SP500StockPriceAnalysis

May 13, 2023

1 A Deep Dive into the S&P 500: Predicting Stock Prices

Kanishk Chinnapapannagari, Aarav Naveen, Avyay Potarlanka, and Melvin Rajendran

1.1 Introduction

In today's evolving financial landscape, both investors and traders are constantly seeking an edge to make informed decisions. The stock market, which contains an intricate web of variables and is influenced by numerous factors, has proven to be a difficult environment to navigate.

In the past, investment-related decisions were often made based on analysis of historical trends. However, the advancement of data science and machine learning techniques has introduced a new opportunity to potentially predict future stock prices with reasonable accuracy and thus gain valuable insights.

This data science project delves into prediction of stock prices within the Standard & Poor's 500 index, otherwise known as the S&P 500. This index contains 500 of the top companies in the United States, and it represents approximately 80% of the U.S. stock market's total value. Hence, it serves as a strong indicator of the movement within the market. To learn more about the S&P 500 and other popular indices in the U.S., read this article: https://www.investopedia.com/insights/introduction-to-stock-market-indices/.

Throughout this project, we will follow a comprehensive data science approach that includes the following steps: * Data collection * Data processing * Exploratory data analysis and data visualization * Data analysis, hypothesis testing, and machine learning (ML) * Insight formation

Our project aims to leverage predictive modeling techniques to provide insights to investors. The analysis herein will identify stocks that are undervalued and thus will increase in price in the near future, meaning investors should consider buying or holding shares. Likewise, it will also identify stocks that are overvalued and will soon decrease in price, indicating that investors should consider selling their position.

[1]:
```python
# Import necessary libraries
from bs4 import BeautifulSoup
from keras.layers import Dense, LSTM
from keras.models import Sequential
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
```

```python
import requests
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

```
2023-05-13 02:33:12.096548: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-05-13 02:33:12.181286: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-05-13 02:33:12.183025: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2023-05-13 02:33:14.622238: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

1.2 Data Collection

1.2.1 Reading in a Kaggle Dataset

To gather information about the S&P 500 companies, we will be using the following dataset: https://www.kaggle.com/datasets/paultimothymooney/stock-market-data. This Kaggle dataset contains the date, volume, and prices for the NASDAQ, NYSE, and S&P 500. For the purposes of this project, we will only analyze the stock prices of companies in the S&P 500.

[2]:
```python
# Initialize an empty data frame to store the stock price data
price_data = pd.DataFrame()

# Initialize the path to the folder containing the data
folder_path = 'sp500-data'

# Iterate across each file in the folder by name
for file_name in os.listdir(folder_path):

    # Check if the current file is a CSV file
    if file_name.endswith('.csv'):

        # Read the current file into a temporary data frame
        temp = pd.read_csv(os.path.join(folder_path, file_name))

        # Extract the symbol from the current file's name
        symbol = file_name[0:-4]

        # Store the symbol in a new column in the temporary data frame
        temp['Symbol'] = symbol
```

```
        # Concatenate the accumulating and temporary data frames
        price_data = pd.concat([price_data, temp], ignore_index = True)

# Print the last five rows of the price data frame
price_data.tail()
```

[2]:

```
                  Date         Low        Open     Volume        High  \
3265995  06-12-2022  152.089996  154.220001  1964800.0  155.500000
3265996  07-12-2022  149.380005  152.960007  2444100.0  153.789993
3265997  08-12-2022  149.199997  150.529999  2267500.0  154.350006
3265998  09-12-2022  152.740005  153.940002  3274900.0  156.330002
3265999  12-12-2022  152.970001  154.070007   301135.0  154.470001

              Close  Adjusted Close Symbol
3265995  153.050003      153.050003    ZTS
3265996  150.250000      150.250000    ZTS
3265997  153.679993      153.679993    ZTS
3265998  153.389999      153.389999    ZTS
3265999  153.625000      153.625000    ZTS
```

1.2.2 Webscraping From Wikipedia

We noticed that the Kaggle dataset does not contain sector data. For this reason, we will supplement our existing data with that which is contained on the following webpage: https://en.wikipedia.org/wiki/List_of_S%26P_500_companies. By scraping this webpage's list of the S&P 500 companies, we can match each company in our existing data to its corresponding GICS sector and sub-industry. This will enable us to perform analysis by sector and/or sub-industry and thus eliminate biases in our modeling.

[3]:

```
# Headers for the HTTP request
headers = {
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/
 ↪537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36',
    'From': 'pleaseletmein@gmail.com'
}

# Make an HTTP request to the Wikipedia URL and store the response
response = requests.get('https://en.wikipedia.org/wiki/
 ↪List_of_S%26P_500_companies', headers = headers)

# Parse the text from the webpage as HTML
soup = BeautifulSoup(response.text, 'html.parser')

# Find the table element containing the data and both extract and store the data
table = soup.find('table')
```

```
# Read the HTML table into a data frame
sector_data = pd.read_html(str(table), flavor = 'html5lib')[0]

# Print the last five rows of the sector data frame
sector_data.tail()
```

[3]:
```
    Symbol              Security             GICS Sector  \
498    YUM           Yum! Brands  Consumer Discretionary
499   ZBRA    Zebra Technologies  Information Technology
500    ZBH         Zimmer Biomet             Health Care
501   ZION  Zions Bancorporation              Financials
502    ZTS                Zoetis             Health Care

                      GICS Sub-Industry   Headquarters Location  Date added  \
498                         Restaurants    Louisville, Kentucky  1997-10-06
499  Electronic Equipment & Instruments  Lincolnshire, Illinois  2019-12-23
500                Health Care Equipment         Warsaw, Indiana  2001-08-07
501                       Regional Banks    Salt Lake City, Utah  2001-06-22
502                      Pharmaceuticals  Parsippany, New Jersey  2013-06-21

         CIK Founded
498  1041061    1997
499   877212    1969
500  1136869    1927
501   109380    1873
502  1555280    1952
```

1.2.3 Webscraping From Slickcharts

We would also like to focus our attention on the top companies of each sector, as these companies drive the movement within their respective sectors. Hence, we will scrape the data from the following webpage: https://www.slickcharts.com/sp500. This webpage contains a list of the S&P 500 companies by weight, where weight is equal to a company's market cap divided by the overall value of the S&P 500. Ultimately, we will select the top companies of each sector by weight.

[4]:
```
# Make an HTTP request to the Slickcharts URL and store the response
response = requests.get('https://www.slickcharts.com/sp500', headers = headers)

# Parse the text from the webpage as HTML
soup = BeautifulSoup(response.text, 'html.parser')

# Find the table element containing the data and both extract and store the data
table = soup.find('table')

# Read the HTML table into a data frame
weight_data = pd.read_html(str(table), flavor = 'html5lib')[0]
```

```
# Print the last five rows of the sector data frame
weight_data.tail()
```

```
[4]:        #                          Company Symbol    Weight  Price   Chg  \
498  499                Newell Brands Inc    NWL  0.010325   9.27 -0.23
499  500           Zions Bancorporation N.A.   ZION  0.009851  22.47 -0.21
500  501              Lincoln National Corp    LNC  0.008901  19.26 -0.68
501  502           News Corporation Class B    NWS  0.005934  18.99  2.16
502  503  DISH Network Corporation Class A   DISH  0.004431   6.15 -0.08

       % Chg
498  (-2.46%)
499  (-0.93%)
500  (-3.42%)
501  (12.80%)
502  (-1.28%)
```

1.3 Data Processing

At this point, we have three data frames containing data that was collected in the previous step. We will merge this data into a single data frame. Then, we will filter our data to include only the top five companies within each sector. As part of this process, we need to clean our data. Data cleaning will involve casting our data to the proper types, removing entries with missing values, and removing unnecessary columns.

1.3.1 Cleaning the Sector Data

```
[5]: # Rename the sector and industry-related columns
sector_data = sector_data.rename(columns = {'GICS Sector': 'Sector', 'GICS␣
 ↪Sub-Industry': 'Industry'})

# Drop unnecessary columns
sector_data = sector_data.drop(['Headquarters Location', 'Date added', 'CIK',␣
 ↪'Founded'], axis = 1)

# Print the last five rows of the data frame
sector_data.tail()
```

```
[5]:    Symbol              Security                  Sector  \
498     YUM           Yum! Brands  Consumer Discretionary
499    ZBRA     Zebra Technologies  Information Technology
500     ZBH         Zimmer Biomet             Health Care
501    ZION   Zions Bancorporation             Financials
502     ZTS                Zoetis             Health Care

                              Industry
498                        Restaurants
```

```
499  Electronic Equipment & Instruments
500                 Health Care Equipment
501                        Regional Banks
502                       Pharmaceuticals
```

1.3.2 Cleaning the Weight Data

[6]:
```
# Drop all columns except Symbol and Weight
weight_data = weight_data.drop(['#', 'Company', 'Price', 'Chg', '% Chg'], axis␣
↪= 1)

# Print the last five rows of the data frame
weight_data.tail()
```

[6]:
```
    Symbol    Weight
498    NWL  0.010325
499   ZION  0.009851
500    LNC  0.008901
501    NWS  0.005934
502   DISH  0.004431
```

1.3.3 Merging the Three Data Frames

[7]:
```
# Perform an inner join (merge) on all three data frames to create a single␣
↪data frame
data = pd.merge(pd.merge(price_data, sector_data, on = 'Symbol'), weight_data,␣
↪on = 'Symbol')

# Reindex the columns of the data frame
data = data.reindex(columns = ['Symbol', 'Security', 'Sector', 'Industry',␣
↪'Weight', 'Date', 'Open', 'High', 'Low', 'Close', 'Adjusted Close',␣
↪'Volume'])

# Cast the Date column's type to datetime
data['Date'] = pd.to_datetime(data['Date'], dayfirst = True)

# Print the last five rows of the resulting data frame
data.tail()
```

[7]:
```
        Symbol Security       Sector         Industry    Weight       Date  \
2890656    ZTS   Zoetis  Health Care  Pharmaceuticals  0.249449 2022-12-06
2890657    ZTS   Zoetis  Health Care  Pharmaceuticals  0.249449 2022-12-07
2890658    ZTS   Zoetis  Health Care  Pharmaceuticals  0.249449 2022-12-08
2890659    ZTS   Zoetis  Health Care  Pharmaceuticals  0.249449 2022-12-09
2890660    ZTS   Zoetis  Health Care  Pharmaceuticals  0.249449 2022-12-12
```

```
               Open        High         Low       Close  Adjusted Close  \
2890656  154.220001  155.500000  152.089996  153.050003      153.050003
2890657  152.960007  153.789993  149.380005  150.250000      150.250000
2890658  150.529999  154.350006  149.199997  153.679993      153.679993
2890659  153.940002  156.330002  152.740005  153.389999      153.389999
2890660  154.070007  154.470001  152.970001  153.625000      153.625000

            Volume
2890656  1964800.0
2890657  2444100.0
2890658  2267500.0
2890659  3274900.0
2890660   301135.0
```

