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 APPLE inc using LSTM)
```

Getting the Data

The first step is to get the data and load it to memory. We will get our stock data from the Yahoo Finance website. Yahoo Finance is a rich resource of financial market data and tools to find compelling investments. To get the data from Yahoo Finance, we will be using yfinance library which offers a threaded and Pythonic way to download market data from Yahoo. Check this article to learn more about yfinance: [Reliably download historical market data from with Python]()

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 -q yfinance

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

from pandas_datareader.data import DataReader
import yfinance as yf
from pandas_datareader import data as pdr

yf.pdr_override()

from datetime import datetime

# Define a list of tech stocks for analysis
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

# Set up the time range for data retrieval
end_date = datetime.now()
start_date = datetime(end_date.year - 1, end_date.month, end_date.day)

# Download stock data for each tech stock in the list
for stock_symbol in tech_list:
    globals()[stock_symbol] = yf.download(stock_symbol, start_date,
end_date)

# Create a list of company dataframes and corresponding names
company_list = [AAPL, GOOG, MSFT, AMZN]
company_names = ["APPLE", "GOOGLE", "MICROSOFT", "AMAZON"]

# Assign company names to their respective dataframes
for company_df, name in zip(company_list, company_names):
    company_df["company_name"] = name

# Concatenate individual company dataframes into one dataframe
combined_df = pd.concat(company_list, axis=0)

# Display the last 10 rows of the combined dataframe
combined_df.tail(10)
```

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

                  Open        High         Low       Close   Adj Close  \
Date

2024-01-12  155.389999  156.199997  154.009995  154.619995  154.619995

2024-01-16  153.529999  154.990005  152.149994  153.160004  153.160004

2024-01-17  151.490005  152.149994  149.910004  151.710007  151.710007

2024-01-18  152.770004  153.779999  151.820007  153.500000  153.500000

2024-01-19  153.830002  155.759995  152.740005  155.339996  155.339996

2024-01-22  156.889999  157.050003  153.899994  154.779999  154.779999

2024-01-23  154.850006  156.210007  153.929993  156.020004  156.020004

2024-01-24  157.800003  158.509995  156.479996  156.869995  156.869995

2024-01-25  156.949997  158.509995  154.550003  157.750000  157.750000

2024-01-26  158.419998  160.720001  157.910004  159.119995  159.119995


              Volume company_name
Date
2024-01-12  40460300       AMAZON
2024-01-16  41384600       AMAZON
2024-01-17  34953400       AMAZON
2024-01-18  37850200       AMAZON
2024-01-19  51033700       AMAZON
2024-01-22  43687500       AMAZON
2024-01-23  37986000       AMAZON
2024-01-24  48547300       AMAZON
2024-01-25  43638600       AMAZON
2024-01-26  51001100       AMAZON

<google.colab._quickchart_helpers.SectionTitle at 0x79ee793af130>

from matplotlib import pyplot as plt
_df_25['Open'].plot(kind='hist', bins=20, title='Open')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_26['High'].plot(kind='hist', bins=20, title='High')
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_27['Low'].plot(kind='hist', bins=20, title='Low')
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_28['Close'].plot(kind='hist', bins=20, title='Close')
plt.gca().spines[['top', 'right',]].set_visible(False)

<google.colab._quickchart_helpers.SectionTitle at 0x79eece2ba710>

from matplotlib import pyplot as plt
_df_29.plot(kind='scatter', x='Open', y='High', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_30.plot(kind='scatter', x='High', y='Low', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_31.plot(kind='scatter', x='Low', y='Close', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_32.plot(kind='scatter', x='Close', y='Adj Close', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

<google.colab._quickchart_helpers.SectionTitle at 0x79ee7b94ed10>

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['Open']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_33.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Open')
```

```python
from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['High']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_34.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('High')

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['Low']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_35.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Low')

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['Close']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])
```

```
fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_36.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Close')

<google.colab._quickchart_helpers.SectionTitle at 0x79ee7b94e800>

from matplotlib import pyplot as plt
_df_37['Open'].plot(kind='line', figsize=(8, 4), title='Open')
plt.gca().spines[['top', 'right']].set_visible(False)

from matplotlib import pyplot as plt
_df_38['High'].plot(kind='line', figsize=(8, 4), title='High')
plt.gca().spines[['top', 'right']].set_visible(False)

from matplotlib import pyplot as plt
_df_39['Low'].plot(kind='line', figsize=(8, 4), title='Low')
plt.gca().spines[['top', 'right']].set_visible(False)

from matplotlib import pyplot as plt
_df_40['Close'].plot(kind='line', figsize=(8, 4), title='Close')
plt.gca().spines[['top', 'right']].set_visible(False)
```

Reviewing the content of our data, we can see that the data is numeric and the date is the index of the data. Notice also that weekends are missing from the records.

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 Statistics for AAPL
summary_stats_aapl = AAPL.describe()
summary_stats_aapl

             Open        High         Low       Close   Adj Close  \
count  250.000000  250.000000  250.000000  250.000000  250.000000
mean   176.075441  177.626880  174.810200  176.348000  175.974101
std     14.271540   14.065161   14.232043   14.095627   14.265316
min    142.699997  144.339996  141.320007  143.000000  142.205139
```

```
25%     166.677505  168.660000  165.654995  167.494999  166.818306
50%     177.355003  179.404999  176.504997  177.805000  177.421356
75%     188.369995  189.799995  187.472496  188.625004  188.499462
max     198.020004  199.619995  197.000000  198.110001  198.110001

             Volume
count  2.500000e+02
mean   5.807680e+07
std    1.728403e+07
min    2.404830e+07
25%    4.746312e+07
50%    5.405425e+07
75%    6.448900e+07
max    1.543573e+08

<google.colab._quickchart_helpers.SectionTitle at 0x79ee7b94e980>

from matplotlib import pyplot as plt
_df_41['Open'].plot(kind='hist', bins=20, title='Open')
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_42['High'].plot(kind='hist', bins=20, title='High')
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_43['Low'].plot(kind='hist', bins=20, title='Low')
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_44['Close'].plot(kind='hist', bins=20, title='Close')
plt.gca().spines[['top', 'right',]].set_visible(False)

<google.colab._quickchart_helpers.SectionTitle at 0x79ee79989f30>

from matplotlib import pyplot as plt
import seaborn as sns
_df_45.groupby('index').size().plot(kind='barh',
color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)

<google.colab._quickchart_helpers.SectionTitle at 0x79ee797d3b80>

from matplotlib import pyplot as plt
_df_46.plot(kind='scatter', x='Open', y='High', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_47.plot(kind='scatter', x='High', y='Low', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_48.plot(kind='scatter', x='Low', y='Close', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_49.plot(kind='scatter', x='Close', y='Adj Close', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

<google.colab._quickchart_helpers.SectionTitle at 0x79ee79830490>

from matplotlib import pyplot as plt
_df_50['Open'].plot(kind='line', figsize=(8, 4), title='Open')
plt.gca().spines[['top', 'right']].set_visible(False)

from matplotlib import pyplot as plt
_df_51['High'].plot(kind='line', figsize=(8, 4), title='High')
plt.gca().spines[['top', 'right']].set_visible(False)

from matplotlib import pyplot as plt
_df_52['Low'].plot(kind='line', figsize=(8, 4), title='Low')
plt.gca().spines[['top', 'right']].set_visible(False)

from matplotlib import pyplot as plt
_df_53['Close'].plot(kind='line', figsize=(8, 4), title='Close')
plt.gca().spines[['top', 'right']].set_visible(False)

<google.colab._quickchart_helpers.SectionTitle at 0x79ee79830160>

<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.


from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(_df_54['index'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(_df_54, x='Open', y='index', inner='stick',
palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)

<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.


from matplotlib import pyplot as plt
import seaborn as sns
```

```
figsize = (12, 1.2 * len(_df_55['index'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(_df_55, x='High', y='index', inner='stick',
palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.
```

```
from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(_df_56['index'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(_df_56, x='Low', y='index', inner='stick',
palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.
```

```
from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(_df_57['index'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(_df_57, x='Close', y='index', inner='stick',
palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

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 Information for AAPL
general_info_aapl = AAPL.info()
general_info_aapl
```

