# Data Project - Stock Market Analysis

techAnalysis-1000x500.jpg

Time Series data is a series of data points indexed in time order. Time series data is everywhere, so manipulating them is important for any data analyst or data scientist.

In this notebook, we will discover and explore data from the stock market, particularly some technology stocks (Apple, Amazon, Google, and Microsoft). We will learn how to use yfinance to get stock information, and visualize different aspects of it using Seaborn and Matplotlib. we will look at a few ways of analyzing the risk of a stock, based on its previous performance history. We will also be predicting future stock prices through a Long Short Term Memory (LSTM) method!

We'll be answering the following questions along the way:

```
1.) What was the change in price of the stock over time?
2.) What was the daily return of the stock on average?
3.) What was the moving average of the various stocks?
4.) What was the correlation between different stocks'?
5.) How much value do we put at risk by investing in a particular stock?
6.) How can we attempt to predict future stock behavior? (Predicting the closing price stock price of
```

---

# 1. What was the change in price of the stock overtime?

In this section we'll go over how to handle requesting stock information with pandas, and how to analyze basic attributes of a stock.

```
!pip install  yfinance
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.10/dist-packages 
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/dist-packages 
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packages
```

```
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pack
```

```
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)
```

```
yfinance: pandas_datareader support is deprecated & semi-broken so will be removed in a
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```

| Date | Open | High | Low | Close | Adj Close | Volume | company_name |
|---|---|---|---|---|---|---|---|
| **2024-05-28** | 179.929993 | 182.240005 | 179.490005 | 182.149994 | 182.149994 | 29927000 | AMAZON |
| **2024-05-29** | 181.699997 | 184.080002 | 181.550003 | 182.020004 | 182.020004 | 32009300 | AMAZON |
| **2024-05-30** | 181.309998 | 181.339996 | 178.360001 | 179.320007 | 179.320007 | 29249200 | AMAZON |
| **2024-05-31** | 178.300003 | 179.210007 | 173.869995 | 176.440002 | 176.440002 | 58903900 | AMAZON |
| **2024-06-03** | 177.699997 | 178.699997 | 175.919998 | 178.339996 | 178.339996 | 30786600 | AMAZON |
| **2024-06-04** | 177.639999 | 179.820007 | 176.440002 | 179.339996 | 179.339996 | 27198400 | AMAZON |

Import Libraries: Import necessary libraries. Set Styles: Set styles for plotting. Read Stock Data: Set up the stock symbols and download their data from Yahoo Finance. Add Company Names: Add a column to each DataFrame indicating the company name. Concatenate Data: Combine all individual DataFrames into one.

**Quick note:** Using `globals()` is a sloppy way of setting the `DataFrame` names, but it's simple. Now we have our data, let's perform some basic data analysis and check our data.

## Descriptive Statistics about the Data

`.describe()` generates descriptive statistics. Descriptive statistics include those that summarize the central tendency, dispersion, and shape of a dataset's distribution, excluding `NaN` values.

Analyzes both numeric and object series, as well as `DataFrame` column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

```
# Summary Stats
AAPL.describe()
```

| | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| **count** | 251.000000 | 251.000000 | 251.000000 | 251.000000 | 251.000000 | 2.510000e+02 |
| **mean** | 182.746534 | 184.211999 | 181.367410 | 182.821753 | 182.296098 | 5.750452e+07 |
| **std** | 8.738375 | 8.552274 | 8.749900 | 8.690825 | 8.641298 | 1.844647e+07 |
| **min** | 165.350006 | 166.399994 | 164.080002 | 165.000000 | 164.776505 | 1.421504e+07 |
| **25%** | 175.244995 | 177.025002 | 173.660004 | 175.280006 | 174.800789 | 4.657160e+07 |
| **50%** | 183.419998 | 184.899994 | 181.809998 | 183.630005 | 182.988205 | 5.302030e+07 |
| **75%** | 190.239998 | 191.540001 | 189.189995 | 190.139999 | 189.854996 | 6.438910e+07 |
| **max** | 198.020004 | 199.619995 | 197.000000 | 198.110001 | 197.589523 | 1.632241e+08 |

We have only 255 records in one 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.

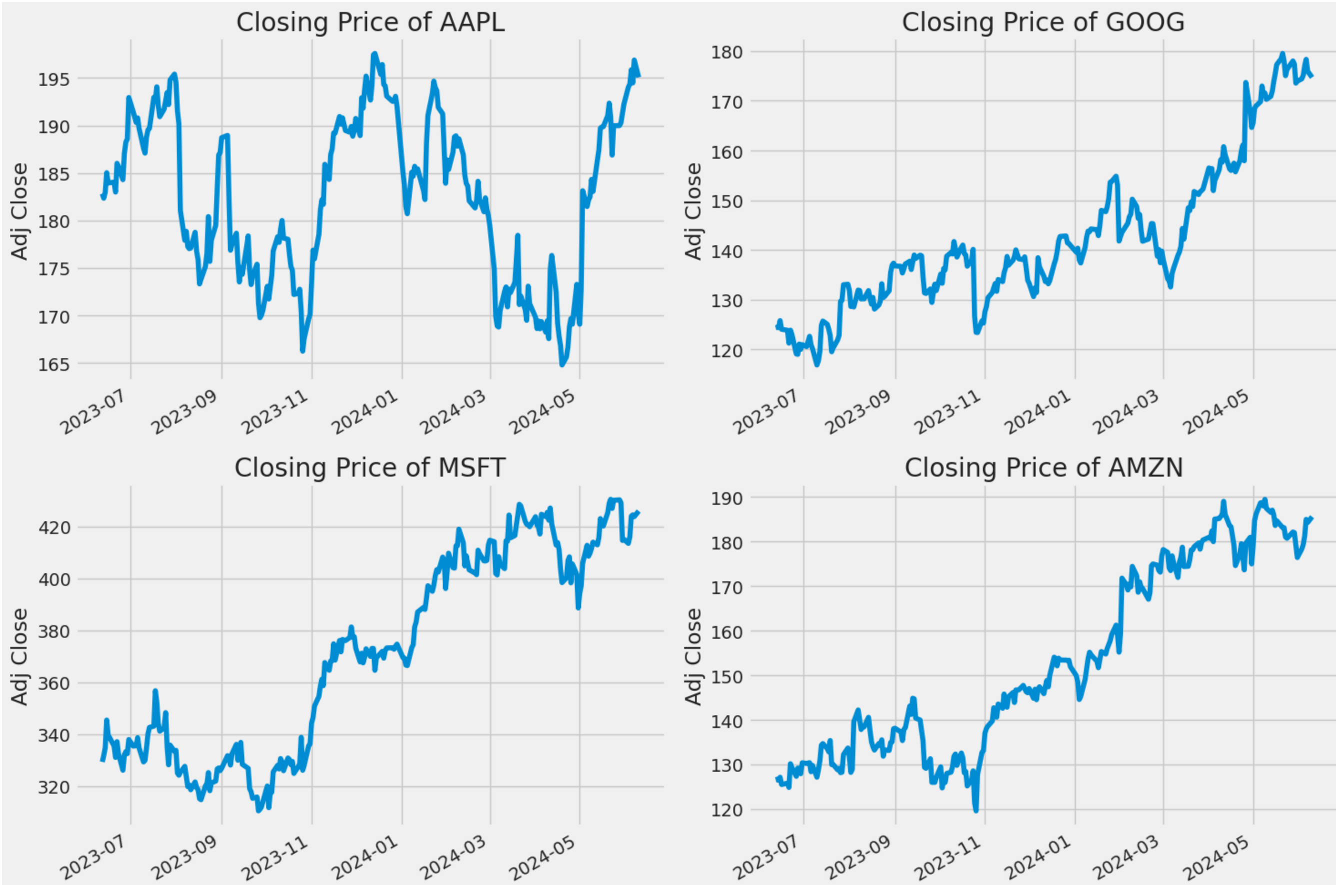
```
# General info
AAPL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2023-06-12 to 2024-06-10
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 the stock is traded during the regular trading day. A stock’s closing price is the standard benchmark used by investors to track its performance over time.

```
# 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()
```

