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.

O‹ ›iew ‹›

14:32

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: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.3.7)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.11.
Requirement already satisfied: cryptography>=3.3.2 in /usr/local/lib/python3.10/dist-packages (from yfinance) (40.0.2)
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.27.1)
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.4.4)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.22.4)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.5.3)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2022.7.1)
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: 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: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography>=3.3.2->yfinanc
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: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26->yfinance)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26->yfir
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26->y
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests>=2.2
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography>=3.3
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_dat
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas->vega_datasets) (1
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->vega_datasets) (20
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas
```

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

en nvironments 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,
        th
        p
```

```
    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 = {"clf":"crude oil", "esf":"E-Mini S&P 500","btcf":"Bitcoin","bzf":"Brent Crude Oil", "ccf":"Cocoa","ctf":"Cott
           "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 Juic
            "rbf":"RBOB Gasoline","rtyf":"E-mini Russell 2000","sbf":"Sugar #11","sif":"Silver","silf":"Micro Silver","ymf":
            "zcf":"Corn","zff":"Five-Year US Treasury Note","zlf":"Soybean Oil Futures","zmf":"Soybean Meal","znf":"10-Year 
            "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()
      = p
      ns.
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
```

```
BTC=F
[*********************100%***********************]  1 of 1 completed
BZ=F
[*********************100%***********************]  1 of 1 completed
CC=F
[*********************100%***********************]  1 of 1 completed
CL=F
[*********************100%***********************]  1 of 1 completed
CT=F
[*********************100%***********************]  1 of 1 completed
ES=F
[*********************100%***********************]  1 of 1 completed
GC=F
[*********************100%***********************]  1 of 1 completed
GF=F
[*********************100%***********************]  1 of 1 completed
HE=F
[*********************100%***********************]  1 of 1 completed
HG=F
[*********************100%***********************]  1 of 1 completed
HO=F
[*********************100%***********************]  1 of 1 completed
KC=F
[*********************100%***********************]  1 of 1 completed
KE=F
 ***                                   **********]  1 of 1 completed
 S=
[***                 100%                ***********]  1 of 1 completed
```

```
LE=F
[*********************100%***********************]  1 of 1 completed
MGC=F
[*********************100%***********************]  1 of 1 completed
NG=F
[*********************100%***********************]  1 of 1 completed
NQ=F
[*********************100%***********************]  1 of 1 completed
OJ=F
[*********************100%***********************]  1 of 1 completed
PA=F
[*********************100%***********************]  1 of 1 completed
PL=F
[*********************100%***********************]  1 of 1 completed
RB=F
[*********************100%***********************]  1 of 1 completed
RTY=F
[*********************100%***********************]  1 of 1 completed
SB=F
[*********************100%***********************]  1 of 1 completed
SI=F
[*********************100%***********************]  1 of 1 completed
SIL=F
[*********************100%***********************]  1 of 1 completed
YM=F
[*********************100%***********************]  1 of 1 completed
ZB=F
[*********************100%***********************]  1 of 1 completed
ZC=F
[*********************100%***********************]  1 of 1 completed
```

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.

\ /ou v 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)
a     art                                  00).encode(
      alt
    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
```

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.