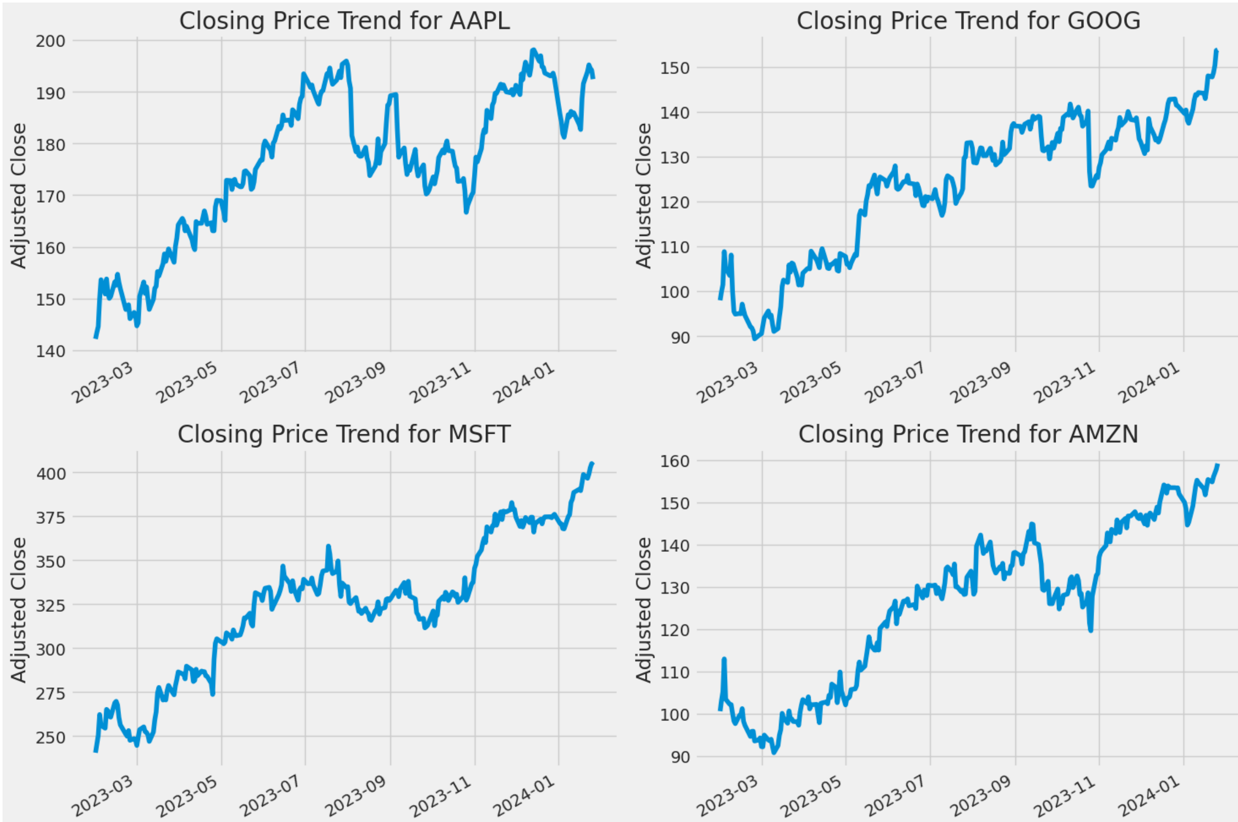
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 250 entries, 2023-01-30 to 2024-01-26
Data columns (total 7 columns):
```

```
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          250 non-null    float64
 1   High          250 non-null    float64
 2   Low           250 non-null    float64
 3   Close         250 non-null    float64
 4   Adj Close     250 non-null    float64
 5   Volume        250 non-null    int64
 6   company_name  250 non-null    object
dtypes: float64(5), int64(1), object(1)
memory usage: 15.6+ 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.

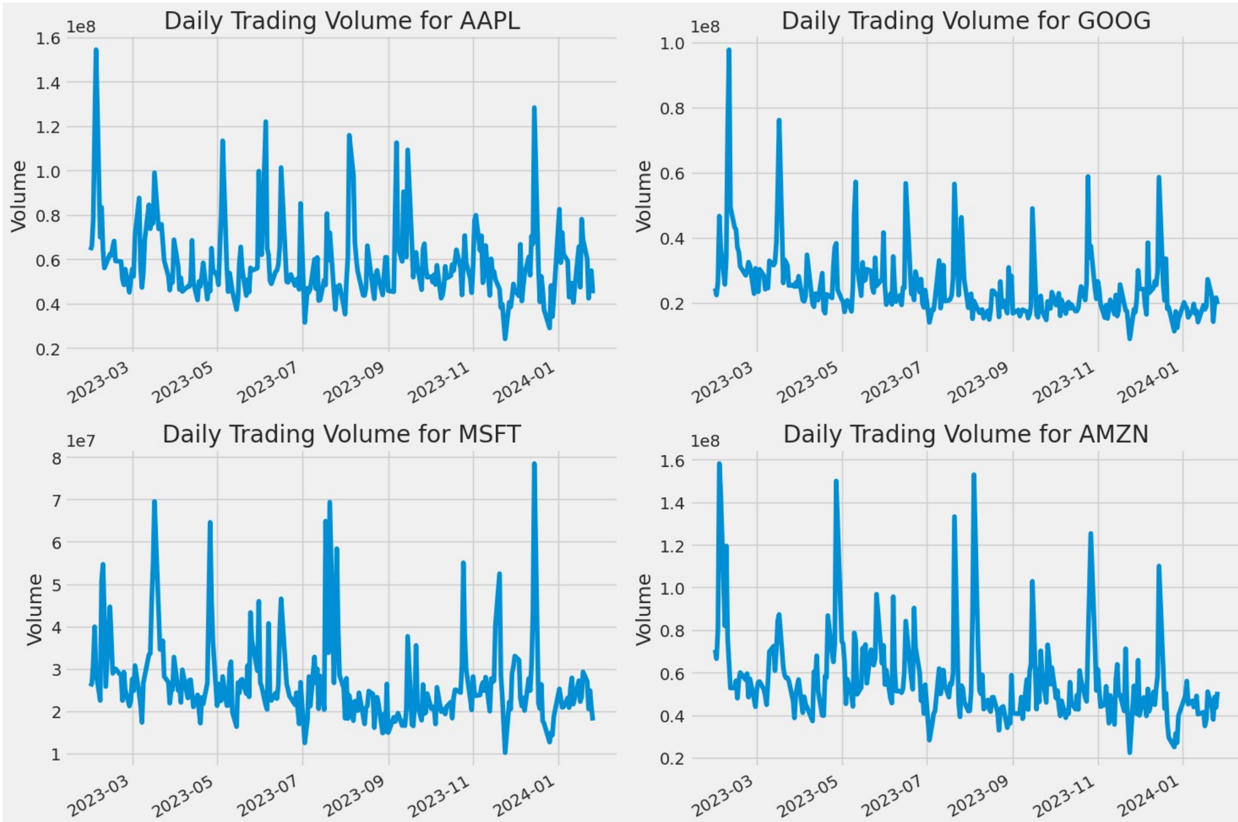
```
# Historical View of Closing Prices
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

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

plt.tight_layout()
```



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.

```
# Plotting Daily Total Stock Trading Volume
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

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

plt.tight_layout()
```



