Mapping the Dynamics of the Gaming Industry: A Cluster-Driven Approach

In the rapidly evolving landscape of the gaming industry, with increasing investments and focus innovative technologies, it becomes crucial to comprehend how gaming companies are categorized and valued in the global market. The gaming industry incorporates elements of physical technology, entertainment intellectual property, and interactive media, creating a unique space for analysis. This project aims to scrutinize gaming companies as integral components of a broader economic ecosystem, akin to established industries like automotive, pharmaceuticals, or consumer electronics.

The gaming industry, with its diverse range of products including video games, mobile games, and esports, along with related industries that focus on hardware and computing power. Recognizing these companies as part of a broader economic sector allows us to apply traditional financial analysis methods, such as those used for evaluating consumer goods or tech companies. This approach is vital for understanding the intrinsic value of gaming companies and their products, which range from blockbuster game titles to in-game purchases and virtual goods.

Why Group Gaming Companies for Analysis?

This analysis can help us understand market behaviors, investment potential, and economic impact. Grouping these companies together for analysis allows us to see how they react to market trends, economic changes, and technological advancements. This comparative approach sheds light on how gaming companies are influenced by and contribute to the global economy.

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

This study samples the recent price behavior of major companies that create blockbuster games, gaming software, or gaming related hardware. It then traces the covariant, linear behavior, matrix style relationships inorder to establish common mover groups. Then used inorder to visualize these trends across the market and between specfic entities.

Overview of Data Science Techniques

The process involves data collection, cleansing, and integration. We use advanced data science techniques, including machine learning and cluster analysis, to categorize companies and understand their market dynamics. The methodology is inspired by established financial analysis techniques but is tailored to address the unique characteristics of the gaming industry.

The data is processed using efficient computational methods, ensuring fast, updatable, and portable data management. The analysis focuses on creating a comprehensive view of the gaming industry, with an emphasis on time-series data, market trends, and predictive modeling. The goal is to identify clusters of gaming companies that exhibit similar market behaviors, allowing for a nuanced understanding of the industry.

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

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.33)
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) (4.11.2)
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) (2.5)
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: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2023.11.17
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) (2023.3.post1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas->vega_datasets) (1.23.5)
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

Utazling the common Yahoo Finance API, we can download data from any stock of commodities that are desrired. We use this to focus on a number of gaming industry related companies, and can alter the data through inputting new ticker symbols of the companies we wish to save.

This data is saved locally on google collab, but can be re-created with current information through re-running the code in this notebook. The data used was sourced in December 2023

```
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
```

Trigger Data Downloads

This code is used to customize the data that we take from the market, creating a list of the companies or ticker symbols that we want to investigate.

The volatility is calculated through the closing priced subtracted from the opening price.

```
symbol_dict = { "nvda":"Nvidia Corp","tcehy":"Tencent Holdings","sony":"Sony Group Corporation", "ntdoy":"Nintendo Co., Ltd.","ea":"Electror
           "rblx":"Roblox Corporation", "ttwo":"Take-Two Interactive Software, Inc. ","logi":"Logitech International S.A.","u":"Unity Softwa
           "amd":"Advanced Micro Devices, Inc","dell":" Dell Technologies ","intc":"Intel Corporation","meta":"Meta Platforms","bili": "Bili

# 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"}
```

```
#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
```

```
AMD
[*********************100%%**********************]  1 of 1 completed
BILI
[*********************100%%**********************]  1 of 1 completed
BRAG
[*********************100%%**********************]  1 of 1 completed
DELL
[*********************100%%**********************]  1 of 1 completed
EA
[*********************100%%**********************]  1 of 1 completed
GME
[*********************100%%**********************]  1 of 1 completed
INTC
[*********************100%%**********************]  1 of 1 completed
LOGI
[*********************100%%**********************]  1 of 1 completed
META
[*********************100%%**********************]  1 of 1 completed
NTDOY
[*********************100%%**********************]  1 of 1 completed
NVDA
[*********************100%%**********************]  1 of 1 completed
OTGLY
[*********************100%%**********************]  1 of 1 completed
RBLX
[*********************100%%**********************]  1 of 1 completed
SKLZ
[*********************100%%**********************]  1 of 1 completed
SONY
[*********************100%%**********************]  1 of 1 completed
TCEHY
[*********************100%%**********************]  1 of 1 completed
TTWO
[*********************100%%**********************]  1 of 1 completed
U
[*********************100%%**********************]  1 of 1 completed
WBD
[*********************100%%**********************]  1 of 1 completed
min length of data:  499
```