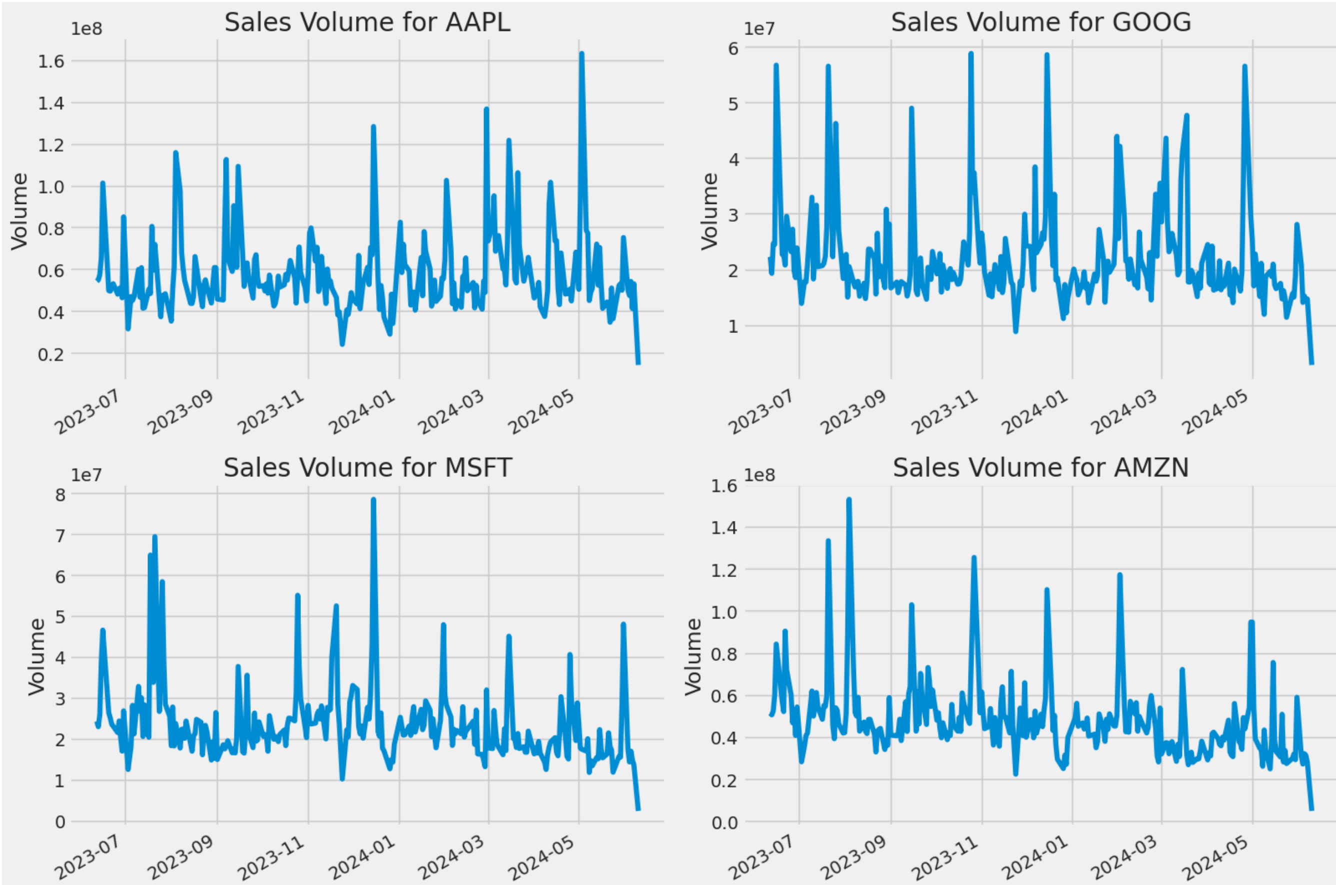


Closing Price of AAPL
Adj Close
195
190
185
180
175
170
165
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
Closing Price of GOOG
Adj Close
180
170
160
150
140
130
120
Closing Price of MSFT
Adj Close
420
400
380
360
340
320
Closing Price of AMZN
Adj Close
190
180
170
160
150
140
130
120

## Volume of Sales

Volume is the amount of an asset or security that changes hands over some period of time, often over the course of a day. For instance, the stock trading volume would refer to the number of shares of security traded between its daily open and close. Trading volume, and changes to volume over the course of time, are important inputs for technical traders.

```
# 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()
```

