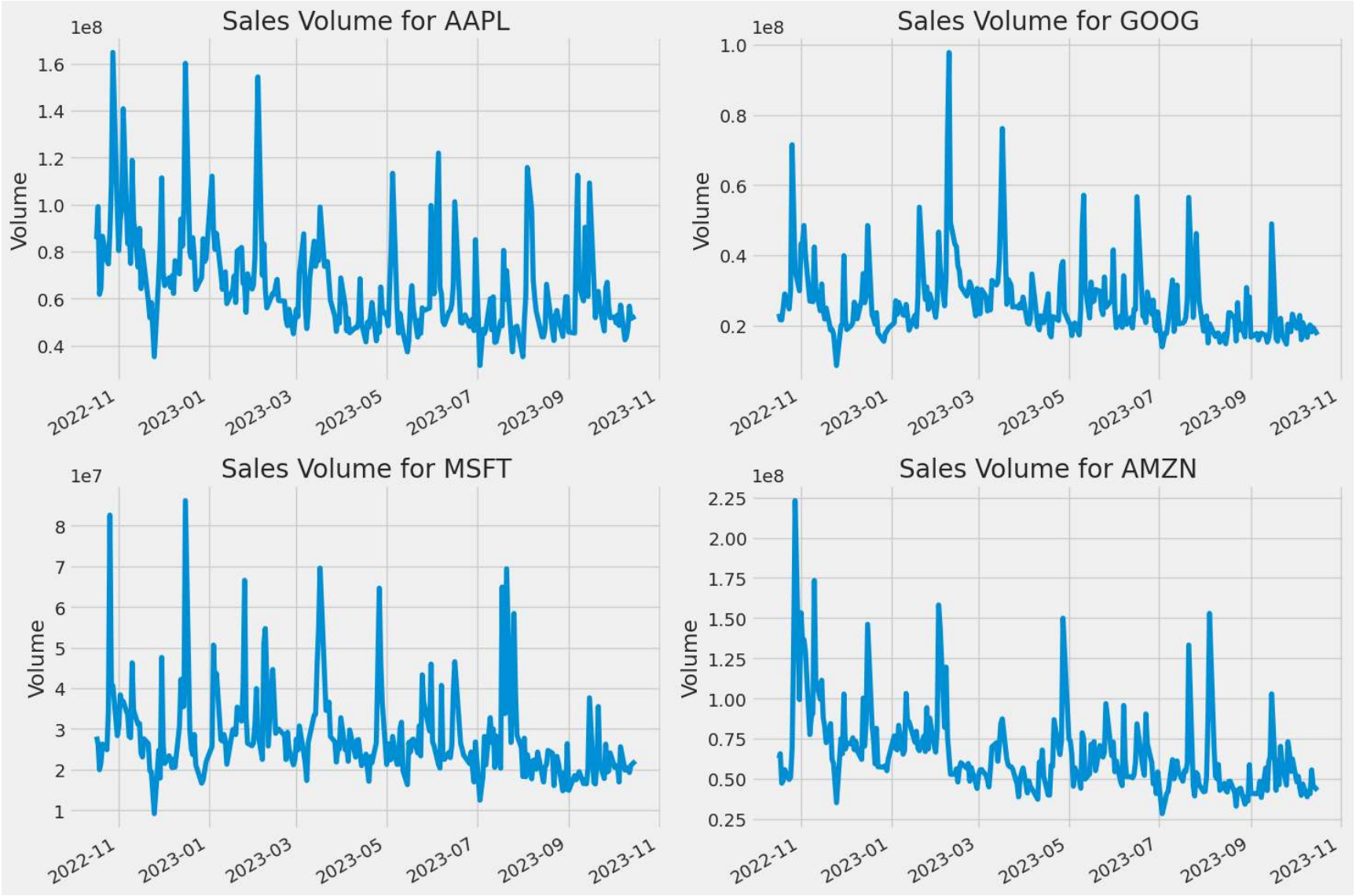


Volume of Sales

Volume represents the quantity of an asset or security exchanged within a specified time frame, typically over the course of a day. For example, stock trading volume refers to the number of shares of a security traded between its daily open and close. Both the trading volume and its changes over time are significant factors for technical traders.

In [6]:

```
# Now let's plot the total volume of stock being traded each day
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Volume'].plot()
    plt.ylabel('Volume')
    plt.xlabel(None)
    plt.title(f"Sales Volume for {tech_list[i - 1]}")

plt.tight_layout()
```

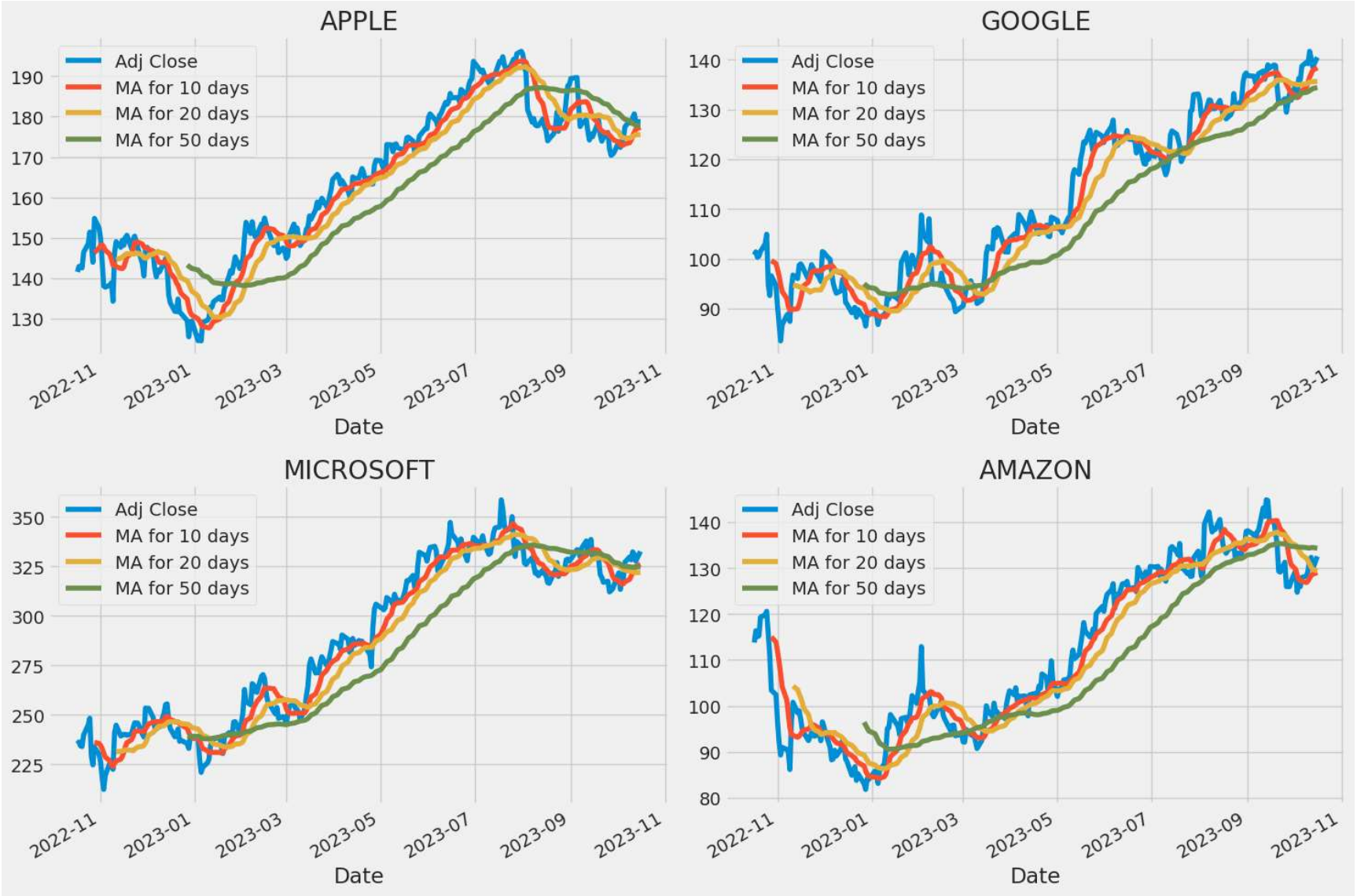


Now that we've seen the visualizations for the closing price and the volume traded each day, let's go ahead and caculate the moving average for the stock.