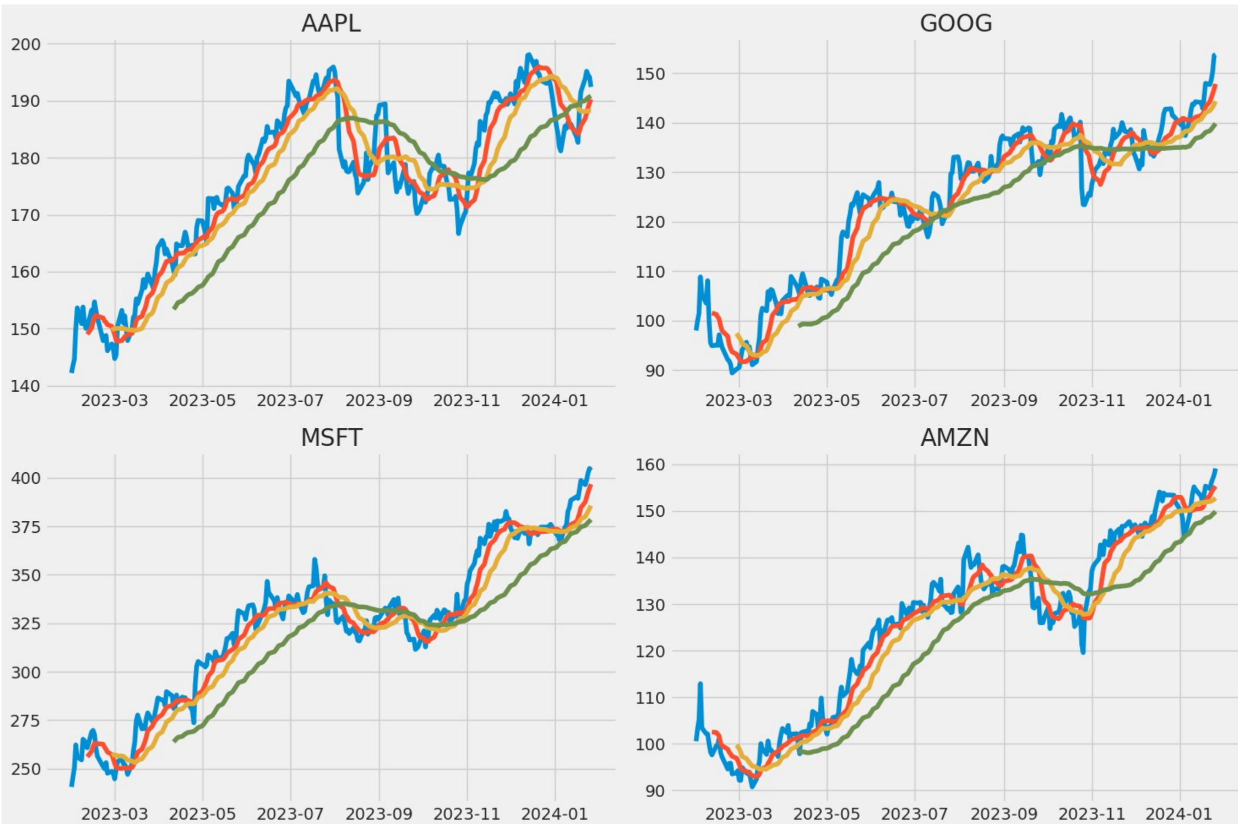
Now that we've seen the visualizations for the closing price and the volume traded each day, let's go ahead and caculate the moving average for the stock.

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

```
# Define Moving Averages for 10, 20, and 50 days
ma_day_list = [10, 20, 50]

# Calculate and add Moving Averages to each company's data
for ma_period in ma_day_list:
    for company_df in company_list:
        ma_column_name = f"MA for {ma_period} days"
        company_df[ma_column_name] = company_df['Adj
Close'].rolling(ma_period).mean()

# Plot the Moving Averages for each company
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))
```

```
for index, (company_df, tech_symbol) in enumerate(zip(company_list, tech_list)):
    axes[index // 2, index % 2].plot(
        company_df[['Adj Close', f'MA for 10 days', f'MA for 20 days', f'MA for 50 days']]
    )
    axes[index // 2, index % 2].set_title(tech_symbol)

fig.tight_layout()
```

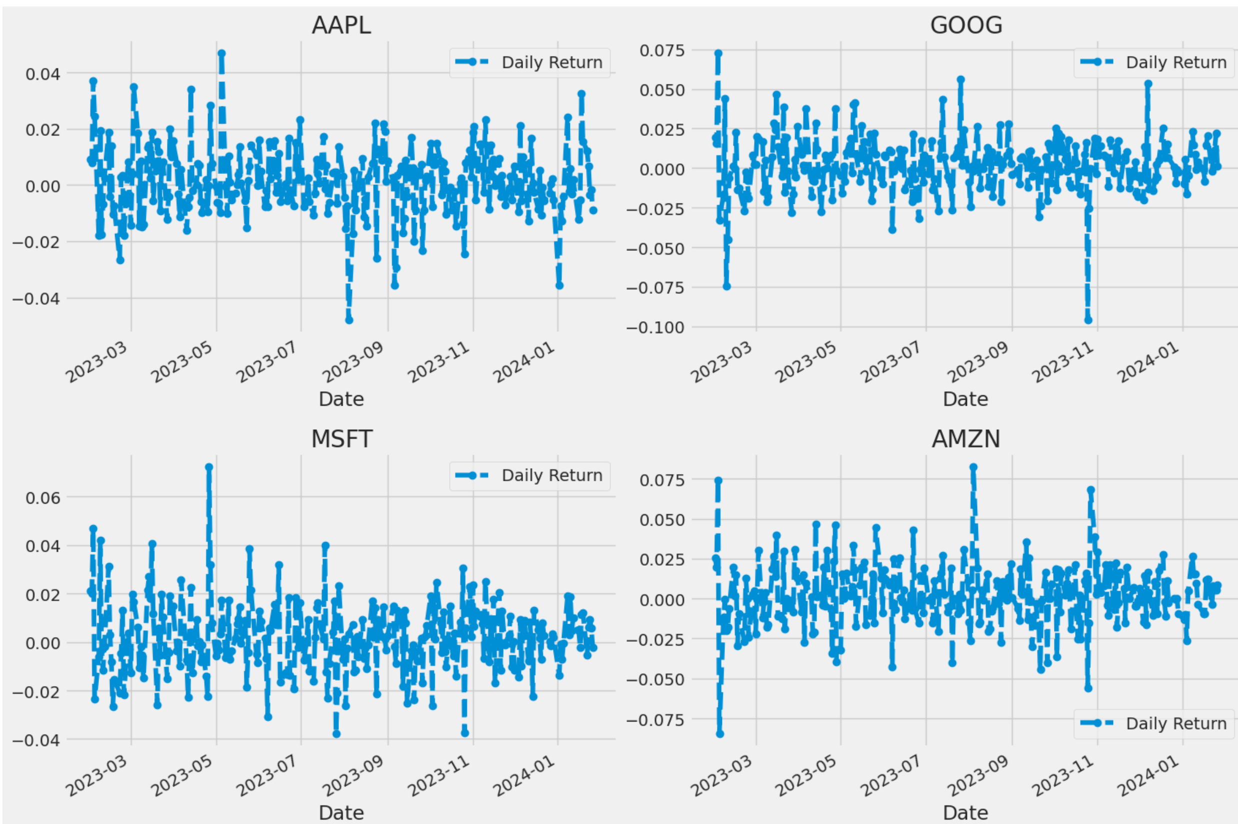


We see in the graph that the best values to measure the moving average are 10 and 20 days because we still capture trends in the data without noise.

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.

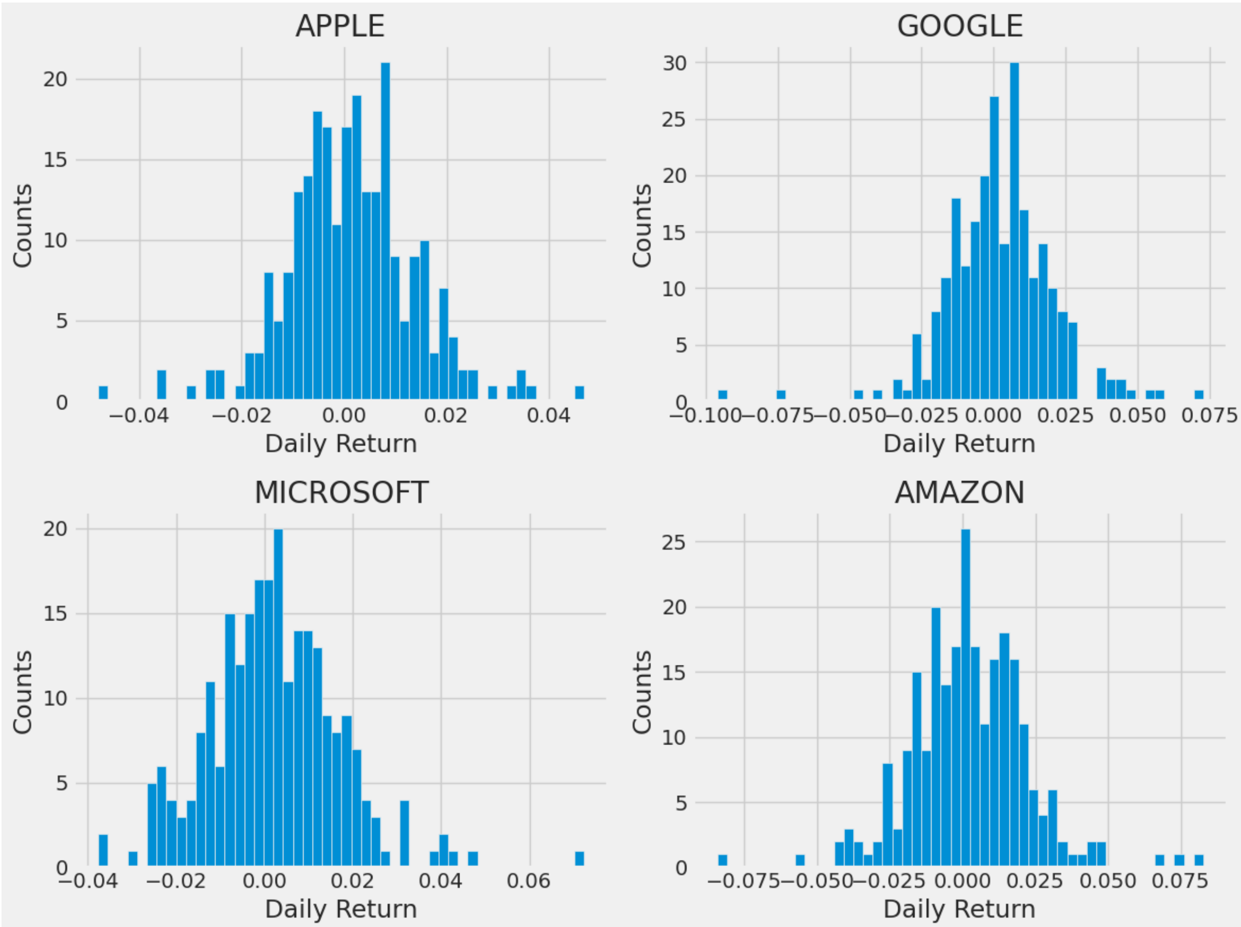
```python
# Calculate daily percentage change using pct_change
for company_df in company_list:
    company_df['Daily Return'] = company_df['Adj Close'].pct_change()

# Plot the daily return percentages for each company
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))
for index, (company_df, tech_symbol) in enumerate(zip(company_list,
tech_list)):
    company_df['Daily Return'].plot(ax=axes[index // 2, index % 2],
legend=True, linestyle='--', marker='o')
    axes[index // 2, index % 2].set_title(tech_symbol)

fig.tight_layout()
```