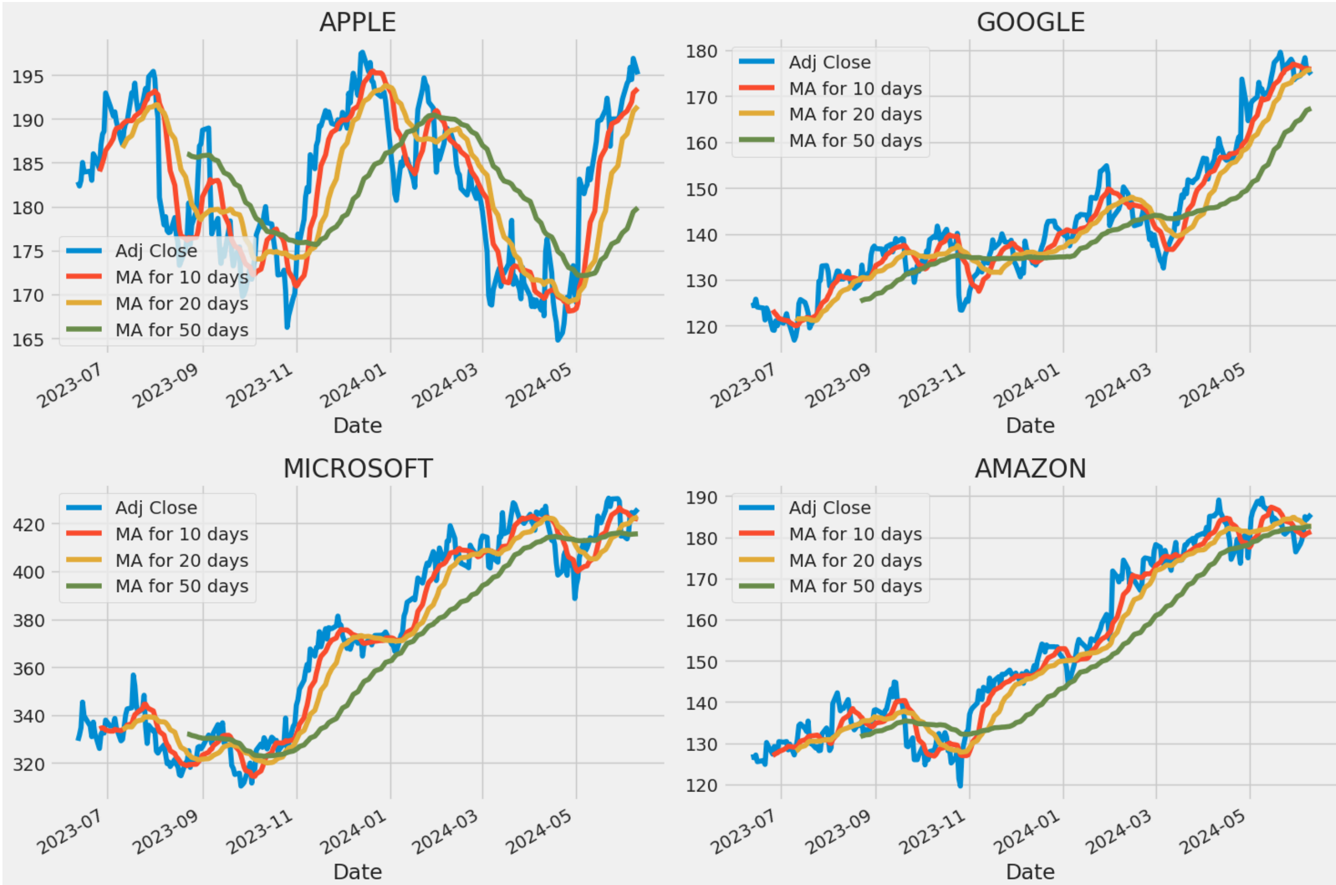

Sales Volume for AAPL
1e8
Volume
Sales Volume for GOOG
1e7
Volume
Sales Volume for MSFT
1e7
Volume
Sales Volume for AMZN
1e8
Volume
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05

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 10 days, 20 minutes, 30 weeks, or any time period the trader chooses.

```
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()
```

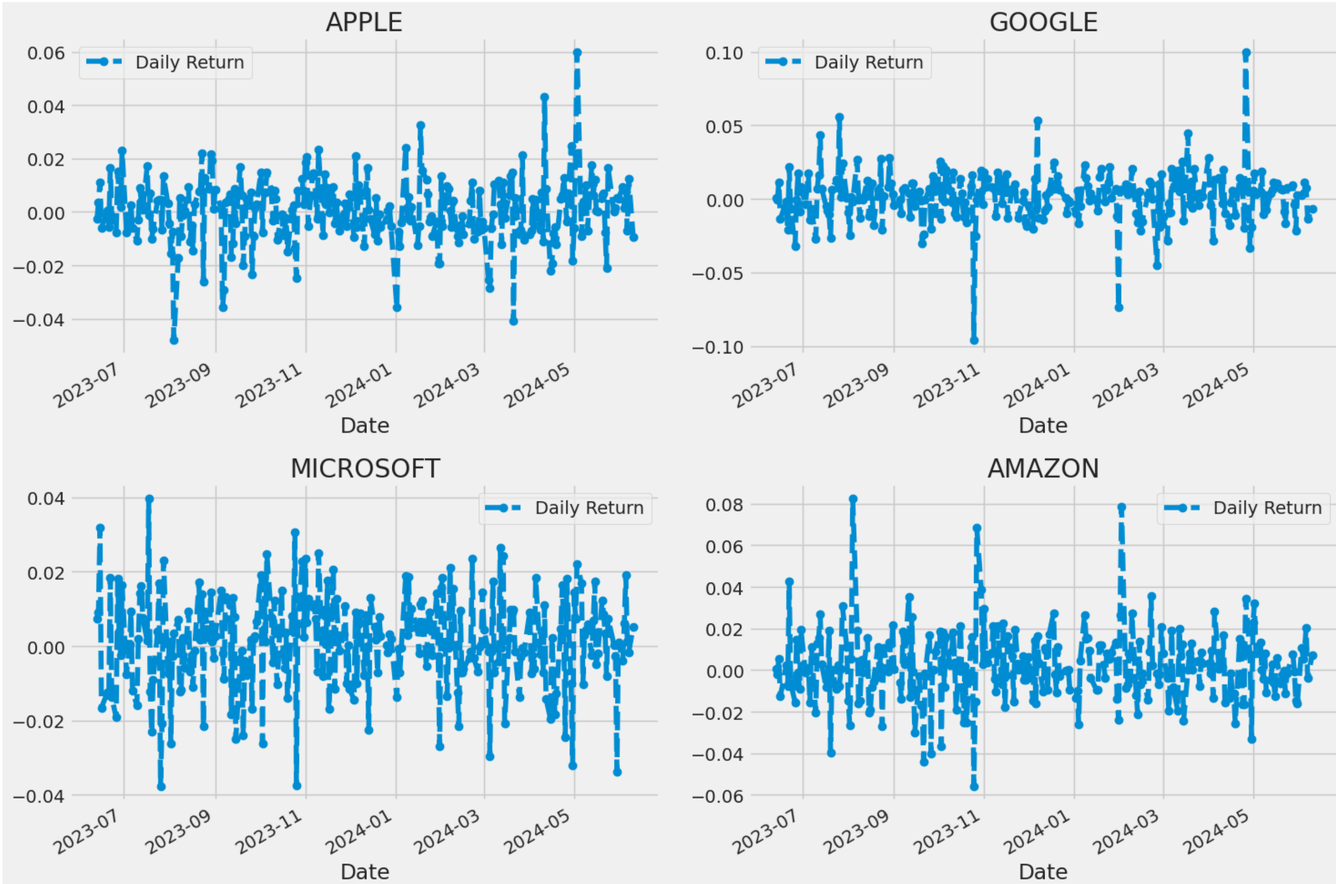

APPLE
GOOGLE
MICROSOFT
AMAZON
Adj Close
MA for 10 days
MA for 20 days
MA for 50 days
Date
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05

We see in the graph 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?

