

Advance Machine Learning CSCI-E82

Clustering and Deep Learning in Portfolio Construction

Team: Nitesh Kumar & Janice Pham

Summary

A brief summary of our methodology in each section of this notebook:

1. Data Collection : Our initial list of stock universe consisted of 1342 stock. We used an api service from tiingo.com to get our fundamental, and daily date. We required 20 years of fundamental data, so we filtered each stock based on the years of quarterly data. We had 293 from this filtering. We had to filter further for missing daily data. This gave us a stock list of 283.
2. Data Processing & Cleaning : We had several columns with missing values out of 49 features. We removed those columns. We further had stocks with missing features. We removed those as well. Our final stock list consisted of 278 stocks.
3. Feature selection of fundamental data : A lot of our features overlapped with other features. We used our domain knowledge to remove the features.
4. We processed our daily data to convert it to daily returns, and then quarterly returns. We also obtained S&P 500 data to use as our benchmark to evaluate the performance of our cluster.
5. Exploratory Data Analysis : We looked at the selected features using PCA to see how much variance they had across features. We looked at TSNE, and UMAP plot as an initial analysis of how many clusters we should have.
6. Unsupervised Learning - Clustering Models : We trained Kmeans, Hierarchical, Gaussian mixture models, and with first quarter data of 2002. We found the best parameters for each of the model using this data. We then trained each quarter data with the best parameters for each model. We used several metrics for our model selection. We used elbow, silhouette score, AIC, BIC and Log

Likelihood criteria based on each model.

7. Visualization : We had a lot of data for different stocks. We needed a way to visualize the stock data across Quarters. We also needed a way to visualize our data for clusters, to check if our clustering approach is any good. We created a visualization tool using d3.js , html, and javascript. We have included screenshots from the tool. For full exploration we have made the tool available at <https://dashing-crisp-38c329.netlify.app/> (<https://dashing-crisp-38c329.netlify.app/>). We used the tool throughout along with pandas filtering to visualize different aspects of our cluster.
8. Model Performance : We evaluated each of the cluster produced by our algorithm for each quarter against benchmark S&P 500. The metric used was sharpe ratio. We also compared all the models sharpe ratio for each quarter.
7. Conclusion : We reported our finding using clustering for portfolio selection. Since we cannot forecast using this approach, but this method has value and merits when it is used with a supervised learning.

Import Libraries

In [120]:

```
# System
import time
import os
import requests
import datetime
from collections import Counter
import itertools
import random as rn
import re
import string
from datetime import datetime
import yfinance as yf

# Libraries
import numpy as np
import pandas as pd
import urllib.request
from matplotlib import pyplot as plt
import seaborn as sns

%matplotlib inline

# Visualization
from sklearn.manifold import TSNE
from sklearn.manifold import MDS
from sklearn.manifold import Isomap
import umap

# Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error,\
mean_absolute_error, mean_absolute_percentage_error
from sklearn.cluster import KMeans, DBSCAN, MeanShift
from sklearn.neighbors import NearestNeighbors, NearestCentroid
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import AgglomerativeClustering
from sklearn.mixture import GaussianMixture as GMM
```

```
# Scipy
import scipy.cluster.hierarchy as hac
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import fcluster
from scipy.cluster.hierarchy import dendrogram, linkage

from tensorflow.keras.models import Model, Sequential, save_model
from tensorflow.keras import layers
from tensorflow.keras import activations
from tensorflow.keras import losses
from tensorflow.keras import optimizers
from tensorflow.keras import initializers
from tensorflow.keras import regularizers

from yellowbrick.cluster import SilhouetteVisualizer
```

Data Collection

Our data consists of fundamental data and daily data of stocks traded in the US. For both fundamental data, and daily data we used api from <https://www.tiingo.com/>.

Our initial stock list consisted of Large cap, and mid cap stocks based on current value. We had 1342 stock in our universe. We required 20 years of quarterly fundamental data for each stock. However, lot of the stocks traded today, did not exist 20 years ago. We filtered our stocks to have atleast 20 years of data, this narrowed our stock universe to 293 tickers.

During our EDA , we discovered few stock with lot of missing values. Our final stock universe included 278 stocks.

The api we used requires a personal token key. If it is required to run the cell, please let us know, we are happy to provide the token.

All the data from each stock were stored in files as this process takes hours. We used the files for our process and ran the code to fetch data from api only few times until we got it right.

```
In [2]: #read the stocks from our universe of 1342 large cap, and mid cap stocks.  
df_stocks = pd.read_excel("large_midcap.xlsx")  
df_stocks.head(2)
```

```
Out[2]:
```

	Symbol	Company Name	Security Type	Security Price	Market Capitalization
0	WFC	Wells Fargo & Co	Common Stock	42.45	\$165.38B
1	LIN	Linde Plc	Common Stock	331.40	\$163.89B

```
In [117]: ## reads the fundamental data for adobe, to make it a standard for getting fundamental data from oth
```

```
In [3]: headers = {
    'Content-Type': 'application/json',
    'Authorization' : 'Token 76026xxxxxxxxb047c'
}

requestResponse = requests.get(f"https://api.tiingo.com/tiingo/fundamentals/{'ADBE'}/statements",
                               headers=headers)

data = requestResponse.json()
row_list = []
for i in range(len(data)):
    row_dict = {}

    if data[i]['quarter'] >0:
        row_dict['date'] = data[i]['date']
        row_dict['quarter'] = data[i]['quarter']

        if 'balanceSheet' in list(data[i]['statementData'].keys()):
            if 'incomeStatement' in list(data[i]['statementData'].keys()):
                balance_list = data[i]['statementData']['balanceSheet']
                income_list = data[i]['statementData']['incomeStatement']

                for item in balance_list:
                    row_dict[f"bal_{item['dataCode']}"] = item['value']

                for item in income_list:
                    row_dict[f"inc_{item['dataCode']}"] = item['value']

    row_list.append(row_dict)

if len(row_list) >=80:

    temp = pd.DataFrame.from_dict(row_list)

    temp['date'] = pd.to_datetime(temp['date'])
    temp.index = temp['date']
    temp.drop('date', axis=1, inplace=True)
    temp.drop_duplicates(inplace=True)
```

```
In [5]: column_integrity_set = set(temp.columns.tolist())
```

```
In [8]: column_integrity_set
```

```
Out[8]: {'bal_accoci',
 'bal_acctPay',
 'bal_acctRec',
 'bal_assetsCurrent',
 'bal_assetsNonCurrent',
 'bal_cashAndEq',
 'bal_debt',
 'bal_debtCurrent',
 'bal_debtNonCurrent',
 'bal_deferredRev',
 'bal_deposits',
 'bal_equity',
 'bal_intangibles',
 'bal_inventory',
 'bal_investments',
 'bal_investmentsCurrent',
 'bal_investmentsNonCurrent',
 'bal_liabilitiesCurrent',
 'bal_liabilitiesNonCurrent',
 'bal_ppeq',
 'bal_retainedEarnings',
 'bal_sharesBasic',
 'bal_taxAssets',
 'bal_taxLiabilities',
 'bal_totalAssets',
 'bal_totalLiabilities',
 'inc_consolidatedIncome',
 'inc_costRev',
 'inc_ebit',
 'inc_ebitda',
 'inc_ebt',
 'inc_eps',
 'inc_epsDil',
 'inc_grossProfit',
 'inc_intexp',
 'inc_netIncComStock',
 'inc_netIncDiscOps',
 'inc_netinc',
 'inc_nonControllingInterests',
 'inc_opex',
```

```
'inc_opinc',
'inc_prefDVDs',
'inc_revenue',
'inc_rnd',
'inc_sga',
'inc_shareswa',
'inc_shareswaDil',
'inc_taxExp',
'quarter'}
```