2. What was the moving average of the various stocks?

The moving average (MA) is a simple technical analysis tool that smooths out price data by creating a constantly updated average price. The average is taken over a specific period of time, like 5 days, 30 minutes, 4 weeks, or any time period the trader chooses.

In [7]:
```
ma_day = [10, 20, 50]

for ma in ma_day:
    for company in company_list:
        column_name = f"MA for {ma} days"
        company[column_name] = company['Adj Close'].rolling(ma).mean()


fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(10)
fig.set_figwidth(15)

AAPL[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[0,0])
axes[0,0].set_title('APPLE')

GOOG[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[0,1])
axes[0,1].set_title('GOOGLE')

MSFT[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[1,0])
axes[1,0].set_title('MICROSOFT')

AMZN[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[1,1])
axes[1,1].set_title('AMAZON')

fig.tight_layout()
```

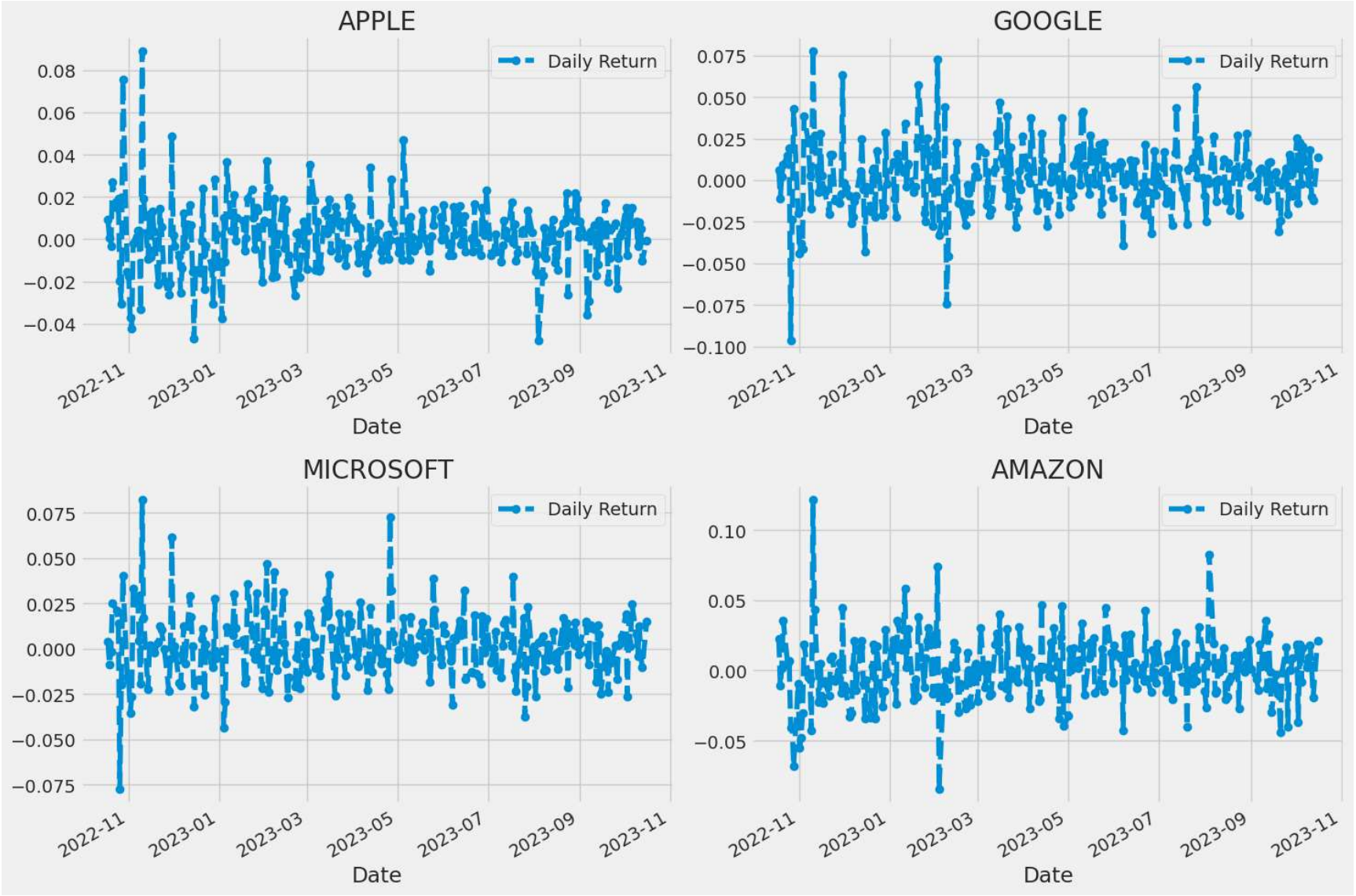


We can see in the above plots that the best values to measure the moving average are 10 and 20 days because we still capture trends in the data without noise.

3. What was the daily return of the stock on average?

Now that we've done some baseline analysis, let's go ahead and dive a little deeper. We're now going to analyze the risk of the stock. In order to do so, we'll need to take a closer look at the daily changes of the stock, and not just its absolute value. Let's go ahead and use pandas to retrieve the daily returns for Apple stock.