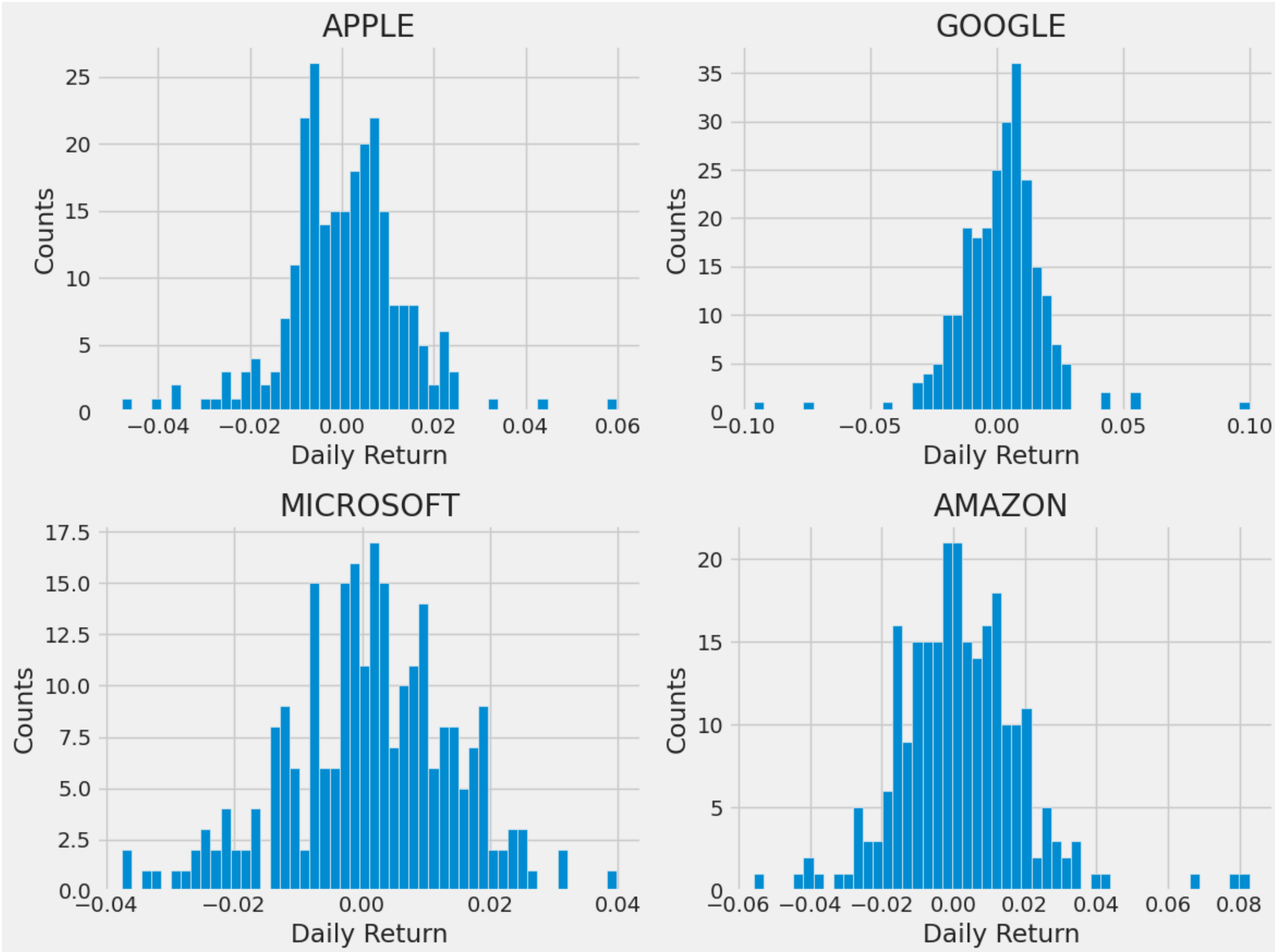
Now that we've done some baseline analysis, let's go ahead and dive a little deeper. We're now going to analyze the risk of the stock. In order to do so we'll need to take a closer look at the daily changes of the stock, and not just its absolute value. Let's go ahead and use pandas to retrieve teh daily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

```
# 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()
```


APPLE
Daily Return
0.06
0.04
0.02
0.00
−0.02
−0.04
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
Date
GOOGLE
Daily Return
0.10
0.05
0.00
−0.05
−0.10
Date
MICROSOFT
Daily Return
0.04
0.02
0.00
−0.02
−0.04
Date
AMAZON
Daily Return
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Date

Great, 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.

```
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()
```


APPLE
GOOGLE
MICROSOFT
AMAZON
Counts
Daily Return

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

Correlation is a statistic that measures the degree to which two variables move in relation to each other which has a value that must fall between -1.0 and +1.0. Correlation measures association, but doesn't show if x causes y or vice versa — or if the association is caused by a third factor[1].

Now what if we wanted to analyze the returns of all the stocks in our list? Let's go ahead and build a DataFrame with all the ['Close'] columns for each of the stocks dataframes.

```
# 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
```

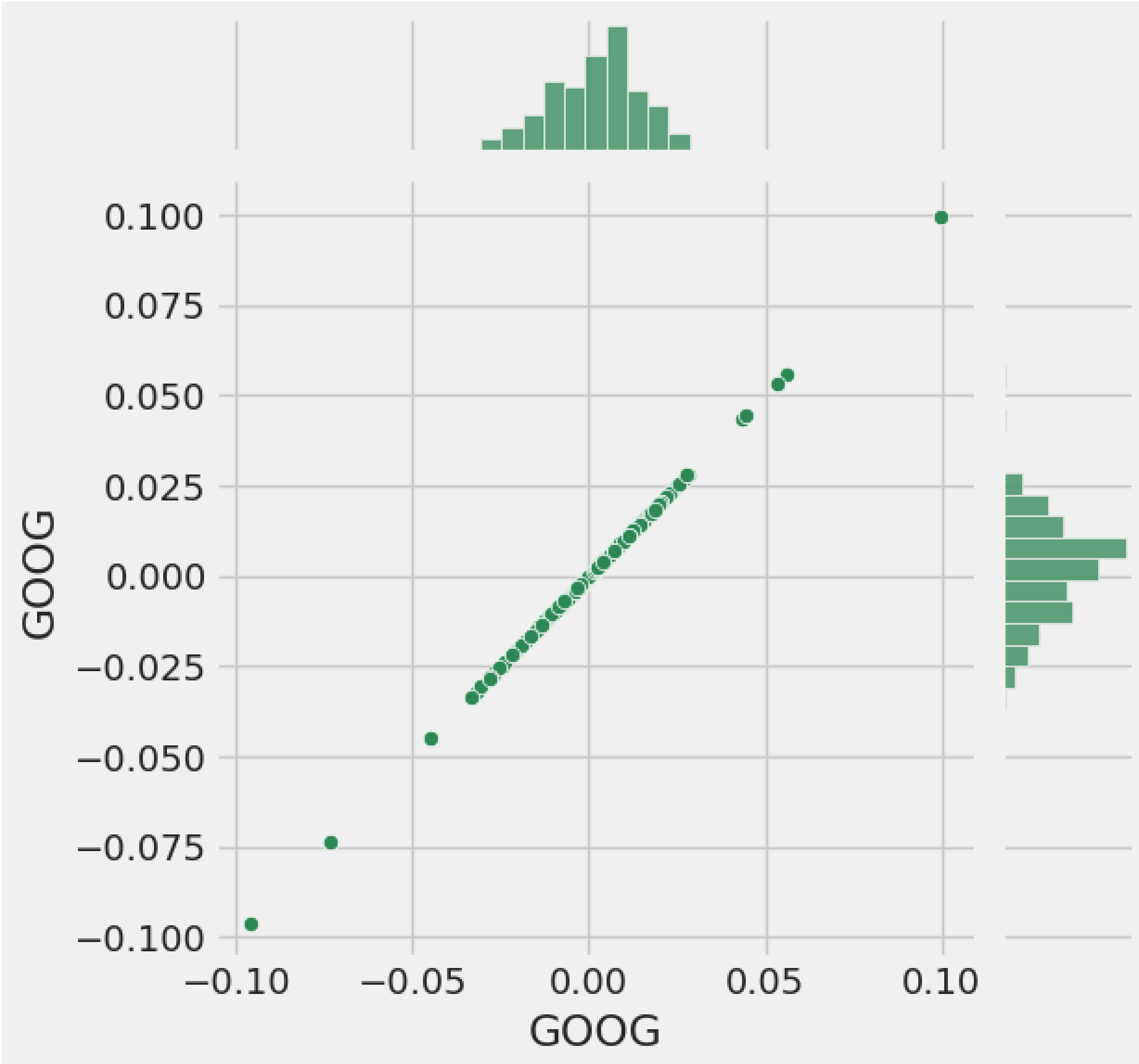
| Ticker | AAPL | AMZN | GOOG | MSFT |
|---|---|---|---|---|
| Date | | | | |
| **2023-06-12** | NaN | NaN | NaN | NaN |
| **2023-06-13** | -0.002612 | 0.000711 | 0.000643 | 0.007353 |
| **2023-06-14** | 0.003491 | -0.001895 | -0.000402 | 0.009124 |
| **2023-06-15** | 0.011199 | 0.005458 | 0.011336 | 0.031897 |
| **2023-06-16** | -0.005860 | -0.012745 | -0.013753 | -0.016576 |

