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

With massive, recent interest in institutional investment in digital commodities, ie cryptocurrencies, US and other regulatory commissions effectively classify such assets as commodities. Given that these risk assets are typically priced in tandem with stock equity, and contrasted against US Treasury instruments, little scholarship has analyzed cryptocurrencies and digital assets as effective commodities, such as Sugar, Timber, Oil products or Grains.

Seeing Bitcoin as a necessary commodity to participate in cross border money exchange, ecommerce, or oil purchasing is necessary to justify considering it as a commodity, rather than a risk asset. For those who analye cryptocurrency as a holding, and analyze it via other valuation methods typically finds the exercise wanting, as valuation tends to look for underlying, fundamental value. The use case, also for Bitcoin and other digital commodities also leaves the analyst to wonder whether they are investing in Ponzi goods; Bitcoin is used to purchase hotel rooms, and at times, yachts or pizza slices, but it remains a held-good such as Gold.

Why Cluster Commodities, to Study Bitcoin (or Hogs)?

When digial commodities are analyzed alongside Oats, Gold, E-Mini Futures and other classical commodities, their prices covariance, against a pool of commodities can be tracked. Unifying digital commodities within pools of other commonly traded daily commodities allows another category of analysis to emerge, where traders simply shift from one commodity to another, as economic winds change, or opportunities simply justify a change of trading venue, ie a trend-shift toward energy away from equity, and we have seen since the start of a hot war in Ukraine.

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

This study samples the recent price behavior of 37 commodities, 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.

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

```
    Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.32)
    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) (1.23.5)
    Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.31.0)
    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) (4.9.3)
```

```
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) (2023.3.post1)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.3.10)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packages (from yfinance) (3.17.0)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.11.
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->y
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (1.16
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->y
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfin
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfin
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_dat
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->vega_datasets) (20
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas->vega_datasets) (1
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas
```

The above code loads in a python script that allows data from Yahoo Finance to be inported into the note book. This is neccasary to complete the project because it is where all of the data we use is taken from. It is really long and I honestly don't understand how it works, but I understand it's function.

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 = "500d",
        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
```

This code actually used the downloaded pip and actually imports the data. It also sets the code for the tickers and allows any ticker from Yahoo finance to be used. You have to run it everytime you open the notebook to redownload the data from Yahoo.

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].

```
symbol_dict = {"dis":"Disney", "amd":"AMD", "aapl":"Apple","nvda":"NVIDIA","amzn":"Amazon", "ea":"EA","para":"Paramount",
            "siri":"Sirius XM","goog":"Alphabet"} #QQ, SPY , TNX, VIX

# symbol_dict = {"AAL":"American Airlines", "DAL":"Delta Airlines", "BTCF":"Bitcoin Futures"}

# symbol_dict  ={"AVAX-USD":"Avalanche", "BTC-USD":"Bitcoin","znf":"US treasury 10yr", "APPL":"Apple"}
```

This square is where you decide the companies you will be using. Professor you may have noticed that I changed my companies to 10 'entertainment' companies. My project 9 it was just popular companies, but I wanted to dig deeper on a specifc field, like the project was inteded for. This is used for the code below to identify the different tickers.

```
#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


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
AMD
[*********************100%%**********************]  1 of 1 completed
AMZN
[*********************100%%**********************]  1 of 1 completed
DIS
[*********************100%%**********************]  1 of 1 completed
```