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.

```python
# Plotting Histograms for Daily Returns
plt.figure(figsize=(12, 9))

for index, company_df in enumerate(company_list, 1):
    plt.subplot(2, 2, index)
    company_df['Daily Return'].hist(bins=50)
    plt.xlabel('Daily Return')
```

```
    plt.ylabel('Counts')
    plt.title(f'{company_name[index - 1]}')

plt.tight_layout()
```



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.

```
# Retrieve the closing prices for the tech stock list and store in a
DataFrame
```

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

# Create a new DataFrame for tech stock returns
tech_returns_df = closing_prices_df.pct_change()
tech_returns_df.head()
```

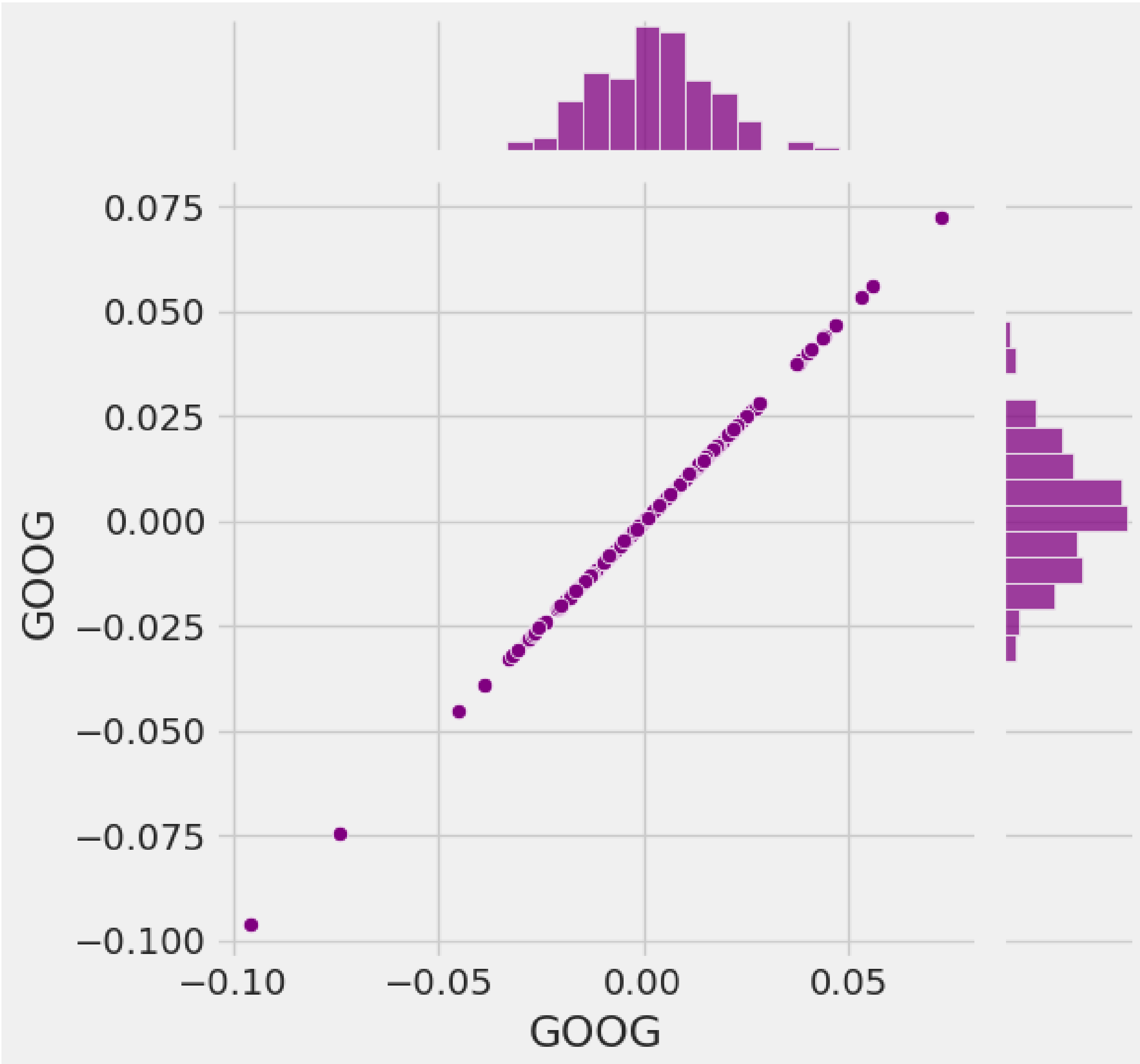
```
[*********************100%%**********************]  4 of 4 completed
```