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: brag
Cluster 2: ea, ttwo
Cluster 3: dell, intc, logi, sony, wbd
Cluster 4: amd, meta, nvda
Cluster 5: otgly
Cluster 6: bili, gme, ntdoy, rblx, sklz, tcehy, u
<ipython-input-10-716215b636ca>:12: FutureWarning: The frame.append method is deprecatec
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-10-716215b636ca>:12: FutureWarning: The frame.append method is deprecatec
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-10-716215b636ca>:12: FutureWarning: The frame.append method is deprecatec
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-10-716215b636ca>:12: FutureWarning: The frame.append method is deprecatec
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-10-716215b636ca>:12: FutureWarning: The frame.append method is deprecatec
  gdf = gdf.append(dict, ignore_index=True, sort=True)
<ipython-input-10-716215b636ca>:12: FutureWarning: The frame.append method is deprecatec
  gdf = gdf.append(dict, ignore_index=True, sort=True)
```

	cluster	names	size	symbols
0	1	[Bragg Gaming Group Inc.]	1	[brag]
1	2	[Electronic Arts Inc., Take-Two Interactive So...	2	[ea, ttwo]
2	3	[Dell Technologies , Intel Corporation, Logit...	5	[dell, intc, logi, sony, wbd]
3	4	[Advanced Micro Devices, Inc, Meta Platforms, ...	3	[amd, meta, nvda]

Visualizing cluster and affine commodities, by volatility

The interactive graphic above, allows the user to investigate the clustered groups by hovering over their mouse over the dots. This allows for the user to investigate the companies within each cluster to infer about relationships between the assets.

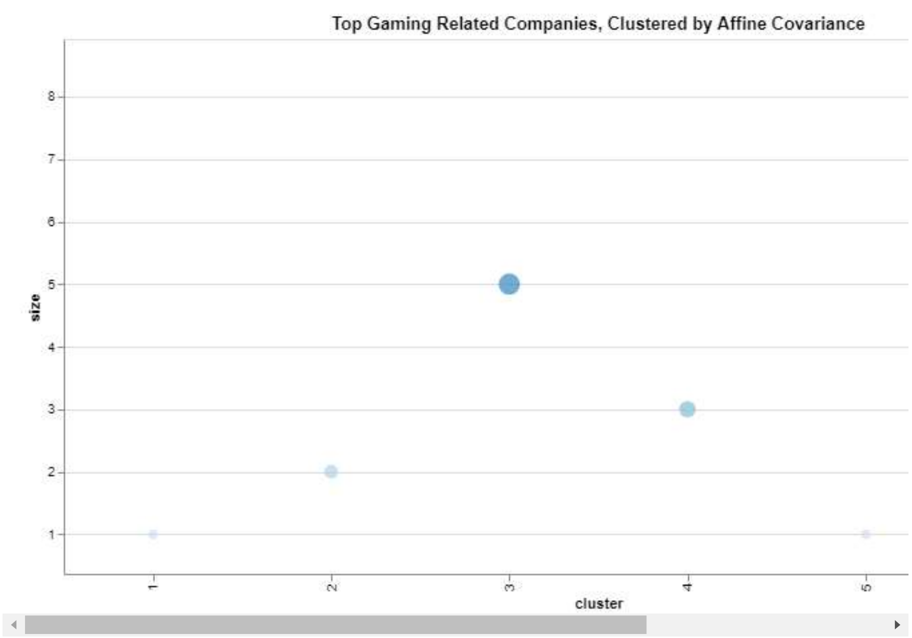
The list above also outlines the same information in a easily digestable table, segmenting the clusters and listing the companies within them.

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

```
cluster  1
['Bragg Gaming Group Inc.'] ,
cluster  2
['Electronic Arts Inc.' 'Take-Two Interactive Software, Inc.\t'] ,
cluster  3
[' Dell Technologies ' 'Intel Corporation' 'Logitech International S.A.'
 'Sony Group Corporation' 'Warner Bros. Discovery, Inc.'] ,
cluster  4
['Advanced Micro Devices, Inc' 'Meta Platforms' 'Nvidia Corp'] ,
cluster  5
['CD Project S.A.'] ,
cluster  6
['Bilibili Inc.' 'GameStop Corp.' 'Nintendo Co., Ltd.'
 'Roblox Corporation' 'Skillz Inc' 'Tencent Holdings'
 'Unity Software Inc.'] ,
```

```
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="Top Gaming Related Companies, 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
```

```
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)
```

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

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