1.3.4 Filtering the Top 5 Companies Within Each Sector

[8]:
```python
# Initialize an empty data frame to contain the filtered data
top_data = pd.DataFrame()

# Iterate across a list of the unique sectors
for sector in data['Sector'].unique():

    # Filter the data by the current sector
    sector_data = data[data['Sector'] == sector]

    # Compile a list of the top five weights in the current sector
    top_five_weights = sorted(sector_data['Weight'].unique(), reverse = True)[:
 ↪5]

    # Filter the data by the top five weights
    sector_data = sector_data[sector_data['Weight'].isin(top_five_weights)]

    # Concatenate the top five companies' data into the accumulating dataframe
    top_data = pd.concat([top_data, sector_data], ignore_index = True)

# Print the last five rows of the filtered data frame
top_data.tail()
```

[8]:
```
       Symbol Security                  Sector  \
518500     VZ  Verizon  Communication Services
518501     VZ  Verizon  Communication Services
518502     VZ  Verizon  Communication Services
518503     VZ  Verizon  Communication Services
518504     VZ  Verizon  Communication Services

                                Industry    Weight        Date       Open  \
518500  Integrated Telecommunication Services  0.457305 2022-12-06  36.990002
```

```
518501  Integrated Telecommunication Services  0.457305 2022-12-07  36.740002
518502  Integrated Telecommunication Services  0.457305 2022-12-08  37.110001
518503  Integrated Telecommunication Services  0.457305 2022-12-09  37.209999
518504  Integrated Telecommunication Services  0.457305 2022-12-12  37.689999

             High        Low      Close  Adjusted Close      Volume
518500  37.070000  36.630001  36.889999       36.889999  26293700.0
518501  37.310001  36.669998  37.169998       37.169998  23065900.0
518502  37.240002  36.869999  37.099998       37.099998  19549100.0
518503  37.630001  36.959999  37.400002       37.400002  20669100.0
518504  37.730000  37.279999  37.615002       37.615002   4698435.0
```

1.4 Exploratory Data Analysis and Data Visualization

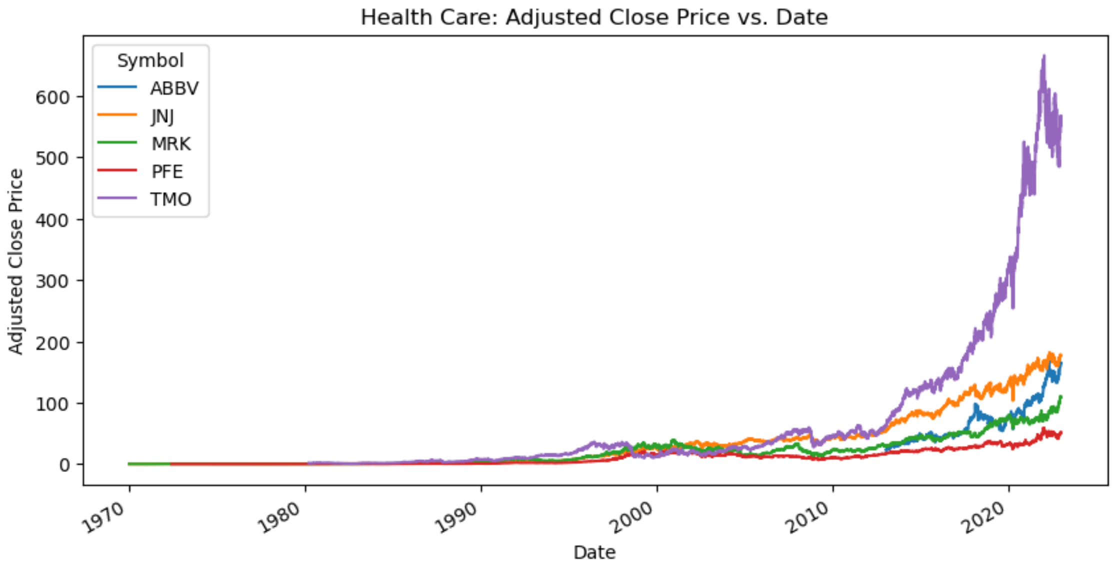
Before we fit a machine learning model to our data, we would like to visualize it by sector and preliminarily determine relationships between the data. In particular, we would like to analyze how strongly the stock prices of companies within the same sector are correlated.

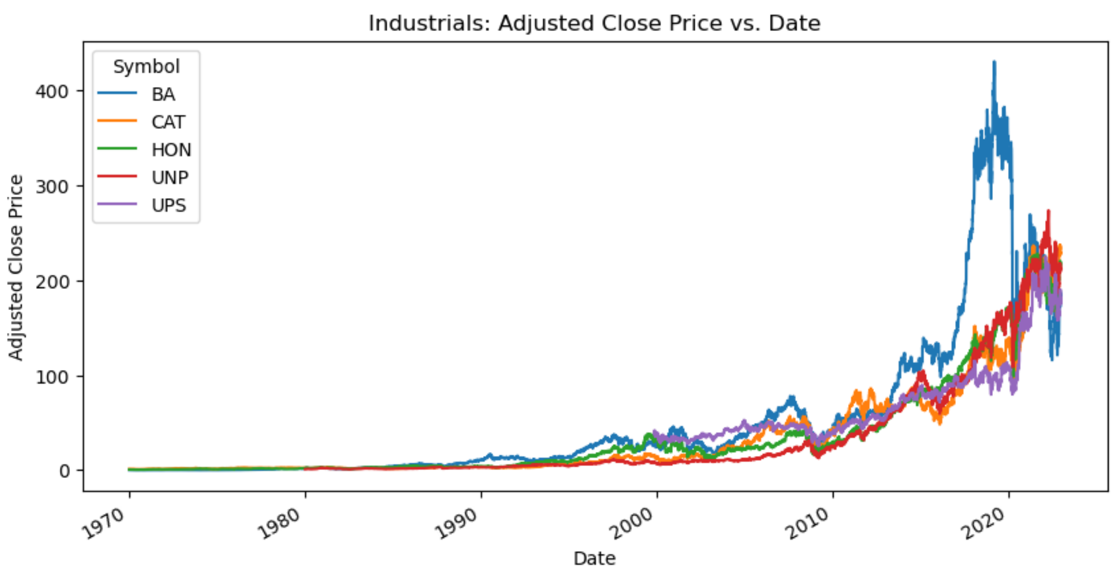
For the remainder of our analysis, we will focus our attention on adjusted close price, which is explained in the following section.

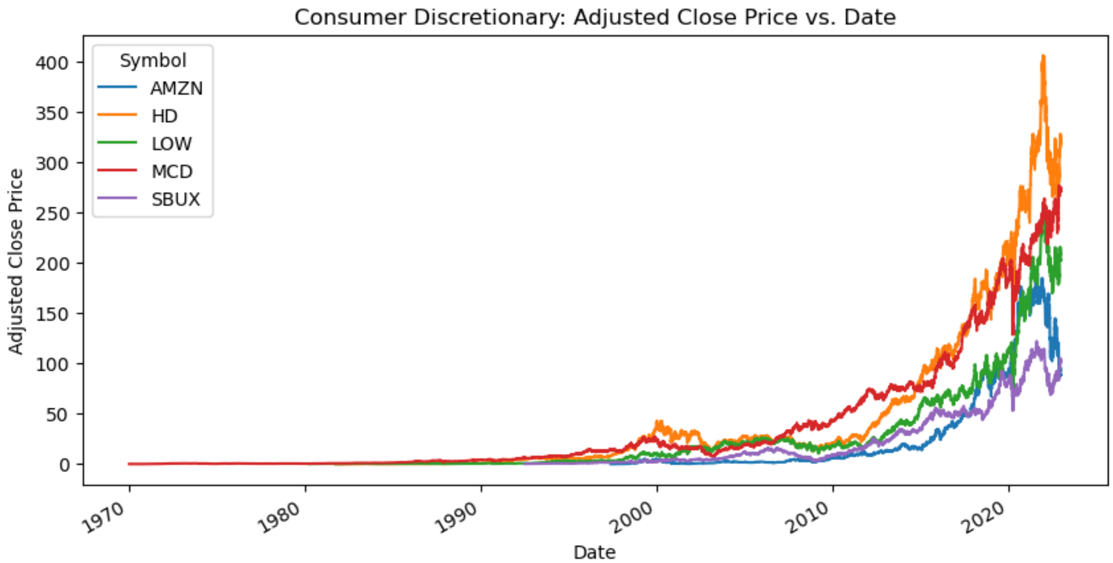
1.4.1 Plotting Adjusted Close Price vs. Date

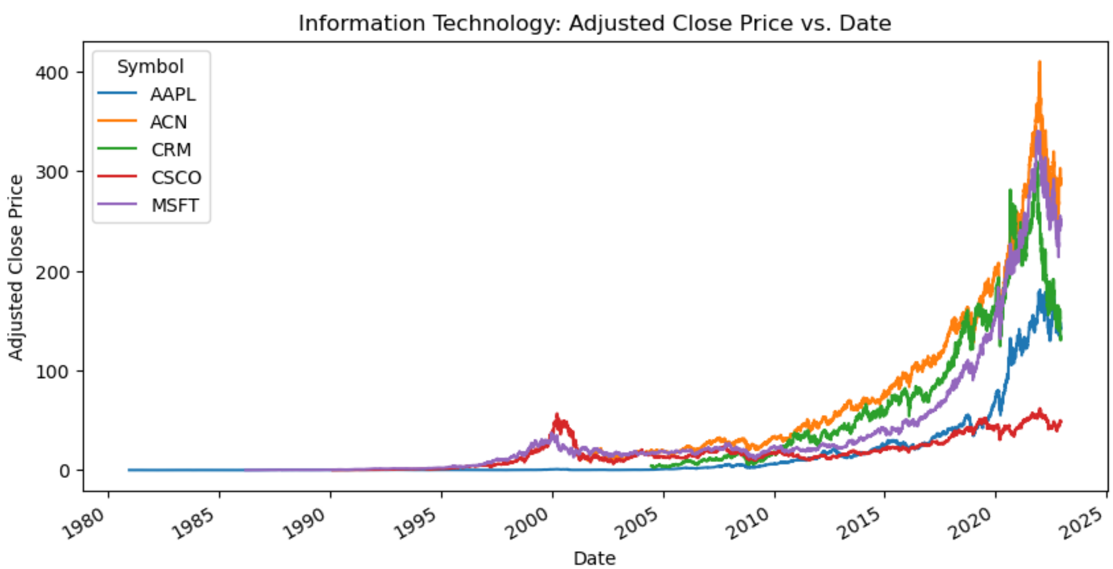
Adjusted close price is the final price at which a security trades at the end of a trading day, adjusting for dividends, stock splits, and new offerings. It is the most accurate representation of a company's stock price, and it is commonly used by investors and traders to track performance.