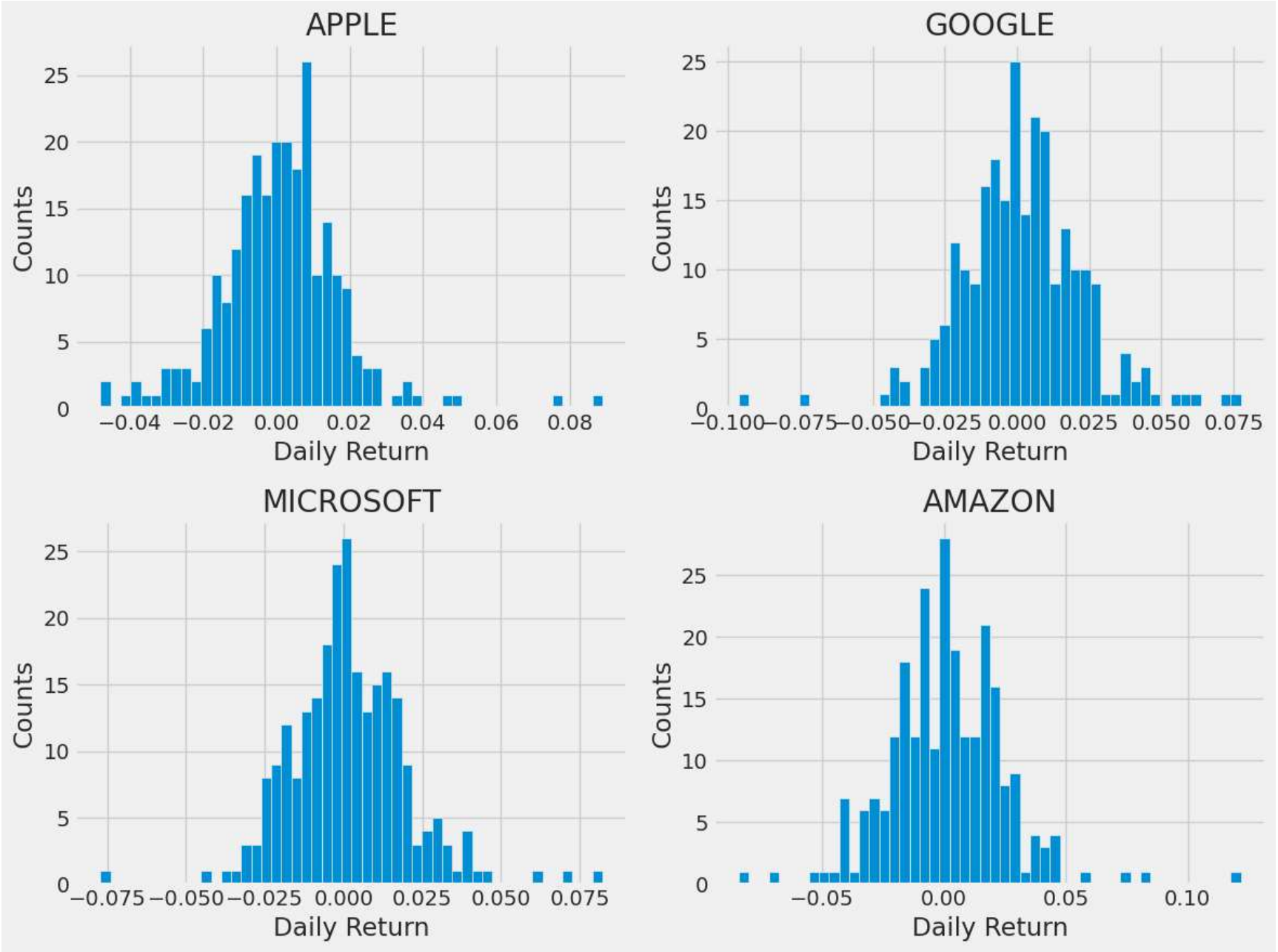
In [8]:
```
# We'll use pct_change to find the percent change for each day
for company in company_list:
    company['Daily Return'] = company['Adj Close'].pct_change()

# Then we'll plot the daily return percentage
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(10)
fig.set_figwidth(15)

AAPL['Daily Return'].plot(ax=axes[0,0], legend=True, linestyle='--', marker='o')
axes[0,0].set_title('APPLE')

GOOG['Daily Return'].plot(ax=axes[0,1], legend=True, linestyle='--', marker='o')
axes[0,1].set_title('GOOGLE')

MSFT['Daily Return'].plot(ax=axes[1,0], legend=True, linestyle='--', marker='o')
axes[1,0].set_title('MICROSOFT')

AMZN['Daily Return'].plot(ax=axes[1,1], legend=True, linestyle='--', marker='o')
axes[1,1].set_title('AMAZON')

fig.tight_layout()
```



Now let's get an overall look at the average daily return using a histogram. We'll use seaborn to create both a histogram and kde plot on the same figure.

In [9]:
```
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()
```



4. What was the correlation between different stocks closing prices?

Correlation is a statistic that quantifies the extent to which two variables move in relation to each other, and its value must fall within the range of -1.0 to +1.0. Correlation measures association but does not indicate causation, meaning it doesn't show if 'x' causes 'y' or vice versa, or if the association is influenced by a third factor.

Now, what if we wanted to analyze the returns of all the stocks in our list? Let's proceed by creating a DataFrame that includes the ['Close'] columns from each of the stock dataframes.

In [10]:
```
# Grab all the closing prices for the tech stock list into one DataFrame

closing_df = pdr.get_data_yahoo(tech_list, start=start, end=end)['Adj Close']

# Make a new tech returns DataFrame
tech_rets = closing_df.pct_change()
tech_rets.head()
```

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

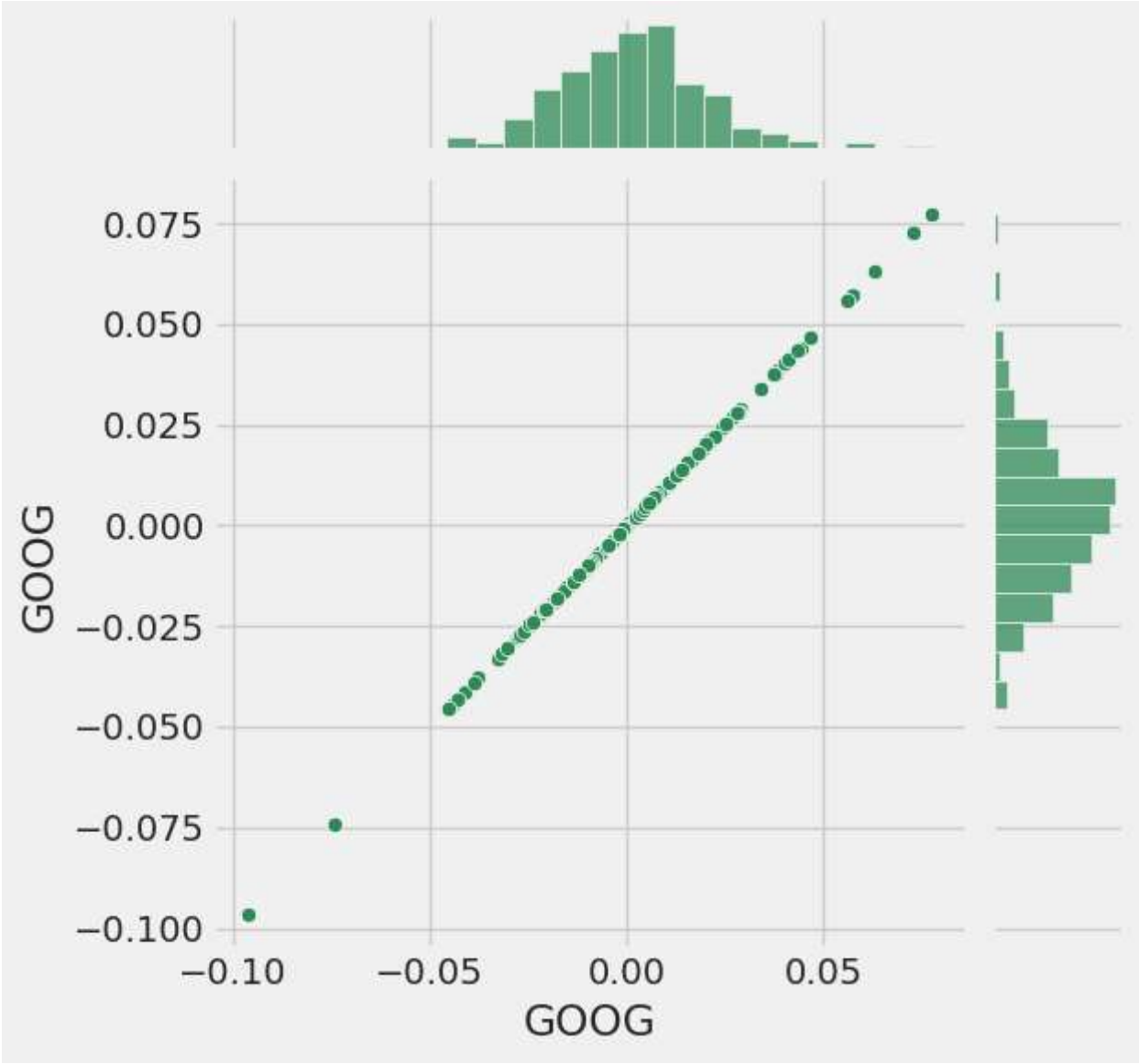
Out[10]:

