Which 10 US Stocks I Chose and Why

The stocks I chose are massive technology companies who are on the Fortune 100. Some of them are focused more on hardware, software, or a mix of the two. I am interested in these companies' stocks because I am a CIS major and would like to one day work in the IT field. Knowing how the biggest tech corporations are related to eachother will allow me to see how the industry works and may inform me on what I would like to specialize in.

What Do They Have In Common?

Apple is both a software and hardware technology company that has recently just started making chips for their Mac computers. So they create software for devices and they create phones and computers as well. This puts them in competition with Intel because Intel is mainly also computer chip manufacturer and even Apple used to use Intel CPUs until they began to make their own. Apple is also similar to Tesla because even though they do not directly complete, Tesla is also a corporation that creates hardware (cars) and also develops the software for these products. Tesla is one of the top companies in terms of developing and using AI. Their company has created a data farm by collecting data from every single one of their cars. They use this data to improve their autonomous driving capability. Google is another tech giant which is a software heavy firm which does heavy data collection to give advertisers the details they want about consumers. Meta does the same thing, however, they own different social media platforms which are both massive. IBM manufactures computer hardware and developes software. They also have cloud computing services. Clearly, all of the US stocks I chose are related to eachother given that they are heavy technology developers and innovators. Seeing if there stock prices move together in a linear way will help me understand which companies are in the growing part of the technology industry. This will help me think about what kind of job I should get.

Conclusion

The companies whose stock prices move together in a covariant way are Apple, Amazon, Google, Meta, and Tesla. This makes a lot of sense to me because these are the biggest tech companies in America. There stock moving the same direction means that people value them similarly and so their value fluctuates simililarly. Because they are the biggest companies I can infer that the most successful segment of the technology industry is the data collection, data analytics, ai development, and software development part of the industry. These types of work are these copmanies bread and butter and it might be wise to try and specialize in some of these jobs. Another group of companies whose stocks rise and fall in a covariant manner are Dell, HP, and Intel. These comanpanies are very similar in that they manufacture computer hardware and less so, software. I think the reason they move in a covariant way is because is because they are in the same lower tier of tech giants. Two companies that move on their own are Cisco Systems and IBM. To be honest, I am not quite sure why they are not covariant with the second tier of tech giants because they are quite similar. However, this may mean that I do not fully understand what their business model is. Overall, this study has led me to believe that there are two tiers of tech giants and that data analytics and AI development are some of the hottest jobs and will continue to grow in the years to come. I conclude this given that these are important skillsets for the top tech companies in America.

```
!pip install yfinance
!pip install vega_datasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.18)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.5.3)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.22.4)
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.27.1)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.9.2)
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.4.4)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2022.7.1)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.3.7)
Requirement already satisfied: cryptography>=3.3.2 in /usr/local/lib/python3.10/dist-packages (from yfinance) (40.0.2)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.11.2)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.4.1)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography>=3.3.2->yfinance) (1.15.1)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26->yfinance) (1.26.15
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26->yfinance) (2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26->yfinance) (2.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26->yfinance) (3.4)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography>=3.3.2->yfinance) (2.
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: vega_datasets in /usr/local/lib/python3.10/dist-packages (0.9.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from vega_datasets) (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->vega_datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->vega_datasets) (2022.7.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas->vega_datasets) (1.22.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->vega_datasets)
```

Data Ingest from Public Markets

The free, common Yahoo Finace API is used to download data from all commodites you wish to see studied. This data will be stored persistently next to your notebook in common environments such as Binder.

Please note that if you deploy this notebook in Google Collab that the 37+ files downloaded will be erased between uses, but can be rebuilt easily each time you operate this notebook.