Next steps: View recommended plots

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

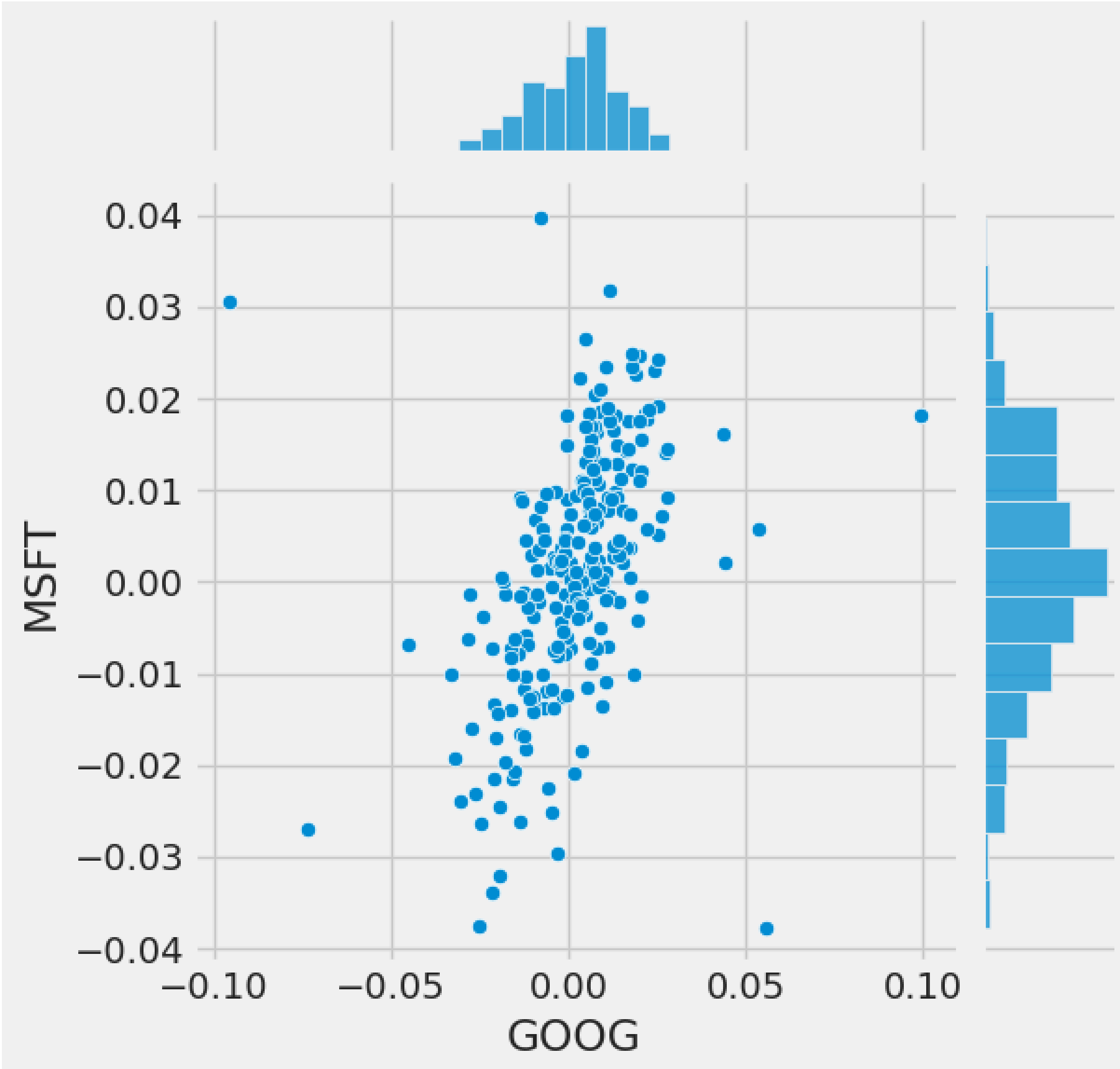
```
# Comparing Google to itself should show a perfectly linear relationship
sns.jointplot(x='GOOG', y='GOOG', data=tech_rets, kind='scatter', color='seagreen')
```

```
<seaborn.axisgrid.JointGrid at 0x7fb67115ff70>
```




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

```
<seaborn.axisgrid.JointGrid at 0x7fb67107b250>
```