call the api to get the fundamental data for 1342 stocks. Store only the stocks that have 20 years of data and has all the columns as defined by our column integrity set.

```
In [ ]: headers = {
    'Content-Type': 'application/json',
    'Authorization' : 'Token 7xxxxxxxxxx1d655b047c'
}

stocks_with_missing_data = set()

def get_stock_data(stock_list):
    count =0
    stocks_with_less_data = []
    for stock in stock_list:
        print(stock)
        requestResponse = \
            requests.get(f"https://api.tiingo.com/tiingo/fundamentals/{stock}/statements",
                         headers=headers)

        if requestResponse.status_code == 200:
            try:
                data =requestResponse.json()

                if requestResponse.status_code ==200:
                    try:

                        if ((len(data) >= 84)):
                            row_list = []

                            for i in range(len(data)):
                                row_dict = {}

                                if data[i]['quarter'] >0:
                                    row_dict['date'] = data[i]['date']
                                    row_dict['quarter'] = data[i]['quarter']

                                if 'balanceSheet' in list(data[i]['statementData'].keys()):
                                    if 'incomeStatement' in\
                                        list(data[i]['statementData'].keys()):
                                        balance_list = \
                                            data[i]['statementData']['balanceSheet']
                                        income_list = \
                                            data[i]['statementData']['incomeStatement']
```

```
        for item in balance_list:
            row_dict[f"bal_{item['dataCode']}"] = \
                item['value']

        for item in income_list:
            row_dict[f"inc_{item['dataCode']}"] = \
                item['value']

    row_list.append(row_dict)

if len(row_list) >=80:

    temp = pd.DataFrame.from_dict(row_list)

    temp['date'] = pd.to_datetime(temp['date'])
    temp.index = temp['date']
    temp.drop('date', axis=1, inplace=True)
    temp.drop_duplicates(inplace=True)

    if len(column_integrity_set.difference\
        (set(temp.columns.tolist()))) == 0:
        temp =temp[temp.index > '2001-12-31']

    if temp.shape[0] ==83:
        print(f"{stock} meets the integrity test.")

    temp.to_excel(f"data/fundamental/{stock}.xlsx")

else:
    stocks_with_missing_data.add(stock)

else:
    stocks_with_missing_data.add(stock)

else:
    stocks_with_missing_data.add(stock)
except:
    pass
```

```
        except:  
            pass  
    return (stocks_with_missing_data)
```

```
In [11]: m = get_stock_data(df_stocks['Symbol'].iloc[:10].tolist())
```

```
WFC  
LIN  
BHP  
TXN  
TXN meets the integrity test.  
PM  
VZ  
VZ meets the integrity test.  
ADBE  
ADBE meets the integrity test.  
TTE  
UPS  
AMGN  
AMGN meets the integrity test.
```

In the above example call of the api call we only tried to fetch the data for 10 stocks. Since our starting universe was 1342 stocks, the api timed out multiple times. We tried it multiple times to get all the stock data in a different notebook

```
In [ ]: m = get_stock_data(df_stocks['Symbol'].tolist())
```

Get daily data for stocks that were filtered and had the fundamental data

```
In [ ]: stock_list = []  
  
for file in os.listdir("data/fundamental/"):   
    stock_list.append(file.split('.')[0])  
  
for stock in stock_list:  
    if stock == '':  
        stock_list.remove(stock)
```

```
In [ ]: headers = {
    'Content-Type': 'application/json',
    'Authorization' : 'Token 760269xxxxxxxxx181d655b047c'
}
stock_with_missing_data = []

def get_daily(stock_list):
    for stock in stock_list:
        print(stock)
        req_string = \
f"https://api.tiingo.com/tiingo/daily/{stock}/prices?startDate=2002-1-1&endDate=2022-12-01&r
requestResponse = requests.get(req_string, headers=headers)

    if requestResponse.status_code ==200:
        try:
            data2 = requestResponse.json()
            if (len(data2) >=5200):
                print(f"working on {stock}")
                temp2 =pd.DataFrame.from_dict(data2)
                temp2 = temp2[['date','adjClose','adjVolume']]
                temp2['date'] = pd.to_datetime(temp2['date'])
                temp2.index = temp2['date'].dt.tz_localize(None)
                temp2.drop( 'date', axis=1, inplace=True)
                temp2.to_excel(f"data/daily/{stock}.xlsx")
            else:
                stock_with_missing_data.append(stock)

        except:
            stock_with_missing_data.append(stock)
            pass

    else:
        stock_with_missing_data.append(stock)
```

```
In [ ]: get_daily(stock_list)
```

```
In [5]: get_daily(stock_list)
```

```
WORKING ON ALV
GILD
working on GILD
MTZ
working on MTZ
C00
working on C00
CHE
working on CHE
WHR
working on WHR

AMT
working on AMT
RS
working on RS
WEC
working on WEC
FISV
working on FISV
HALO
```

```
In [6]: stock_with_missing_data
```

```
Out[6]: ['HALO', 'PARA', 'DKS', 'MTCH', 'WTW', 'TRI', 'CBRE', 'CME', 'HUBB', 'NFLX']
```

The stocks above with missing daily data, were manually removed from the already created files for fundamental data

1. Data Processing & Cleaning

1.1 - Fundamental data

After acquiring all the data, we combine the fundamental data into a data frame by quarter of each year.

```
In [3]: ##### List of all stocks collected in the dataset #####
stock_list = []

for file in os.listdir("data/fundamental/"):
    stock_list.append(file.split('.')[0])

for stock in stock_list:
    if stock == '':
        stock_list.remove(stock)

##### Function to combine all stock data across quarters 2002-2022 #####
def add_key_and_combine_df(df,stock_list):
    """
    Combine all stocks and add key to each row i.e. 2022_3 representing the year and quarter
    """
    for stock in stock_list[1:]:
        temp = pd.read_excel(f"data/fundamental/{stock}.xlsx",index_col='date')
        temp.index = pd.date_range(start='03/31/2002', end='09/30/2022', periods =83)
        temp['stock'] = stock
        temp['key'] = [f"{temp.index[i].year}_{temp.index[i].quarter}" for i in range(temp.shape[0])]
        temp['date'] = temp.index.strftime("%Y-%m-%d")

        temp.drop('quarter', axis=1, inplace=True)

        df = pd.concat([df,temp], axis=0,ignore_index=True)

    return df

print(f"Number of stocks with fundamental data {len(stock_list)}")
```

Number of stocks with fundamental data 283

```
In [4]: ##### Dataframe contain all stock data across quarters #####
stock = stock_list[0]
clustering_all_data = pd.read_excel(f"data/fundamental/{stock}.xlsx", index_col='date')
clustering_all_data.index = pd.date_range(start='03/31/2002', end='09/30/2022', periods=83)
clustering_all_data['stock'] = stock
clustering_all_data['key'] = [f"{clustering_all_data.index[i].year}_{clustering_all_data.index[i].quarter}" for i in range(clustering_all_data.shape[0])]
clustering_all_data['date'] = clustering_all_data.index.strftime("%Y-%m-%d")
clustering_all_data.drop('quarter', axis=1, inplace=True)

clustering_all_data = add_key_and_combine_df(clustering_all_data.copy(), stock_list)

# print result
print(f"Number of stock tickers before cleaning: {len(clustering_all_data['stock'].unique())}")
clustering_all_data.head(3)
```

Number of stock tickers before cleaning: 283

Out[4]:

	bal_totalAssets	bal_taxAssets	bal_acctRec	bal_investments	bal_accoci	bal_liabilitiesCurrent	bal_debtCurrent	bal_deposits	bal...
0	1.595600e+10	0.0	836000000.0	566000000.0	-942000000.0	3.627000e+09	7000000.0	0.0	
1	1.587900e+10	0.0	876000000.0	561000000.0	-855000000.0	3.474000e+09	7000000.0	0.0	
2	1.591100e+10	0.0	800000000.0	578000000.0	-779000000.0	3.337000e+09	7000000.0	0.0	

3 rows × 51 columns

```
In [5]: # Check for NaN  
clustering_all_data.isna().sum()
```

```
Out[5]: bal_totalAssets           4  
bal_taxAssets                  4  
bal_acctRec                   4  
bal_investments                4  
bal_accoci                     4  
bal_liabilitiesCurrent         394  
bal_debtCurrent                317  
bal_deposits                  4  
bal_investmentsCurrent         315  
bal_debtNonCurrent              390  
bal_retainedEarnings             891  
bal_investmentsNonCurrent       315  
bal_intangibles                 4  
bal_cashAndEq                  4  
bal_taxLiabilities              4  
bal_liabilitiesNonCurrent       394  
bal_inventory                  4  
bal_sharesBasic                 3  
bal_assetsNonCurrent             315  
bal_assetsCurrent                315  
bal_totalLiabilities              4  
bal_ppeq                       4  
bal_acctPay                    4  
bal_equity                      4  
bal_deferredRev                 4  
bal_debt                        4  
inc_opex                        0  
inc_opinc                       0  
inc_netinc                      0  
inc_taxExp                      0  
inc_prefDVDs                    0  
inc_intexp                      0  
inc_epsDil                      2  
inc_sga                          0  
inc_netIncDisc0ps                0  
inc_grossProfit                  0  
inc_costRev                     0  
inc_rnd                          0  
inc_eps                          1
```

```
inc_nonControllingInterests      0
inc_ebt                          0
inc_consolidatedIncome          0
inc_shareswa                     0
inc_ebit                         0
inc_revenue                      0
inc_ebitda                       1
inc_netIncComStock               0
inc_shareswaDil                 3198
stock                            0
key                               0
date                             0
dtype: int64
```

There were few stocks with lot of missing values, and some features had most of the data missing.

```
In [6]: ##### Remove features with a lot of NaN #####
missing_features = ['bal_investmentsNonCurrent', 'bal_investmentsCurrent', 'bal_assetsCurrent', \
                     'bal_liabilitiesNonCurrent', 'bal_debtNonCurrent', 'bal_debtCurrent', \
                     'bal_liabilitiesCurrent', 'bal_assetsNonCurrent', 'bal_retainedEarnings', \
                     'inc_shareswaDil']

]

for feature in missing_features:
    if feature in clustering_all_data.columns.tolist():
        clustering_all_data.drop(feature, axis=1, inplace=True)

# Check NaN again
clustering_all_data.isna().sum()
```