Date	AAPL	AMZN	GOOG	MSFT
2022-10-17	NaN	NaN	NaN	NaN
2022-10-18	0.009409	0.022585	0.006053	0.004084
2022-10-19	0.000765	-0.011086	-0.010849	-0.008470
2022-10-20	-0.003267	0.001564	0.002393	-0.001395
2022-10-21	0.027059	0.035315	0.009450	0.025280

Now we can compare the daily percentage return of two stocks to check how correlated. First let's see a sotck compared to itself.

In [11]:
```
# Comparing Google to itself should show a perfectly linear relationship
sns.jointplot(x='GOOG', y='GOOG', data=tech_rets, kind='scatter', color='seagreen')
```

Out[11]:
```
<seaborn.axisgrid.JointGrid at 0x7fe7f4a6a0b0>
```



In [12]:
```
# We'll use joinplot to compare the daily returns of Google and Microsoft
sns.jointplot(x='GOOG', y='MSFT', data=tech_rets, kind='scatter')
```

Out[12]:
```
<seaborn.axisgrid.JointGrid at 0x7fe7f4066440>
```

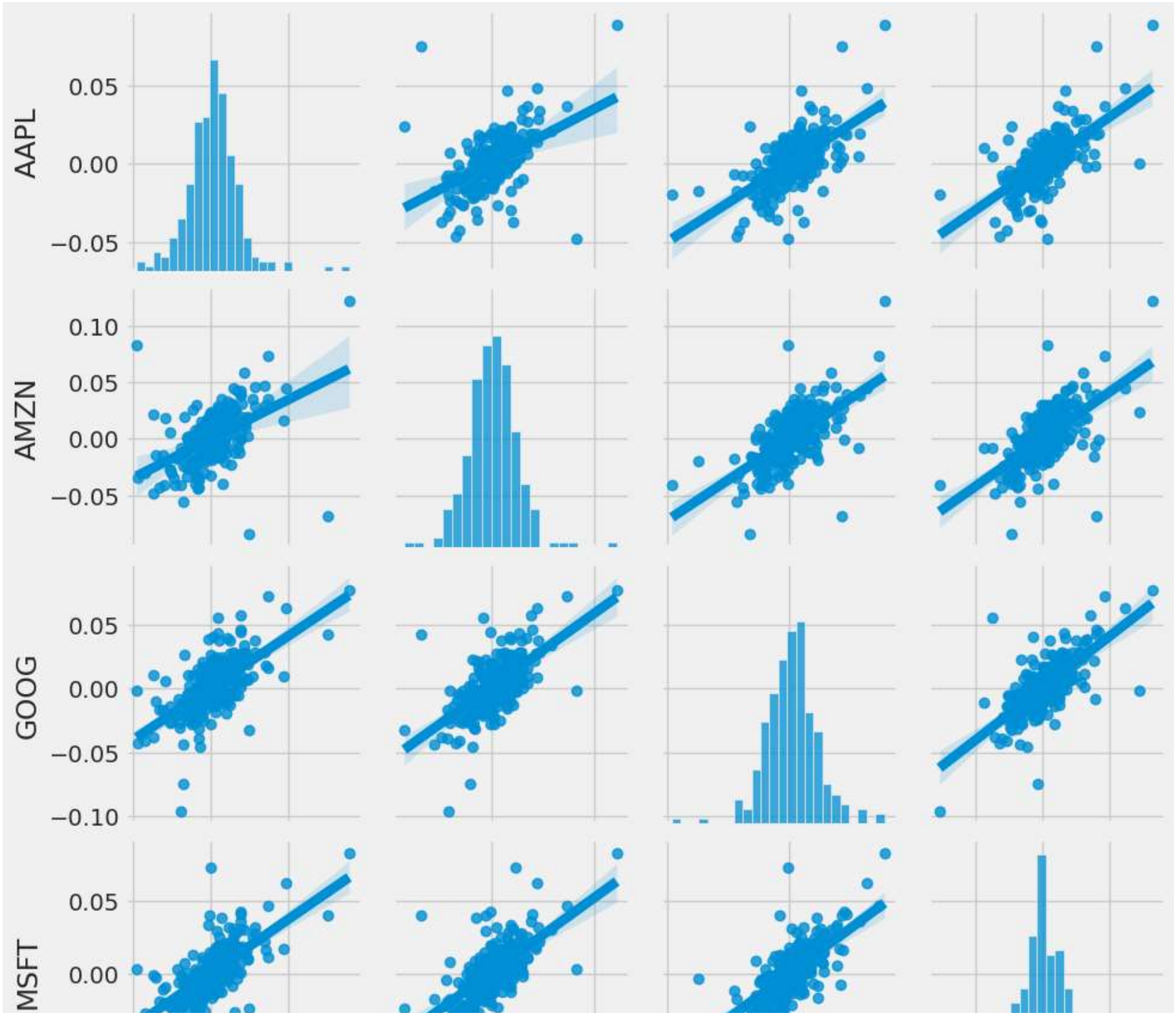
So now we can see that if two stocks are perfectly and positively correlated with each other, a linear relationship between their daily return values should occur.

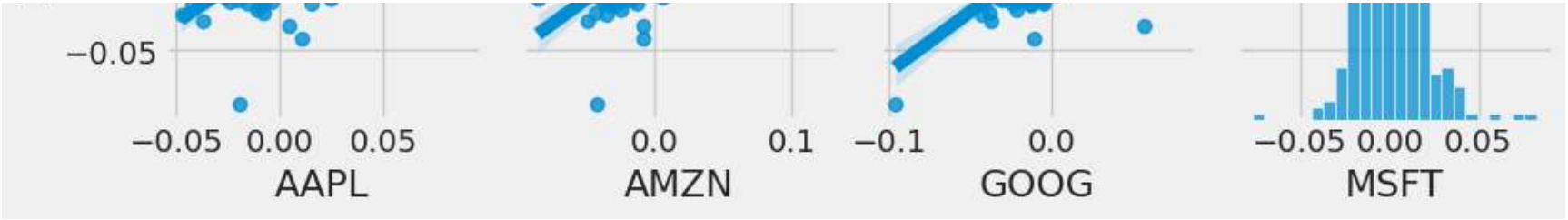
Seaborn and Pandas make it very easy to perform this comparison analysis for every possible combination of stocks in our technology stock ticker list. We can use sns.pairplot() to automatically generate this plot.