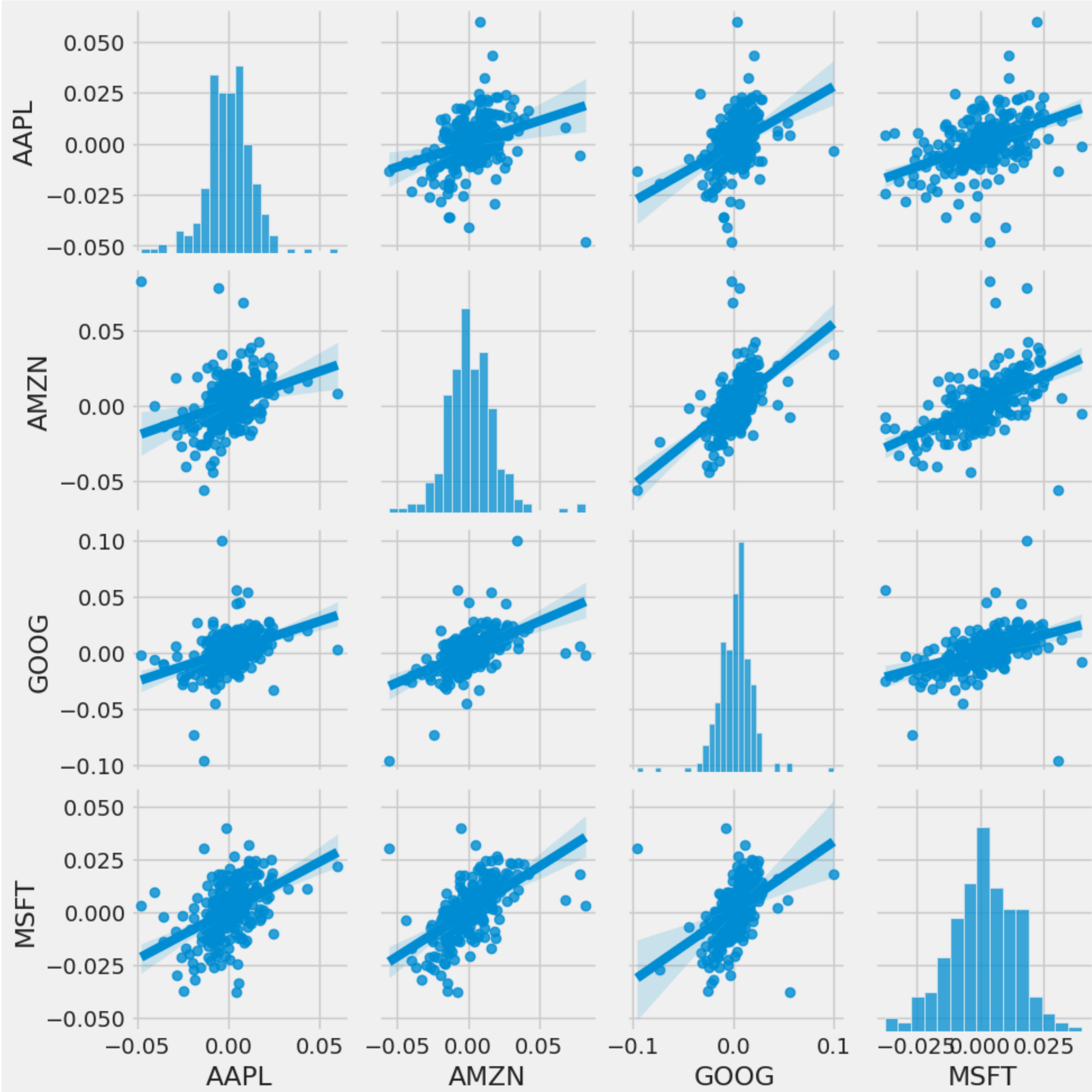



So now we can see that if two stocks are perfectly (and positivley) correlated with each other a linear relationship bewteen its daily return values should occur.

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

```
# We can simply call pairplot on our DataFrame for an automatic visual analysis
# of all the comparisons

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