The data you download becomes permanently usable, and the ingest request below can be customized in order to grab more, or less data and at different intervals.[2]

I have included several exceptions to the download and renaming technique, in order to tolerate commodities with differing ticker symbols.

```
import yfinance as yf
from time import time,ctime, clock_gettime
from time import gmtime, time, time_ns

def ifs(input):
    ni = ''
    if input =='gff':
        input = 'GFF'
        ni = "GF=F"
    elif input == 'zff':
        input = 'ZFF'
        ni = "ZF=F"
    else:
        input = input.upper()
        ins = "="
        before = "F"
        ni = input.replace(before, ins + before , 1)
    print(ni)
    data = yf.download(
        tickers = ni,
        period = "365d",
        interval = "1d",
        group_by = 'ticker',
        auto_adjust = True,
        prepost = True,
        threads = True,
        proxy = None
    )
    epoch = ctime()
    filename = input
    data.to_csv(filename)
#!ls #only in jupy
```

Trigger Data Downloads

The following code customizes the commodities under investigation. In order to compare every commodity's price history versus the rest in your matrix, the lengths of the data captures are minimized to the length of the smallest data set. Thus, larger sets are only captured at the length of the smallest set.

The volatility of every price tick is calculated via [close price minus open price].

```
#read in csv data from each commodity capture, gather
#assign 'open' to an array, create df from arrays
import numpy as np
import pandas as pd
from  scipy.stats import pearsonr
symbol_dict = {"TSLA":"Tesla","AMZN":"Amazon","AAPL":"Apple","GOOG":"Alphabet","CSCO":"Cisco Systems","META":"Meta Platforms","DELL":"Dell Te
    #"clf":"crude oil", "esf":"E-Mini S&P 500","btcf":"Bitcoin","bzf":"Brent Crude Oil", "ccf":"Cocoa","ctf":"Cotton","gcf":"Gold",
          # "gff":"Feeder Cattle", "hef":"Lean Hogs","hgf":"Copper","hof":"Heating Oil","kcf":"Coffee","kef":"KC HRW Wheat",
          # "lbsf":"Lumber","lef":"Live Cattle","mgcf":"Micro Gold","ngf":"Natural Gas","nqf":"Nasdaq 100","ojf":"Orange Juice","paf":"Pallad
           # "rbf":"RBOB Gasoline","rtyf":"E-mini Russell 2000","sbf":"Sugar #11","sif":"Silver","silf":"Micro Silver","ymf":"Mini Dow Jones
           # "zcf":"Corn","zff":"Five-Year US Treasury Note","zlf":"Soybean Oil Futures","zmf":"Soybean Meal","znf":"10-Year T-Note","zof":"O
           # "zsf":"Soybean","ztf":"2-Year T-Note"} #QQ, SPY , TNX, VIX

sym, names = np.array(sorted(symbol_dict.items())).T

for i in sym:    #build all symbol csvs, will populate/appear in your binder. Use linux for efficient dp
    ifs(i)

quotes = []
lens = []
```

```
for symbol in sym:
    symbol = symbol.upper()
    t = pd.read_csv(symbol)
    lens.append(t.shape[0])
mm = np.amin(lens)-1
print("min length of data: ",mm)

for symbol in sym:
    symbol = symbol.upper()
    t = pd.read_csv(symbol)
    t= t.truncate(after=mm)
    quotes.append(t)
mi = np.vstack([q["Close"] for q in quotes]) #min
ma = np.vstack([q["Open"] for q in quotes]) #max

volatility = ma - mi
```

```
AAPL
[*********************100%***********************]  1 of 1 completed
AMZN
[*********************100%***********************]  1 of 1 completed
CSCO
[*********************100%***********************]  1 of 1 completed
DELL
[*********************100%***********************]  1 of 1 completed
GOOG
[*********************100%***********************]  1 of 1 completed
HPQ
[*********************100%***********************]  1 of 1 completed
IBM
[*********************100%***********************]  1 of 1 completed
INTC
[*********************100%***********************]  1 of 1 completed
META
[*********************100%***********************]  1 of 1 completed
TSLA
[*********************100%***********************]  1 of 1 completed
min length of data:  364
```

Data Format

After downloading this massive store of data, you should click on a file, in your project. Using the file browser, you will see a large quantity of new files.