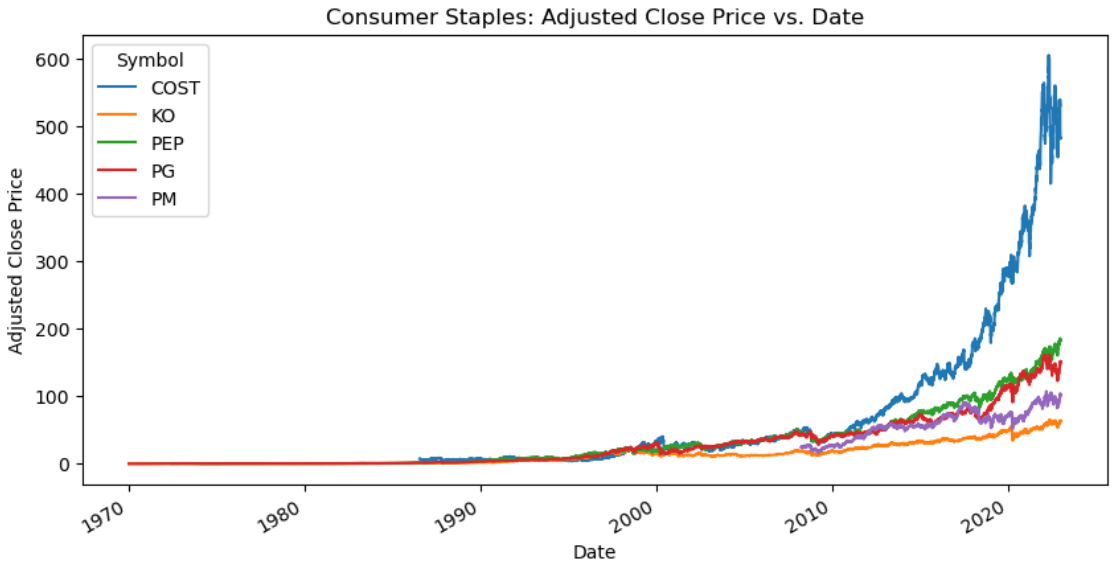
[9]:
```
# Generate a plot for the top five companies in each sector
for sector in top_data['Sector'].unique():

    # Filter the data for the current sector
    sector_data = top_data[top_data['Sector'] == sector]

    # Reshape the data for plotting purposes
    sec_as_row = sector_data.pivot(index = 'Date', columns = 'Symbol', values =
 ↪'Adjusted Close')

    # Generate plot
    sec_as_row.plot(title = f'{sector}: Adjusted Close Price vs. Date', legend
 ↪= True, xlabel = 'Date', ylabel = 'Adjusted Close Price', figsize = (10, 5))
```

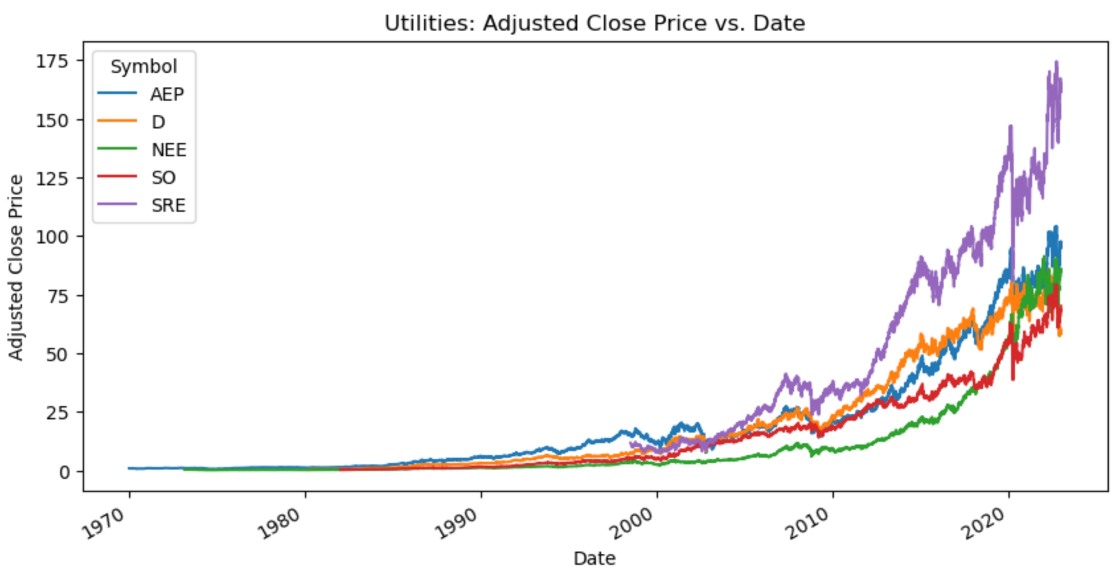

Health Care: Adjusted Close Price vs. Date
Symbol
ABBV
JNJ
MRK
PFE
TMO
Adjusted Close Price
0
100
200
300
400
500
600
1970
1980
1990
2000
2010
2020
Date

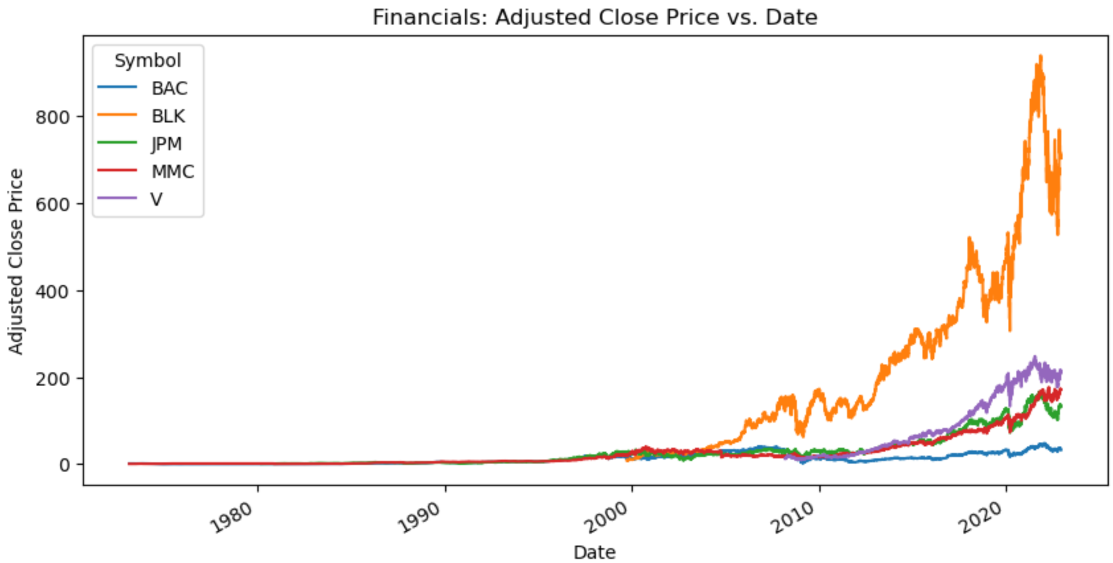

Industrials: Adjusted Close Price vs. Date
Symbol
BA
CAT
HON
UNP
UPS
Adjusted Close Price
0
100
200
300
400
1970
1980
1990
2000
2010
2020
Date

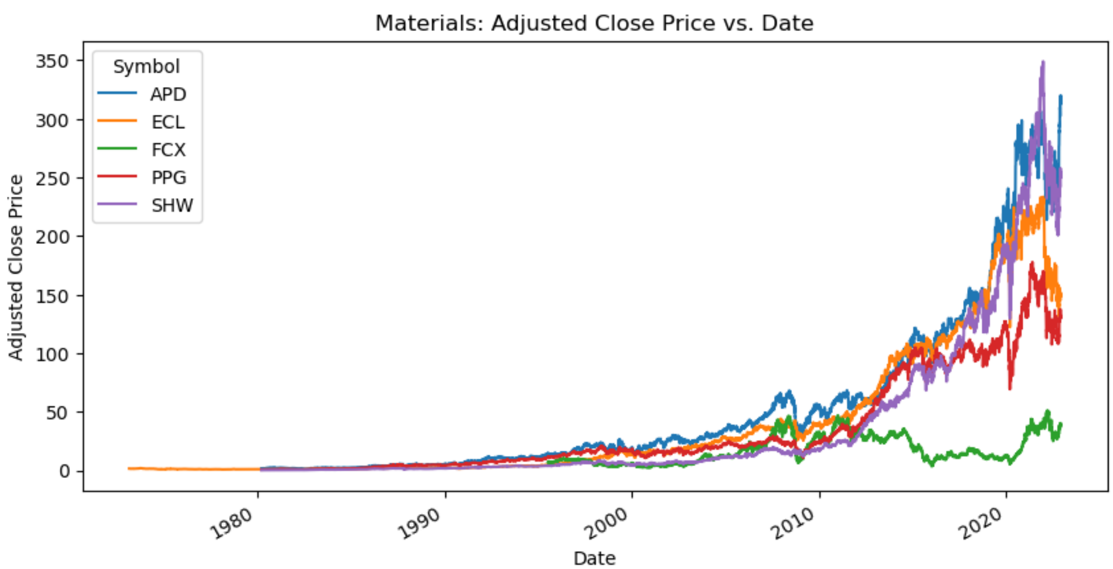

Consumer Discretionary: Adjusted Close Price vs. Date
Symbol
AMZN
HD
LOW
MCD
SBUX
Adjusted Close Price
0
50
100
150
200
250
300
350
400
1970
1980
1990
2000
2010
2020
Date


Information Technology: Adjusted Close Price vs. Date
Symbol
AAPL
ACN
CRM
CSCO
MSFT
Adjusted Close Price
0
100
200
300
400
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
Date

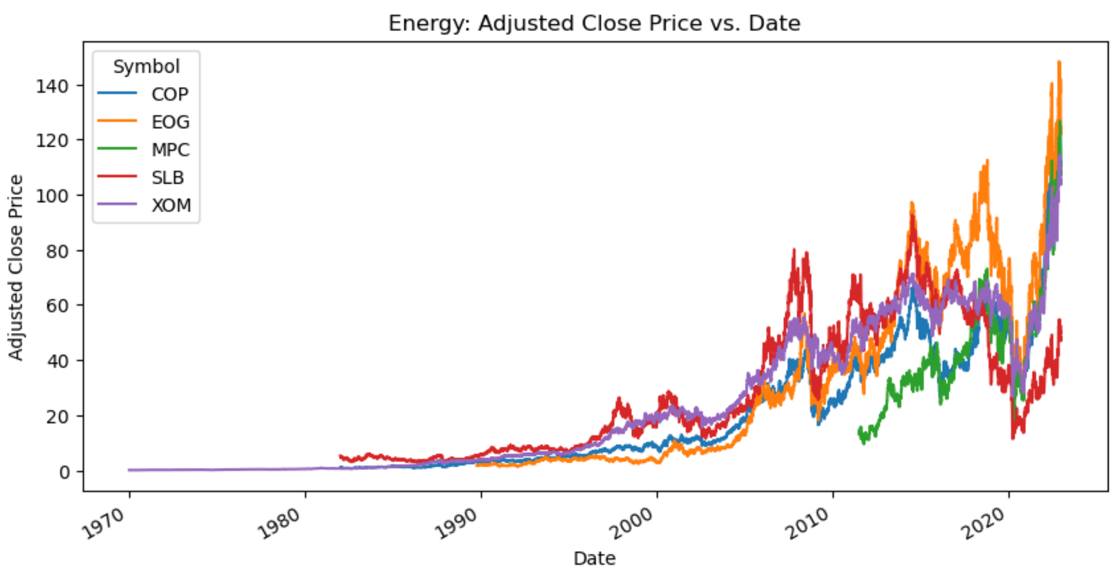

Consumer Staples: Adjusted Close Price vs. Date
Symbol
COST
KO
PEP
PG
PM
Adjusted Close Price
0
100
200
300
400
500
600
1970
1980
1990
2000
2010
2020
Date

