Locating Novel Digital Stocks Within a Cluster-Driven Model for Global Stocks

The economy is very unpredictable right now. Everyone talks about a reccesion and there is many evidence of that. However, there is strong evidence that it may not be in the near future. By clustering stocks and seeing their relationship with one another, we are able to make a strong prediction of where they will go. I picked banking, commercial real estate, health care, and railroad stocks for this assignment. Union Pacific is the back bone of America. They transport goods from shore to shore, and the country does not function without Union Pacific. Their performace is a good indication of where the country is headed. Banking and Commerical real estate simultaneously follow each other. Just like real estate follows after stocks. Commerical real estate is a big sector that has may types of products. Such as offices, apartments, industrial, and hotels to name a few. Currently health care leases are prospering so I added healh care stocks as well. However, when there is a recession, Health care stocks are considered a defensive hedge agaisnt inflation. They have shown to provide stable earnings during a recession.

Why Cluster Commodities, to Study Stocks?

By clustering companies together, we are able to determine their correlations with one another This is extremely important because it gives us a better understanding of how different sectors of the economy correlate with each other. By having this information we can make better investments and build a diversified portfolio. It also allows us to combine like compaanies into a group so we can focus on each group rather than trying to decide on indivdual stock. We are able to see how the sector as a whole is performing.

Using Cluster Matrices to Study Covariant, Affine Price Behaviors between Stocks and Other Flows

This study samples the recent price behavior of 10 companies, then traces the covariant, linear behavior, matrix style. Affine, or common mover groups are established, and presented interactively, for the viewer in a visual milieu.

Discussion of data pipeline used, and the subsequent data transformations needed in order to create this affine matrix, as well as the technical tools to facilitate this.

Overview of Data Science Techniques

The pipeline includes downloading data, introducing processing efficiencies, model building and cross validation, and cluster expression. I outline my steps as I take them, to arrive at a matrix of pricing which affords the following advantages.

The experiement was adapted from scikit-learn's own documentation, where the techniques were applied to the US stock market. My rendition creates several departures while adapting the advantage of Varoquaux's pipeline.[1]

1. The data ingest is fast, efficient, updateable and portable. Anyone may use this code to build a working model of US-traded commodities, and add symbols they wish to see, where I have missed them.
2. Data represent public, recently settled trades.
3. Local CPU resources are used in order to use notebook memory efficiently, and leverage local Linux resources.
4. Data remains in perpetuity for the analyst, or it may be rebuilt, using updated, daily trade series.
5. Data is built as a time series, in the OHLC format, where Opening, Closing, High and daily Low prices are located.
6. Clustering is aimed toward predictive use, where clusters can achieve whatever size is needed, to cluster affine, covariant items
7. Every commodity under consideration is measured for covariance against each other, to locate a product that trades in the same linear way

8. Sparse Inverse Covariance is the technique used to identify relationships between every item in the Matrix, and thus explose clusters of products, trading similarly. This is a list of connected items, trading conditionally upon the others.Thus the list is a useable, probable list of items which trade in the same way, over a week of US business.
9. An edge model exposes the borders for classification, and locates clusters at its discretion. Thus, no supervised limits are imposed in cluster formation.
10. Hyperparameters are determined via search with a predetermined number of folds, where each subset is used to locate model parameters, which are averaged at the close of the run.
11. Given the large volume of colinear features, a cross validation technique is used to 'lasso' model features.

Building the Data Science Environment for Linux and Python

Use the following commands to interface with your underlying linux environment. These may not need to be commented out, but will remain necessary each time a new kernel boot, in your notebook, takes place.

```
!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) (2.0.12)
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) (2.21)
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) (1.16.0)
```

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 = {"JPM":"JPMorgan", "BAC":"Bank of America", "GS":"Goldman Sachs", "TDOC":"Teladoc", "CVS":"CVS Health", "CAH":"Cardinal Health", "UNP":"Union Pacific" } #QQ, SP