```
Ticker           AAPL      AMZN      GOOG      MSFT
Date
2023-01-30        NaN       NaN       NaN       NaN
2023-01-31  0.009021  0.025659  0.019602  0.021013
2023-02-01  0.007901  0.019587  0.015620  0.019935
2023-02-02  0.037063  0.073799  0.072661  0.046884
2023-02-03  0.024400 -0.084315 -0.032904 -0.023621
```

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

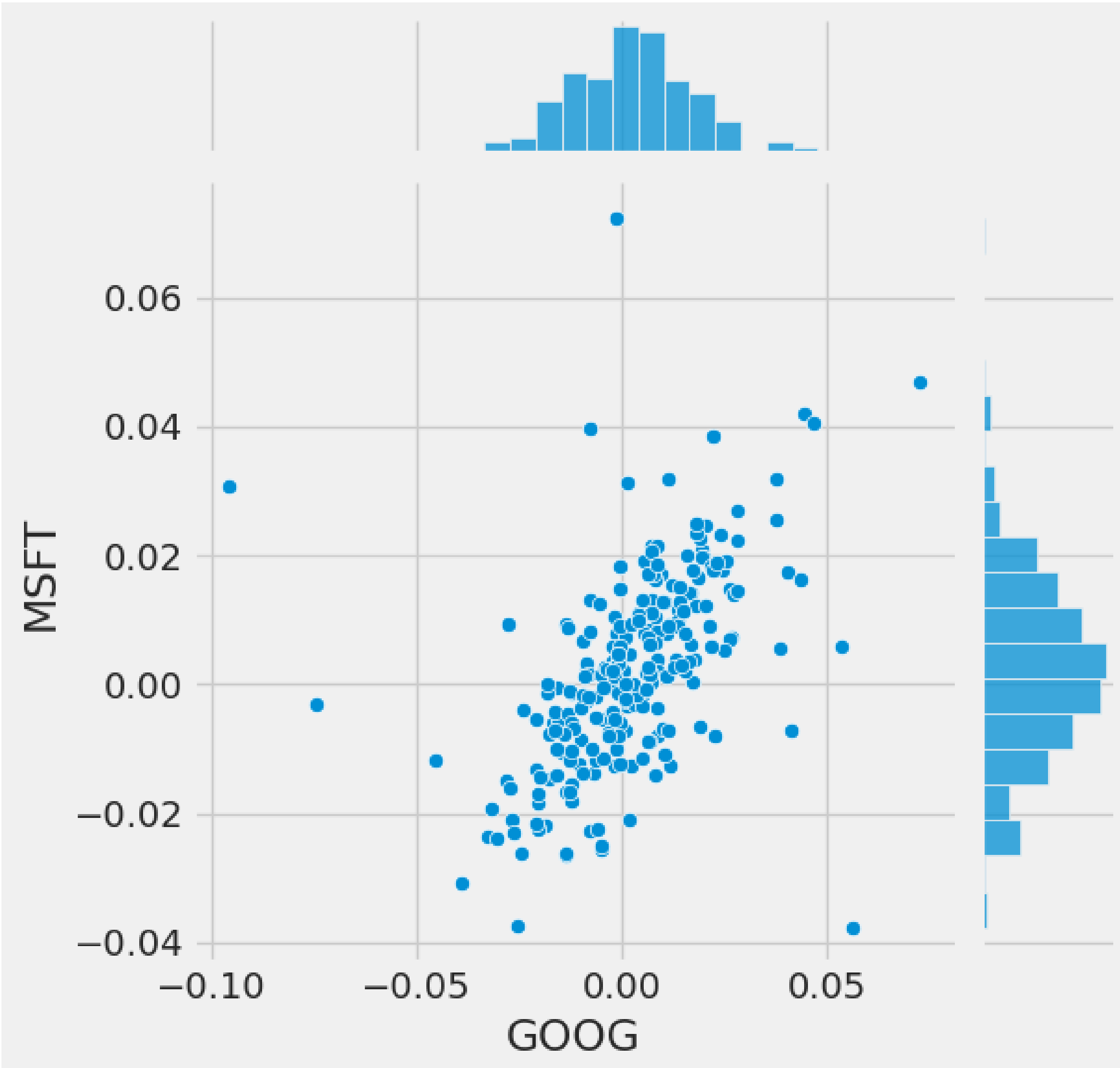
```
# Comparing Google to itself should reveal a perfectly linear
relationship
sns.jointplot(x='GOOG', y='GOOG', data=tech_returns_df,
kind='scatter', color='purple')
```

```
<seaborn.axisgrid.JointGrid at 0x79eee2709a80>
```




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

```
<seaborn.axisgrid.JointGrid at 0x79eee26dfee0>
```

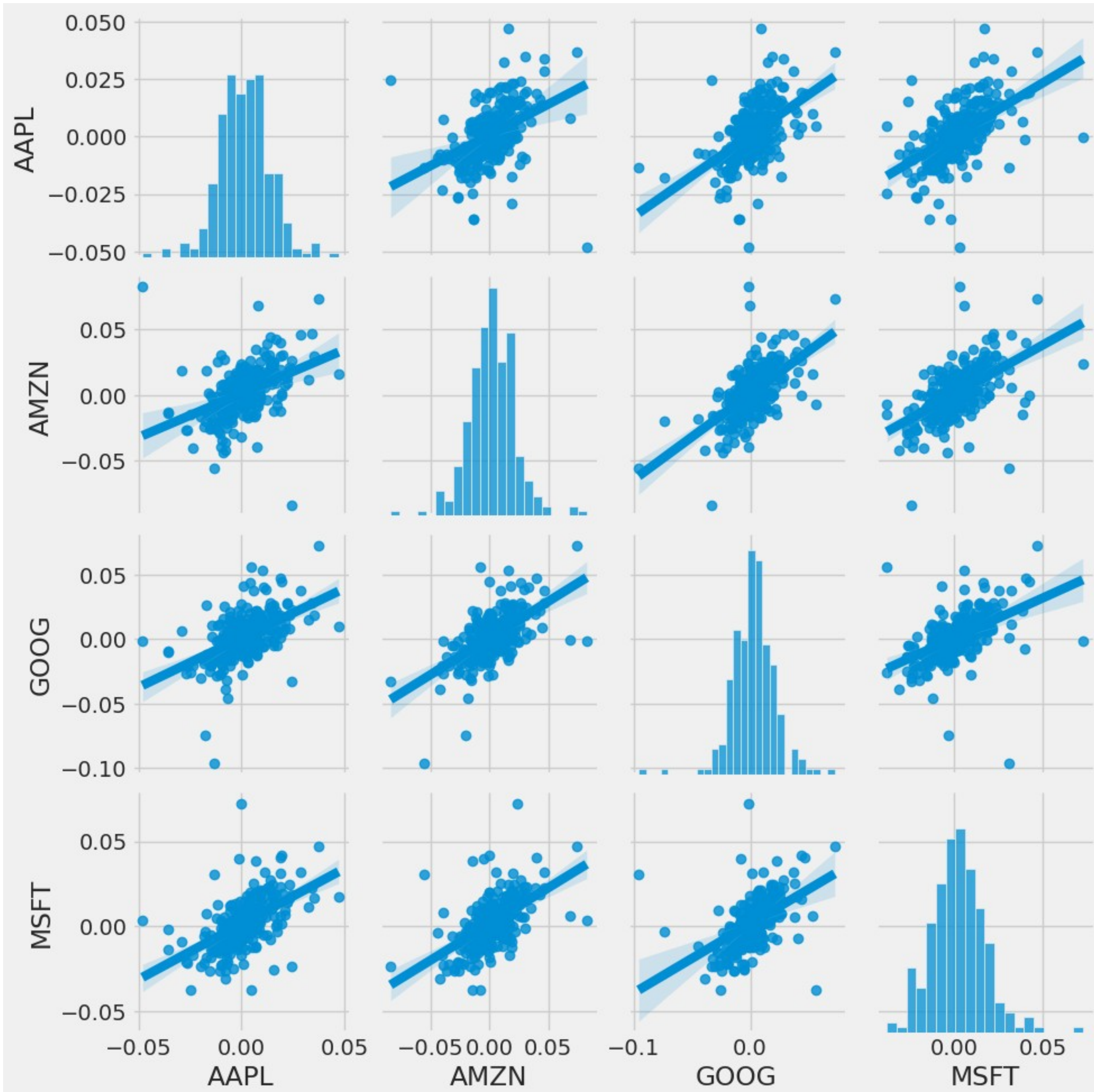


So now we can see that if two stocks are perfectly (and positivley) correlated with each other a linear relationship bewteen its daily return values should occur.

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

```
# Conveniently use pairplot on the DataFrame for an automatic visual
analysis of all the comparisons with regression lines
sns.pairplot(tech_returns_df, kind='reg')
```