```
<seaborn.axisgrid.PairGrid at 0x7fb6734a2860>
```

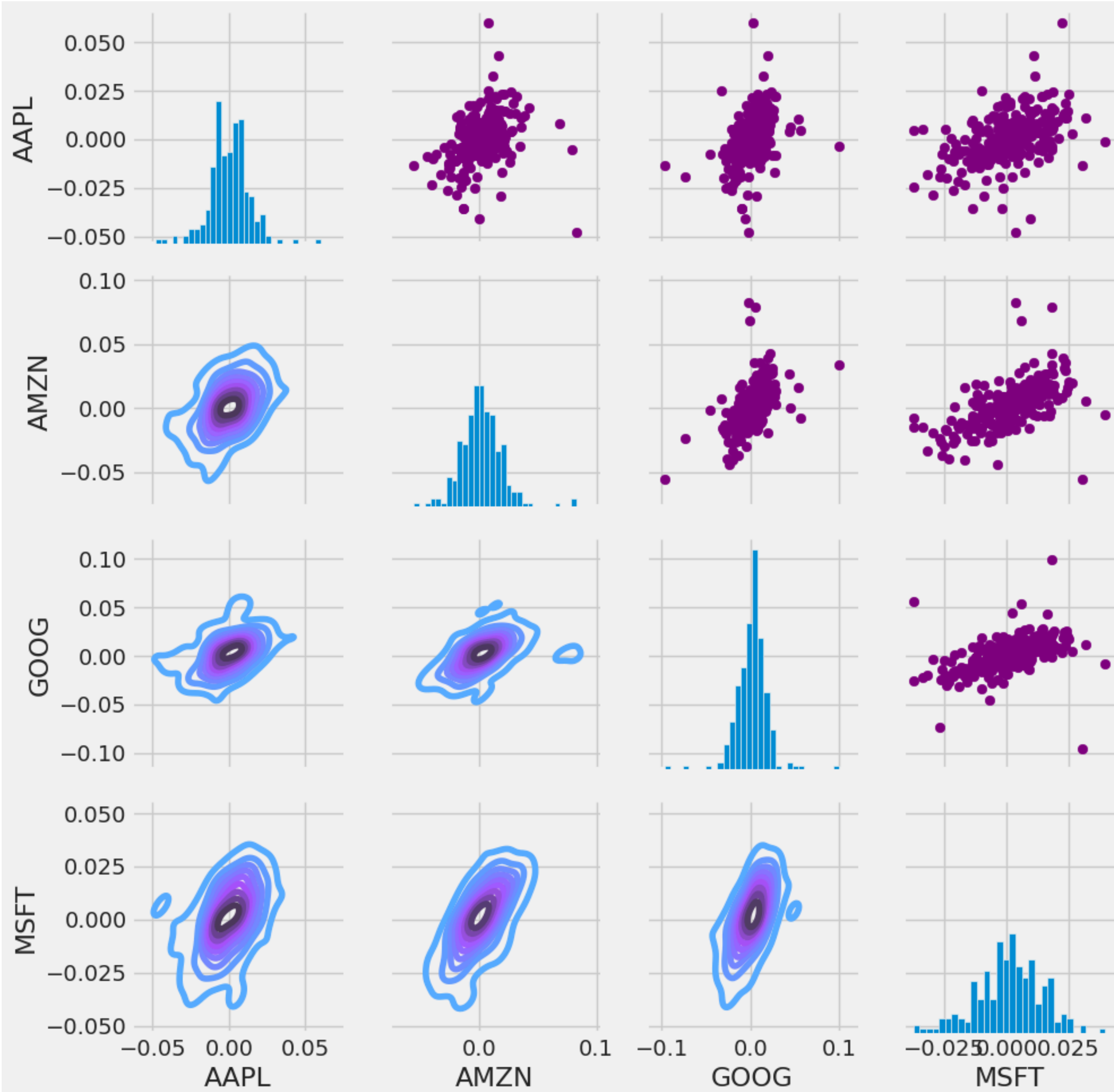


Above we can see all the relationships on daily returns between all the stocks. A quick glance shows an interesting correlation between Google and Amazon daily returns. It might be interesting to investigate that individual comaprison.

While the simplicity of just calling `sns.pairplot()` is fantastic we can also use `sns.PairGrid()` for full control of the figure, including what kind of plots go in the diagonal, the upper triangle, and the lower triangle. Below is an example of utilizing the full power of seaborn to achieve this result.

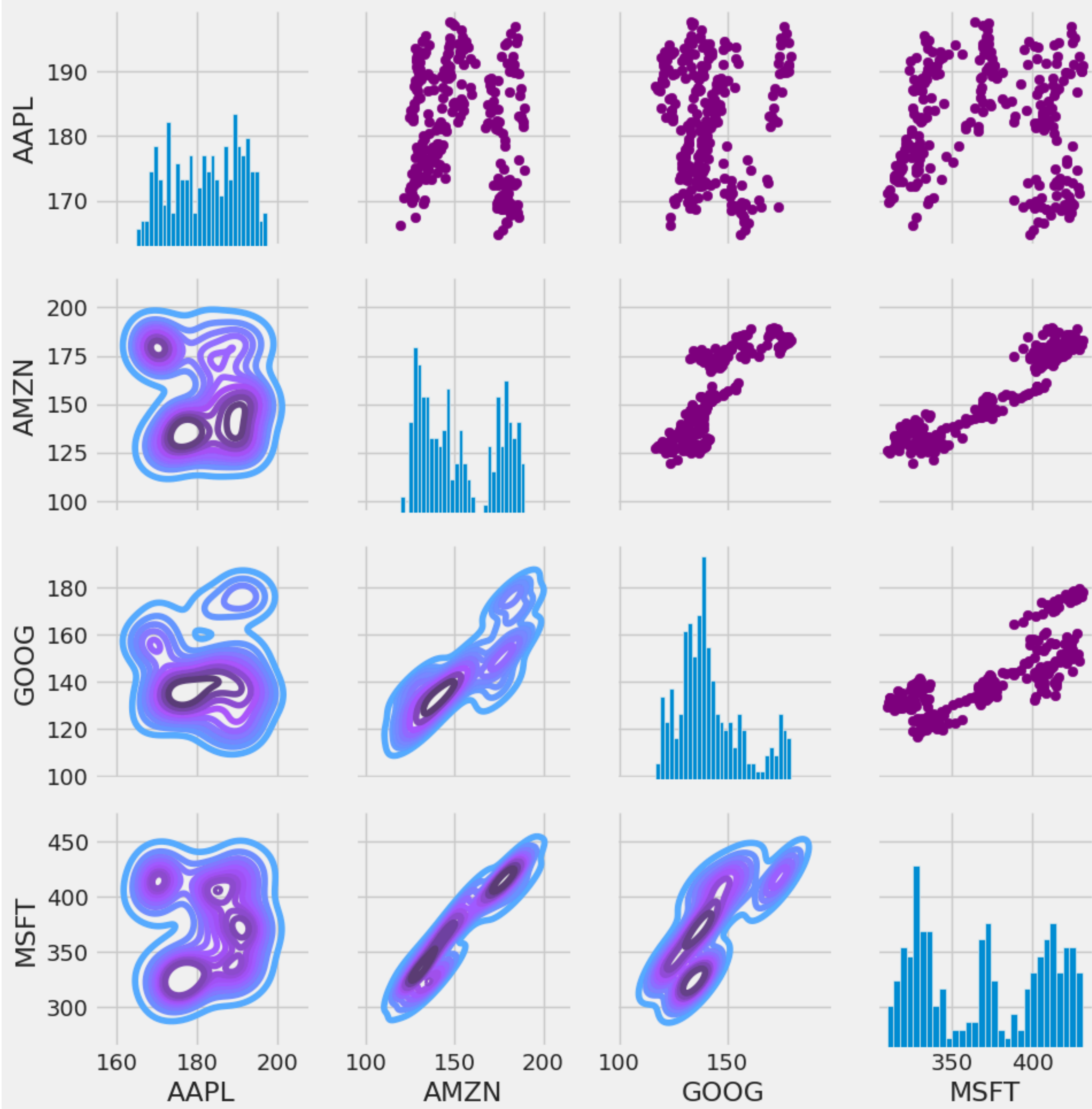
```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
return_fig = sns.PairGrid(tech_rets.dropna())

# 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)
```

<seaborn.axisgrid.PairGrid at 0x7fb670305660>



```
# 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
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)
```

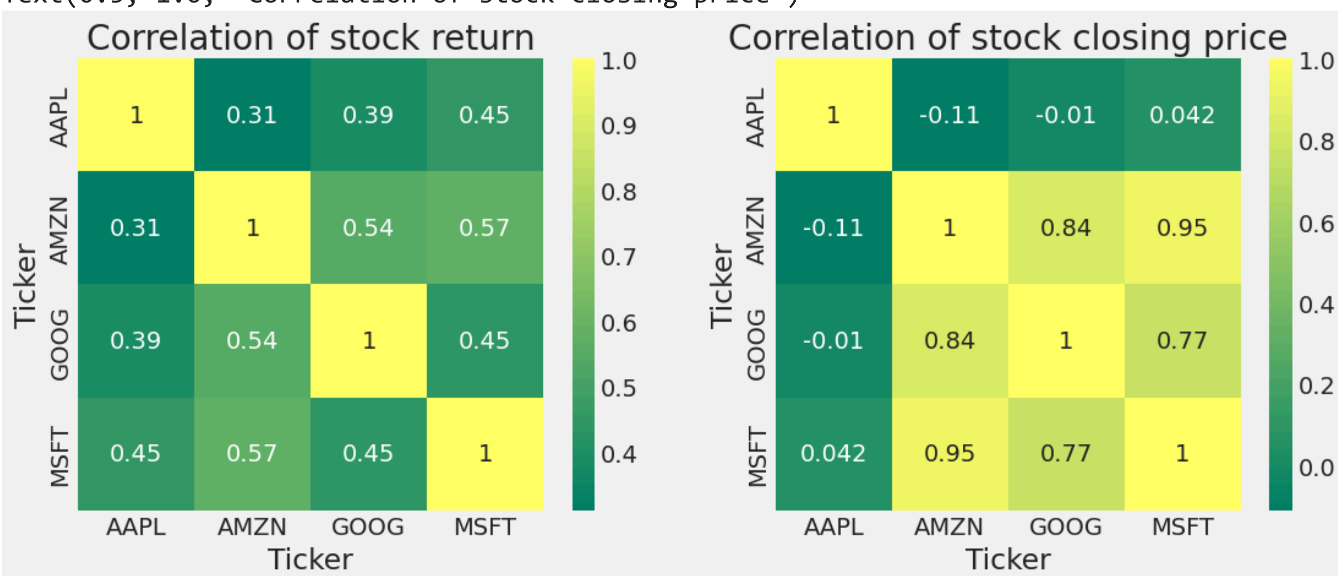
```
<seaborn.axisgrid.PairGrid at 0x7fb66991a6b0>
```



Finally, we could also do a correlation plot, to get actual numerical values for the correlation between the stocks' daily return values. By comparing the closing prices, we see an interesting relationship between Microsoft and Apple.