```
4.39397056076079
```

```
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: bzf, clf, ctf, hef, hgf, hof, ngf, ojf, rbf, sbf
Cluster 2: btcf, ccf, esf, nqf, rtyf, ymf
Cluster 3: gcf, kcf, mgcf
Cluster 4: gff, lbsf, lef
Cluster 5: paf, plf, sif, silf
Cluster 6: zbf, zff, znf, ztf
Cluster 7: kef, zcf, zlf, zmf, zof, zrf, zsf
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be rem
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be rem
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be rem
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be rem
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be rem
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be rem
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-5-3e2cbe7f4ace>:12: FutureWarning: The frame.append method is deprecated and will be rem
  gdf = gdf.append(dict, ignore_index=True, sort=True)
```

	cluster	names	size	symbols
0	1	[Brent Crude Oil, crude oil, Cotton, Lean Hogs...	10	[bzf, clf, ctf, hef, hgf, hof, ngf, ojf, rbf, ...
1	2	[Bitcoin, Cocoa, E-Mini S&P 500, Nasdaq 100, E...	6	[btcf, ccf, esf, nqf, rtyf, ymf]
2	3	[Gold, Coffee, Micro Gold]	3	[gcf, kcf, mgcf]

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.

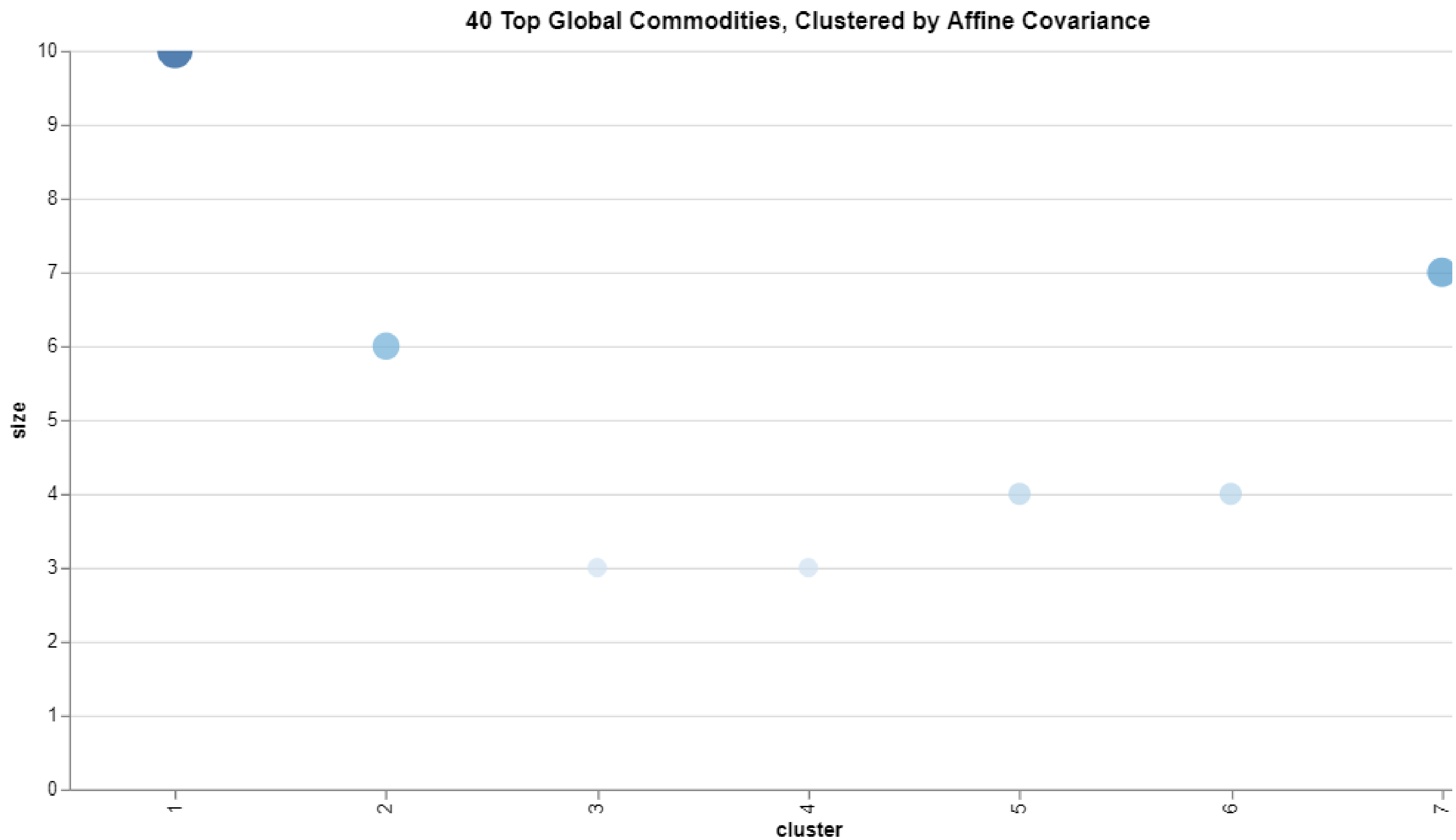
T' ser i' n commodity may sit, in close relation to other globally relevant commodities.

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