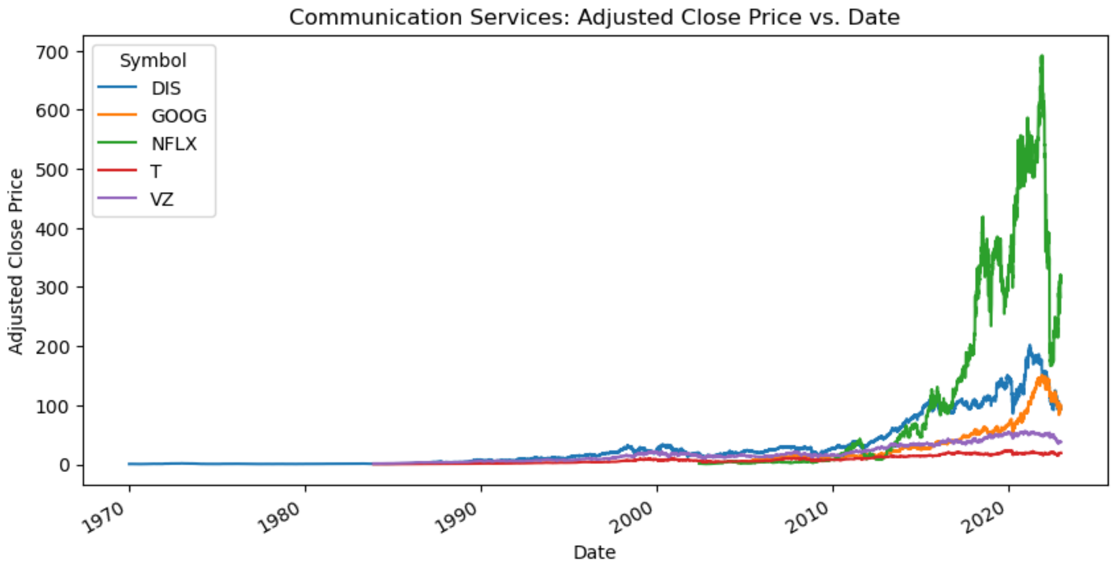

Utilities: Adjusted Close Price vs. Date
Symbol
AEP
D
NEE
SO
SRE
Adjusted Close Price
0
25
50
75
100
125
150
175
1970
1980
1990
2000
2010
2020
Date


Financials: Adjusted Close Price vs. Date
Symbol
BAC
BLK
JPM
MMC
V
Adjusted Close Price
800
600
400
200
0
1980
1990
2000
2010
2020
Date


Materials: Adjusted Close Price vs. Date
Symbol
APD
ECL
FCX
PPG
SHW
Adjusted Close Price
350
300
250
200
150
100
50
0
1980
1990
2000
2010
2020
Date


Real Estate: Adjusted Close Price vs. Date
Symbol
AMT
CCI
EQIX
O
PLD
Adjusted Close Price
800
600
400
200
0
1996
2000
2004
2008
2012
2016
2020
2024
Date


Energy: Adjusted Close Price vs. Date
Symbol
COP
EOG
MPC
SLB
XOM
Adjusted Close Price
140
120
100
80
60
40
20
0
1970
1980
1990
2000
2010
2020
Date



Above are 11 line plots of adjusted close price vs. date for the the top five companies (by weight) in each sector.

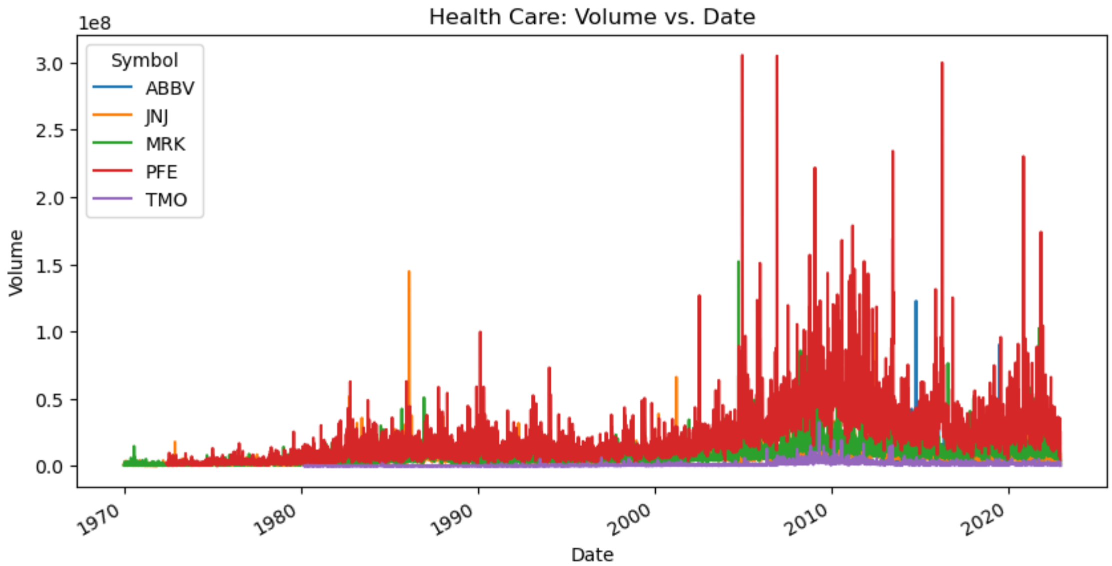
In the Health Care sector, one company had a much higher close price while the other four were closely correlated with one another. This case of the top company having a significantly greater closing price while the other four were much lower but closer to each other is a general trend that is visible through several of these graphs. In addition to the Health Care sector, this trend is present in the Financials sector, Consumer Staples sector, and Industrials sector, but interestingly in the Industrials sector as the top company's adjusted close price began to fall, the other four companies' adjusted close price rose together rather than one company taking over and continuing the trend. Other sectors have closer adjusted close prices amongst the top 5 companies: for example, in the Information Technology sector, ACN, MSFT, and CRM follow similar growth trends and maintain a similar price over the years while CSCO and AAPL trail behind. Also, in the Energy sector, MPC, XOM, COP, and EOG, essentially follow the same trend and stock price while SLB is consistently lower, so within this sector four companies are equally competitive rather than the trend of one company dominance that was seen in other sectors.

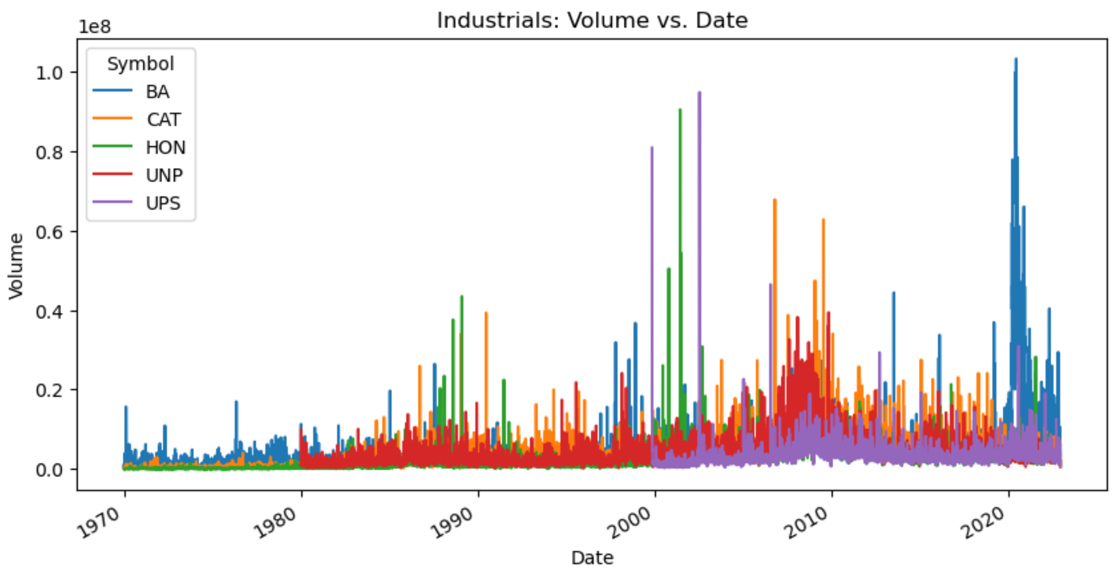
1.4.2 Plotting Volume vs. Date

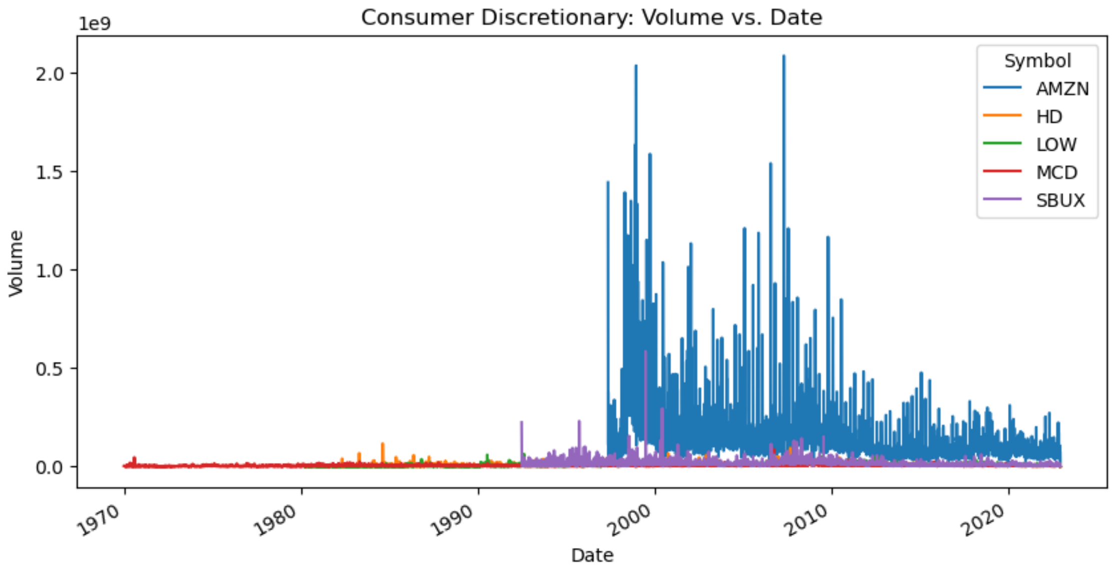
Volume traded is the number of shares that are transferred between constituents during the trading day. This is an important metric for investors and traders to consider.