```
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')
```

```
Text(0.5, 1.0, 'Correlation of stock closing price')
```

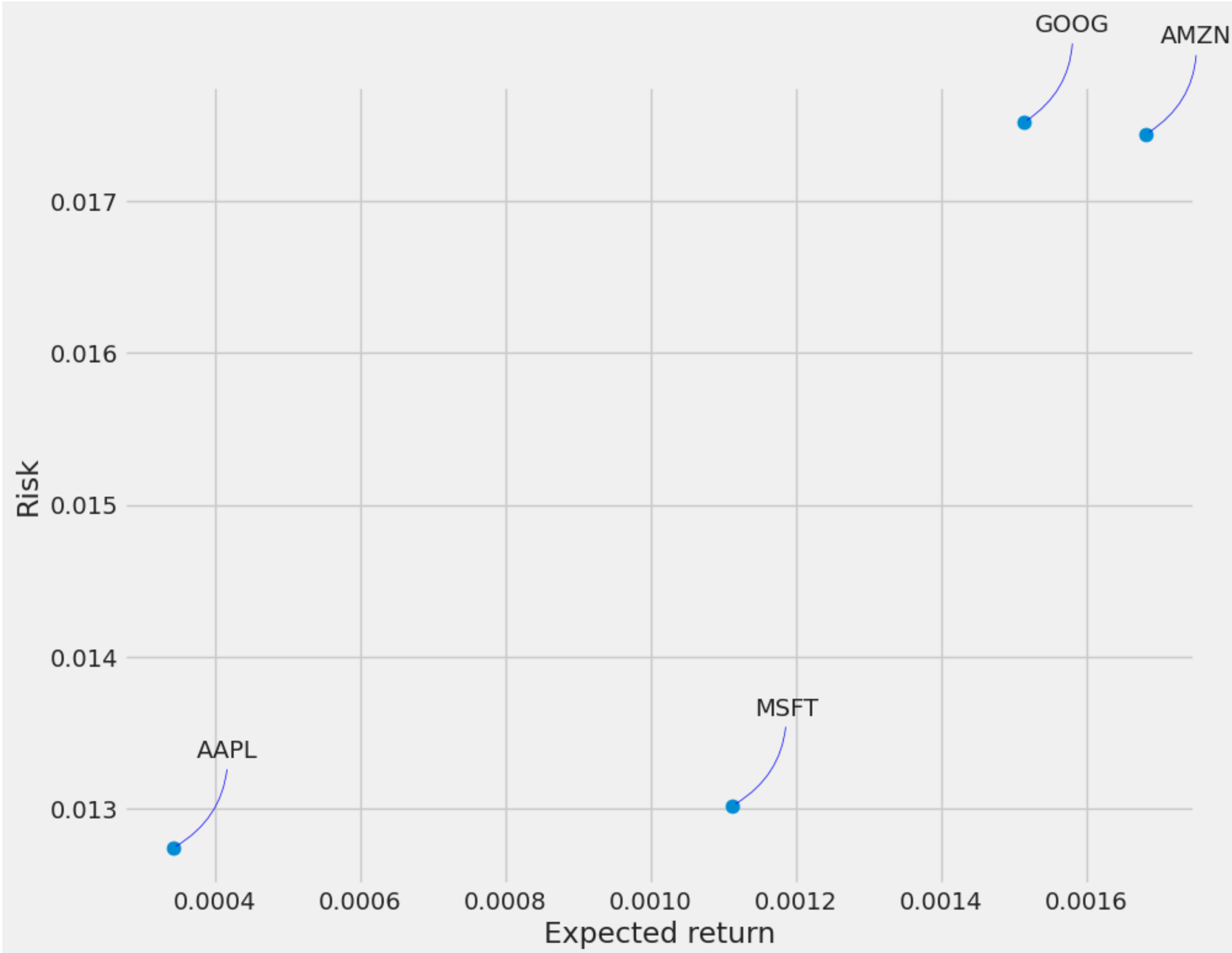



Just like we suspected in our `PairPlot` we see here numerically and visually that Microsoft and Amazon had the strongest correlation of daily stock return. It's also interesting to see that all the technology comapnies are positively correlated.

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

There are many ways we can quantify risk, one of the most basic ways using the information we've gathered on daily percentage returns is by comparing the expected return with the standard deviation of the daily returns.

```
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',
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-0.
```




## 6. Predicting the closing price stock price of APPLE inc:

```
pip install arch
```

```
Collecting arch
  Downloading arch-7.0.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (983
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 983.4/983.4 kB 5.5 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.22.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scipy>=1.8 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pandas>=1.4 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: statsmodels>=0.12 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from pats
Installing collected packages: arch
Successfully installed arch-7.0.0
```

```
# ARIMA and GARCH models
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model
```

```
# 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
```

| | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| **Date** | | | | | | |
| **2012-01-03** | 14.621429 | 14.732143 | 14.607143 | 14.686786 | 12.416981 | 302220800 |
| **2012-01-04** | 14.642857 | 14.810000 | 14.617143 | 14.765714 | 12.483709 | 260022000 |
| **2012-01-05** | 14.819643 | 14.948214 | 14.738214 | 14.929643 | 12.622306 | 271269600 |
| **2012-01-06** | 14.991786 | 15.098214 | 14.972143 | 15.085714 | 12.754258 | 318292800 |
| **2012-01-09** | 15.196429 | 15.276786 | 15.048214 | 15.061786 | 12.734029 | 394024400 |
| **...** | ... | ... | ... | ... | ... | ... |
| **2024-06-04** | 194.639999 | 195.320007 | 193.029999 | 194.350006 | 194.350006 | 47471400 |
| **2024-06-05** | 195.399994 | 196.899994 | 194.869995 | 195.869995 | 195.869995 | 54156800 |
| **2024-06-06** | 195.690002 | 196.500000 | 194.169998 | 194.479996 | 194.479996 | 41181800 |
| **2024-06-07** | 194.649994 | 196.940002 | 194.139999 | 196.889999 | 196.889999 | 53044700 |
| **2024-06-10** | 197.199997 | 197.281693 | 194.830002 | 195.128998 | 195.128998 | 14358294 |

3129 rows × 6 columns

Next steps: View recommended plots

```
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()
```

```
# 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
```

```
2973
```

```
# Scale the data
from sklearn.preprocessing import MinMaxScaler

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

scaled_data
```

```
array([[0.00401431],
       [0.00444289],
       [0.00533302],
       ...,
       [0.98028912],
       [0.99337541],
       [0.98381319]])
```

```
# 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([0.00401431, 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,
        0.00620958, 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]
```