'''
"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":"Palladium","plf":"Chicago Ethanol (Platt
            "rbf":"RBOB Gasoline","rtyf":"E-mini Russell 2000","sbf":"Sugar #11","sif":"Silver","silf":"Micro Silver","ymf":"Mini Dow Jones Indus","zbf":"U.S. Treasury Bond F
            "zcf":"Corn","zff":"Five-Year US Treasury Note","zlf":"Soybean Oil Futures","zmf":"Soybean Meal","znf":"10-Year T-Note","zof":"Oat Futures","zrf":"Rough Rice",
            "zsf":"Soybean","ztf":"2-Year T-Note"
'''

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
```

```
BAC
[*********************100%***********************]  1 of 1 completed
CAH
[*********************100%***********************]  1 of 1 completed
CVS
[*********************100%***********************]  1 of 1 completed
GS
[*********************100%***********************]  1 of 1 completed
JPM
[*********************100%***********************]  1 of 1 completed
TDOC
[*********************100%***********************]  1 of 1 completed
```

```
UNP
[*********************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

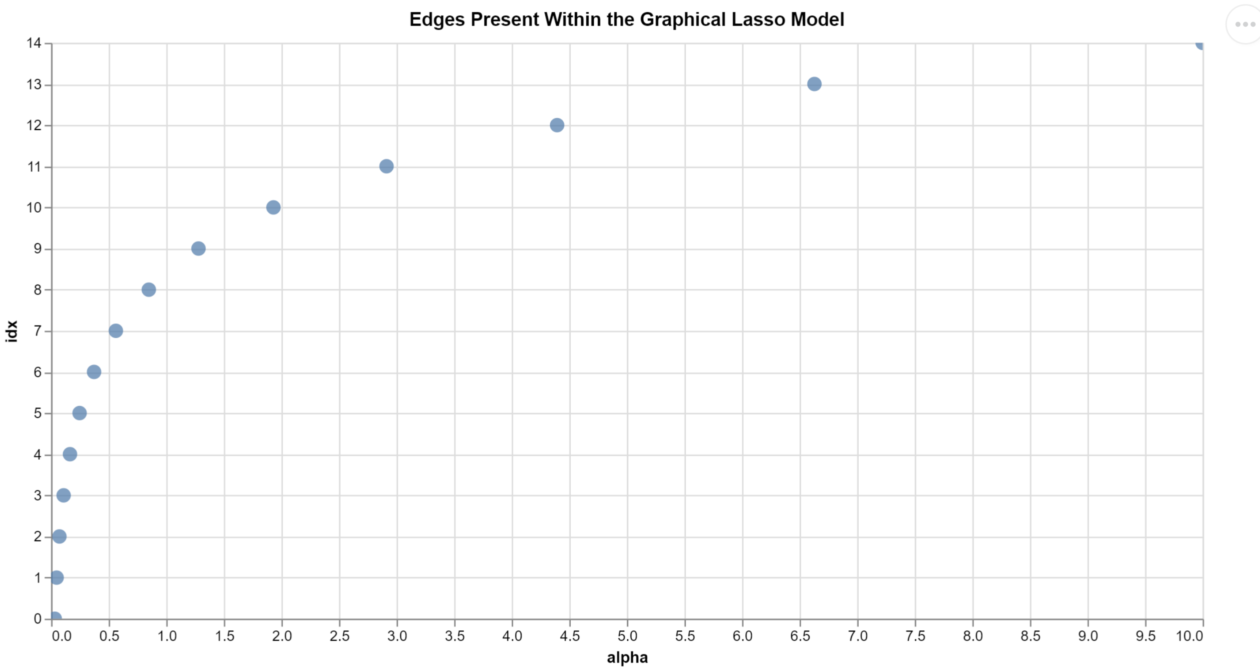
Feel free to move your mouse into the graph, then roll your mouse. This will drill in/out and allow you to hover over data points. They will mape to the edges of the clusters, under investigation.

```
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
```