```
EA
[*********************100%%**********************]  1 of 1 completed
GOOG
[*********************100%%**********************]  1 of 1 completed
NVDA
[*********************100%%**********************]  1 of 1 completed
PARA
[*********************100%%**********************]  1 of 1 completed
SIRI
[*********************100%%**********************]  1 of 1 completed
min length of data:  499
```

This code uses the ticker data and actually pulls the data from Yahoo Finance and downloads it directly into the notebook. It also tells you when the data download is completed with a percent. That was really helpful because I was originally using data that contained an 'F' which broke the code and the percent display allowed me to identify the issue.

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.

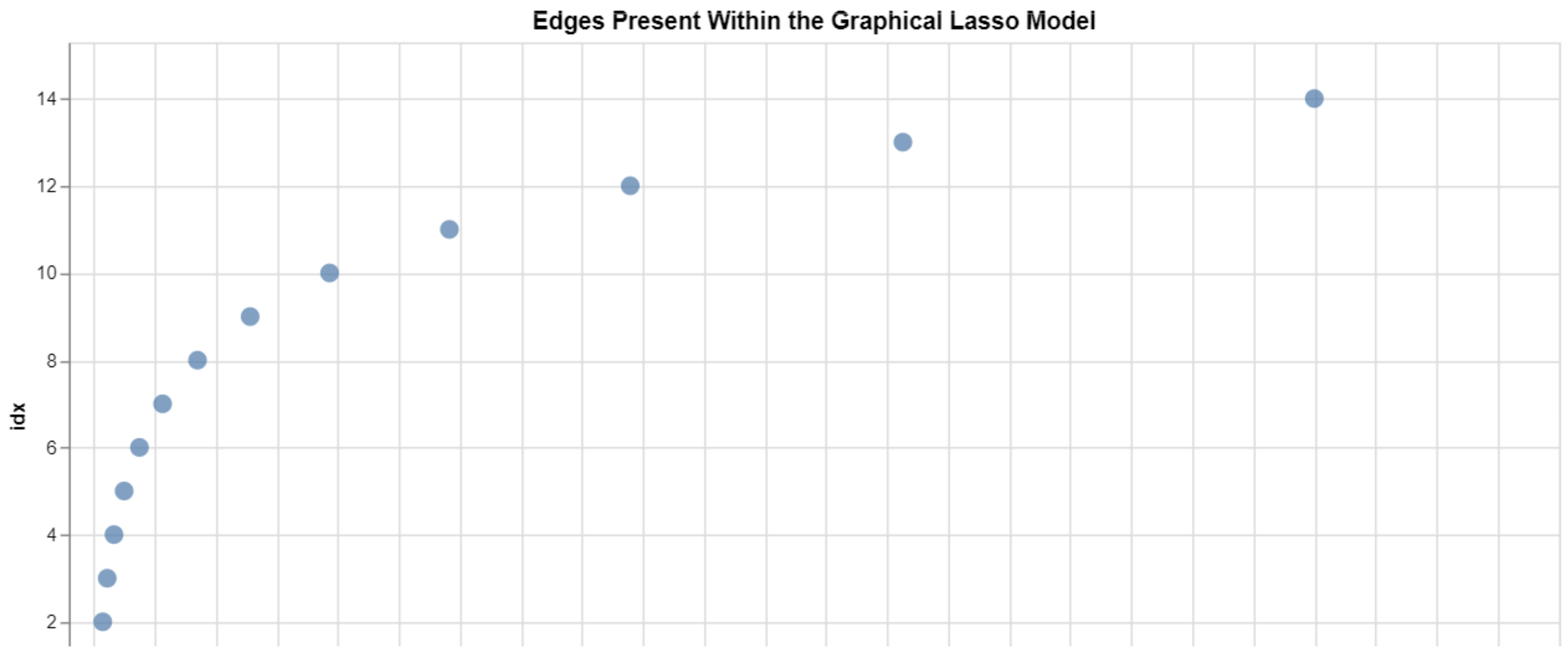
Using the Interactive Graph

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
```



This code took the predownladed data (sample data folder) and created a cluster diagram from the covariences of the imported data.

Defininig 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, amd, amzn, dis, goog, nvda
Cluster 2: ea
Cluster 3: para
Cluster 4: siri
<ipython-input-41-716215b636ca>:12: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