When you open one, you will see the rows of new data.

Cross Validate for Optimal Parameters: the Lasso

Varoquaux's pipeline involves steps in the following two cells.

A set of clusters is built using a set of predefined edges, called the edge model. The volatility of every OHLC tick is fed into the edge model, in order to establish every commodity's covariance to eachother.

The advantages of the Graphical Lasso model is that a cross validated average set of hyperparameters is located, then applied to cluster each commodity. Thus, every commodity is identified with other commodities which move in tandem, together, over seven days. I print the alpha edges below, and visualize this group.

Depending upon the markets when you run this study, more intensive clustering may take place at either end of the spectrum. This exposes the covariance between different groups, while exposing outlier clusters.

Using the Interactive Graph

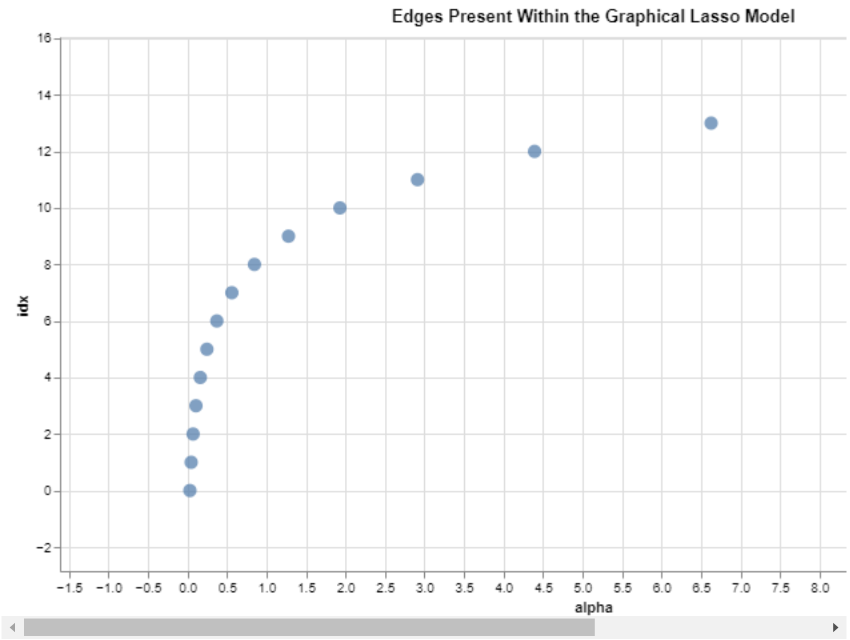
Feel free to move your mouse into the graph, then roll your mouse. This will drill in/out and allow you to hover over data points. They will mape to the edges of the clusters, under investigation.

```
from sklearn import covariance
import altair as alt
alphas = np.logspace(-1.5, 1, num=15)
edge_model = covariance.GraphicalLassoCV(alphas=alphas)
X = volatility.copy().T
X /= X.std(axis=0)
l =edge_model.fit(X)
n= []
print(type(l.alphas))
for  i in range(len(l.alphas)):
```

```
        print(l.alphas[i])
        dict = {"idx":i , "alpha":l.alphas[i]}
        n.append(dict)

dd = pd.DataFrame(n)
alt.Chart(dd).mark_point(filled=True, size=100).encode(
    y=alt.Y('idx'),
    x=alt.X('alpha'),tooltip=['alpha'],).properties(
        width=800,
        height=400,
        title="Edges Present Within the Graphical Lasso Model"
    ).interactive()
```

```
<class 'numpy.ndarray'>
0.03162277660168379
0.047705826961439296
0.07196856730011521
0.10857111194022041
0.16378937069540642
0.2470911227985605
0.372759372031494
0.5623413251903491
0.8483428982440722
1.279802213997954
1.9306977288832505
2.9126326549087382
4.39397056076079
6.628703161826448
10.0
```