Out[6]:	bal_totalAssets	4
	bal_taxAssets	4
	bal_acctRec	4
	bal_investments	4
	bal_accoci	4
	bal_deposits	4
	bal_intangibles	4
	bal_cashAndEq	4
	bal_taxLiabilities	4
	bal_inventory	4
	bal_sharesBasic	3
	bal_totalLiabilities	4
	bal_ppeq	4
	bal_acctPay	4
	bal_equity	4
	bal_deferredRev	4
	bal_debt	4
	inc_opex	0
	inc_opinc	0
	inc_netinc	0
	inc_taxExp	0
	inc_prefDVDs	0
	inc_intexp	0
	inc_epsDil	2
	inc_sga	0
	inc_netIncDiscOps	0

```
inc_grossProfit          0  
inc_costRev             0  
inc_rnd                 0  
inc_eps                  1  
inc_nonControllingInterests 0  
inc_ebt                  0  
inc_consolidatedIncome    0  
inc_shareswa             0  
inc_ebit                  0  
inc_revenue                0  
inc_ebitda                 1  
inc_netIncComStock        0  
stock                      0  
key                        0  
date                      0  
dtype: int64
```

We observed few more missing values.

```
In [7]: ##### Dealing with stocks with small number of NaN #####
# Find stocks with missing values
search_features = ['bal_ppeq', 'bal_deferredRev', 'bal_sharesBasic', 'bal_acctRec', \
                    'bal_equity', 'bal_acctPay', 'bal_totalAssets', 'bal_totalLiabilities', \
                    'bal_cashAndEq', 'bal_deposits', 'bal_deposits', 'bal_investments', \
                    'bal_accoci', 'bal_taxAssets', 'bal_debt', 'bal_inventory', \
                    'bal_taxLiabilities', 'bal_intangibles', 'inc_ebitda', 'inc_epsDil', \
                    'inc_eps', \
                    ]
temp1 = clustering_all_data.iloc[:,10:][clustering_all_data['bal_totalAssets'].isna()]
temp2 = clustering_all_data.iloc[:,10:][clustering_all_data['bal_sharesBasic'].isna()]
temp3 = clustering_all_data.iloc[:,10:][clustering_all_data['inc_ebitda'].isna()]
temp4 = clustering_all_data.iloc[:,10:][clustering_all_data['inc_epsDil'].isna()]
print('*'*50 + '\n1st List of NaN Stocks\n' + '*'*50)
display(temp1)
print('*'*50 + '\n2nd List of NaN Stocks\n' + '*'*50)
display(temp2)
print('*'*50 + '\n3rd List of NaN Stocks\n' + '*'*50)
display(temp3)
print('*'*50 + '\n4th List of NaN Stocks\n' + '*'*50)
display(temp4)
```

=====

1st List of NaN Stocks

=====

	bal_sharesBasic	bal_totalLiabilities	bal_ppeq	bal_acctPay	bal_equity	bal_deferredRev	bal_debt	inc_opex	inc_opinc	inc
12581	84977766.0		NaN	NaN	NaN	NaN	NaN	95301000	13029000	3
12582	84754333.0		NaN	NaN	NaN	NaN	NaN	95953000	35391000	26
12583	83187187.0		NaN	NaN	NaN	NaN	NaN	112164000	44462000	37
20660	782931936.0		NaN	NaN	NaN	NaN	NaN	902000000	185000000	106

4 rows × 31 columns

=====

2nd List of NaN Stocks

=====

	bal_sharesBasic	bal_totalLiabilities	bal_ppeq	bal_acctPay	bal_equity	bal_deferredRev	bal_debt	inc_opex	inc_c
16182	NaN	4.101000e+09	3.935000e+09	1.805000e+09	4.945000e+09	86000000.0	579000000.0	291000000	19500
16183	NaN	3.948000e+09	3.795000e+09	1.710000e+09	4.937000e+09	87000000.0	575000000.0	306000000	27200
16184	NaN	3.620000e+09	3.608000e+09	1.618000e+09	4.611000e+09	0.0	655000000.0	279000000	26300

3 rows × 31 columns

=====

3rd List of NaN Stocks

=====

	bal_sharesBasic	bal_totalLiabilities	bal_ppeq	bal_acctPay	bal_equity	bal_deferredRev	bal_debt	inc_opex	inc_opinc	inc_ne
12583	83187187.0	NaN	NaN	NaN	NaN	NaN	NaN	112164000	44462000	37920

1 rows × 31 columns

=====

4th List of NaN Stocks

=====

	bal_sharesBasic	bal_totalLiabilities	bal_ppeq	bal_acctPay	bal_equity	bal_deferredRev	bal_debt	inc_opex
1156	3.054803e+08	1.843872e+10	1.114020e+10	5.587300e+08	1.450754e+09	9.563800e+07	1.066192e+10	157263000
23157	4.413238e+09	1.733750e+11	5.355500e+10	1.224100e+10	8.029600e+10	2.757000e+09	9.762300e+10	32275000000

2 rows × 31 columns

Since 7 out of 8 tickers either link to more than 2 features or 2 quarters that have missing values, we remove these stocks from the dataset. This leaves us with 1 ticker which is CMCSA. However, since the missing value is the 1st quarter of 2002, the starting quarter of the dataset, we cannot use the average of previous quarters to fill in the NaN for this quarter as we do not have the data for previous quarters. Therefore, we also remove this stock from the dataset.

```
In [8]: ##### Stocks to remove from dataset #####
stocks_with_missing_features = ['TTWO', 'LUV', 'MGA', 'CNP', 'CMCSA']
for stock in stocks_with_missing_features:
    clustering_all_data = clustering_all_data[clustering_all_data['stock'] != stock]
    stock_list.remove(stock)
```

```
In [9]: # Final check for missing value  
clustering_all_data.isna().sum()
```

```
Out[9]: bal_totalAssets          0  
bal_taxAssets                  0  
bal_acctRec                   0  
bal_investments                0  
bal_accoci                     0  
bal_deposits                   0  
bal_intangibles                 0  
bal_cashAndEq                  0  
bal_taxLiabilities              0  
bal_inventory                  0  
bal_sharesBasic                 0  
bal_totalLiabilities             0  
bal_ppeq                        0  
bal_acctPay                     0  
bal_equity                      0  
bal_deferredRev                 0  
bal_debt                         0  
inc_opex                        0  
inc_opinc                       0  
inc_netinc                      0  
inc_taxExp                      0  
inc_prefDVDs                    0  
inc_intexp                      0  
inc_epsDil                      0  
inc_sga                          0  
inc_netIncDiscOps                0  
inc_grossProfit                  0  
inc_costRev                     0  
inc_rnd                          0  
inc_eps                          0  
inc_nonControllingInterests      0  
inc_ebt                          0  
inc_consolidatedIncome            0  
inc_shareswa                    0  
inc_ebit                         0  
inc_revenue                      0  
inc_ebitda                      0  
inc_netIncComStock                0  
stock                            0
```

```
key          0  
date         0  
dtype: int64
```

At this point we had fundamental data for all stocks with no missing values. We are left with 278 stocks

```
In [10]: len(stock_list)
```

```
Out[10]: 278
```

1.2- Daily Data

Combine Daily data into a dataframe.

```
In [11]: ##### Function to combine all daily stock prices data #####  
def stock_price_df(df, stock_price_list):  
    for stock in stock_price_list[1:]:  
        temp = pd.read_excel(f"data/daily/{stock}.xlsx", index_col='date').iloc[:,0]  
        df = pd.concat([df,temp], axis=1, ignore_index=True)  
    return df  
  
stock_price_first_i = stock_list[0]  
stock_daily_df = pd.read_excel(f"data/daily/{stock_price_first_i}.xlsx", index_col='date').iloc[:,0]  
stock_daily_df = stock_price_df(stock_daily_df.copy(), stock_list)
```

```
In [12]: stock_daily_df.shape
```

```
Out[12]: (5267, 278)
```

```
In [13]: stock_daily_df.head(2)
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9	...	268
date												
2002-01-02	17.303168	7.190484	14.061759	15.719889	42.305134	9.580771	15.507481	15.848582	12.972503	28.300198	...	6.523670
2002-01-03	17.075227	7.219362	14.117687	15.512482	40.759946	9.449340	16.929093	16.465801	12.967265	27.945847	...	6.583968

2 rows × 278 columns

Calculate daily returns for each stock