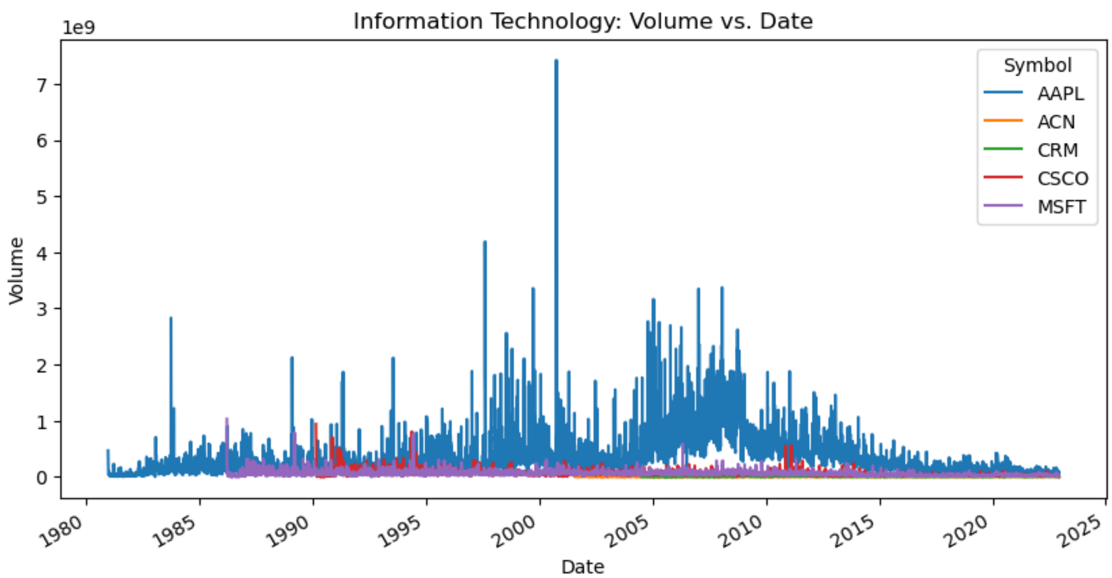
[10]:
```
# Generate a plot for the top five companies in each sector
for sector in top_data['Sector'].unique():

    # Filter the data for the current sector
    sector_data = top_data[top_data['Sector'] == sector]

    # Reshape the data for plotting purposes
```

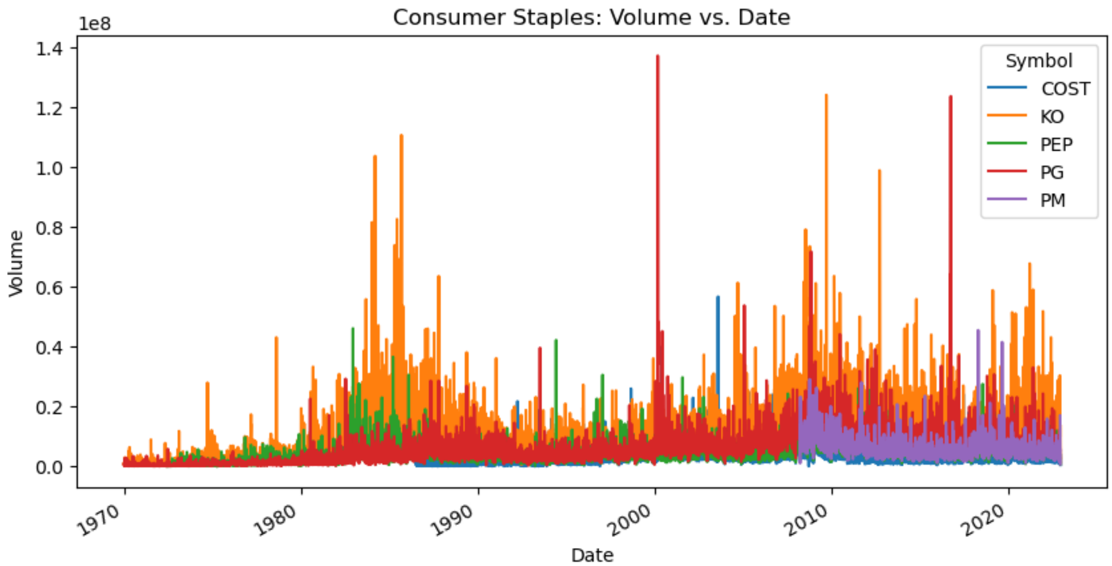
```
    sec_as_row = sector_data.pivot(index = 'Date', columns = 'Symbol', values = 'Volume')

    # Generate plot
    sec_as_row.plot(title = f'{sector}: Volume vs. Date', legend = True, xlabel = 'Date', ylabel = 'Volume', figsize = (10, 5))
```

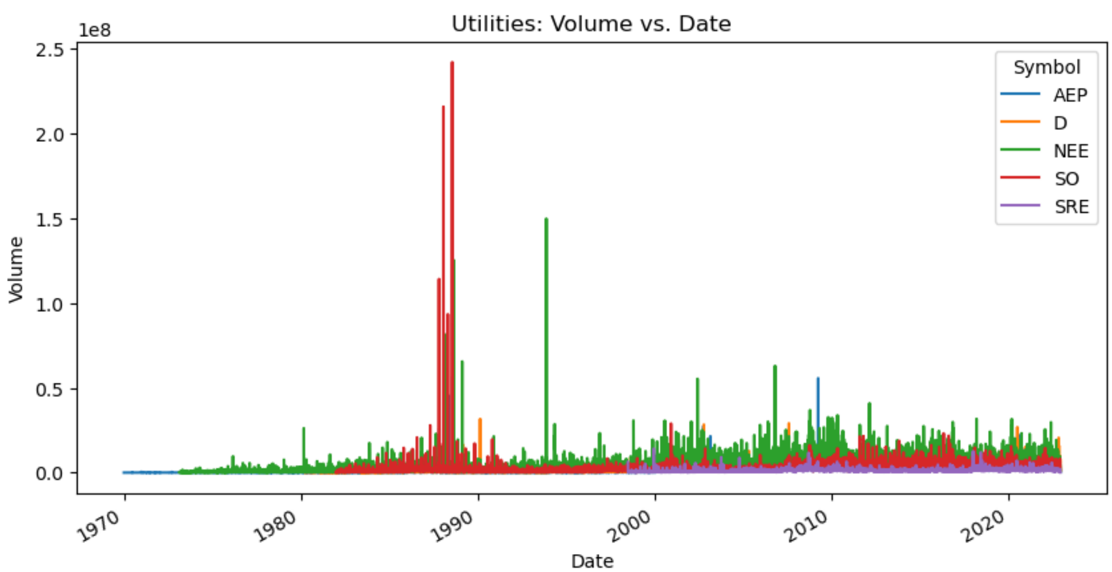


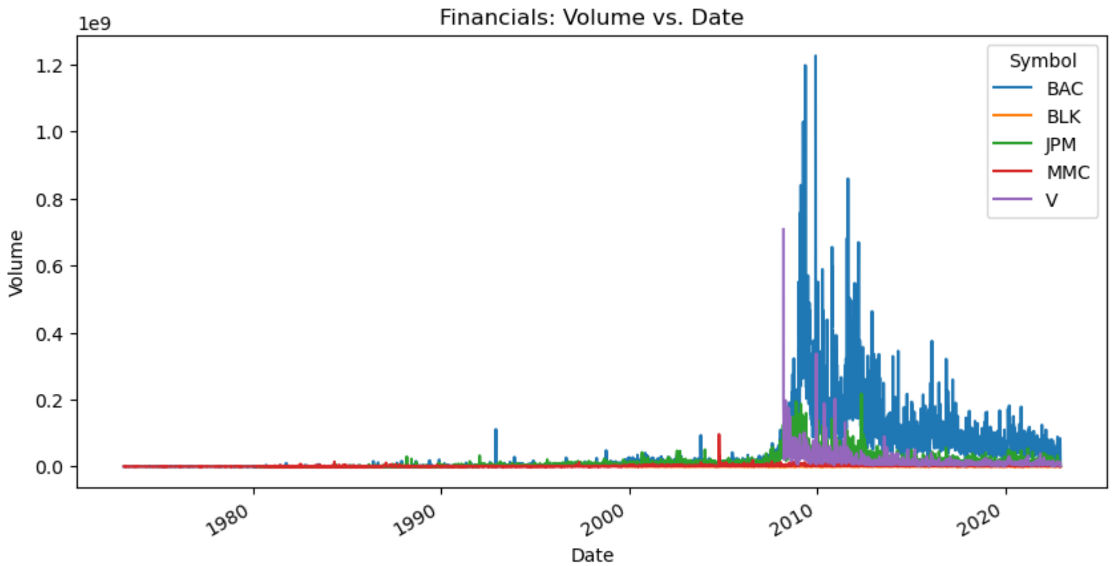


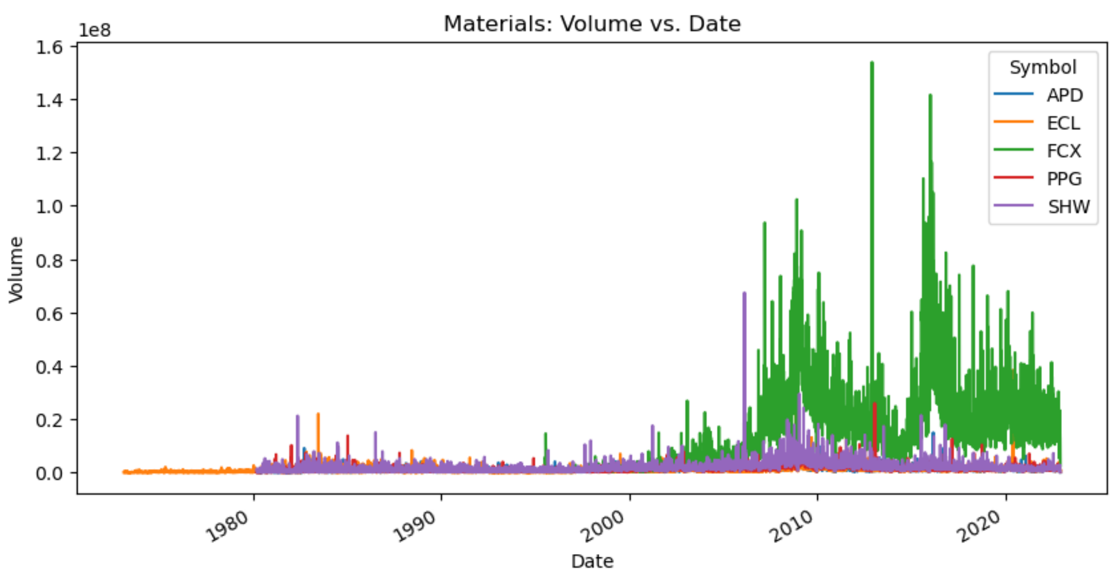

Consumer Discretionary: Volume vs. Date
1e9
2.0
1.5
1.0
0.5
0.0
Volume
1970
1980
1990
2000
2010
2020
Date
Symbol
AMZN
HD
LOW
MCD
SBUX