Definining cluster Membership, by Covariant Affinity

Clusters of covariant, affine moving commodities are established. This group is then passed into a dataframe so that the buckets of symbols can become visible.

```
from sklearn import cluster
                                                    #each symbol, at index, is labeled with a cluster id:
_, labels = cluster.affinity_propagation(edge_model.covariance_, random_state=0)
n_labels = labels.max()                             #integer limit to list of clusters ids
# print("names: ",names,"  symbols: ",sym)
gdf = pd.DataFrame()
for i in range(n_labels + 1):
    print(f"Cluster {i + 1}: {', '.join(np.array(sym)[labels == i])}")
    l = np.array(sym)[labels == i]
    ss = np.array(names)[labels == i]
    dict = {"cluster":(i+1), "symbols":l, "size":len(l), "names":ss}
    gdf = gdf.append(dict, ignore_index=True, sort=True)

gdf.head(15)
```

```
Cluster 1: AAPL, AMZN, GOOG, META, TSLA
Cluster 2: CSCO
Cluster 3: DELL, HPQ, INTC
Cluster 4: IBM
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is depre
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is depre
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is depre
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is depre
  gdf = gdf.append(dict, ignore_index=True, sort=True)
```

	cluster	names	size	symbols
0	1	[Apple, Amazon, Alphabet, Meta Platforms, Tesla]	5	[AAPL, AMZN, GOOG, META, TSLA]
1	2	[Cisco Systems]	1	[CSCO]
2	3	[Dell Technologies, HP Inc, Intel]	3	[DELL, HPQ, INTC]

Visualizing cluster and affine commodities, by volatility

The interactive graphic requires the user to hover over each dot, in teh scatter chart. The size of the commodity cluster pushes it to the top, where the user can study the members, whose prices move in covariant fashion.

I have experimented with laying the text of the commodity group over the dots, but I find that the above table is most helpful, in identifying markets which move in tandem, and with similar price graphs. Also, as groups expand and contract, overlaying text on the chart below may prevent certain clusters from appearing. I appreciate spacing them out, and not congesting the chart.

The user is free to study where his or her chosen commodity may sit, in close relation to other globally relevant commodities.

```
for i in gdf['cluster']:
    print("cluster ",i)
    d = gdf[gdf['cluster'].eq(i)]
    for j in d.names:
        print(j, ", ")
```

```
cluster  1
['Apple' 'Amazon' 'Alphabet' 'Meta Platforms' 'Tesla'] ,
cluster  2
['Cisco Systems'] ,
cluster  3
['Dell Technologies' 'HP Inc' 'Intel'] ,
cluster  4
['IBM Corp'] ,
```

```
import altair as alt
def runCluster():
    c = alt.Chart(gdf).mark_circle(size=60).encode(
        x= alt.X('cluster:N'),
        y= alt.Y('size:Q'),
        color='size:Q',
        tooltip=['names'],
        size=alt.Size('size:Q')
    ).properties(
        width=800,
        height=400,
        title="40 Top Global Commodities, Clustered by Affine Covariance"
    ).interactive()
    #.configure_title("40 Top Global Commodities, Clustered by Affine Covariance")

    chart =c
    return chart
runCluster()
```

Double-click (or enter) to edit

References

1. Gael Varoquaux. Visualizing the Stock Market Structure. Scikit-Learn documentation pages, https://scikit-learn.org/stable/auto_examples/applications/plot_stock_market.html
2. Ran Aroussi. YFinance API documents. https://github.com/ranaroussi/yfinance
3. The Altair Charting Toolkit. https://altair-viz.github.io/index.html

```
!pip install plotly
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.13.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (8.2.2)
```

```
import plotly.graph_objects as go

import pandas as pd
from datetime import datetime

df = pd.read_csv('AMZN')
```

```
df.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume'], dtype='object')
```

```
df.head()
```