<ipython-input-41-716215b636ca>:12: FutureWarning:
```

This code takes the downloaded data from Yahoo Finance, analyzes it's coefficients, and then organizes them in 4 clusters based of a range of values set by the code. Then it displays those clusters that display the names and symbols.

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.

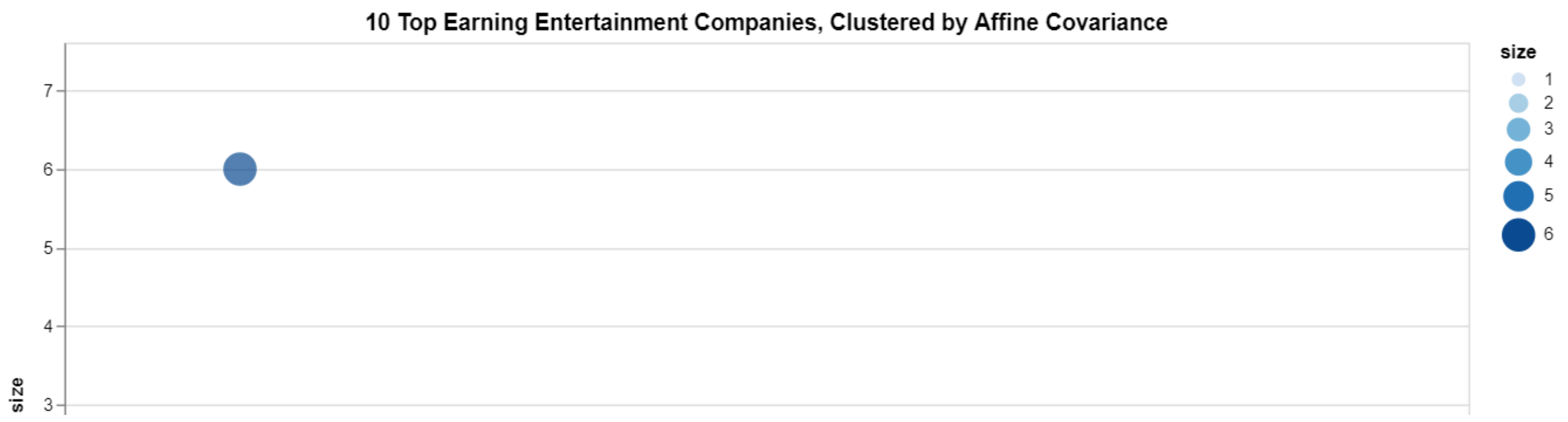
```
for i in gdf['cluster']:
    print("cluster ",i)
    d = gdf[gdf['cluster'].eq(i)]
    for j in d.names:
        print(j, ", ")
```

```
cluster  1
['Apple' 'AMD' 'Amazon' 'Disney' 'Alphabet' 'NVIDIA'] ,
cluster  2
['EA'] ,
cluster  3
['Paramount'] ,
cluster  4
['Sirius XM'] ,
```

The above code classifies the clusters using the name instead of the ticker symbol in a raw data format.

```
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="10 Top Earning Entertainment Companies, Clustered by Affine Covariance"
    ).interactive()
    #.configure_title("40 Top Global Commodities, Clustered by Affine Covariance")

    chart =c
    return chart
runCluster()
```



The above code takes all of the cluster data created from the last 2 blocks and actually creates the cluster chart. It defines the different sizes of circles and their meaning and the color associated with the size. It also defines the size of the graph.

My analysis: I think it's clear from the cluster chart that entertainment companies tend to move together. The main outlier was Sirius XM, which is mainly known for it's radio service. The reason I included them as an entertainment company is because they also offer a streaming service for sports and some local channels. It's hard for them to compete with the big dogs, like Apple, so it makes sense that their prices aren't directly correlated.

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
```

```
    Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15.0)
    Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (8.2.3)
    Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly) (23.2)
```

This code installs the plotly code necessary for creating the graphs.

```
import plotly.graph_objects as go
import pandas as pd
from datetime import datetime

df_symbol = pd.read_csv('DIS')    #no .csv
```

This runs the code imports the downloaded code from above and allows python to read it. It also defines where the df_symbol (the chart) is getting it's data from. I chose Disney because I am interested in the companies growth (or decline) from the past years of having (arguably) poorly perfoming movies.

```
df_symbol.columns
```