In [13]:
```
# We can simply call pairplot on our DataFrame for an automatic visual analysis of all the comparisons

sns.pairplot(tech_rets, kind='reg')
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x7fe7f4cc4b20>
```


AAPL
AMZN
GOOG
MSFT
0.05
0.00
−0.05
0.10
0.05
0.00
−0.05
−0.10
0.05
0.00
−0.05
−0.10
0.05
0.00

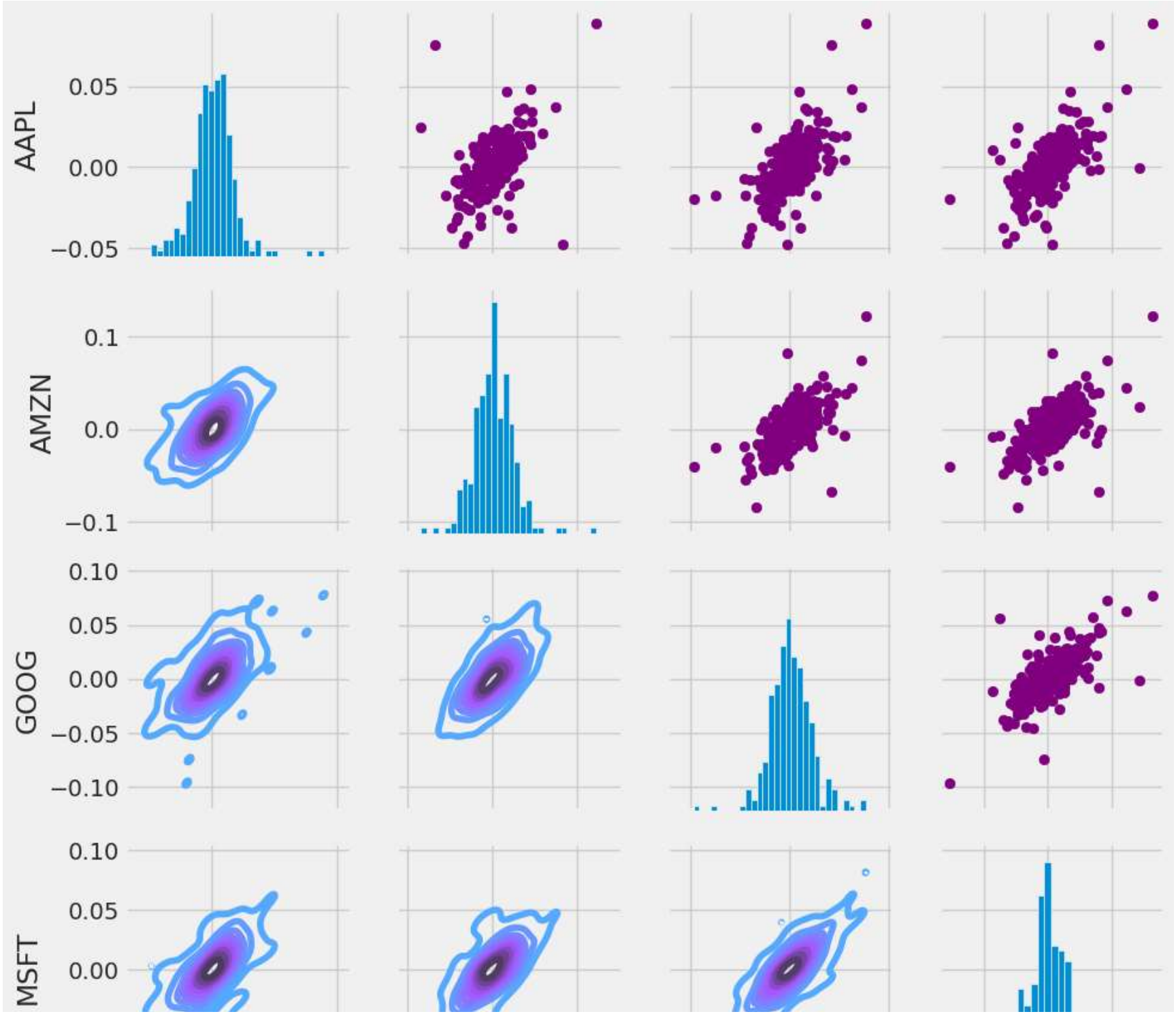


From the above plots, we can observe all the relationships in daily returns between the stocks. A quick glance reveals an interesting correlation between Google and Amazon daily returns, which warrants further investigation.

While the simplicity of just calling sns.pairplot() is fantastic, we can also utilize sns.PairGrid() for full control over the figure, including the type of plots in the diagonal, upper triangle, and lower triangle. Below is an example of harnessing the full power of Seaborn to achieve this result.

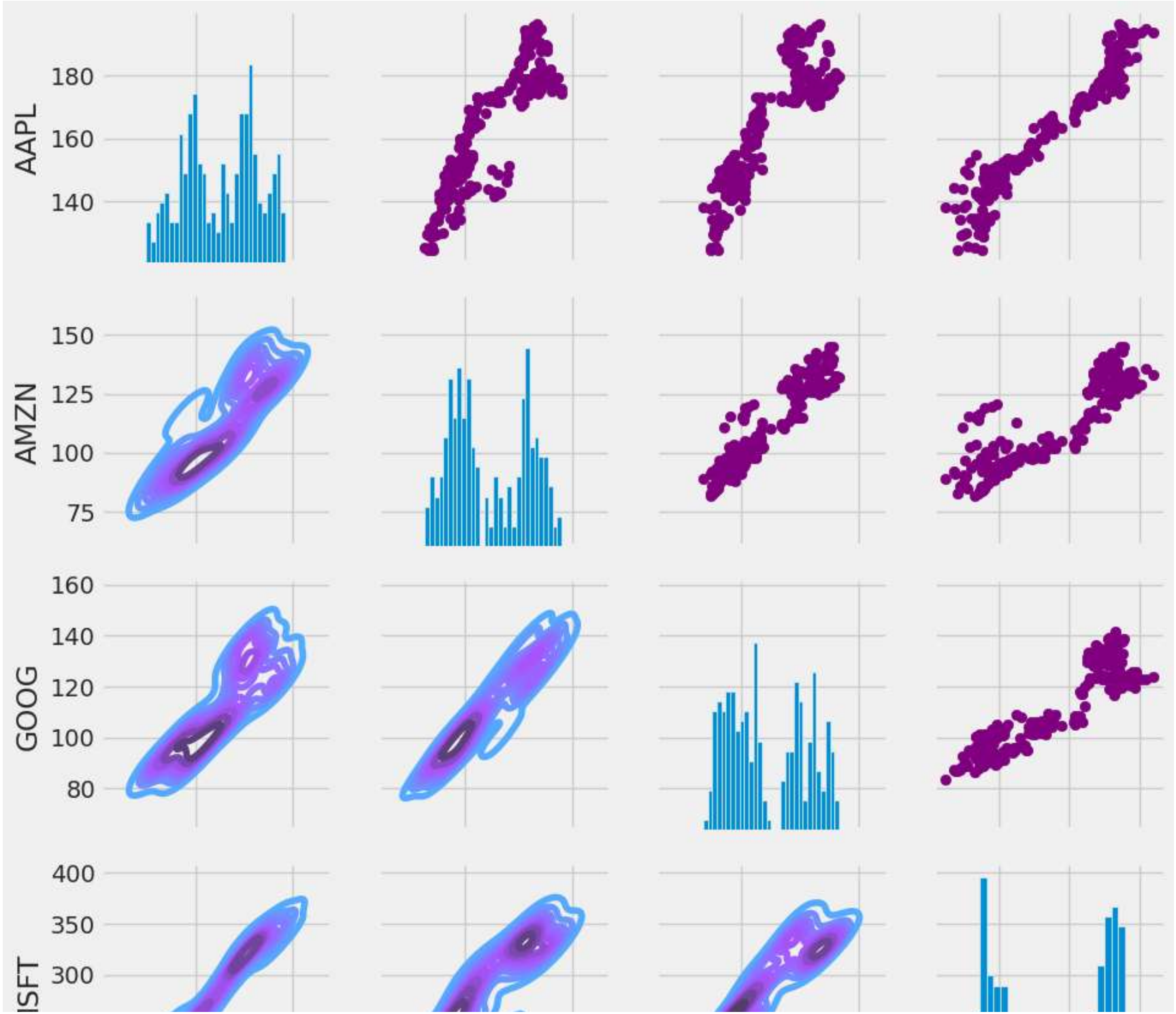
In [14]:
```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
return_fig = sns.PairGrid(tech_rets.dropna())

# Using map_upper we can specify what the upper triangle will look like.
return_fig.map_upper(plt.scatter, color='purple')

# We can also define the lower triangle in the figure, inclufing the plot type (kde) or the color map (BluePurple)
return_fig.map_lower(sns.kdeplot, cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily return
return_fig.map_diag(plt.hist, bins=30)
```