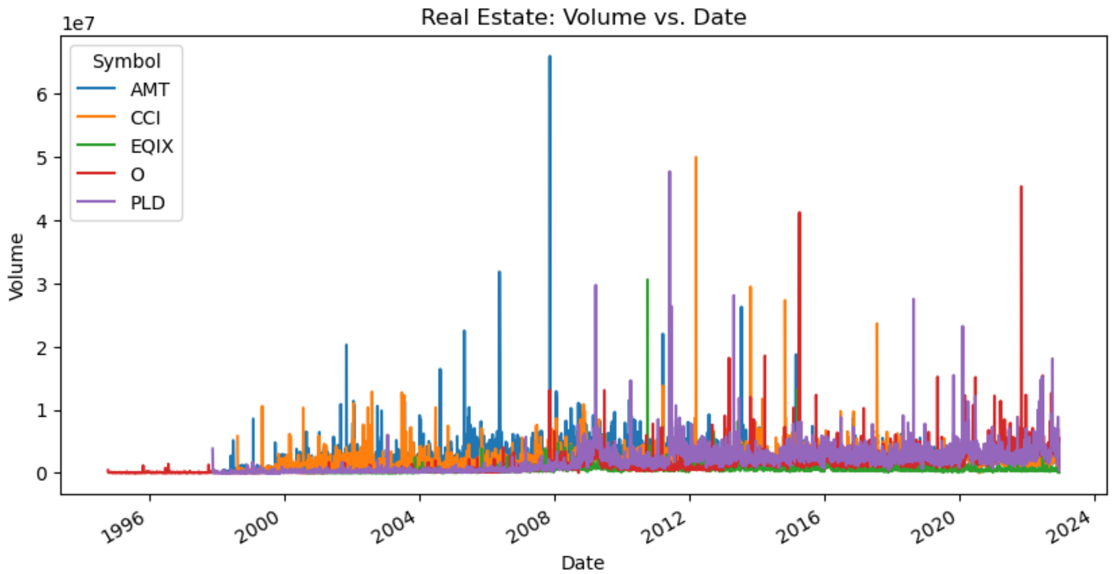

Information Technology: Volume vs. Date
1e9
7
6
5
4
3
2
1
0
Volume
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
Date
Symbol
AAPL
ACN
CRM
CSCO
MSFT

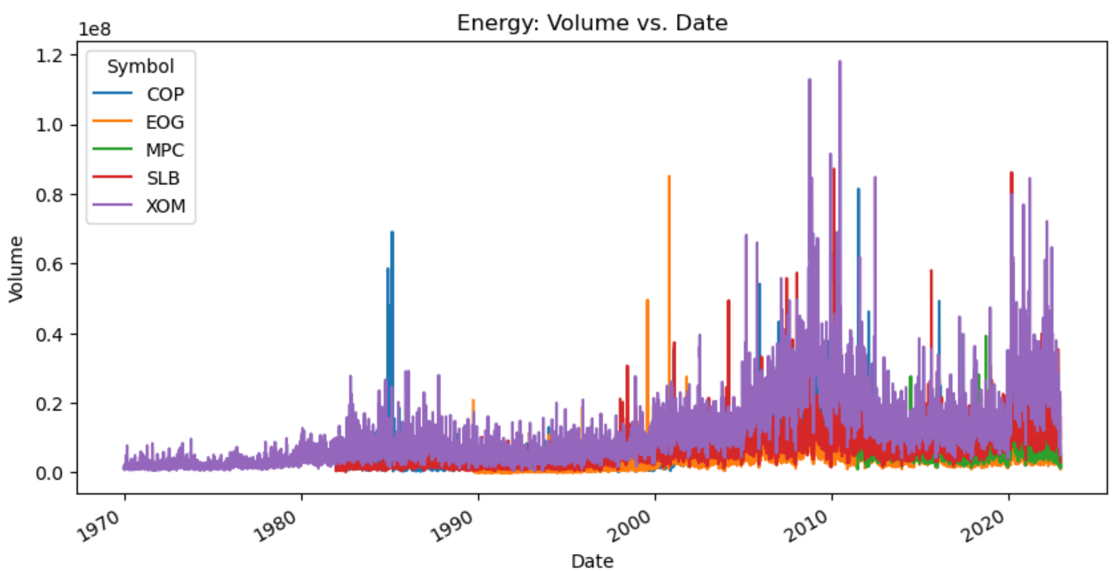

Consumer Staples: Volume vs. Date
1e8
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Volume
Symbol
COST
KO
PEP
PG
PM
1970
1980
1990
2000
2010
2020
Date

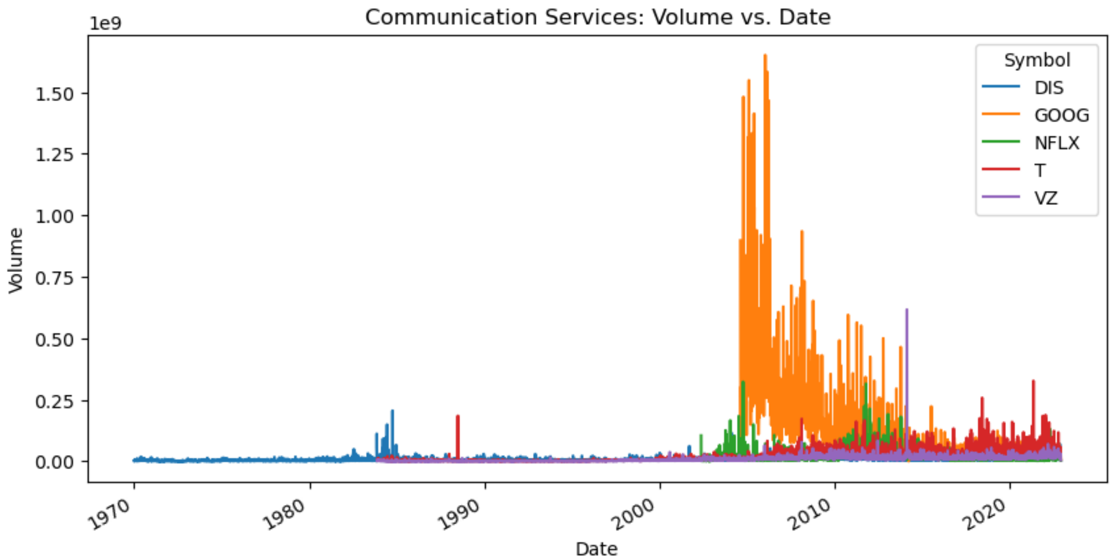

Utilities: Volume vs. Date
1e8
2.5
2.0
1.5
1.0
0.5
0.0
Volume
Symbol
AEP
D
NEE
SO
SRE
1970
1980
1990
2000
2010
2020
Date


Financials: Volume vs. Date
1e9
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Volume
1980
1990
2000
2010
2020
Date
Symbol
BAC
BLK
JPM
MMC
V


Materials: Volume vs. Date
1e8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Volume
1980
1990
2000
2010
2020
Date
Symbol
APD
ECL
FCX
PPG
SHW


Real Estate: Volume vs. Date
1e7
Symbol
AMT
CCI
EQIX
O
PLD
Volume
0
1
2
3
4
5
6
Date
1996
2000
2004
2008
2012
2016
2020
2024


Energy: Volume vs. Date
1e8
Symbol
COP
EOG
MPC
SLB
XOM
Volume
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Date
1970
1980
1990
2000
2010
2020



Above are 11 line plots of volume traded vs. date for the the top five companies (by weight) in each sector.

It is apparent that while there are correlations amongst the companies within each sector, one company often dominates the volume traded or has strong, isolated shifts. For example, in the Consumer Discretionary sector, Amazon (ticker AMZN) has the greatest volume traded since approximately 2000. It has had up to 2 billion dollar trading volumes at certain points. Similarly, in the Financials sector, Bank of America (ticker BAC) has the greatest volume traded since approximately 2010. It has had up to 1 billion dollar trading volumes at certain points, whereas its competitors have only had up to 600 million dollar trading volumes.

1.4.3 Calculating Various Moving Averages

Moving average standardizes the price of a stock by converting it to a constantly updated average price. This average is calculated over a predetermined time period. The most relevant and commonly used time periods for calculating moving average are 10 days and 20 days.

[11]:
```
# Lengths of moving averages (in days) to calculate
moving_averages = [10, 20]

# Iterate across the moving averages
for ma in moving_averages:

    # Iterate across each company
    for security in top_data['Security'].unique():

        # Filter the data for the current company
        security_data = top_data[top_data['Security'] == security]
```

```
        # Add a column containing the current company's moving average
        top_data[f'{ma}-Day Moving Average'] = top_data['Adjusted Close'].
↪rolling(ma).mean()

# Print the last five rows of the data frame
top_data.tail()
```

[11]:
```
       Symbol Security                  Sector  \
518500     VZ  Verizon  Communication Services
518501     VZ  Verizon  Communication Services
518502     VZ  Verizon  Communication Services
518503     VZ  Verizon  Communication Services
518504     VZ  Verizon  Communication Services

                                     Industry    Weight       Date       Open  \
518500  Integrated Telecommunication Services  0.457305 2022-12-06  36.990002
518501  Integrated Telecommunication Services  0.457305 2022-12-07  36.740002
518502  Integrated Telecommunication Services  0.457305 2022-12-08  37.110001
518503  Integrated Telecommunication Services  0.457305 2022-12-09  37.209999
518504  Integrated Telecommunication Services  0.457305 2022-12-12  37.689999

             High        Low      Close  Adjusted Close      Volume  \
518500  37.070000  36.630001  36.889999       36.889999  26293700.0
518501  37.310001  36.669998  37.169998       37.169998  23065900.0
518502  37.240002  36.869999  37.099998       37.099998  19549100.0
518503  37.630001  36.959999  37.400002       37.400002  20669100.0
518504  37.730000  37.279999  37.615002       37.615002   4698435.0

        10-Day Moving Average  20-Day Moving Average
518500                38.3170               38.23500
518501                38.1140               38.20000
518502                37.9320               38.17400
518503                37.7700               38.11800
518504                37.7075               38.08375
```

1.4.4 Calculating Daily Returns

Daily return is the percentage change in the price of stock over the course of a trading day. This will help us assess the risk of investing in a particular company.