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

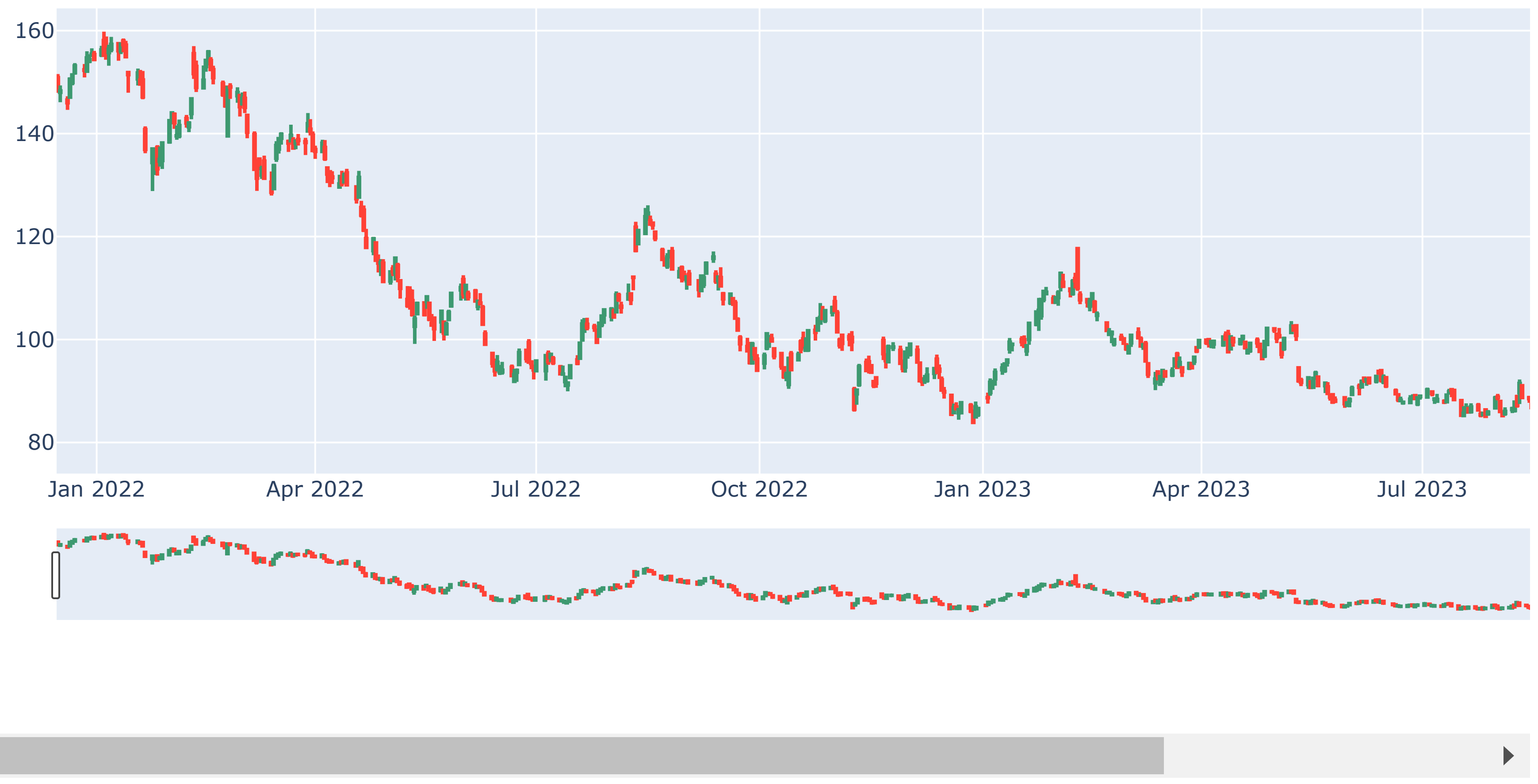
This code assigns names to the columns of the data to reflect each columns respective data.