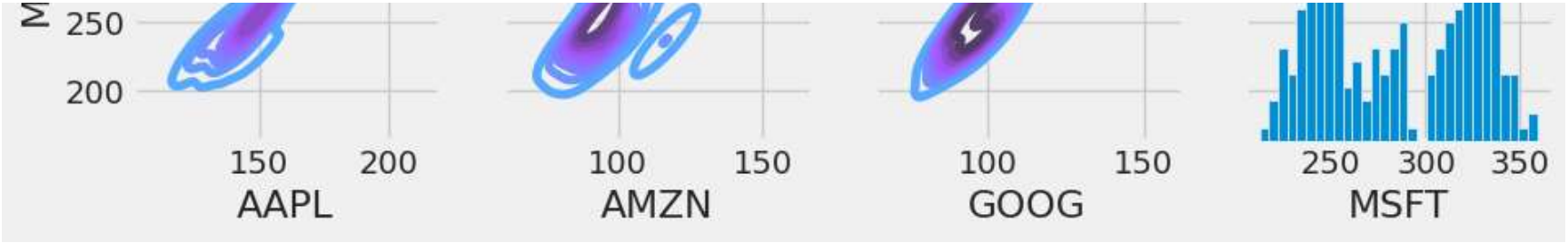
Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x7fe7f4bdbf40>
```


AAPL
AMZN
GOOG
MSFT
0.05
0.00
−0.05
0.1
0.0
−0.1
0.10
0.05
0.00
−0.05
−0.10
0.10
0.05
0.00

In [15]:
```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
returns_fig = sns.PairGrid(closing_df)

# Using map_upper we can specify what the upper triangle will look like.
returns_fig.map_upper(plt.scatter,color='purple')