Defining 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: BAC, GS, JPM, UNP
Cluster 2: CAH, CVS
Cluster 3: TDOC
<ipython-input-34-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be removed from panda
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-34-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be removed from panda
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-34-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be removed from panda
  gdf = gdf.append(dict, ignore_index=True, sort=True)
```

	cluster	names	size	symbols
0	1	[Bank of America, Goldman Sachs, JPMorgan, Uni...	4	[BAC, GS, JPM, UNP]
1	2	[Cardinal Health, CVS Health]	2	[CAH, CVS]

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.

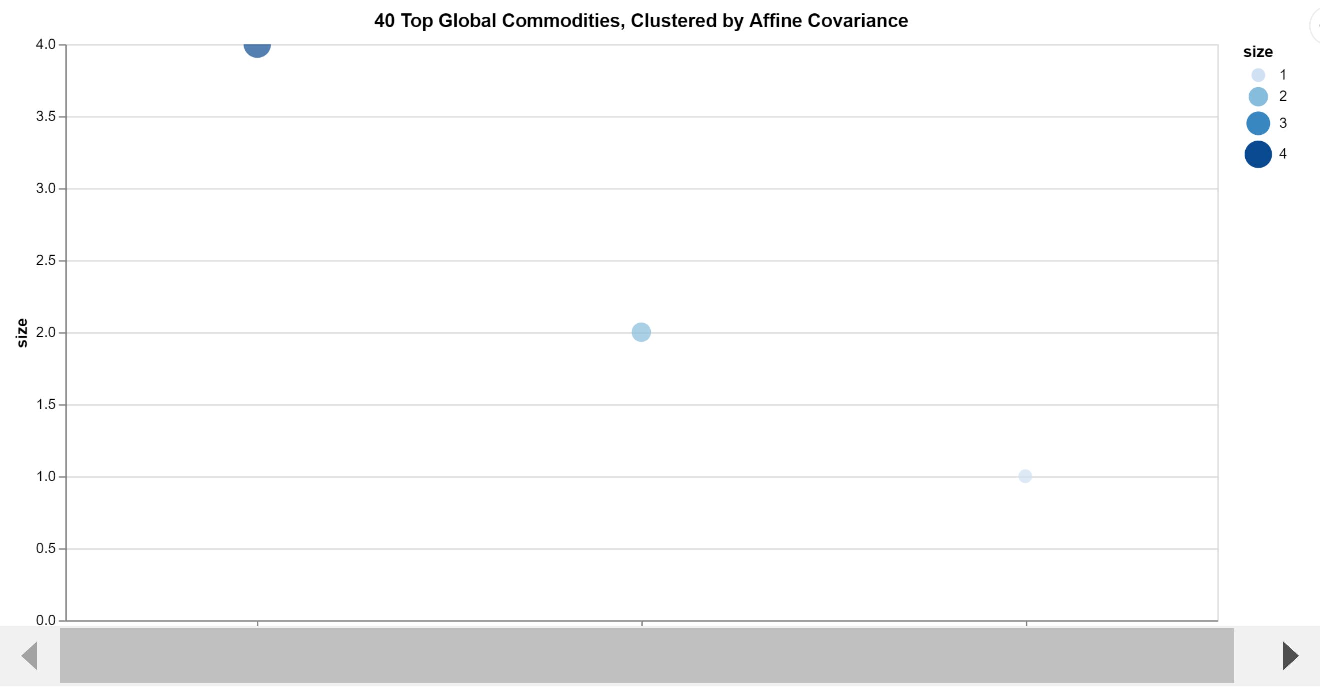
```
for i in gdf['cluster']:
    print("cluster ",i)
    d = gdf[gdf['cluster'].eq(i)]
    for j in d.names:
        print(j, ", ")
```

```
cluster  1
['Bank of America' 'Goldman Sachs' 'JPMorgan' 'Union Pacific'] ,
cluster  2
```

```
['Cardinal Health' 'CVS Health'] ,
cluster  3
['Teladoc'] ,
```

```python
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_symbol = pd.read_csv('GS')    #no .csv
```