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

	Date	Open	High	Low	Close	Volume
0	2021-12-16	150.858345	151.566030	148.097339	148.266785	11145600
1	2021-12-17	148.027561	149.353242	146.093876	148.276749	13785000

This code gives the rows names. After this code is run, the chart is displayed with all the columns and rows labeled properly.

```
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()
```

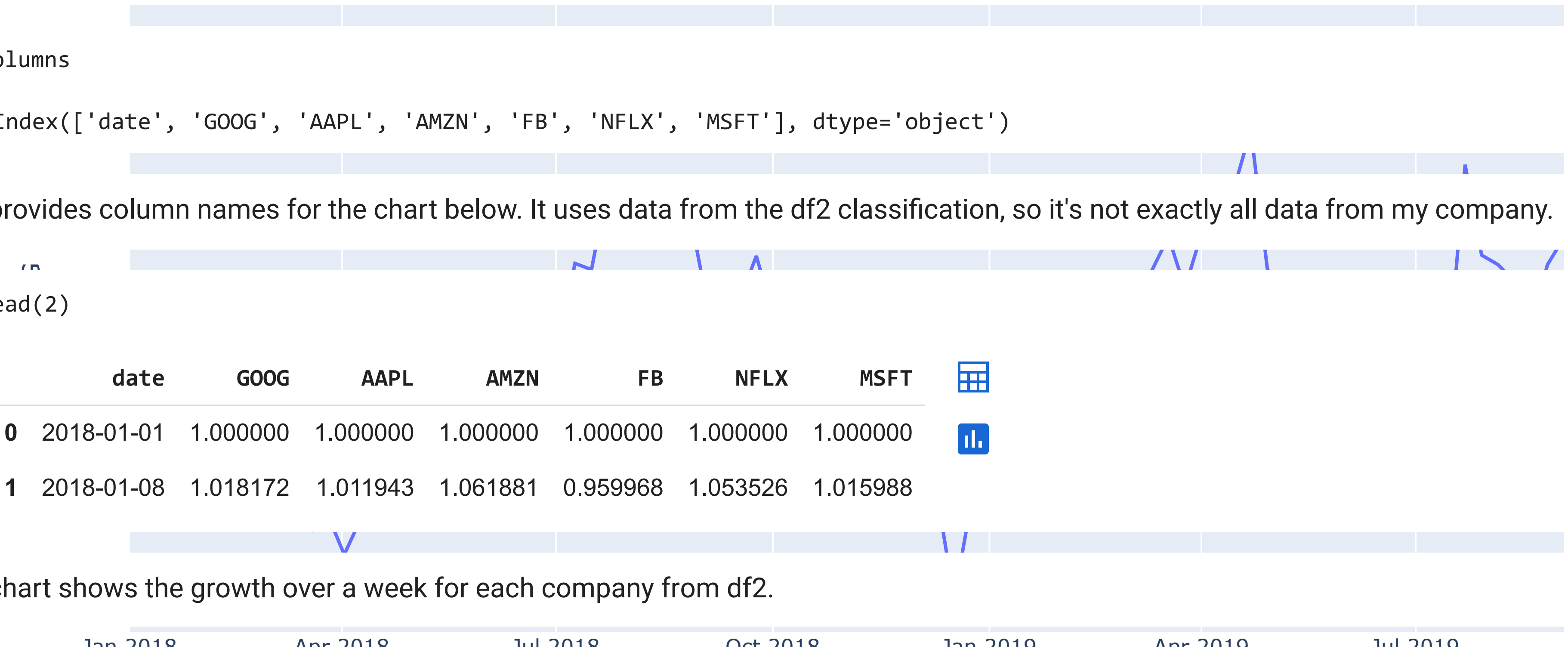


This takes the data from df_sybmol (Disney's data) and makes it into a candle stick chart. It shows the opening and closing prices from January 2023, to today December 12th 2023. You can also adjust the window. From this chart it's very easy to see that Disney's stock has significantly dropped over the past year. This chart is what option traders use to predict market activity and movement to make purchses throught the open hours of the stock exhange.

```
# Using plotly.express
import plotly.express as px

df2 = px.data.stocks()
fig = px.line(df2, x='date', y="GOOG")
fig.show()
```

This took data from Google and created a trendline from Jan 2018-Dec 2019. From this it's easy to identify the high/low points of the stock over this period. Its also easy to see when the stock drops its tends to come back higher and drop less the next time.



```
df2.columns
```