	Date	Open	High	Low	Close	Volume
0	2021-12-07	174.600006	177.499496	173.334503	176.164505	66410000
1	2021-12-08	176.150497	177.179993	174.750504	176.158005	45254000
2	2021-12-09	175.750000	176.969498	174.139496	174.171005	46062000
3	2021-12-10	175.417007	175.927002	170.500000	172.212006	60690000
4	2021-12-13	172.000000	172.100006	169.130005	169.567505	62170000

```
fig = go.Figure(data=[go.Candlestick(x=df['Date'],
                open=df['Open'],
                high=df['High'],
                low=df['Low'],
                close=df['Close'])])

fig.show()
```


```
import plotly.express as px

#df2 = px.data.stocks()
fig = px.line(df, x='Date', y="Close") # Contains AMZN daily price series
fig.show()
```


```
df2.columns
```

```
Index(['date', 'GOOG', 'AAPL', 'AMZN', 'FB', 'NFLX', 'MSFT'], dtype='object')
```

Plotting the Clustered Commodities

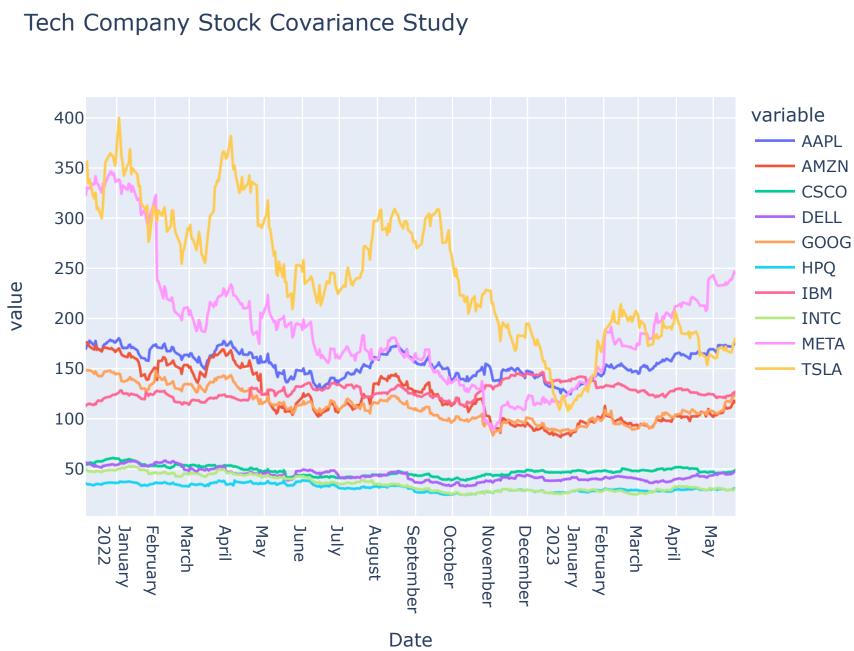
```
def getDateColumn():
  df = pd.read_csv('AMZN')
  return df['Date']


symUpper = [x.upper() for x in sym]
gdf = pd.DataFrame(columns=symUpper)
gdf['Date'] =getDateColumn()
for i in range(len(symUpper)):
  df_x = pd.read_csv(symUpper[i])
  gdf[symUpper[i]] = df_x['Close']
print(gdf.head(3))
```

```
         AAPL        AMZN       CSCO       DELL        GOOG        HPQ  \
0  169.698044  176.164505  55.449554  56.542660  148.036499  35.808460
1  173.564301  176.158005  54.437561  56.053158  148.720505  35.251507
2  173.048782  174.171005  54.943558  55.333298  148.106003  34.713753

          IBM       INTC        META        TSLA        Date
0  112.890305  49.750713  322.809998  350.583344  2021-12-07
1  114.227379  48.974689  330.559998  356.320007  2021-12-08
2  114.738075  47.772800  329.820007  334.600006  2021-12-09
```

```
fig = px.line(gdf, x="Date", y=gdf.columns,
              hover_data={"Date": "|%B %d, %Y"},
              title='Tech Company Stock Covariance Study')
fig.update_xaxes(
    dtick="M1",
    tickformat="%B\n%Y")
fig.show()
```



✓ 0s completed at 9:40 PM