[12]:
```
# Initialize an empty data frame to contain the daily return values
return_data = pd.DataFrame()

# Iterate across the sectors
for security in top_data['Security'].unique():

    # Filter the data for the current security
```

```
    security_data = top_data[top_data['Security'] == security]

    # Calculate the percent change i.e. daily return
    security_rets = pd.DataFrame(security_data['Adjusted Close'].pct_change())

    # Append this data to the accumulating data frame
    return_data = pd.concat([return_data, security_rets], ignore_index = True)

# Add the daily return values to the top company data frame
top_data['Daily Return'] = return_data

# Print the last five rows of the top data frame
top_data.tail()
```

[12]:

```
       Symbol Security                  Sector  \
518500     VZ  Verizon  Communication Services
518501     VZ  Verizon  Communication Services
518502     VZ  Verizon  Communication Services
518503     VZ  Verizon  Communication Services
518504     VZ  Verizon  Communication Services

                                     Industry    Weight        Date       Open  \
518500  Integrated Telecommunication Services  0.457305  2022-12-06  36.990002
518501  Integrated Telecommunication Services  0.457305  2022-12-07  36.740002
518502  Integrated Telecommunication Services  0.457305  2022-12-08  37.110001
518503  Integrated Telecommunication Services  0.457305  2022-12-09  37.209999
518504  Integrated Telecommunication Services  0.457305  2022-12-12  37.689999

             High        Low      Close  Adjusted Close      Volume  \
518500  37.070000  36.630001  36.889999       36.889999  26293700.0
518501  37.310001  36.669998  37.169998       37.169998  23065900.0
518502  37.240002  36.869999  37.099998       37.099998  19549100.0
518503  37.630001  36.959999  37.400002       37.400002  20669100.0
518504  37.730000  37.279999  37.615002       37.615002   4698435.0

        10-Day Moving Average  20-Day Moving Average  Daily Return
518500                38.3170               38.23500     -0.004856
518501                38.1140               38.20000      0.007590
518502                37.9320               38.17400     -0.001883
518503                37.7700               38.11800      0.008086
518504                37.7075               38.08375      0.005749
```

1.4.5 Plotting and Comparing the Daily Returns of Various Stocks

Next, we will plot the daily returns of various stocks against one another. This will help us assess whether the stock prices of companies in the same sector are strongly correlated or not. We expect that they are linearly and positively correlated.

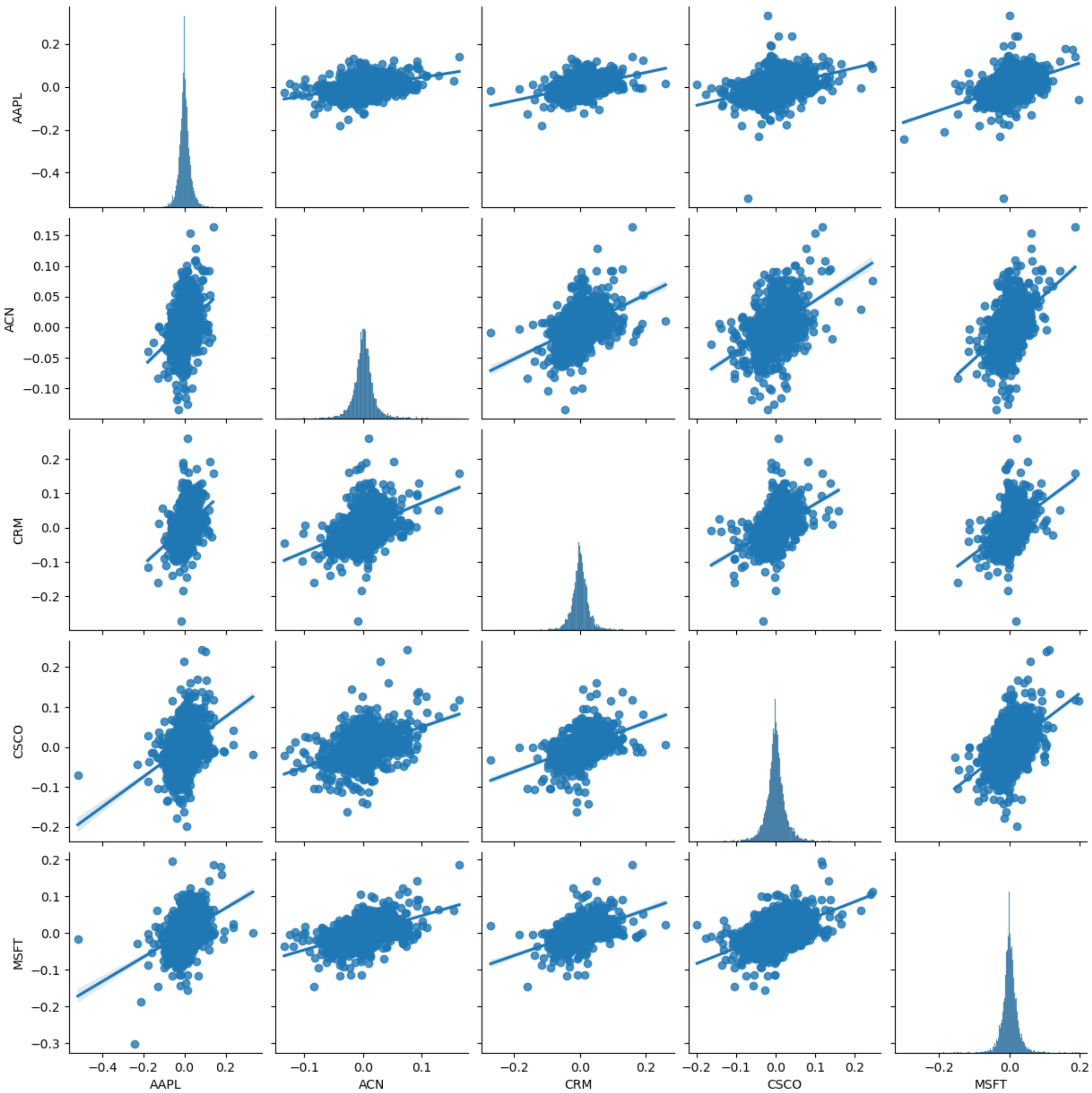
For the purposes of this project, we will only show the plot for the Information Technology sector. This will avoid long, repetitive outputs.

[13]:
```python
# Initialize a data frame to contain the formatted data for plotting
formatted_data = top_data[['Symbol', 'Date', 'Daily Return']]

# Pivots the ticker symbols from a column's entries to column headers
formatted_data = formatted_data.pivot(index = 'Date', columns = 'Symbol',␣
 ↪values = 'Daily Return')

# Print the last five rows of the formatted data frame
formatted_data.tail()
```

[13]:
```
Symbol          AAPL      ABBV       ACN       AEP       AMT      AMZN  \
Date
2022-12-06 -0.025370 -0.001342 -0.025039  0.019573 -0.014331 -0.030326
2022-12-07 -0.013785  0.010261  0.004485  0.003113 -0.006635  0.002380
2022-12-08  0.012133  0.003567  0.019045  0.010862  0.005400  0.021366
2022-12-09 -0.003435 -0.017652 -0.012802 -0.011666  0.007491 -0.013946
2022-12-12  0.000563  0.002112  0.010090  0.004608  0.001356 -0.006566

Symbol           APD        BA       BAC       BLK  …       SLB        SO  \
Date                                                …
2022-12-06 -0.009112 -0.036035 -0.042646  0.003404  … -0.006347  0.014819
2022-12-07  0.011150 -0.010817 -0.007879 -0.001591  … -0.021226  0.000292
2022-12-08  0.013562  0.014618 -0.009163 -0.008066  …  0.002410  0.002628
2022-12-09 -0.017039  0.002569 -0.001849  0.004990  … -0.059094 -0.004659
2022-12-12  0.003037  0.038265  0.005405 -0.003105  …  0.023206  0.025819

Symbol           SRE         T       TMO       UNP       UPS         V  \
Date
2022-12-06  0.010141  0.022400 -0.011988  0.000379 -0.033451 -0.021527
2022-12-07 -0.017720  0.006781  0.013823  0.005115 -0.002456 -0.006074
2022-12-08 -0.001970 -0.009326  0.017576  0.003817  0.028396  0.006208
2022-12-09 -0.005861 -0.001569 -0.013593 -0.007886 -0.011078 -0.001913
2022-12-12  0.009246  0.005500  0.010951  0.004353  0.019815  0.022377

Symbol            VZ       XOM
Date
2022-12-06 -0.004856 -0.027796
2022-12-07  0.007590 -0.002214
2022-12-08 -0.001883  0.007429
2022-12-09  0.008086 -0.008427
2022-12-12  0.005749  0.017288

[5 rows x 55 columns]
```

[14]:
```
# Pair plot the comparisons of daily returns for all companies in the
↪Information Technology sector
sns.pairplot(formatted_data[['AAPL', 'ACN', 'CRM', 'CSCO', 'MSFT']], kind =
↪'reg')
```

[14]: <seaborn.axisgrid.PairGrid at 0x7f54f8a6e4a0>



The above plot is a pair plot for the daily returns of the top five companies (by weight) in the Information Technology sector. It is apparent that there is a somewhat linear correlation between the companies, with some outlying cases. In general, as one company's daily return increases, the daily returns of the other companies increase. Hence, we can proceed with our modeling.

1.5 Data Analysis, Hypothesis Testing, and Machine Learning

We will be implementing the Sequential machine learning model from the Keras library. This works by having several different types of layers that sequentially feed into eachother during the training process. Each layer performs a unique computation on the input data and the model feeds the output of that layer to another layer. Source: https://www.educba.com/keras-sequential/

1.5.1 Filtering the Data to Google

At this point in our analysis, we will focus our attention on a company that is driving a lot of movement within the S&P 500: Google. Google is an industry leader in the Communication Services sector and its respective industry. This decision was made by our team because of computational limitations with regard to training the following machine learning model. This process is both time-consuming and resource-intensive. Note that this modeling could be applied to any company in the S&P 500.

[15]:
```
# Filter the data to only contain the symbol and adjusted close
goog_data = top_data.filter(['Date', 'Adjusted Close', 'Symbol'])

# Filter the data to only contain the data for Google
goog_data = goog_data[goog_data['Symbol'] == 'GOOG']

# Drop the symbol column
goog_data = goog_data.drop(columns = ['Symbol'])

# Set date to the data frame's index
goog_data.set_index('Date', inplace = True)

# Print the last five rows of the data frame
goog_data.head()
```

[15]:

Date	Adjusted Close
2004-08-19	2.499133
2004-08-20	2.697639
2004-08-23	2.724787
2004-08-24	2.611960
2004-08-25	2.640104

1.5.2 Long Short-Term Memory Modeling

Our goal is to predict the future stock prices of Google (ticker GOOG). In order to do so, we need to train and fit a model that could then estimate stock prices in the immediate future. We can use an LTSM model, or a Long Short-Term Memory model, designed by Fares Sayah at Kaggle, and modify it so our data could be used instead.

In this case, our goal is to predict the future stock prices of Google.

Thanks to Fares Sayah for his documentation of this particular model! To read more: https://www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm/notebook.

Organizing the Training Data Ideally, we can use 95% of our data to train our model.