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

This provides column names for the chart below. It uses data from the df2 classification, so it's not exactly all data from my company.

```
df2.head(2)
```

	date	GOOG	AAPL	AMZN	FB	NFLX	MSFT
0	2018-01-01	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1	2018-01-08	1.018172	1.011943	1.061881	0.959968	1.053526	1.015988

This chart shows the growth over a week for each company from df2.

```
df2['AMZN']
```

```
0      1.000000
1      1.061881
2      1.053240
3      1.140676
4      1.163374
         ...
100    1.425061
101    1.432660
102    1.453455
103    1.521226
104    1.503360
Name: AMZN, Length: 105, dtype: float64
```

This chart shows the percentage growth of Amazon over 105 days, imported from df2

```
df_symbol.columns
```

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

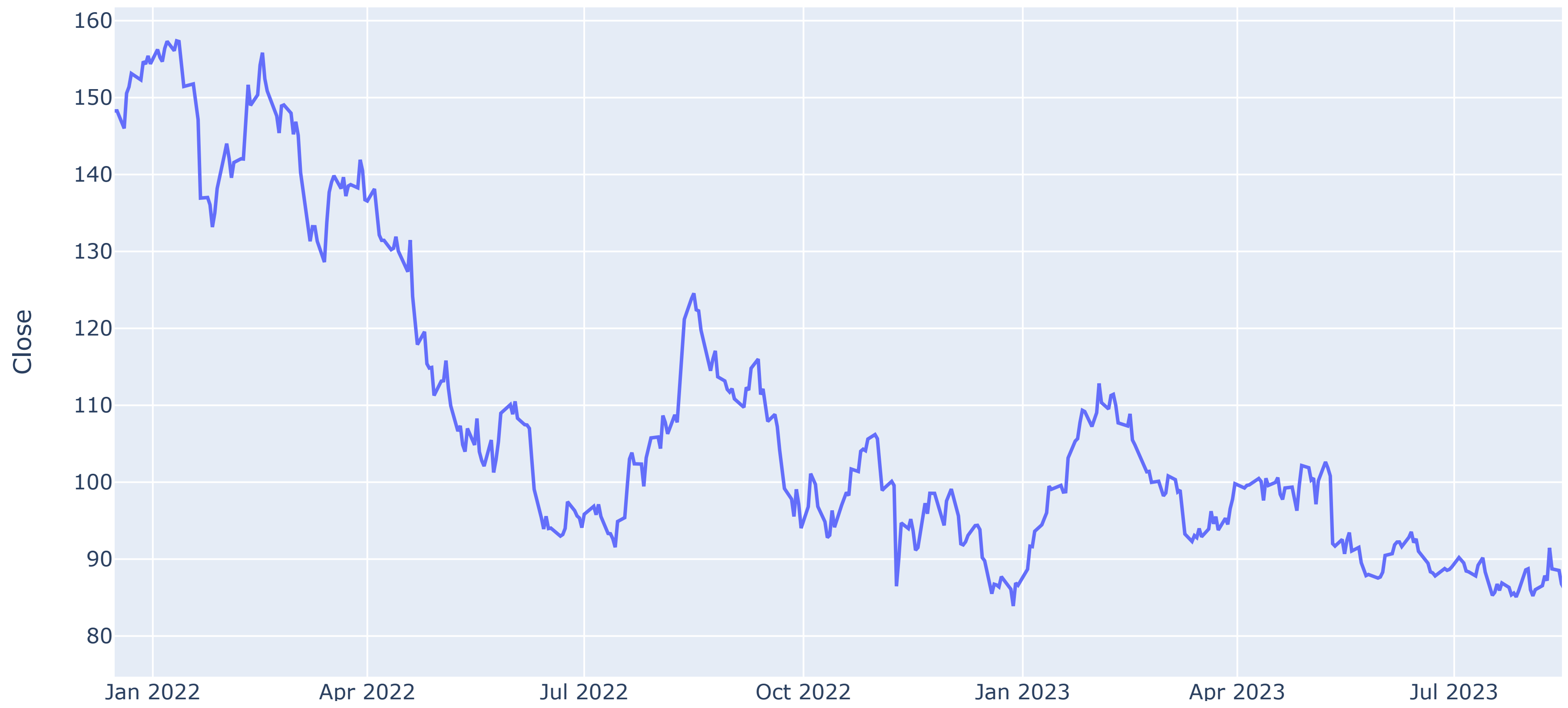
This provides names for the chart below to label which area it is focuing on.

```
df_symbol['Open']
```

```
0      150.858345
1      148.027561
2      146.502524
3      147.080643
4      149.931363
          ...
495     90.814026
496     91.501787
497     92.059998
498     92.730003
499     92.120003
Name: Open, Length: 500, dtype: float64
```