```
In [14]: ##### Getting daily stock returns #####
stock_daily_returns = pd.read_csv('stock_daily_rets.csv')
stock_daily_returns.index = stock_daily_returns['date']
stock_daily_returns.index = pd.to_datetime(stock_daily_returns.index)
stock_daily_returns = stock_daily_returns.drop(columns=['date'])
print(f'Shape of data frame: {stock_daily_returns.shape}')
stock_daily_returns.head()
```

Shape of data frame: (5266, 278)

Out[14]:

	NOV	JBL	BIIB	STZ	EXPD	FIS	MSI	DVN	ALV	PPL	...	LRCX	T
date													
2002-01-03	-0.015452	0.021944	-0.027693	0.003843	0.001370	0.001724	0.009946	-0.009076	0.014135	0.000997	...	0.032672	-0.0058
2002-01-04	0.004536	0.004537	0.005601	0.008969	0.019830	-0.004199	-0.009667	0.008051	0.009793	-0.033085	...	0.005954	-0.0011
2002-01-07	0.000676	0.005176	-0.016816	0.018537	-0.004451	-0.013484	-0.010745	0.011360	0.000412	0.007192	...	-0.004956	-0.0005
2002-01-08	-0.011411	-0.007264	0.013383	0.012759	0.003943	-0.002439	-0.010724	-0.004127	-0.008522	-0.001986	...	0.000498	-0.0005
2002-01-09	-0.005352	0.014747	-0.007155	0.003929	-0.003576	0.009422	-0.010093	0.002347	-0.001682	0.005799	...	0.000332	0.0044

5 rows × 278 columns

Get Daily S&P for Evaluation our cluster

```
In [15]: ##### DAILY S&P500 DATA PROCESSING #####
#####----- Get data for S&P500 -----#####
sp_start = datetime(2002, 1, 1)
sp_end = datetime(2022, 9, 30)

sp500_ticker = yf.Ticker('^GSPC')

sp500_daily_prices = pd.DataFrame(sp500_ticker.history(start=sp_start, \
                                                       end=sp_end)['Close'])
print('*'*40 + '\nS&P 500 Daily Prices\n' + '*'*40)
display(sp500_daily_prices.head())

#####----- Compute S&P 500 Returns -----#####
sp500_daily_rets = pd.DataFrame(np.log(sp500_daily_prices.Close) - \
                                 np.log(sp500_daily_prices.Close.shift(1)))
sp500_daily_rets.columns = ['S&P500']
sp500_daily_rets = sp500_daily_rets.iloc[1:,:]
sp500_daily_rets.head(3)
```

=====

S&P 500 Daily Prices

=====

Date	Close
2002-01-02 00:00:00-05:00	1154.670044
2002-01-03 00:00:00-05:00	1165.270020
2002-01-04 00:00:00-05:00	1172.510010
2002-01-07 00:00:00-05:00	1164.890015
2002-01-08 00:00:00-05:00	1160.709961

Out[15]:

S&P500

Date	
2002-01-03 00:00:00-05:00	0.009138
2002-01-04 00:00:00-05:00	0.006194
2002-01-07 00:00:00-05:00	-0.006520

In [16]: clustering_all_data.shape

Out[16]: (23074, 41)

2. Feature Selection for Fundamental Data

Our Fundamental data consists of 41 features. Some of the features overlapped with each other. Using our domain knowledge, we removed features that overlapped. Example , some features were the ratio of two features.

```
In [17]: ##### Remove overlapped features #####
dropped_features = [
    'bal_ppeq', 'bal_deferredRev', 'bal_sharesBasic', 'bal_acctRec', 'bal_acctPay', 'bal_cashAndEq',
    'bal_deposits', 'bal_accoci', 'bal_taxAssets', 'bal_taxLiabilities', 'bal_intangibles', 'inc_sha
    'inc_rnd', 'inc_sga', 'inc_epsDil', 'inc_netIncDisc0ps', 'inc_ebt',
    'inc_nonControllingInterests', 'inc_taxExp', 'inc_prefDVDs', 'inc_intexp'
]

shortlist_modified_clustering_agg = clustering_all_data.copy()

for feature in dropped_features:
    if feature in shortlist_modified_clustering_agg.columns.tolist():
        shortlist_modified_clustering_agg.drop(feature, axis=1, inplace=True)

print(f'Number of features in modified cluster aggregate data: {shortlist_modified_clustering_agg.sh
```

Number of features in modified cluster aggregate data: 20

Next we grouped our data in to a dictionary by each quarter, and scaled our data in preparation for model training, and visualization**

```
In [18]: data_scaler = StandardScaler()
scaled_shortlist_modified_clustering_agg = data_scaler.fit_transform(shortlist_modified_clustering_a
updated_shortlist_modified_clustering_agg = pd.DataFrame(scaled_shortlist_modified_clustering_agg)
updated_shortlist_modified_clustering_agg.columns = shortlist_modified_clustering_agg.columns[:-3]
updated_shortlist_modified_clustering_agg = pd.concat([updated_shortlist_modified_clustering_agg, \
shortlist_modified_clustering_agg.iloc[:, -3:].reset_index(drop=True)], axis=1)
```

```
In [19]: print("DataFrame after scaling\n")
updated_shortlist_modified_clustering_agg.head(3)
```

DataFrame after scaling

Out[19]:

	bal_totalAssets	bal_investments	bal_inventory	bal_totalLiabilities	bal_equity	bal_debt	inc_opex	inc_opinc	inc_netinc	inc_gross
0	-0.059803	-0.113510	0.807935	-0.108007	0.087025	-0.159333	-0.301192	-0.052917	0.009751	-0.2
1	-0.061980	-0.114056	0.784224	-0.112412	0.090631	-0.159122	-0.288172	-0.052917	-0.001460	-0.2
2	-0.061075	-0.112201	0.761121	-0.115602	0.101356	-0.152848	-0.270356	-0.087274	-0.035095	-0.2

```
In [20]: ##### Create dict to split data by quarters #####
by_quarter_grouping = updated_shortlist_modified_clustering_agg.groupby(\n                        updated_shortlist_modified_clustering_agg['key'])

by_quarter_dict = dict(list(by_quarter_grouping))

# Print result
print(f'Length of dict: {len(by_quarter_dict)}')
print(f"Shape of quarterly data frame for 2002 Quarter 1: {by_quarter_dict['2002_1'].shape}")
display(by_quarter_dict['2002_1'].head(3))
```

Length of dict: 83

Shape of quarterly data frame for 2002 Quarter 1: (278, 20)

	bal_totalAssets	bal_investments	bal_inventory	bal_totalLiabilities	bal_equity	bal_debt	inc_opex	inc_opinc	inc_netinc	inc_grc
0	-0.059803	-0.113510	0.807935	-0.108007	0.087025	-0.159333	-0.301192	-0.052917	0.009751	-0.000100
83	-0.436607	-0.174458	-0.126406	-0.345833	-0.566396	-0.282938	-0.333741	-0.181527	-0.106709	-0.000100
166	1.702950	-0.175277	-0.352391	0.788463	3.652767	0.922843	0.243575	0.138337	0.043386	0.000100

```
In [21]: ##### Create a dictionary without the features stock, key, and date #####
by_quarter_dict_clean = {}
for i in by_quarter_dict.keys():
    by_quarter_dict_clean[i] = by_quarter_dict[i].drop(columns=['stock', 'key', 'date'])
```

3. Further processing of daily data and S&P 500 benchmark daily data and Helper methods for performance evaluation

This step further processes our daily data to convert it into quarterly returns to evaluate our cluster which is based on Quarterly fundamental data. This section also includes helper methods to calculate sharpe ratio for stocks in each cluster for each of the quarter. We also calculate our benchmark sharpe ratio for S&P 500.


```
In [22]: ##### Group daily S&P500 returns by quarter #####  
sp500_daily_rets_by_quarter = {}  
for k, v in by_quarter_dict_clean.items():  
    daily_ls = []  
    if k[-1] == '1':  
        for i in sp500_daily_rets.T.columns:  
            if (str(i).startswith(k[:4]+'-01') == True) or \  
                (str(i).startswith(k[:4]+'-02') == True) or \  
                (str(i).startswith(k[:4]+'-03') == True):  
                col = sp500_daily_rets.T[i]  
                daily_ls.append(col)  
                daily_df = pd.DataFrame(daily_ls)  
            else:  
                pass  
        sp500_daily_rets_by_quarter[k] = daily_df  
  
    elif k[-1] == '2':  
        for i in sp500_daily_rets.T.columns:  
            if (str(i).startswith(k[:4]+'-04') == True) or \  
                (str(i).startswith(k[:4]+'-05') == True) or \  
                (str(i).startswith(k[:4]+'-06') == True):  
                col = sp500_daily_rets.T[i]  
                daily_ls.append(col)  
                daily_df = pd.DataFrame(daily_ls)  
            else:  
                pass  
        sp500_daily_rets_by_quarter[k] = daily_df  
  
    elif k[-1] == '3':  
        for i in sp500_daily_rets.T.columns:  
            if (str(i).startswith(k[:4]+'-07') == True) or \  
                (str(i).startswith(k[:4]+'-08') == True) or \  
                (str(i).startswith(k[:4]+'-09') == True):  
                col = sp500_daily_rets.T[i]  
                daily_ls.append(col)  
                daily_df = pd.DataFrame(daily_ls)  
            else:  
                pass  
        sp500_daily_rets_by_quarter[k] = daily_df  
  
    elif k[-1] == '4':  
        for i in sp500_daily_rets.T.columns:
```

```
if (str(i).startswith(k[:4] + '-10') == True) or \
(str(i).startswith(k[:4] + '-11') == True) or \
(str(i).startswith(k[:4] + '-12') == True):
    col = sp500_daily_rets.T[i]
    daily_ls.append(col)
    daily_df = pd.DataFrame(daily_ls)
else:
    pass
sp500_daily_rets_by_quarter[k] = daily_df

else:
    pass
```

```
In [23]: #####----- Display result -----#####
print('*'*50 + '\nDaily S&P 500 returns grouped by quarter - 2022Q3\n' + '*'*50)
print('\nFirst 3 days\n')
display(sp500_daily_rets_by_quarter['2022_3'].head(3))
print('\nLast 3 days\n')
display(sp500_daily_rets_by_quarter['2022_3'].tail(3))
```