```
df_symbol.columns
```

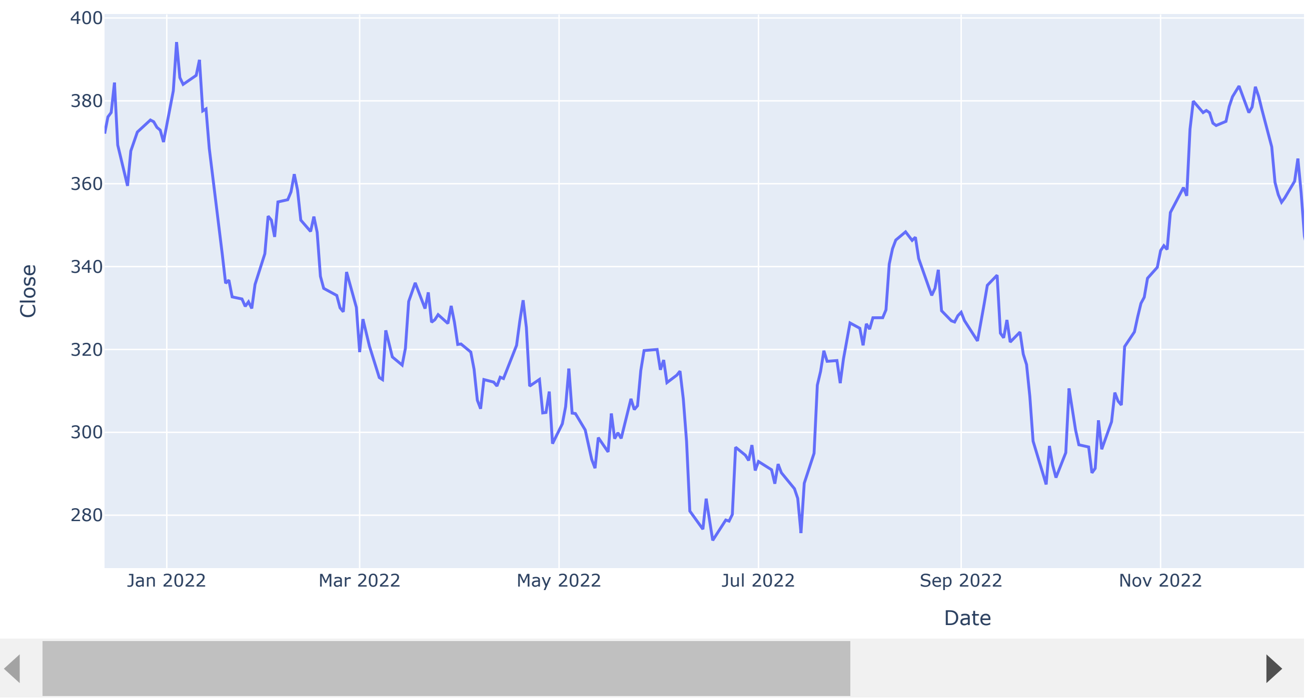
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume'], dtype='object')
```

```
df_symbol.head(2)
```

	Date	Open	High	Low	Close	Volume
0	2021-12-13	378.424906	378.424906	370.173253	372.088654	2313300
1	2021-12-14	371.043895	379.692173	370.753696	376.132263	2788500

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


```
# Using plotly.express
import plotly.express as px
fig = px.line(df_symbol, x='Date', y="Close")
fig.show()
```




```
df_symbol.columns
```

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

```
df_symbol.head(15)
```