This gives you the Open price of Amazon over the last 500 days, the words on the bottom are takes from the index above. You change change Open to any word from [61] Index and it will give you the data.

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



This code downloaded another px file and shows the daily price of Bitcoin from Jan 2022 to December 2023. This chart shows how Bitcoin has tanked over the last 2 years.

Plotting the Clustered Commodities

```
#generate a Date column in gdf
def getDateColumn():
  df = pd.read_csv('PARA')  #CHOOSE an equity or vehicle for which you possess a Date index
  return df['Date']  #pandas series
```

This code generates the date column for the graph below. I also changed the date index from BTC to Paramount because I use it in my data set. This gives the graph data for which dates to use.

```
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)
```

```
         AAPL         AMD        AMZN         DIS          EA        GOOG  \
0  170.314026  138.639999  168.871002  148.266785  127.868828  144.838501
1  169.206680  137.750000  170.017502  148.276749  126.020813  142.802994
2  167.832382  135.800003  167.078995  145.994186  128.392593  142.401505

         NVDA       PARA      SIRI
0  283.479034  28.048346  5.894934
1  277.627106  27.840857  5.932126
2  276.808197  27.048637  5.904232
```

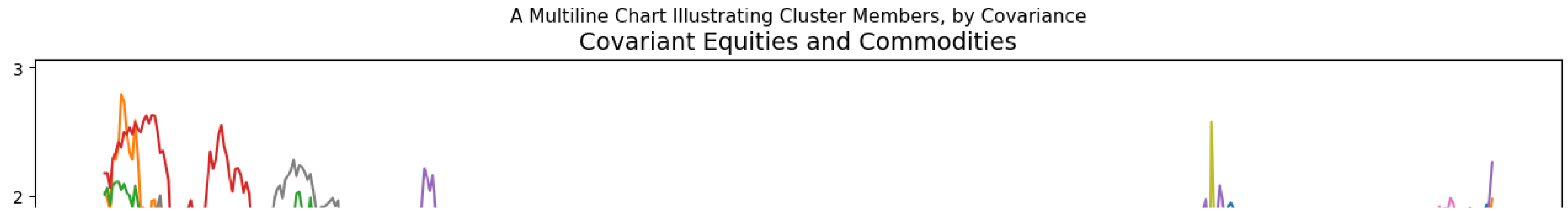
This code imports the already generated commodity clusters for the graph to use.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# scale the data
scaler = StandardScaler()
scaled_gdf = pd.DataFrame(scaler.fit_transform(gdf), columns=gdf.columns)

# plot the dataframe
fig, ax = plt.subplots(figsize=(16, 8))
scaled_gdf.plot.line(ax=ax)

# add title and subtitle
ax.set_title('Covariant Equities and Commodities', fontsize=14)
ax.text(0.5, 1.05, 'A Multiline Chart Illustrating Cluster Members, by Covariance',
        horizontalalignment='center',
        fontsize=11,
        transform=ax.transAxes)
# show the plot
plt.show()
```



This code takes all the imported data and actually presents the data in a overlapping line chart. After completing this project I actually feel like Paul Bunyan. I learned a lot from this project. I also agree with your comments about having less data makes this chart easier to read. It's not as cluttered, which is why I opted to change my data set for project 10. You can actually see the specific companies and the correlation between them.