```
<seaborn.axisgrid.PairGrid at 0x79eee576efe0>
```



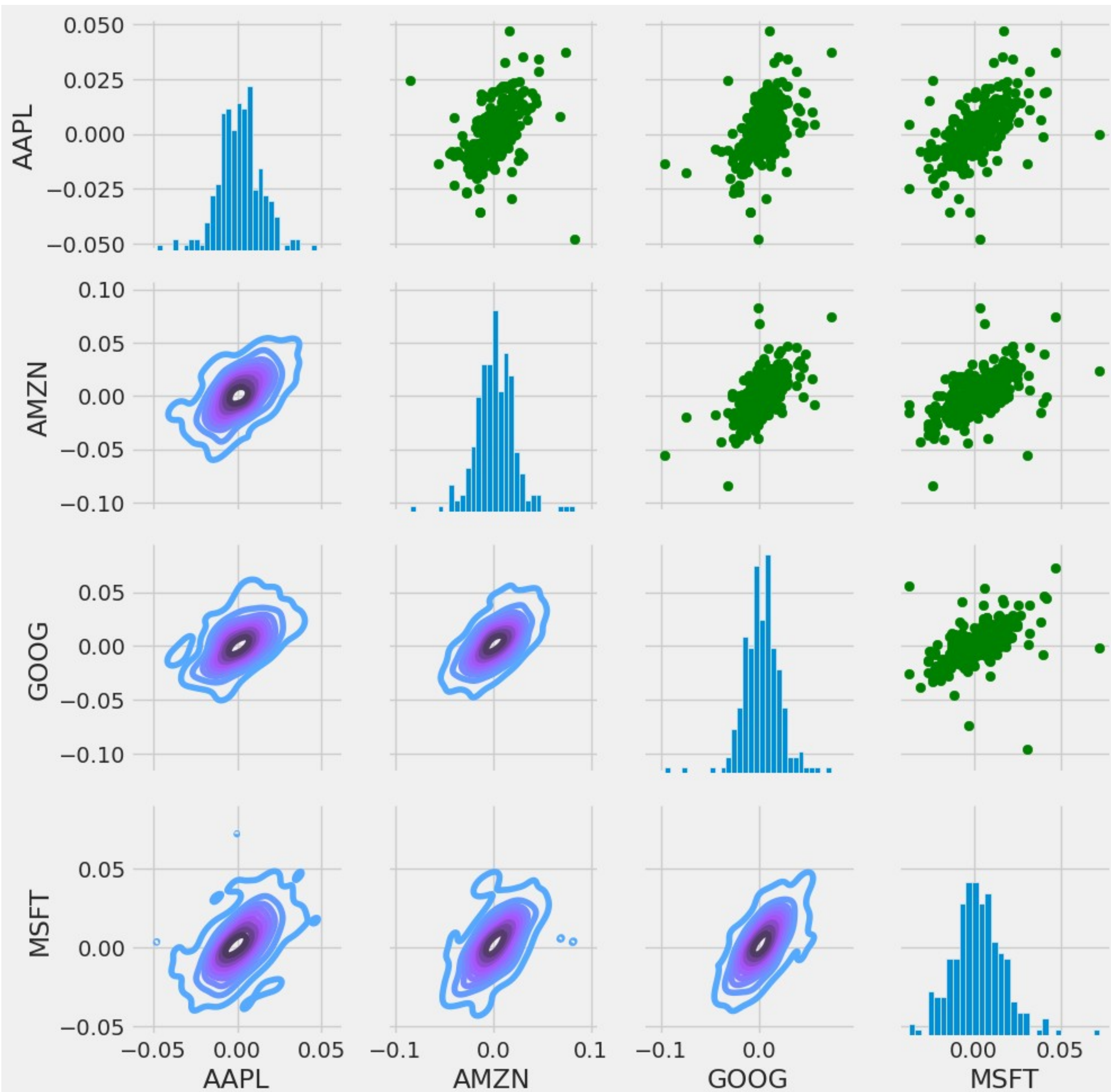
Above we can see all the relationships on daily returns between all the stocks. A quick glance shows an interesting correlation between Google and Amazon daily returns. It might be interesting to investigate that individual comaprison.

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 a figure named 'returns_fig' and use PairGrid on the
DataFrame
returns_fig = sns.PairGrid(tech_returns_df.dropna())
```

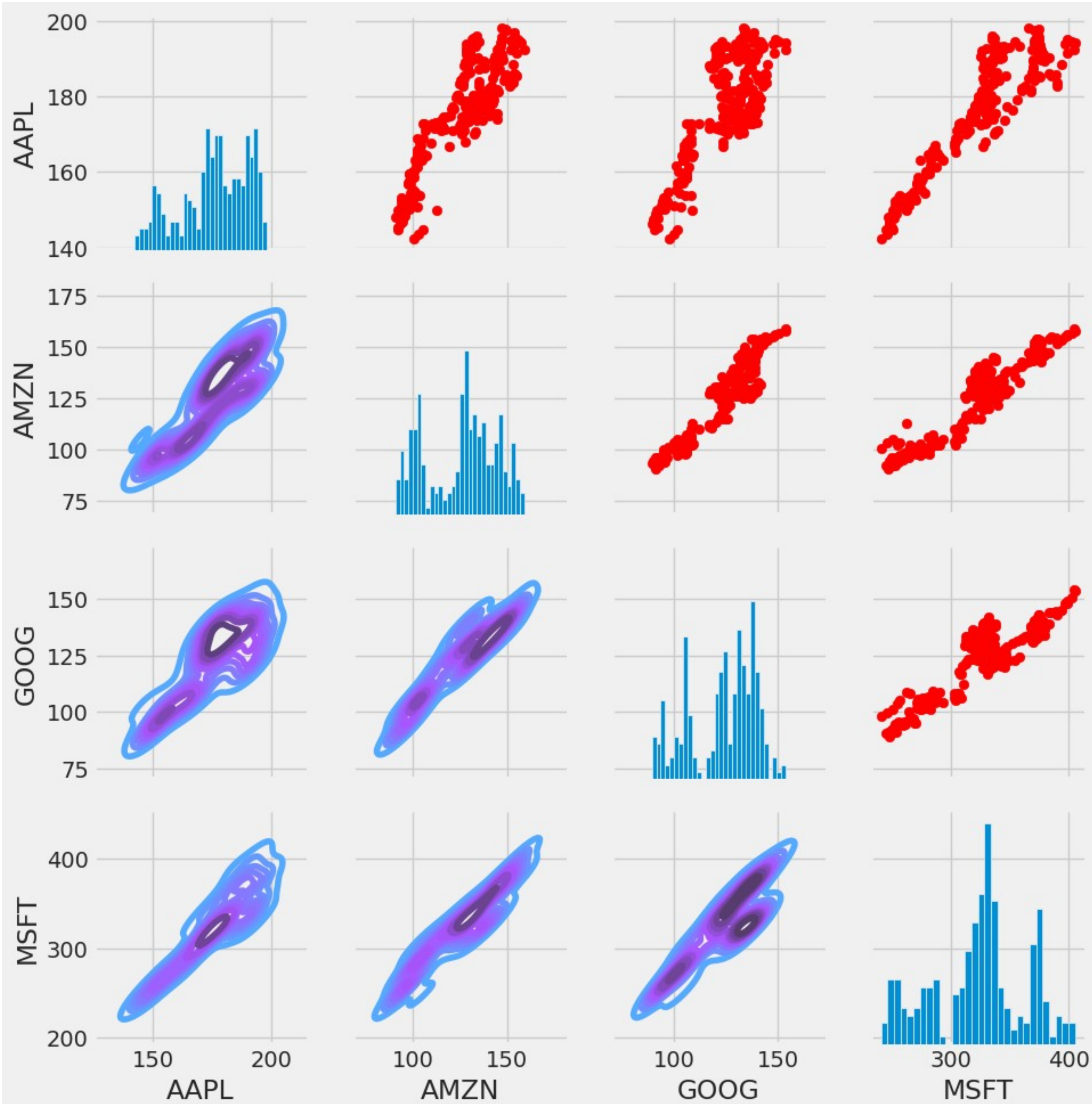
```
# Customize the upper triangle with scatter plots in purple
returns_fig.map_upper(plt.scatter, color='green')

# Define the lower triangle with kernel density plots using the
'cool_d' colormap
returns_fig.map_lower(sns.kdeplot, cmap='cool_d')

# Define the diagonal as a series of histogram plots for daily returns
with 30 bins
returns_fig.map_diag(plt.hist, bins=30)
```

```
<seaborn.axisgrid.PairGrid at 0x79eee5367df0>
```



```
# Set up a figure named 'returns_fig' and use PairGrid on the
DataFrame of closing prices
returns_fig = sns.PairGrid(closing_df)

# Customize the upper triangle with scatter plots in purple
returns_fig.map_upper(plt.scatter, color='red')

# Define the lower triangle with kernel density plots using the
'cool_d' colormap
returns_fig.map_lower(sns.kdeplot, cmap='cool_d')

# Define the diagonal as a series of histogram plots for daily returns
with 30 bins
returns_fig.map_diag(plt.hist, bins=30)
```

```
<seaborn.axisgrid.PairGrid at 0x79eed051f520>
```