	Date	Open	High	Low	Close	Volume
0	2021-12-13	378.424906	378.424906	370.173253	372.088654	2313300
1	2021-12-14	371.043895	379.692173	370.753696	376.132263	2788500
2	2021-12-15	376.790051	378.628046	368.180477	377.186676	2802300
3	2021-12-16	382.236333	386.792630	379.014979	384.403229	3578200
4	2021-12-17	376.509544	380.824030	368.209534	369.341339	7058600
5	2021-12-20	363.266240	363.372666	354.685686	359.483826	3735600
6	2021-12-21	364.146529	369.621839	362.531040	367.909607	2942800
7	2021-12-22	368.006356	371.479206	366.893859	369.854004	1487900
8	2021-12-23	371.266406	374.961731	371.053583	372.475616	1624000
9	2021-12-27	374.361961	376.412778	372.436907	375.377716	1430400
10	2021-12-28	376.238665	378.560343	373.636441	374.961731	1528200
11	2021-12-29	375.309980	376.751349	371.875833	373.597748	1327800
12	2021-12-30	375.097153	377.060910	372.630375	372.939911	1160100
13	2021-12-31	372.436929	374.371666	368.412672	370.066864	1601300
14	2022-01-03	376.306379	386.270264	374.700544	382.429810	3334300

Plotting the Clustered Commodities

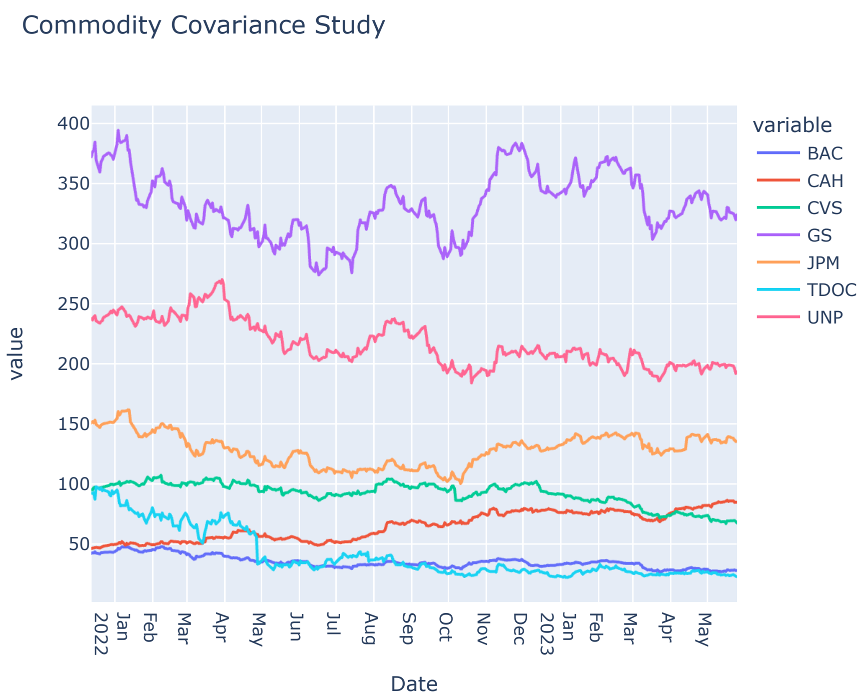
```
#generate a Date column in gdf
def getDateColumn():
  df = pd.read_csv('GS')
  return df['Date']  #pandas series
```

```
symUpper = [x.upper() for x in sym] #make all symbols in sym to uppercase
# print(symUpper)
gdf = pd.DataFrame(columns=symUpper) #form a new global dataframe, gdf, for purpose of graphing
gdf['Date'] = getDateColumn()          #get a common index for dates, for every commodity or equity
for i in range(len(symUpper)):         #iterate the length of the uppercase symbols
  df_x = pd.read_csv( symUpper[i])     #create one dataframe to hold the csv contents
  gdf[symUpper[i]] = df_x['Close']     #extract the price series from the 'Closed' column
print(gdf.head(3))                     #print the resulting top three rows from the new gdf
# print(gdf.columns)
```

```
         BAC        CAH        CVS          GS         JPM       TDOC  \
0  42.313931  46.186237  95.270760  372.088654  150.782211  92.540001
1  42.847950  46.653446  95.155167  376.132263  151.937531  92.169998
2  42.663464  46.920414  96.860214  377.186676  150.801300  92.940002

          UNP        Date
0  237.521729  2021-12-13
1  236.429550  2021-12-14
2  238.855576  2021-12-15
```

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



0s completed at 8:41 PM