```
=====
Daily S&P 500 returns grouped by quarter - 2022Q3
=====
```

First 3 days

S&P500	
2022-07-01 00:00:00-04:00	0.010499
2022-07-05 00:00:00-04:00	0.001583
2022-07-06 00:00:00-04:00	0.003567

Last 3 days

S&P500	
2022-09-27 00:00:00-04:00	-0.002123
2022-09-28 00:00:00-04:00	0.019481
2022-09-29 00:00:00-04:00	-0.021353

```
In [24]: ##### Group daily stock returns by quarter #####
transposed_stock_daily_rets = stock_daily_returns.T

daily_rets_by_quarter = {}
for k, v in by_quarter_dict_clean.items():
    daily_ls = []
    if k[-1] == '1':
        for i in transposed_stock_daily_rets.columns:
            if (str(i).startswith(k[:4]+'-01') == True) or \
                (str(i).startswith(k[:4]+'-02') == True) or \
                (str(i).startswith(k[:4]+'-03') == True):
                col = transposed_stock_daily_rets[i]
                daily_ls.append(col)
                daily_df = pd.DataFrame(daily_ls)
            else:
                pass
        daily_rets_by_quarter[k] = daily_df

    elif k[-1] == '2':
        for i in transposed_stock_daily_rets.columns:
            if (str(i).startswith(k[:4]+'-04') == True) or \
                (str(i).startswith(k[:4]+'-05') == True) or \
                (str(i).startswith(k[:4]+'-06') == True):
                col = transposed_stock_daily_rets[i]
                daily_ls.append(col)
                daily_df = pd.DataFrame(daily_ls)
            else:
                pass
        daily_rets_by_quarter[k] = daily_df

    elif k[-1] == '3':
        for i in transposed_stock_daily_rets.columns:
            if (str(i).startswith(k[:4]+'-07') == True) or \
                (str(i).startswith(k[:4]+'-08') == True) or \
                (str(i).startswith(k[:4]+'-09') == True):
                col = transposed_stock_daily_rets[i]
                daily_ls.append(col)
                daily_df = pd.DataFrame(daily_ls)
            else:
                pass
        daily_rets_by_quarter[k] = daily_df
```

```
elif k[-1] == '4':
    for i in transposed_stock_daily_rets.columns:
        if (str(i).startswith(k[:4]+'-10') == True) or \
           (str(i).startswith(k[:4]+'-11') == True) or \
           (str(i).startswith(k[:4]+'-12') == True):
            col = transposed_stock_daily_rets[i]
            daily_ls.append(col)
            daily_df = pd.DataFrame(daily_ls)
        else:
            pass
    daily_rets_by_quarter[k] = daily_df

else:
    pass
```

```
In [25]: print('*'*50 + '\nDaily stock returns grouped by quarter - 2022Q3\n' + '*'*50)
print('\nFirst 3 days\n')
display(daily_rets_by_quarter['2022_3'].head(3))
print('\nLast 3 days\n')
display(daily_rets_by_quarter['2022_3'].tail(3))
```

=====
Daily stock returns grouped by quarter - 2022Q3
=====

First 3 days

	NOV	JBL	BIIB	STZ	EXPD	FIS	MSI	DVN	ALV	PPL	...	LRCX	TA
2022-07-01	-0.000257	-0.001019	0.014018	0.004394	0.000534	0.008956	0.000993	0.000000	0.006744	0.009970	...	-0.033152	0.00875
2022-07-05	-0.024038	-0.004873	0.004002	0.002355	-0.004834	0.006541	0.004422	-0.026147	-0.007959	-0.014799	...	-0.006717	-0.00219
2022-07-06	-0.000816	0.002229	0.006891	0.004980	0.004522	-0.000824	0.009493	-0.007513	-0.001829	0.003547	...	0.003250	-0.00086

3 rows × 278 columns

Last 3 days

	NOV	JBL	BIIB	STZ	EXPD	FIS	MSI	DVN	ALV	PPL	...	LRCX	T
2022-09-28	0.010403	0.019956	0.145664	0.008683	0.004899	0.002818	0.007555	0.029009	0.009872	0.003987	...	0.005670	0.0061
2022-09-29	0.014876	-0.007285	-0.019804	-0.006933	0.002021	-0.004121	-0.008249	0.006603	-0.019909	-0.022979	...	-0.012818	-0.0013
2022-09-30	0.011974	-0.010483	0.004447	-0.007480	-0.010495	-0.005709	-0.002571	-0.006025	-0.003376	-0.010159	...	-0.005366	-0.0016

3 rows × 278 columns


```
In [27]: def calculate_performance_cluster_by_quarter(df, cluster_name, model):
    """
    Helper method to calculate the sharpe ratio for each cluster in
    each of the quarter. Sharpe ratio is our evaluation metric for
    evaluation of our clusters. https://www.investopedia.com/terms/s/sharperatio.asp
    params:
    df : dataframe that includes cluster number for all the data.
    cluster_name: name of the column that has the cluster assigned
    model: model for which the sharpe ratio are being calculated.

    returns:
    new_df : A dataframe with the calculations such as sharpe ratio, quarter, volatility

    """
    key_list = []
    cluster_list = []
    quarter_return_list = []
    quarter_vol_list = []
    quarter_sharpe_list = []

    for key in df['key'].unique().tolist():
        temp = df[['key', 'stock', cluster_name]][df['key'] == key].copy()

        for cluster in temp[cluster_name].unique().tolist():
            temp2 = temp[temp[cluster_name] == cluster]
            a = [daily_rets_by_quarter[key][m] for m in temp2['stock']]
            daily_ret_df_ = pd.DataFrame(a)
            daily_ret_clus_ = daily_ret_df_.sum()
            quarter_ret_clus_ = (daily_ret_clus_.sum()) / len(daily_ret_clus_) * 63 # 63 trading days in
            clust_vol = (daily_ret_clus_.std()) * (252**0.5) # 252 trading days in a year
            clust_sharpe = quarter_ret_clus_ / clust_vol
            key_list.append(key)
            cluster_list.append(cluster)
            quarter_return_list.append(quarter_ret_clus_)
            quarter_vol_list.append(clust_vol)
            quarter_sharpe_list.append(clust_sharpe)

    new_df = pd.DataFrame({'key':key_list,
                           'cluster':cluster_list,
                           'quarter_return':quarter_return_list,
                           'quarter_vol': quarter_vol_list,
                           'quarter_sharpe': quarter_sharpe_list})
```

```

        'quarter_sharpe': quarter_sharpe_list})

new_df['model'] = model

return new_df

```

In [28]: *Compute S&P 500 Performance Across Quarters*

```

sp500_quarter_sharpe = {}
for k, v in sp500_daily_rets_by_quarter.items():
    rets_ = (v.sum()) / len(v) * 63
    vol_ = (v.std()) * (252 ** 0.5)
    sharpe_ = rets_ / vol_
    sp500_quarter_sharpe[k] = sharpe_

sp500_performance_df = pd.DataFrame(sp500_quarter_sharpe).T
sp500_performance_df.head()

```

Out[28]:

S&P500	
2002_1	-0.039426
2002_2	-0.735674
2002_3	-0.538130
2002_4	0.276784
2003_1	-0.157044

4. Exploratory Data Analysis

In this section we used dimensionality reduction techniques to do our EDA. PCA was used for looking at each features. TSNE and UMAP was used for looking at quarterly fundamental data for quarters. This helped us in coming up in understanding how different stocks are organized in the 2-D space. Since we had many stocks in our universe, and the data changes every quarter. We explored opportunities to visualize the stocks. We created a tool based on d3.js to visualize the stocks. The tool can be seen here <https://dashing-crisp-38c329.netlify.app/> (<https://dashing-crisp-38c329.netlify.app/>)

4.1 By Features Data

In [29]: *####----- Group & Create dict of quarters -----###*

```
clustering_all_data = updated_shortlist_modified_clustering_agg.copy()  
by_quarter_grouping_full = clustering_all_data.groupby(clustering_all_data['key'])  
by_quarter_dict_full = dict(list(by_quarter_grouping_full))
```

```
In [30]: ##### Create dict of factors across stocks and quarters #####
by_factors_dict_full = {}
for i in clustering_all_data.columns[:-3]:
    by_factors_dict_full[i] = {}

for k, v in by_quarter_dict_full.items():
    for i in by_factors_dict_full.keys():
        by_factors_dict_full[i][k] = v[i].reset_index(drop=True)