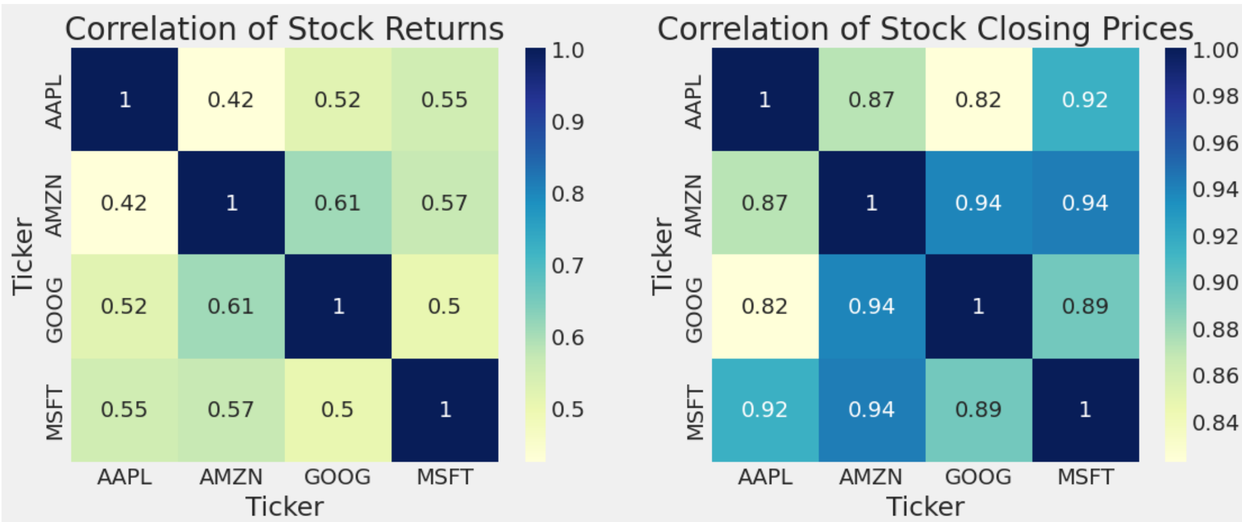


Finally, we could also do a correlation plot, to get actual numerical values for the correlation between the stocks' daily return values. By comparing the closing prices, we see an interesting relationship between Microsoft and Apple.

```
# Set up a figure with a size of 12x10
plt.figure(figsize=(12, 10))

# Create the first subplot and plot the correlation heatmap for stock
returns with a summer color map
plt.subplot(2, 2, 1)
sns.heatmap(tech_returns_df.corr(), annot=True, cmap='YlGnBu')
plt.title('Correlation of Stock Returns')
```

```
# Create the second subplot and plot the correlation heatmap for stock
closing prices with a summer color map
plt.subplot(2, 2, 2)
sns.heatmap(closing_df.corr(), annot=True, cmap='YlGnBu')
plt.title('Correlation of Stock Closing Prices')
```

```
Text(0.5, 1.0, 'Correlation of Stock Closing Prices')
```



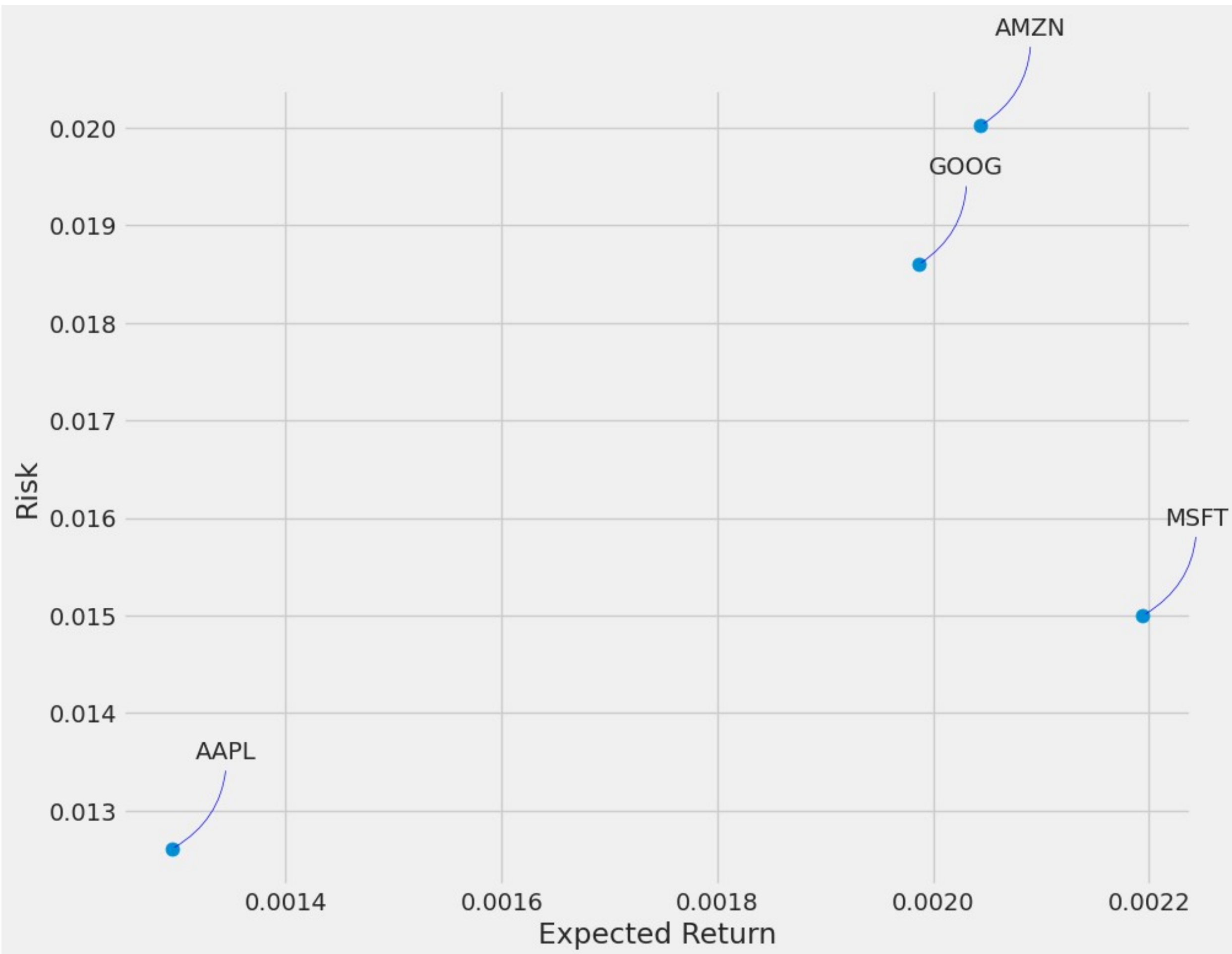
Just like we suspected in our `PairPlot` we see here numerically and visually that Microsoft and Amazon had the strongest correlation of daily stock return. It's also interesting to see that all the technology comapnies are positively correlated.

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.

```
# Remove NaN values from tech stock returns
returns = tech_returns_df.dropna()

# Define the area for scatter points
marker_area = np.pi * 20

# Set up a figure with a size of 10x8
plt.figure(figsize=(10, 8))

# Scatter plot of mean vs. standard deviation for expected return and
risk
```

```
plt.scatter(returns.mean(), returns.std(), s=marker_area)
plt.xlabel('Expected Return')
plt.ylabel('Risk')

# Annotate each point with the corresponding stock label
for label, mean_return, risk in zip(returns.columns, returns.mean(),
returns.std()):
    plt.annotate(label, xy=(mean_return, risk), 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 stock price of APPLE inc:

```
# Obtain the stock quote data for AAPL from Yahoo Finance starting
from January 1, 2012, to the current date
```

```python
stock_data = pdr.get_data_yahoo('AAPL', start='2012-01-01',
end=datetime.now())