# We can also define the lower triangle in the figure, inclufing the plot type (kde) or the color map (BluePurple)
returns_fig.map_lower(sns.kdeplot,cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily return
returns_fig.map_diag(plt.hist,bins=30)
```

Out[15]:
```
<seaborn.axisgrid.PairGrid at 0x7fe7e43c8580>
```

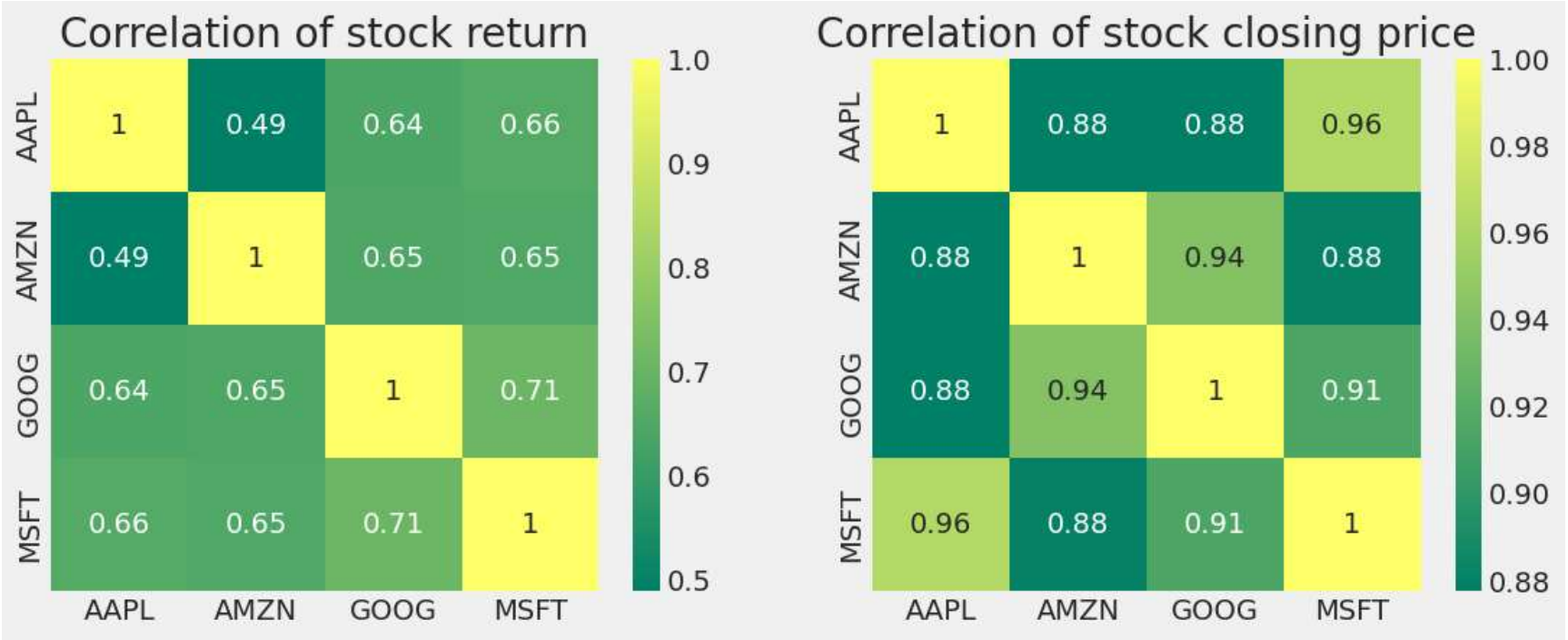

AAPL
180
160
140
AMZN
150
125
100
75
GOOG
160
140
120
100
80
400
350
300



Finally, we can create a correlation plot to obtain actual numerical values for the correlation between the stocks' daily return values. When comparing the closing prices, we notice an interesting relationship between Microsoft and Apple.

In [16]:
```
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')
```

Out[16]:
```
Text(0.5, 1.0, 'Correlation of stock closing price')
```



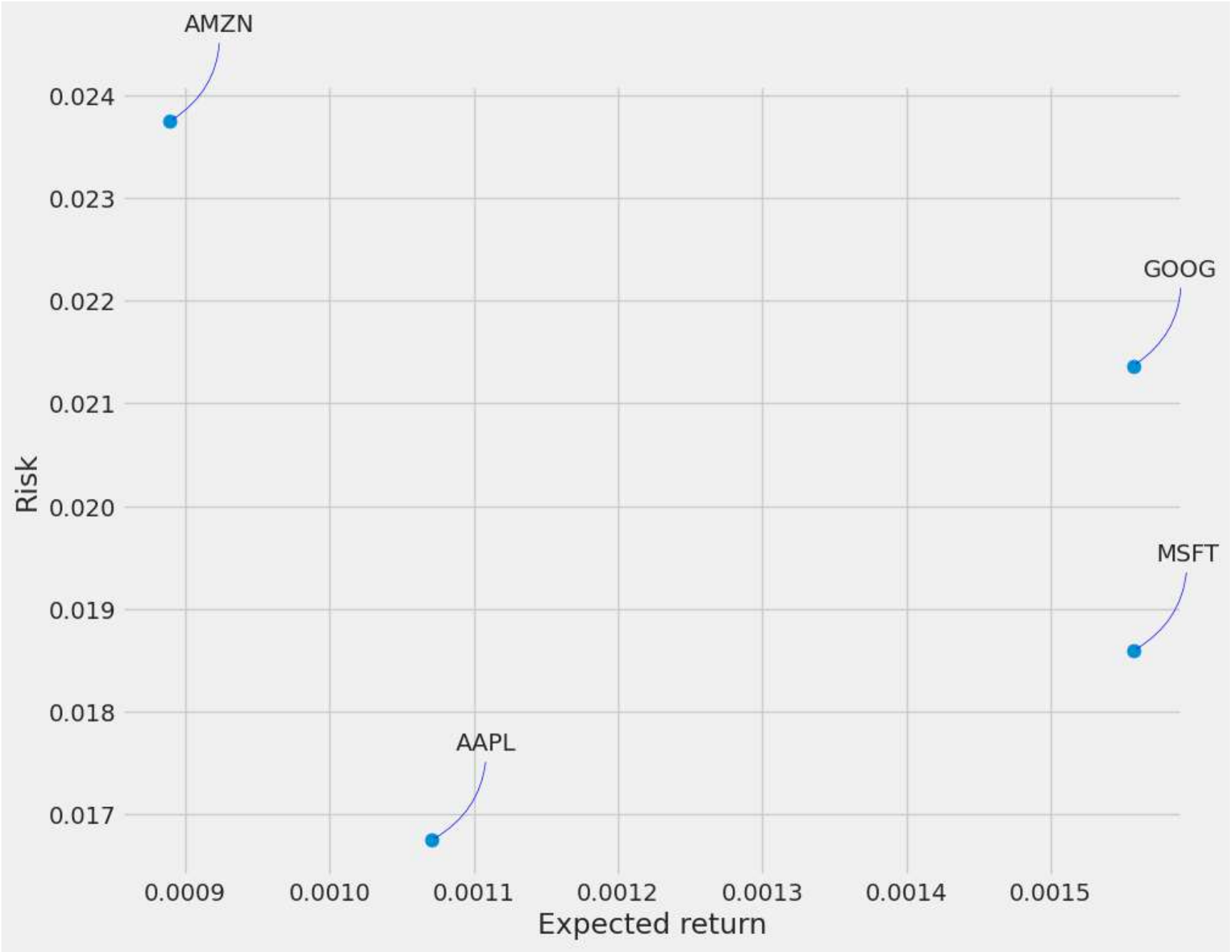
Just as we suspected from our 'PairPlot,' we can see here, both numerically and visually, that Microsoft and Amazon have the strongest correlation in daily stock returns. It's also interesting to note that all the technology companies are positively correlated.

5. How much value do we put at risk by investing in a particular stock?

We can quantify risk in various ways, and one of the most basic methods, using the information we've gathered on daily percentage returns, is to compare the expected return with the standard deviation of these returns.

In [17]:
```
rets = tech_rets.dropna()

area = np.pi * 20

plt.figure(figsize=(10, 8))
plt.scatter(rets.mean(), rets.std(), s=area)
plt.xlabel('Expected return')
plt.ylabel('Risk')
```

```
for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points', ha='right', va='bottom',
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-0.3'))
```



6. Predicting the closing price of APPLE inc:

In [18]:
```
# Get the stock quote
df = pdr.get_data_yahoo('AAPL', start='2012-01-01', end=datetime.now())
# Show teh data
df
```

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

Out[18]:

Date	Open	High	Low	Close	Adj Close	Volume
2012-01-03	14.621429	14.732143	14.607143	14.686786	12.466093	302220800
2012-01-04	14.642857	14.810000	14.617143	14.765714	12.533088	260022000
2012-01-05	14.819643	14.948214	14.738214	14.929643	12.672231	271269600
2012-01-06	14.991786	15.098214	14.972143	15.085714	12.804702	318292800
2012-01-09	15.196429	15.276786	15.048214	15.061786	12.784390	394024400
...	...	...	...	...	...	...
2023-10-10	178.100006	179.720001	177.949997	178.389999	178.389999	43698000
2023-10-11	178.199997	179.850006	177.600006	179.800003	179.800003	47551100
2023-10-12	180.070007	182.339996	179.039993	180.710007	180.710007	56743100
2023-10-13	181.419998	181.929993	178.139999	178.850006	178.850006	51427100
2023-10-16	176.750000	179.080002	176.509995	178.720001	178.720001	52472600

2966 rows × 6 columns

In [19]:
```
plt.figure(figsize=(16,6))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.show()
```

In [20]:
```
# Create a new dataframe with only the 'Close column
data = df.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
```

Out[20]:
```
2818
```

In [21]:
```
# Scale the data
from sklearn.preprocessing import MinMaxScaler

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

scaled_data
```

Out[21]:
```
array([[0.00405082],
       [0.0044833 ],
       [0.00538153],
       ...,
       [0.91375466],
       [0.90356301],
       [0.90285067]])
```

In [22]:
```
# Create the training data set
# Create the scaled training data set
train_data = scaled_data[0:int(training_data_len), :]
# Split the data into x_train and y_train data sets
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))
# x_train.shape
```

```
[array([0.00405082, 0.0044833 , 0.00538153, 0.0062367 , 0.00610559,
       0.00640108, 0.00626606, 0.00603905, 0.00572986, 0.0066868 ,
       0.0075498 , 0.00728366, 0.00582575, 0.00721712, 0.00584728,
       0.01098419, 0.01058694, 0.01110552, 0.01222684, 0.01290588,
       0.01284914, 0.01263975, 0.0135321 , 0.01437162, 0.01532269,
       0.01685887, 0.02008583, 0.02013475, 0.02193121, 0.02327365,
       0.02096645, 0.02185489, 0.02183728, 0.02432844, 0.02397423,
       0.02462979, 0.02580786, 0.02646344, 0.02835186, 0.02972757,
       0.03012483, 0.03026377, 0.02791156, 0.02734404, 0.0274282 ,
       0.02963952, 0.03026182, 0.0315984 , 0.03474903, 0.0389525 ,
       0.03816582, 0.03816777, 0.04120687, 0.04215794, 0.04148084,
       0.04086246, 0.04021863, 0.04235754, 0.04382523, 0.04443971])]
[0.04292113229660477]