##### Create dict of df of stocks across quarters
by_factors_dict_df = {}

for k, v in by_factors_dict_full.items():
    by_factors_dict_df[k] = pd.DataFrame(v).T
    by_factors_dict_df[k].columns = list(by_quarter_dict_full['2002_1']['stock'])

print(f'Number of features: {len(by_factors_dict_df)}\n')
print(f'Feature bal_ppeq across stocks and quarters:\n')
display(by_factors_dict_df['bal_totalAssets'].head(3))
```

Number of features: 17

Feature bal_ppeq across stocks and quarters:

	TXT	LII	FIS	HES	AMGN	CHDN	AMAT	ADBE	SRE	IFF	...	WTRG	DVN
2002_1	-0.059803	-0.436607	1.702950	0.100951	1.289773	-0.378708	0.228661	0.245140	1.625103	0.465425	...	-0.078917	0.155054
2002_2	-0.061980	-0.435668	1.739782	0.087863	1.165229	-0.380248	0.208817	0.233325	1.625640	0.600173	...	-0.085521	0.144793
2002_3	-0.061075	-0.441381	1.784387	0.048798	1.162459	-0.424986	0.207941	0.223431	1.606984	0.612695	...	-0.090303	0.104767

3 rows × 278 columns

```
In [31]: by_factors_dict_df['bal_totalAssets']
```

Out[31]:

	TXT	LII	FIS	HES	AMGN	CHDN	AMAT	ADBE	SRE	IFF	...	WTRG
2002_1	-0.059803	-0.436607	1.702950	0.100951	1.289773	-0.378708	0.228661	0.245140	1.625103	0.465425	...	-0.078917
2002_2	-0.061980	-0.435668	1.739782	0.087863	1.165229	-0.380248	0.208817	0.233325	1.625640	0.600173	...	-0.085521
2002_3	-0.061075	-0.441381	1.784387	0.048798	1.162459	-0.424986	0.207941	0.223431	1.606984	0.612695	...	-0.090303
2002_4	-0.063450	-0.449437	1.833374	0.069066	1.218116	-0.426550	0.219163	0.259189	1.525660	0.610179	...	-0.096486
2003_1	-0.079223	-0.450805	1.798295	0.040092	1.326322	-0.426778	0.181116	0.228180	1.481196	0.626970	...	-0.108409
...
2021_3	-0.050673	-0.465764	-0.490533	-0.116139	0.188197	-0.497769	-0.227457	-0.480081	0.030086	-0.446079	...	-0.461881
2021_4	-0.072297	-0.468215	-0.490983	-0.135954	0.180474	-0.497603	-0.221808	-0.481104	0.061349	-0.447719	...	-0.462294
2022_1	-0.064156	-0.466126	-0.491660	-0.125637	0.160258	-0.497694	-0.222230	-0.479599	-0.025402	-0.447922	...	-0.463989
2022_2	-0.058786	-0.463109	-0.492233	-0.118994	-0.252538	-0.496767	-0.232197	-0.478178	-0.027127	-0.446220	...	-0.465022
2022_3	-0.057796	-0.466059	-0.492532	-0.110514	-0.254593	-0.497902	-0.233527	-0.482848	-0.022434	-0.447531	...	-0.466247

83 rows × 278 columns

4.2 PCA by features

In [32]: #####

BY FEATURES PCA ###
#####

```
# Fit PCA model to each feature and store in dictionary
pca_by_features = {}

for k, v in by_factors_dict_df.items():
    # Standardize factors
    scaler_feat = StandardScaler()
    scaled_data_feat = scaler_feat.fit_transform(v)

    # Fit PCA model
    pca_by_features[k] = PCA()
    pca_by_features[k].fit_transform(scaled_data_feat)

# Scree plot across all currencies

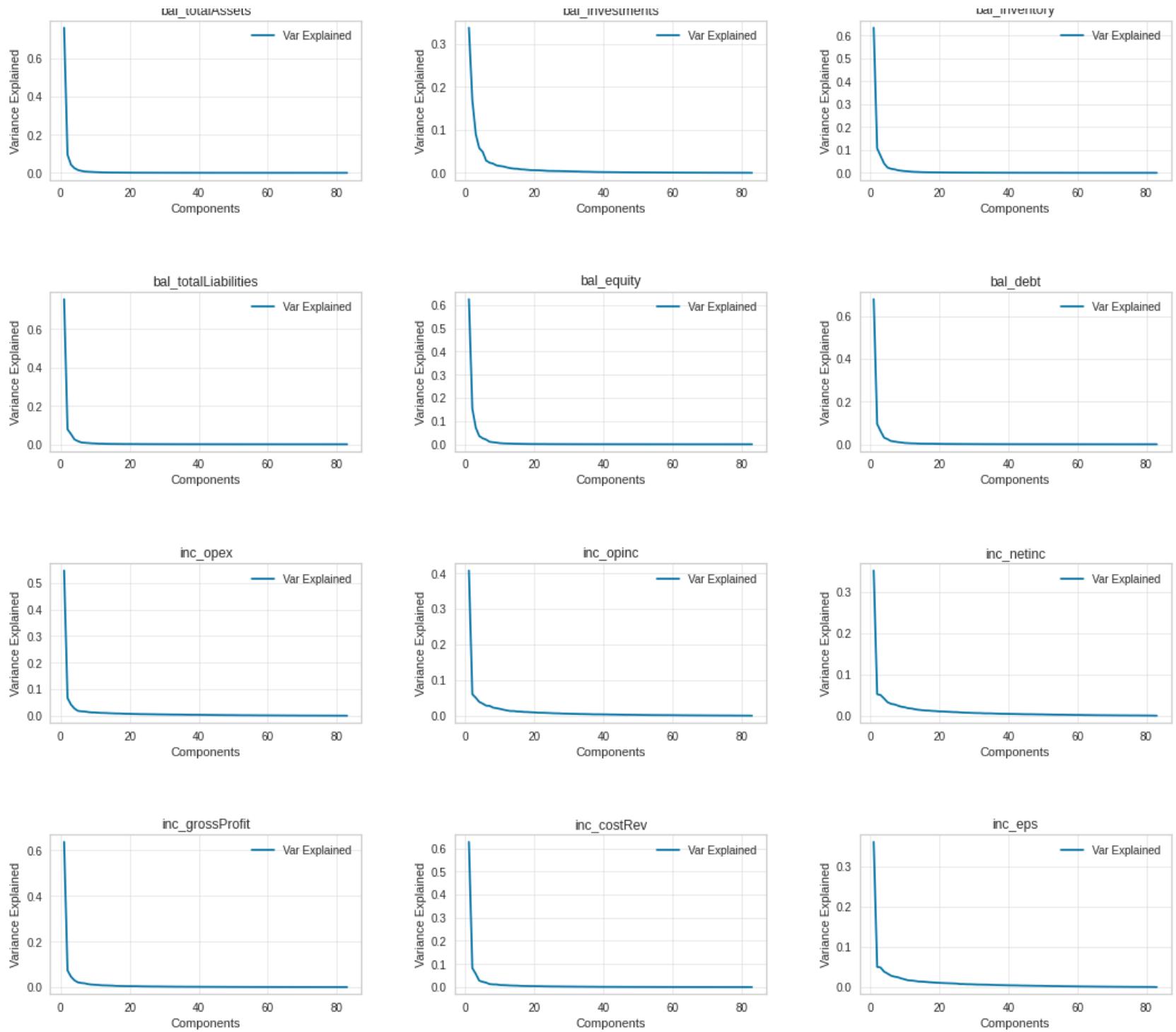
fig, axes = plt.subplots(6, 3, figsize=(18, 25))