```
df_symbol.columns
```

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

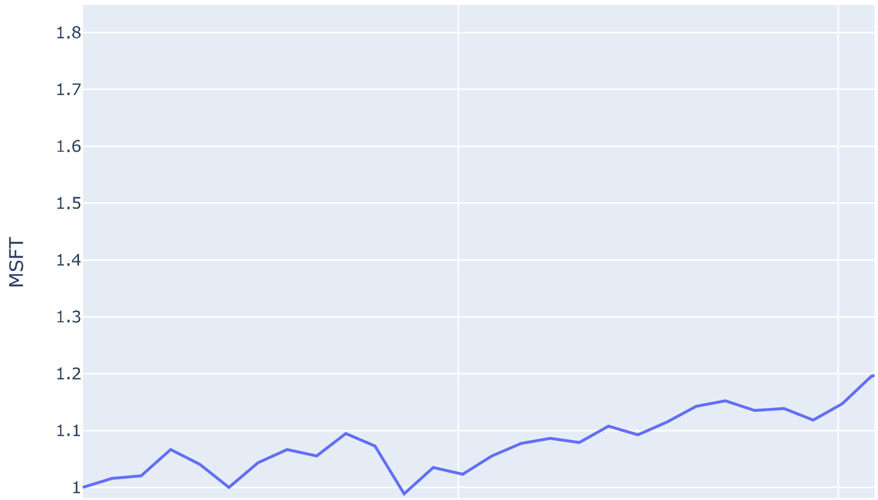
```
df_symbol.head(2)
```

	Date	Open	High	Low	Close	Volume
0	2021-12-21	178.240005	179.970001	176.440002	179.440002	869900
1	2021-12-22	179.789993	180.369995	176.389999	177.619995	1078800

```
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

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




```
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
1	2018-01-08	1.018172	1.011943	1.061881	0.959968	1.053526	1.015988

```
df2['MSFT']
```

```
0      1.000000
1      1.015988
2      1.020524
3      1.066561
4      1.040708
         ...
100    1.720717
101    1.752239
102    1.784896
103    1.802472
104    1.788185
Name: MSFT, Length: 105, dtype: float64
```

```
df_symbol.columns
```

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

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

```
0      179.440002
1      177.619995
2      177.960007
3      179.559998
4      179.389999
          ...
495    152.559998
496    157.199997
497    163.119995
498    163.889999
499    160.199997
Name: Close, Length: 500, dtype: float64
```

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

Plotting the Clustered Commodities

```
#generate a Date column in gdf
def getDateColumn():
  df = pd.read_csv('nvda')  #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)
```

```
          AMD       BILI   BRAG       DELL          EA        GME       INTC  \
0  144.250000  49.310001  5.480  52.557339  130.616135  39.529999  47.718105
1  143.880005  47.000000  5.400  52.481510  129.924362  38.500000  47.906078
2  146.139999  45.910000  5.685  52.964905  130.981796  38.035000  48.225643

        LOGI        META  NTDOY        NVDA      OTGLY        RBLX  \
0  80.951752  334.200012  12.190  290.349579  11.403989  102.589996
1  82.193466  330.450012  12.022  293.595093  11.668970  102.769997
2  81.683304  335.239990  12.182  295.991760  11.502131  101.820000

         SKLZ        SONY      TCEHY        TTWO           U        WBD
0  173.800003  120.550003  54.363251  179.440002  145.839996  23.530001
1  170.000000  123.099998  53.443745  177.619995  144.889999  23.629999
2  170.199997  123.860001  56.540028  177.960007  145.570007  24.420000
```

Start coding or generate with AI.

Final Response

The chart below showcases the overall market trends of the equities within the gaming industry that we chose to focus on. These companies all together show that the industry as awhole moves pretty much in the same cluster throughout the market history that we covered. Near the end the companies start to dissapate, signialing that their direct connection is becoming less and less strong. This can be influenced by the industry as a whole growing, while utalzing technology and resources from outside the industry.

Cluster Analysis

Above we broke the idustry down into clusters, that correlation to the volitilty of the stocks and how they follow the same trends. These clusters largely correlate to the function that those equites execute within the broader gaming industry. For example, cluster 2 consists of companies that are striclty producers and distrubitors of blockbuster titles within the gaming industry, while cluster 3 focuses on companies that create gaming related hardware (computer parts, keybaords, headsets, mice, etc.)

```
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=(30, 8))
scaled_gdf.plot.line(ax=ax)

# add title and subtitle
ax.set_title('Covariant Equities Related to Gaming Industry', 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()
```