[array([0.00405082, 0.0044833 , 0.00538153, 0.0062367 , 0.00610559,
       0.00640108, 0.00626606, 0.00603905, 0.00572986, 0.0066868 ,
       0.0075498 , 0.00728366, 0.00582575, 0.00721712, 0.00584728,
       0.01098419, 0.01058694, 0.01110552, 0.01222684, 0.01290588,
       0.01284914, 0.01263975, 0.0135321 , 0.01437162, 0.01532269,
       0.01685887, 0.02008583, 0.02013475, 0.02193121, 0.02327365,
       0.02096645, 0.02185489, 0.02183728, 0.02432844, 0.02397423,
       0.02462979, 0.02580786, 0.02646344, 0.02835186, 0.02972757,
       0.03012483, 0.03026377, 0.02791156, 0.02734404, 0.0274282 ,
       0.02963952, 0.03026182, 0.0315984 , 0.03474903, 0.0389525 ,
       0.03816582, 0.03816777, 0.04120687, 0.04215794, 0.04148084,
       0.04086246, 0.04021863, 0.04235754, 0.04382523, 0.04443971]), array([0.0044833 , 0.00538153, 0.0062367 , 0.00610
559, 0.00640108,
       0.00626606, 0.00603905, 0.00572986, 0.0066868 , 0.0075498 ,
       0.00728366, 0.00582575, 0.00721712, 0.00584728, 0.01098419,
       0.01058694, 0.01110552, 0.01222684, 0.01290588, 0.01284914,
       0.01263975, 0.0135321 , 0.01437162, 0.01532269, 0.01685887,
       0.02008583, 0.02013475, 0.02193121, 0.02327365, 0.02096645,
       0.02185489, 0.02183728, 0.02432844, 0.02397423, 0.02462979,
       0.02580786, 0.02646344, 0.02835186, 0.02972757, 0.03012483,
       0.03026377, 0.02791156, 0.02734404, 0.0274282 , 0.02963952,
       0.03026182, 0.0315984 , 0.03474903, 0.0389525 , 0.03816582,
       0.03816777, 0.04120687, 0.04215794, 0.04148084, 0.04086246,
       0.04021863, 0.04235754, 0.04382523, 0.04443971, 0.04292113])]
[0.04292113229660477, 0.04090355083781154]
```

In [23]:

```
from keras.models import Sequential
from keras.layers import Dense, LSTM

# Build the LSTM model
```

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

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

# Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)
```

Out[23]:
```
2758/2758 [==============================] - 26s 7ms/step - loss: 0.0012
<keras.callbacks.History at 0x7fe7301a83a0>
```

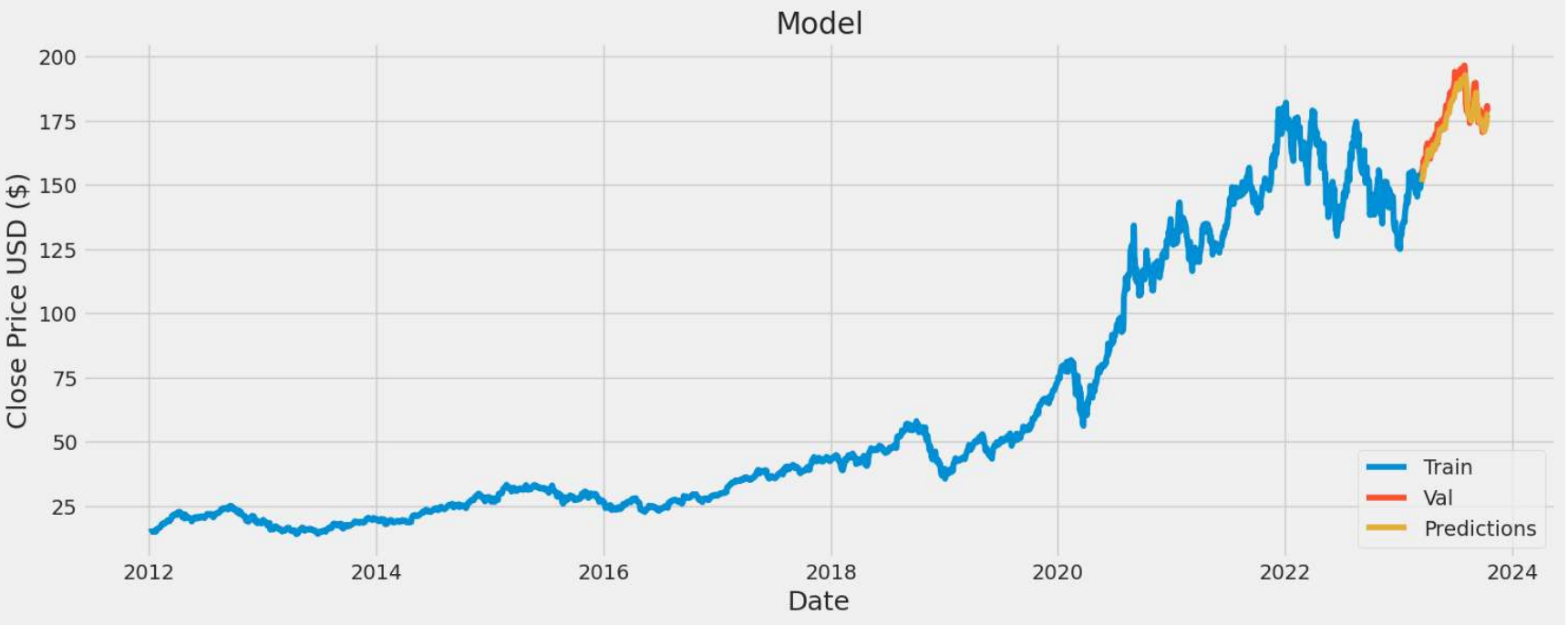
In [24]:
```
# Create the testing data set
# Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data[training_data_len - 60: , :]
# Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 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 ))

# 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
```

Out[24]:
```
5/5 [==============================] - 1s 5ms/step
3.716612883961819
```

In [25]:
```
# Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
```

```
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,6))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```

```
/tmp/ipykernel_28/2388977846.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
```



In [26]:
```
# Show the valid and predicted prices
valid
```

Out[26]:

Date	Close	Predictions
2023-03-16	155.850006	151.059204
2023-03-17	155.000000	152.141739
2023-03-20	157.399994	153.065460
2023-03-21	159.279999	154.243790
2023-03-22	157.830002	155.643723
...	...	...
2023-10-10	178.389999	174.853928
2023-10-11	179.800003	175.888870
2023-10-12	180.710007	176.836334
2023-10-13	178.850006	177.694641
2023-10-16	178.720001	177.903366

148 rows × 2 columns

Summary

We can learn the following information from this project:

- How to load stock market data from the YAHOO Finance website using yfinance.
- How to explore and visualize time-series data using Pandas, Matplotlib, and Seaborn.
- How to measure the correlation between stocks.
- How to measure the risk of investing in a particular stock.