[16]:
```
# Select the values to be trained
pre_train = goog_data.values

# Calculate the length of the training data
training_data_len = int(np.ceil(len(pre_train) * .95))

# Print this length
training_data_len
```

[16]:
```
4382
```

Fitting the Data and Getting the Training Data We need the data from the range of (0,1) to make sure that all of the values are in a similar range to make our machine learning model more accurate. We can then prepare the training data to be used by our model

[17]:
```
# Create a scaler
scaler = MinMaxScaler(feature_range = (0, 1))

# Create scaled data
scaled_data = scaler.fit_transform(pre_train)

# Print the scaled data frame
scaled_data
```

[17]:
```
array([[5.54601395e-05],
       [1.39474256e-03],
       [1.57790515e-03],
       …,
       [6.17057513e-01],
       [6.11120335e-01],
       [6.07105972e-01]])
```

Building and Training the Model The machine learning model used here is the Sequential model from the Keras library. This model is useful when there is one precisely input and one output. This holds in our case since the model process stock prices over time in order to make a singular prediction for a singular point in time.

[18]:
```
# Total training data
train_data = scaled_data[0:int(training_data_len), :]

x_train = []
y_train = []

# Split training data for x and y axis
for i in range(60, len(train_data)):
```

```python
    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))

# 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)
```

```
[array([5.54601395e-05, 1.39474256e-03, 1.57790515e-03, 8.16681841e-04,
       1.00656502e-03, 1.32752381e-03, 1.03177125e-03, 3.36085743e-04,
       3.96579086e-04, 4.03331846e-05, 2.52062297e-04, 0.00000000e+00,
       2.63827349e-04, 3.84818860e-04, 3.86498202e-04, 8.93979873e-04,
       1.25862571e-03, 1.92911142e-03, 2.01480938e-03, 2.34584744e-03,
       2.93735417e-03, 3.25158754e-03, 2.99616656e-03, 3.08690416e-03,
       3.49692925e-03, 3.33056652e-03, 3.06674561e-03, 4.51189259e-03,
       5.22102410e-03, 4.97232210e-03, 5.47308479e-03, 5.88982081e-03,
       6.44603560e-03, 6.22926685e-03, 6.52669393e-03, 6.33849170e-03,
       5.92343341e-03, 6.28303960e-03, 6.87117960e-03, 7.05602153e-03,
       7.41058809e-03, 8.25919514e-03, 8.05418662e-03, 6.80228150e-03,
       8.29616160e-03, 1.21695001e-02, 1.46850625e-02, 1.37440385e-02,
       1.44447685e-02, 1.56765022e-02, 1.52295171e-02, 1.61352524e-02,
       1.59403296e-02, 1.54025956e-02, 1.42313536e-02, 1.16519311e-02,
       1.21896587e-02, 1.15427095e-02, 1.14015514e-02, 1.39490471e-02])]
[0.013777644707187047]

[array([5.54601395e-05, 1.39474256e-03, 1.57790515e-03, 8.16681841e-04,
       1.00656502e-03, 1.32752381e-03, 1.03177125e-03, 3.36085743e-04,
       3.96579086e-04, 4.03331846e-05, 2.52062297e-04, 0.00000000e+00,
       2.63827349e-04, 3.84818860e-04, 3.86498202e-04, 8.93979873e-04,
```

```
       1.25862571e-03, 1.92911142e-03, 2.01480938e-03, 2.34584744e-03,
       2.93735417e-03, 3.25158754e-03, 2.99616656e-03, 3.08690416e-03,
       3.49692925e-03, 3.33056652e-03, 3.06674561e-03, 4.51189259e-03,
       5.22102410e-03, 4.97232210e-03, 5.47308479e-03, 5.88982081e-03,
       6.44603560e-03, 6.22926685e-03, 6.52669393e-03, 6.33849170e-03,
       5.92343341e-03, 6.28303960e-03, 6.87117960e-03, 7.05602153e-03,
       7.41058809e-03, 8.25919514e-03, 8.05418662e-03, 6.80228150e-03,
       8.29616160e-03, 1.21695001e-02, 1.46850625e-02, 1.37440385e-02,
       1.44447685e-02, 1.56765022e-02, 1.52295171e-02, 1.61352524e-02,
       1.59403296e-02, 1.54025956e-02, 1.42313536e-02, 1.16519311e-02,
       1.21896587e-02, 1.15427095e-02, 1.14015514e-02, 1.39490471e-02]),
array([1.39474256e-03, 1.57790515e-03, 8.16681841e-04, 1.00656502e-03,
       1.32752381e-03, 1.03177125e-03, 3.36085743e-04, 3.96579086e-04,
       4.03331846e-05, 2.52062297e-04, 0.00000000e+00, 2.63827349e-04,
       3.84818860e-04, 3.86498202e-04, 8.93979873e-04, 1.25862571e-03,
       1.92911142e-03, 2.01480938e-03, 2.34584744e-03, 2.93735417e-03,
       3.25158754e-03, 2.99616656e-03, 3.08690416e-03, 3.49692925e-03,
       3.33056652e-03, 3.06674561e-03, 4.51189259e-03, 5.22102410e-03,
       4.97232210e-03, 5.47308479e-03, 5.88982081e-03, 6.44603560e-03,
       6.22926685e-03, 6.52669393e-03, 6.33849170e-03, 5.92343341e-03,
       6.28303960e-03, 6.87117960e-03, 7.05602153e-03, 7.41058809e-03,
       8.25919514e-03, 8.05418662e-03, 6.80228150e-03, 8.29616160e-03,
       1.21695001e-02, 1.46850625e-02, 1.37440385e-02, 1.44447685e-02,
       1.56765022e-02, 1.52295171e-02, 1.61352524e-02, 1.59403296e-02,
       1.54025956e-02, 1.42313536e-02, 1.16519311e-02, 1.21896587e-02,
       1.15427095e-02, 1.14015514e-02, 1.39490471e-02, 1.37776447e-02])]
[0.013777644707187047, 0.014259921756710325]


2023-05-13 02:35:28.503510: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with
dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-13 02:35:28.509576: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype
int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-13 02:35:28.511614: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with
dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
2023-05-13 02:35:28.892036: I tensorflow/core/common_runtime/executor.cc:1197]
```

```
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with
dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-13 02:35:28.896459: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype
int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-13 02:35:28.901289: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with
dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
2023-05-13 02:35:29.634912: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with
dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-13 02:35:29.638368: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype
int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-13 02:35:29.641234: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with
dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
2023-05-13 02:35:29.958713: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with
dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-13 02:35:29.961697: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype
int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-13 02:35:29.965383: I tensorflow/core/common_runtime/executor.cc:1197]
```

```
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with
dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
2023-05-13 02:35:31.199359: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with
dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-13 02:35:31.203932: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype
int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-13 02:35:31.207815: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with
dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
2023-05-13 02:35:31.678822: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with
dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-13 02:35:31.682767: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype
int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-13 02:35:31.689594: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with
dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]

4322/4322 [==============================] - 183s 41ms/step - loss: 8.2505e-04
```

[18]:
```
<keras.callbacks.History at 0x7f54f854d870>
```

[19]:
```python
# Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data[training_data_len - 60: , :]
```

```python
# Create the data sets x_test and y_test
x_test = []
y_test = pre_train[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
```

```
2023-05-13 02:38:57.334339: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with
dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-13 02:38:57.339835: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype
int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-13 02:38:57.344173: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with
dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
2023-05-13 02:38:57.751703: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with
dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-13 02:38:57.755639: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
```

```
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype
int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-13 02:38:57.760419: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with
dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]

8/8 [==============================] - 2s 43ms/step
```

[19]:
```
4.739062835437262
```

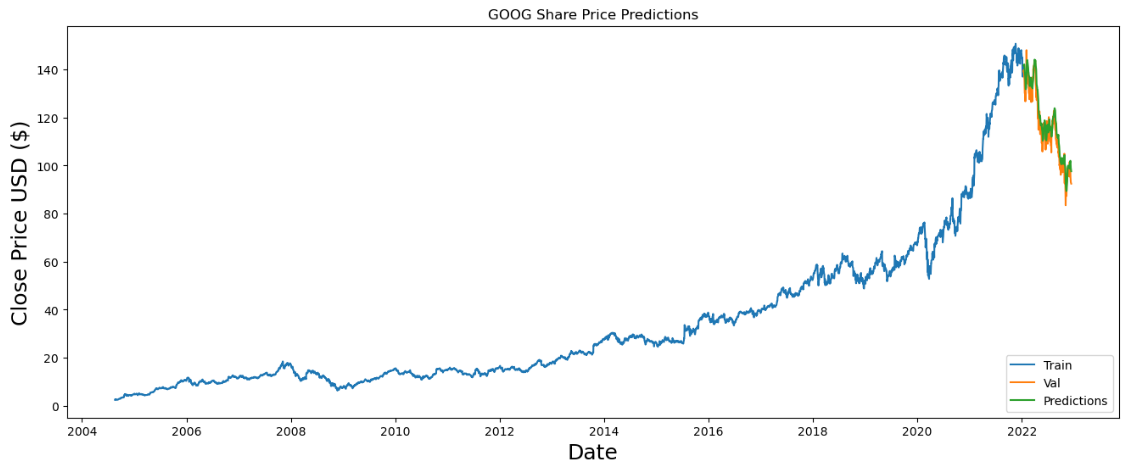
[20]:
```python
# Calculate range of stock price in order to contextualize RMSE

min_price = min(goog_data['Adjusted Close'])
max_price = max(goog_data['Adjusted Close'])

print(f'The GOOG stock price ranges from {min_price} to {max_price}')
```

```
The GOOG stock price ranges from 2.490912914276123 to 150.70899963378906
```

Root Mean Squared Error is a statistic used to evaluate the accuracy of the model's predictions. In this case the root mean squared error is 5.65 which is alright given the large range of stock prices. In the case of the GOOG stock price, the overall range is from about 2.491 to about 150.709 which is a relatively large range. The root mean squared error says on average the prediction varies from the actual value by 5.51 points. While this is not that high meaning the prediction was generally pretty good, it isn't excellent either, especially in high risk environments such as trading stocks.

Plotting the Predicted GOOG Stock Prices

[21]:
```python
# Plot the data
train = top_data[top_data['Symbol'] == 'GOOG'][:training_data_len]
valid = top_data[top_data['Symbol'] == 'GOOG'][training_data_len:]
valid['Predictions'] = predictions

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



1.6 Insights

In general this project involved a large amount of cleaning and processing data, urging us to learn how to omptimally categorize information in a way that makes trends/patterns in the data visually understandable instead of being overwhelmed by the sheer quantity of data provided. The collection process was also extensive as we learned that all the necessary data to carry out the operation is not always easily found in one source. Instead we had to combine several sources to make sure the necessary information was there. When it comes to the predictive algorithm we learned about selcting the right model and the extensive process required to modify data into the correct format required for that machine learning model. However, Applying the model, seeing the predictions, and understanding the accuracy of those predictions was truly exciting. Our model has proven to be moderately accurate in predicting stock prices as shown by the showcase of the prediction on GOOG. However, given the high stakes nature of stock trading, predictions provided are held to a signficantly higher standard and so while out model is pretty good at predicting stock prices, it isn't something that can be applied to real world cases just yet. The finance industry is the foundation of modern capitalism and as technology continues to rapidly develop, data science is used to make predicitions that optimize stock trades beyond the capabilites of humans allowing the industry to further develop and flourish. Therefore data science and the concepts discussed throughout this project have powerful real world applications, directly impacting the way our world runs today and develops in the future.