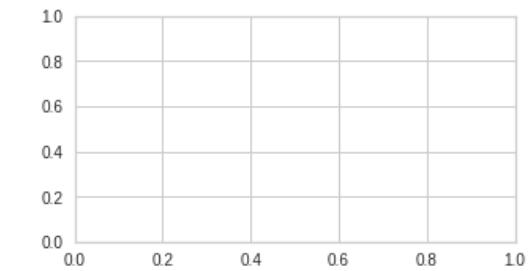
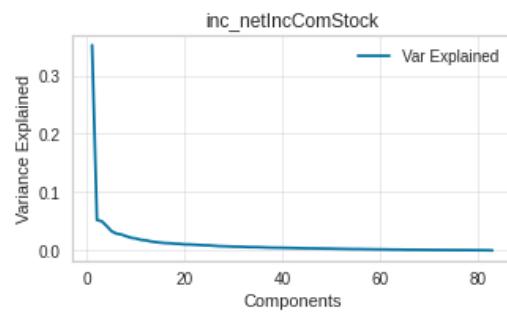
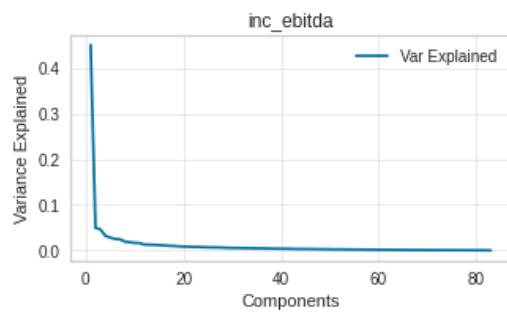
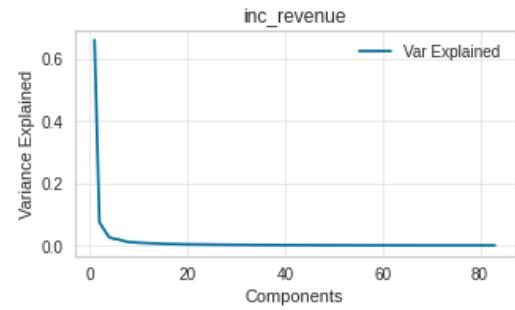
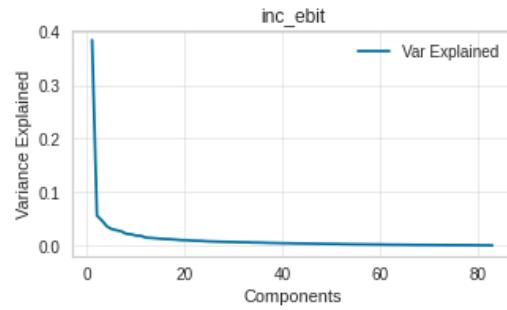
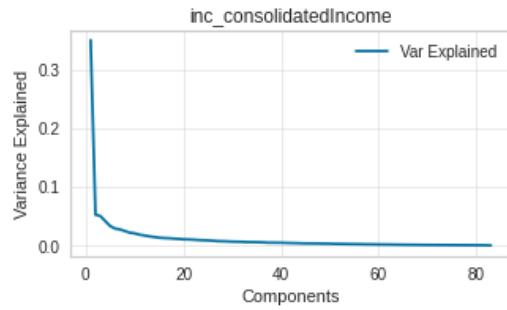
for i, ((k, v), axes) in enumerate(zip(pca_by_features.items(), axes.flat)):
    axes.plot(np.arange(1, len(v.explained_variance_ratio_)+1), v.explained_variance_ratio_, label='Var'
    axes.set_title(k)
    axes.legend()
    axes.set_xlabel('Components')
    axes.set_ylabel('Variance Explained')
    axes.grid(':', alpha=0.4)

plt.suptitle('PCA Across Features', fontsize=20)
plt.subplots_adjust(wspace=0.3, hspace=0.7)
plt.show()
```

PCA Across Features

bal_treasury
bal_inventories
bal_inventories





4.3 - TSNE - Quarter visualization for stocks

```
In [33]: # Standardize factors
tsne_list = ['2002_1', '2009_1', '2015_1', '2022_1']
#tsne_scaled_data = {}
#for i in tsne_list:
#    tsne_scaler = StandardScaler()
#    tsne_scaled_data[i] = tsne_scaler.fit_transform(by_quarter_dict_clean[i])

tsne_data = {}
for i in tsne_list:
    tsne_data[i] = by_quarter_dict_clean[i]

# Fit t-SNE model
tsne_fit = {}
for k, v in tsne_data.items():
    tsne_ = TSNE(n_components=2, perplexity=15)
    tsne_fit[k] = tsne_.fit_transform(v)

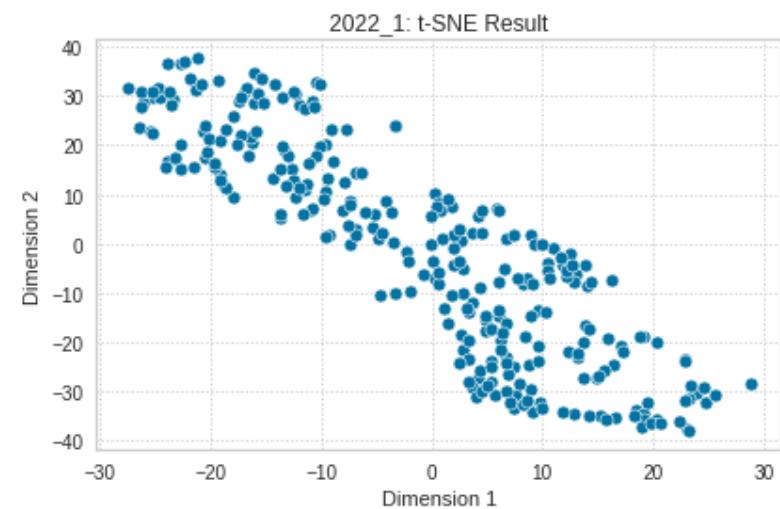
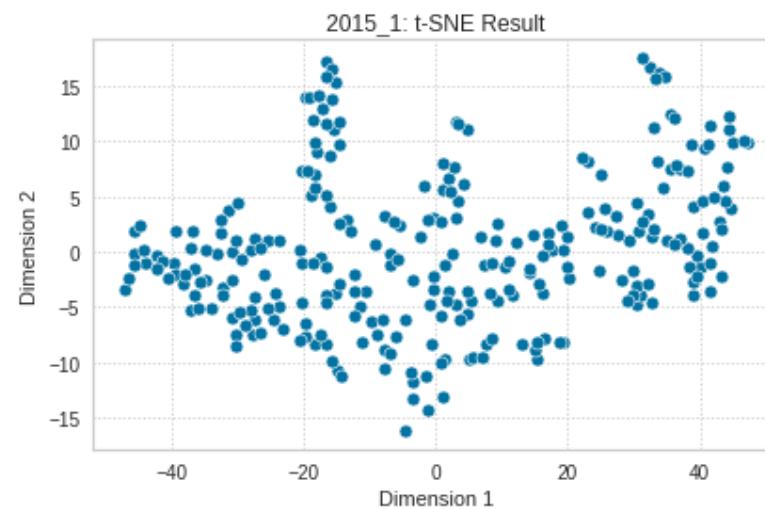
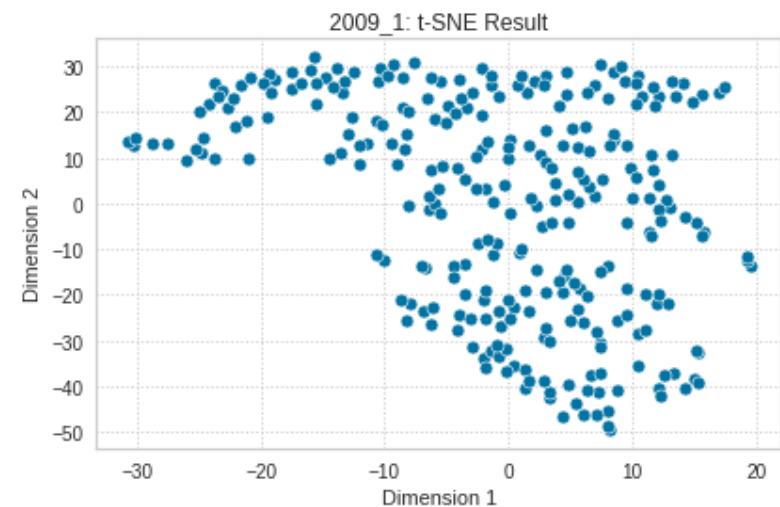
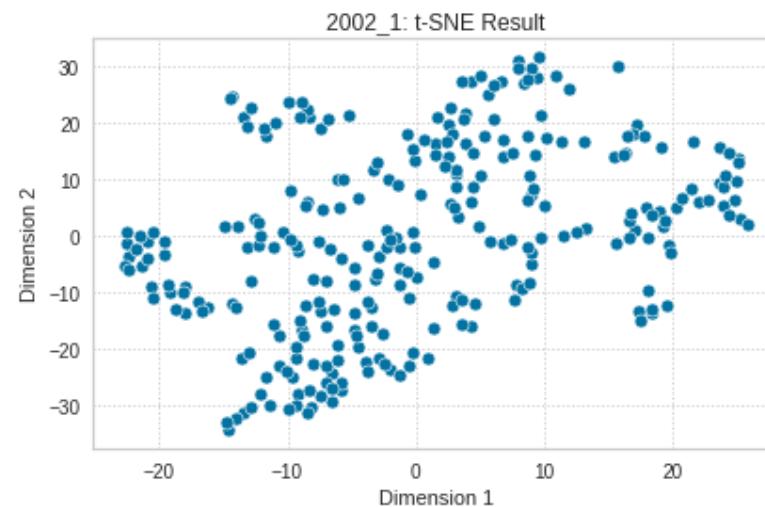
tsne_df = {}
for k, v in tsne_fit.items():
    df_tsne_ = pd.concat([by_quarter_dict_full[k]['stock'].reset_index(drop=True), pd.DataFrame(v)], axis=1)
    df_tsne_.columns=['Stocks', 'Dimension 1', 'Dimension 2']
    tsne_df[k] = df_tsne_