# Display the acquired stock data
stock_data
```

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

                  Open        High         Low       Close   Adj Close
\
Date

2012-01-03   14.621429   14.732143   14.607143   14.686786   12.449695

2012-01-04   14.642857   14.810000   14.617143   14.765714   12.516594

2012-01-05   14.819643   14.948214   14.738214   14.929643   12.655555

2012-01-06   14.991786   15.098214   14.972143   15.085714   12.787854

2012-01-09   15.196429   15.276786   15.048214   15.061786   12.767572

...                ...         ...         ...         ...         ...

2024-01-22  192.300003  195.330002  192.259995  193.889999  193.889999

2024-01-23  195.020004  195.750000  193.830002  195.179993  195.179993

2024-01-24  195.419998  196.380005  194.339996  194.500000  194.500000

2024-01-25  195.220001  196.270004  193.110001  194.169998  194.169998

2024-01-26  194.270004  194.759995  191.940002  192.419998  192.419998


               Volume
Date
2012-01-03  302220800
2012-01-04  260022000
2012-01-05  271269600
2012-01-06  318292800
2012-01-09  394024400
...               ...
2024-01-22   60133900
2024-01-23   42355600
2024-01-24   53631300
2024-01-25   54822100
2024-01-26   44553400

[3036 rows x 6 columns]
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x79eececbf790>

from matplotlib import pyplot as plt
_df_0['Open'].plot(kind='hist', bins=20, title='Open')
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_1['High'].plot(kind='hist', bins=20, title='High')
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_2['Low'].plot(kind='hist', bins=20, title='Low')
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_3['Close'].plot(kind='hist', bins=20, title='Close')
plt.gca().spines[['top', 'right',]].set_visible(False)

<google.colab._quickchart_helpers.SectionTitle at 0x79eee5644c10>

from matplotlib import pyplot as plt
_df_4.plot(kind='scatter', x='Open', y='High', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_5.plot(kind='scatter', x='High', y='Low', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_6.plot(kind='scatter', x='Low', y='Close', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_7.plot(kind='scatter', x='Close', y='Adj Close', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

<google.colab._quickchart_helpers.SectionTitle at 0x79eeceb875e0>

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['Open']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_8.sort_values('Date', ascending=True)
```

```
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Open')

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['High']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_9.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('High')

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['Low']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_10.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Low')

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
```

```
  ys = series['Close']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_11.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Close')
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x79eeceb877f0>
```

```
from matplotlib import pyplot as plt
_df_12['Open'].plot(kind='line', figsize=(8, 4), title='Open')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_13['High'].plot(kind='line', figsize=(8, 4), title='High')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_14['Low'].plot(kind='line', figsize=(8, 4), title='Low')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_15['Close'].plot(kind='line', figsize=(8, 4), title='Close')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
# Set up a figure with a size of 16x6
plt.figure(figsize=(16, 6))

# Plot the Close Price History
plt.title('Close Price History')
plt.plot(stock_data['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.show()
```


```python
# Create a new dataframe containing only the 'Close' column
filtered_data = stock_data.filter(['Close'])

# Convert the dataframe to a numpy array
data_array = filtered_data.values

# Determine the number of rows for training the model
training_data_length = int(np.ceil(len(data_array) * 0.95))

training_data_length
```

```
2885
```

```python
data = stock_data.filter(['Close'])
dataset = data.values

# Scale the data using Min-Max Scaling
from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler object with a feature range of (0, 1)
data_scaler = MinMaxScaler(feature_range=(0, 1))

# Transform the dataset using the scaler
scaled_data = data_scaler.fit_transform(dataset)

scaled_data
```

```
array([[0.00401431],
       [0.00444289],
       [0.00533302],
       ...,
       [0.98039774],
       [0.97860584],
       [0.96910336]])
```

```python
# Create the training dataset
# Obtain the scaled training dataset
training_data = scaled_data[0:int(training_data_length), :]

# Split the data into x_train and y_train datasets
x_train = []
y_train = []

# Iterate to create sequences for training
for i in range(60, len(training_data)):
    x_train.append(training_data[i-60:i, 0])
    y_train.append(training_data[i, 0])
    if i <= 61:
        print(x_train)
        print(y_train)
        print()

# Convert 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))
# Print the shape of x_train
# 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, 0.00533302, 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]
```

```python
# Import necessary modules from Keras
from keras.models import Sequential
from keras.layers import Dense, LSTM

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

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

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

```
2825/2825 [==============================] - 133s 46ms/step - loss:
0.0011

<keras.src.callbacks.History at 0x79ee7cbc3d00>
```

```python
# Create the testing dataset
# Obtain a new array containing scaled values from index 1543 to 2002
testing_data = scaled_data[training_data_length - 60:, :]

# Create the datasets x_test and y_test
x_test = []
y_test = dataset[training_data_length:, :]

# Iterate to create sequences for testing
for i in range(60, len(testing_data)):
    x_test.append(testing_data[i-60:i, 0])
```

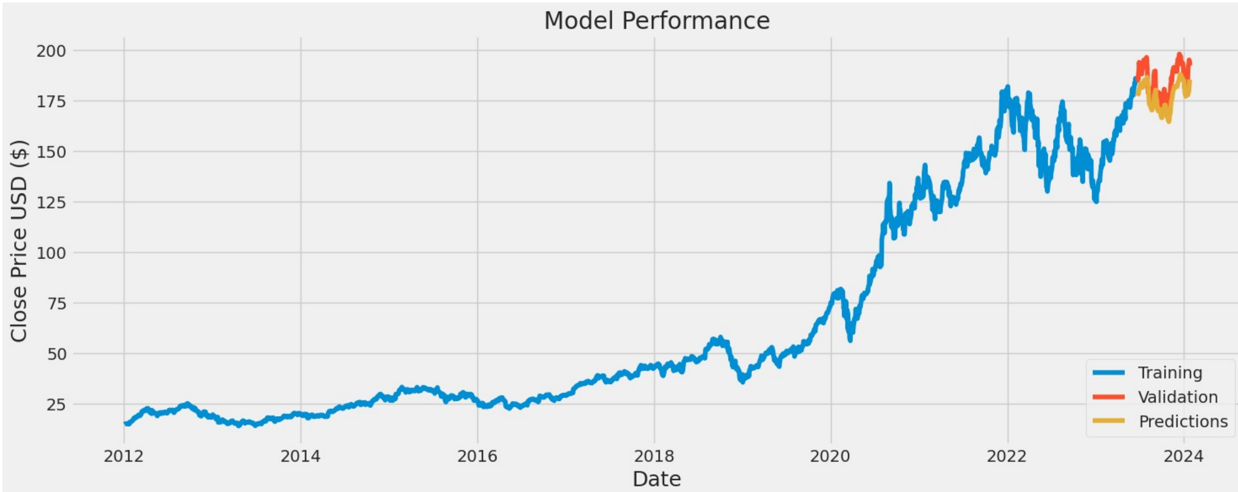
```python
# Convert the data to a numpy array
x_test = np.array(x_test)

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

# Obtain the model's predicted price values
predictions = lstm_model.predict(x_test)
predictions = data_scaler.inverse_transform(predictions)

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

```
5/5 [==============================] - 3s 79ms/step
```

```
8.098923644539846
```

```python
# Plotting the data
training_set = data[:training_data_length]
validation_set = data[training_data_length:]
validation_set['Predictions'] = predictions

# Visualizing the data
plt.figure(figsize=(16, 6))
plt.title('Model Performance')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(training_set['Close'])
plt.plot(validation_set[['Close', 'Predictions']])
plt.legend(['Training', 'Validation', 'Predictions'], loc='lower
right')
plt.show()
```

```
<ipython-input-43-3242ffc95c9e>: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
  validation_set['Predictions'] = predictions
```




```
# Display the actual and predicted closing prices for the validation set
validation_set

                 Close  Predictions
Date
2023-06-22  187.000000   177.357178
2023-06-23  186.679993   177.797791
2023-06-26  185.270004   178.235077
2023-06-27  188.059998   178.292526
2023-06-28  189.250000   178.791321
...                ...          ...
2024-01-22  193.889999   179.873993
2024-01-23  195.179993   181.771194
2024-01-24  194.500000   183.619888
2024-01-25  194.169998   184.838806
2024-01-26  192.419998   185.447876

[151 rows x 2 columns]

<google.colab._quickchart_helpers.SectionTitle at 0x79ee793ae5c0>

from matplotlib import pyplot as plt
_df_16['Close'].plot(kind='hist', bins=20, title='Close')
plt.gca().spines[['top', 'right',]].set_visible(False)

from matplotlib import pyplot as plt
_df_17['Predictions'].plot(kind='hist', bins=20, title='Predictions')
plt.gca().spines[['top', 'right',]].set_visible(False)

<google.colab._quickchart_helpers.SectionTitle at 0x79ee793af2b0>

from matplotlib import pyplot as plt
_df_18.plot(kind='scatter', x='Close', y='Predictions', s=32,
```

```
alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x79ee7bd77b20>
```

```
from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['Close']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_19.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Close')
```

```
from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['Predictions']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_20.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Predictions')
```

```
from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  counted = (series['Date']
```

```
                .value_counts()
              .reset_index(name='counts')
              .rename({'index': 'Date'}, axis=1)
              .sort_values('Date', ascending=True))
  xs = counted['Date']
  ys = counted['counts']
  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_21.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('count()')

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Date']
  ys = series['Close']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_22.sort_values('Date', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Date')
_ = plt.ylabel('Close')

<google.colab._quickchart_helpers.SectionTitle at 0x79ee7bd76ef0>

from matplotlib import pyplot as plt
_df_23['Close'].plot(kind='line', figsize=(8, 4), title='Close')
plt.gca().spines[['top', 'right']].set_visible(False)

from matplotlib import pyplot as plt
_df_24['Predictions'].plot(kind='line', figsize=(8, 4),
title='Predictions')
plt.gca().spines[['top', 'right']].set_visible(False)
```