```
cluster  1
['Brent Crude Oil' 'crude oil' 'Cotton' 'Lean Hogs' 'Copper' 'Heating Oil'
 'Natural Gas' 'Orange Juice' 'RBOB Gasoline' 'Sugar #11'] ,
cluster  2
['Bitcoin' 'Cocoa' 'E-Mini S&P 500' 'Nasdaq 100' 'E-mini Russell 2000'
 'Mini Dow Jones Indus'] ,
cluster  3
['Gold' 'Coffee' 'Micro Gold'] ,
cluster  4
['Feeder Cattle' 'Lumber' 'Live Cattle'] ,
cluster  5
['Palladium' 'Chicago Ethanol (Platts)' 'Silver' 'Micro Silver'] ,
cluster  6
['U.S. Treasury Bond Futures' 'Five-Year US Treasury Note'
 '10-Year T-Note' '2-Year T-Note'] ,
cluster  7
['KC HRW Wheat' 'Corn' 'Soybean Oil Futures' 'Soybean Meal' 'Oat Futures'
 'Rough Rice' 'Soybean'] ,
```

```
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"
     nt
     con                                  ties, Clustered by Affine Covariance")
```

```
    chart =c
    return chart
runCluster()
```



Double-click (or enter) to edit

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('BTCF')    #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	2021-09-20	47415.0	47620.0	42365.0	43780.0	8330
1	2021-09-21	43410.0	43870.0	40085.0	42030.0	8523

```
fig = go.Figure(data=[go.Candlestick(x=df_symbol['Date'],
                open=df_symbol['Open'],
                high=df_symbol['High']
```

```
                close=df_symbol['Close'])])
fig.show()
```


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

df   px.d
f     px.
fi   ow()
```


```
df2.columns
```

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

```
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

```
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
```

```
df_symbol.columns
```

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

```
df_symbol['Close']
```

```
0      43780.000000
1      42030.000000
2      43385.000000
3      44785.000000
4      42359.398438
           ...
404    27900.000000
405    29690.000000
406    29177.000000
407
408
Name: Close, Length: 409, dtype: float64
```

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

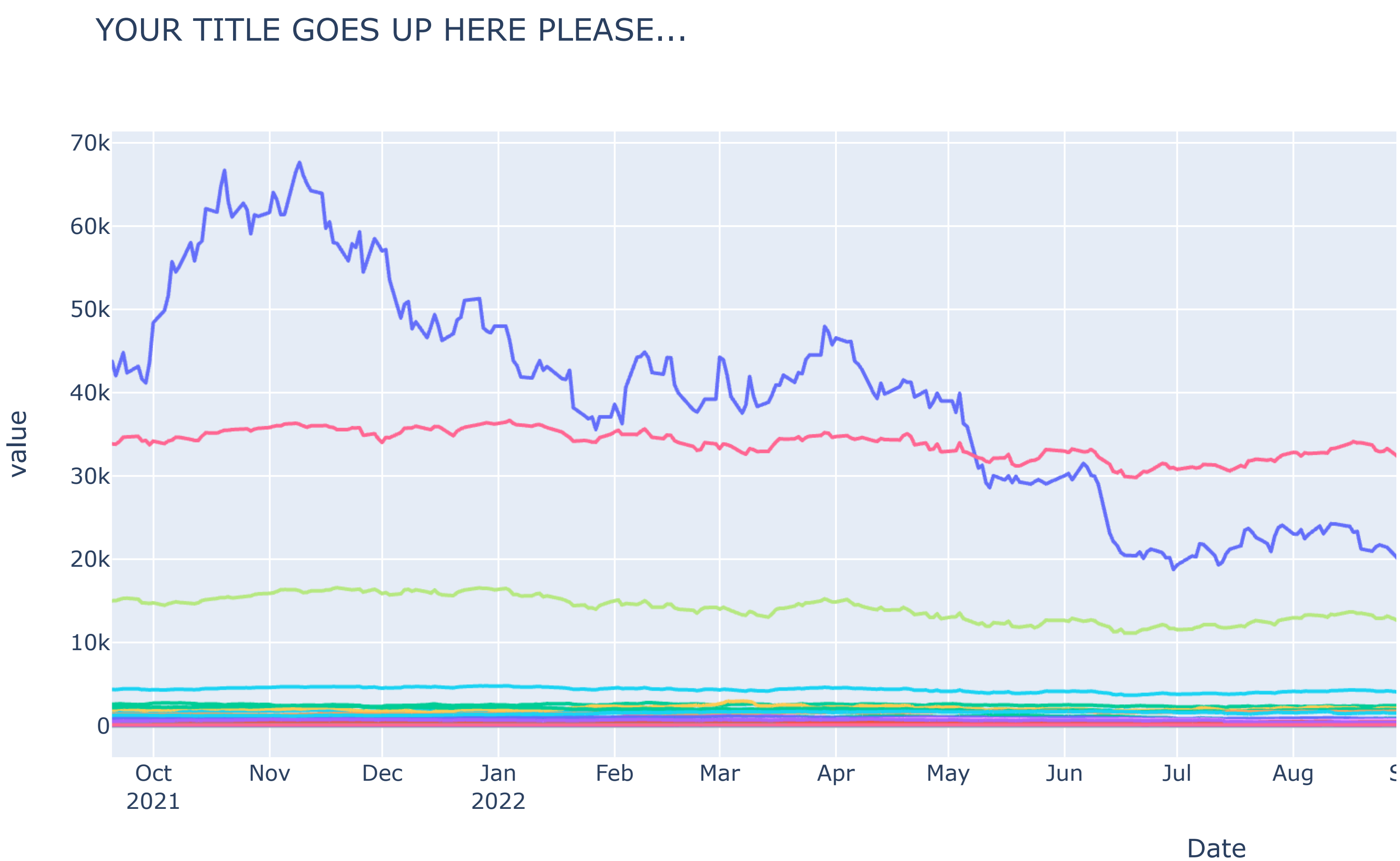
tir odities

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

```
      BTCF        BZF     CCF        CLF        CTF      ESF          GCF  \
0  43780.0  73.919998  2593.0  70.290001  89.889999  4348.25  1761.800049
1  42030.0  74.360001  2605.0  70.559998  91.180000  4343.25  1776.000000
2  43385.0  76.190002  2652.0  72.230003  91.830002  4384.00  1776.699951

          GFF        HEF     HGF  ...     ZCF         ZFF        ZLF  \
0  155.000000  84.974998  4.1170  ...  521.75  124.000000  54.910000
1  154.850006  84.375000  4.1280  ...  517.00  123.960938  55.320000
2  154.800003  83.849998  4.2525  ...  525.50  123.742188  56.389999

          ZMF         ZNF     ZOF     ZRF      ZSF         ZTF        Date
0  336.100006  133.843750  532.25  1372.5  1262.50  110.320312  2021-09-20
1  337.899994  133.859375  532.75  1385.0  1274.00  110.328125  2021-09-21
2  337.899994  132.984375  557.75  1388.5  1282.75  110.273438  2021-09-22

[3 rows x 38 columns]
```

```
fig = px.line(gdf, x="Date", y=gdf.columns,
                                                   %Y"},
                                                     PLEASE...')
fig.update_
```

```
        dtick="M1",
        tickformat="%b\n%Y")
fig.show()
```




```
type(gdf.columns)
```

```
    pandas.core.indexes.base.Index
```