# Plot t-SNE
fig, ax = plt.subplots(2, 2, figsize=(15, 10))

for ((k,v), axes) in zip(tsne_df.items(), ax.flat):
    sns.scatterplot(data=v, x=v['Dimension 1'], y=v['Dimension 2'], ax=axes)
    axes.set_title(f'{k}: t-SNE Result')
    #ax.legend(bbox_to_anchor=(1.02, 1))
    axes.grid(ls=':')

plt.suptitle('t-SNE Visualization for Q1 of 2002/2009/2015/2022', fontsize=20)
plt.subplots_adjust(wspace=0.3, hspace=0.5)
plt.show()
```

t-SNE Visualization for Q1 of 2002/2009/2015/2022



4.4 - UMAP - Quarter visualization for stocks

This section shows visualization for UMAP. For visualization across all quarters please see our tool at <https://dashing-crisp-38c329.netlify.app/> (<https://dashing-crisp-38c329.netlify.app/>)

```
In [34]: random_state =1  
viz_path =f"~/Documents/__fall2022/advanced_machinelearning/hw/vizualization/hw/week16/data/"  
viz_path  
  
Out[34]: '~/Documents/__fall2022/advanced_machinelearning/hw/vizualization/hw/week16/data/'
```

4.4.1 Create PCA, UMAP, And TSNE features reduced to 2 dimension for Visualization

```
In [35]: umap_obj = umap.UMAP(random_state=random_state)
tsne_obj = TSNE(n_components=2, random_state=random_state)
pca_obj = PCA(n_components=2, random_state=random_state )

scaled_df = pd.DataFrame()

for key, val in by_quarter_dict.items():
    scaled_features = val.iloc[:, :-3]
    umap_fit = umap_obj.fit_transform(scaled_features)
    tsne_fit = tsne_obj.fit_transform(scaled_features)
    pca_fit = pca_obj.fit_transform(scaled_features)
    temp = pd.DataFrame(scaled_features, columns=val.columns[:-3], index=val.index)
    temp['key'] = val['key']
    temp['stock'] = val['stock']
    temp['umap_D1'] = umap_fit[:, 0]
    temp['umap_D2'] = umap_fit[:, 1]
    temp['tsne_D1'] = tsne_fit[:, 0]
    temp['tsne_D2'] = tsne_fit[:, 1]
    temp['pca_D1'] = pca_fit[:, 0]
    temp['pca_D2'] = pca_fit[:, 1]

scaled_df = pd.concat([scaled_df, temp], axis=0)
```

```
In [36]: print(f"Shape of our scaled df is {scaled_df.shape}")
scaled_df.head()
```

Shape of our scaled df is (23074, 25)

Out[36]:

	bal_totalAssets	bal_investments	bal_inventory	bal_totalLiabilities	bal_equity	bal_debt	inc_opex	inc_opinc	inc_netinc	inc_grc
0	-0.059803	-0.113510	0.807935	-0.108007	0.087025	-0.159333	-0.301192	-0.052917	0.009751	-0.000100
83	-0.436607	-0.174458	-0.126406	-0.345833	-0.566396	-0.282938	-0.333741	-0.181527	-0.106709	-0.000100
166	1.702950	-0.175277	-0.352391	0.788463	3.652767	0.922843	0.243575	0.138337	0.043386	0.000100
249	0.100951	-0.175277	-0.267882	0.053569	0.160714	0.228835	0.036632	0.769361	0.416170	0.000100
332	1.289773	0.040361	1.093685	1.823042	-0.200428	2.317625	1.209765	2.655561	2.697719	0.000100

5 rows × 25 columns

```
In [37]: scaled_df.isna().sum()
```

```
Out[37]: bal_totalAssets      0  
bal_investments      0  
bal_inventory        0  
bal_totalLiabilities 0  
bal_equity           0  
bal_debt             0  
inc_opex             0  
inc_opinc            0  
inc_netinc           0  
inc_grossProfit      0  
inc_costRev          0  
inc_eps              0  
inc_consolidatedIncome 0  
inc_ebit              0  
inc_revenue           0  
inc_ebitda            0  
inc_netIncComStock   0  
key                  0  
stock                0  
umap_D1               0  
umap_D2               0  
tsne_D1               0  
tsne_D2               0  
pca_D1               0  
pca_D2               0  
dtype: int64
```

Save the file as it takes some time to generate the pca, umap, and tsne data

```
In [38]: scaled_df.to_csv("scaled_df.csv")
```

Read the file from csv

```
In [39]: scaled_df = pd.read_csv("scaled_df.csv", index_col=0)
```

In [40]: `scaled_df.head(2)`

Out[40]:

	bal_totalAssets	bal_investments	bal_inventory	bal_totalLiabilities	bal_equity	bal_debt	inc_opex	inc_opinc	inc_netinc	inc_g
0	-0.059803	-0.113510	0.807935	-0.108007	0.087025	-0.159333	-0.301192	-0.052917	0.009751	
83	-0.436607	-0.174458	-0.126406	-0.345833	-0.566396	-0.282938	-0.333741	-0.181527	-0.106709	

2 rows × 25 columns

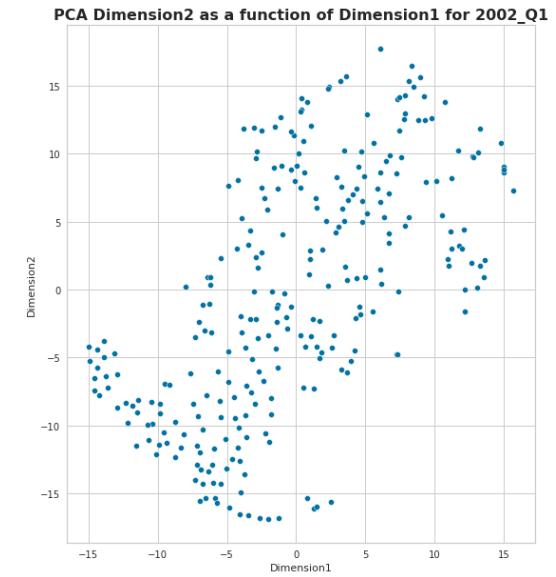
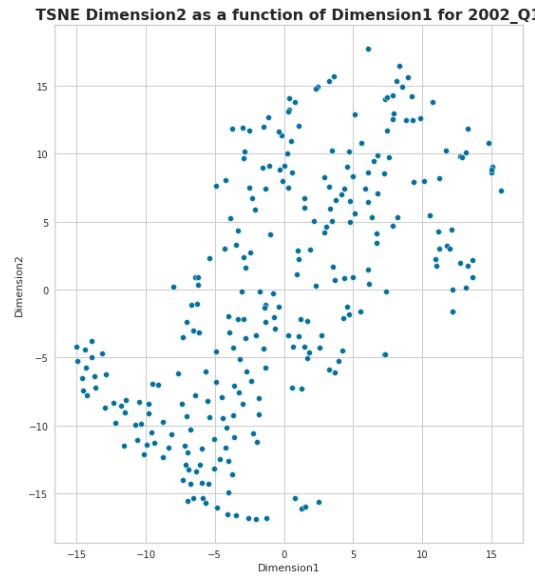
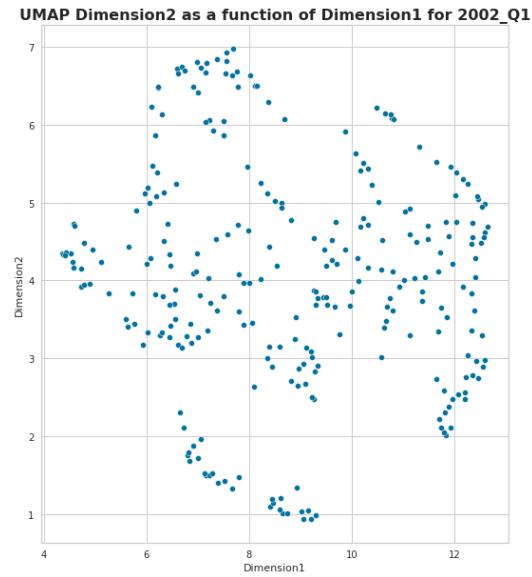
```
In [41]: fig, axs = plt.subplots(1,3, figsize=(30,10))
axs = axs.ravel()
sns.set_theme(style="ticks")

sns.scatterplot(x ='umap_D1',
                y='umap_D2',
                data = scaled_df[scaled_df['key'] =='2002_1'],
                ax = axs[0],
                legend= True,
                )
axs[0].set_xlabel('Dimension1')
axs[0].set_ylabel('Dimension2')
axs[0].set_title("UMAP Dimension2 as a function of Dimension1 for 2002_Q1", fontsize=16,fontweight='b'

sns.scatterplot(x ='tsne_D1',
                y='tsne_D2',
                data = scaled_df[scaled_df['key'] =='2002_1'],
                ax = axs[1],
                legend= True,
                )
axs[1].set_xlabel('Dimension1')
axs[1].set_ylabel('Dimension2')
axs[1].set_title("TSNE Dimension2 as a function of Dimension1 for 2002_Q1", fontsize=16,fontweight='b

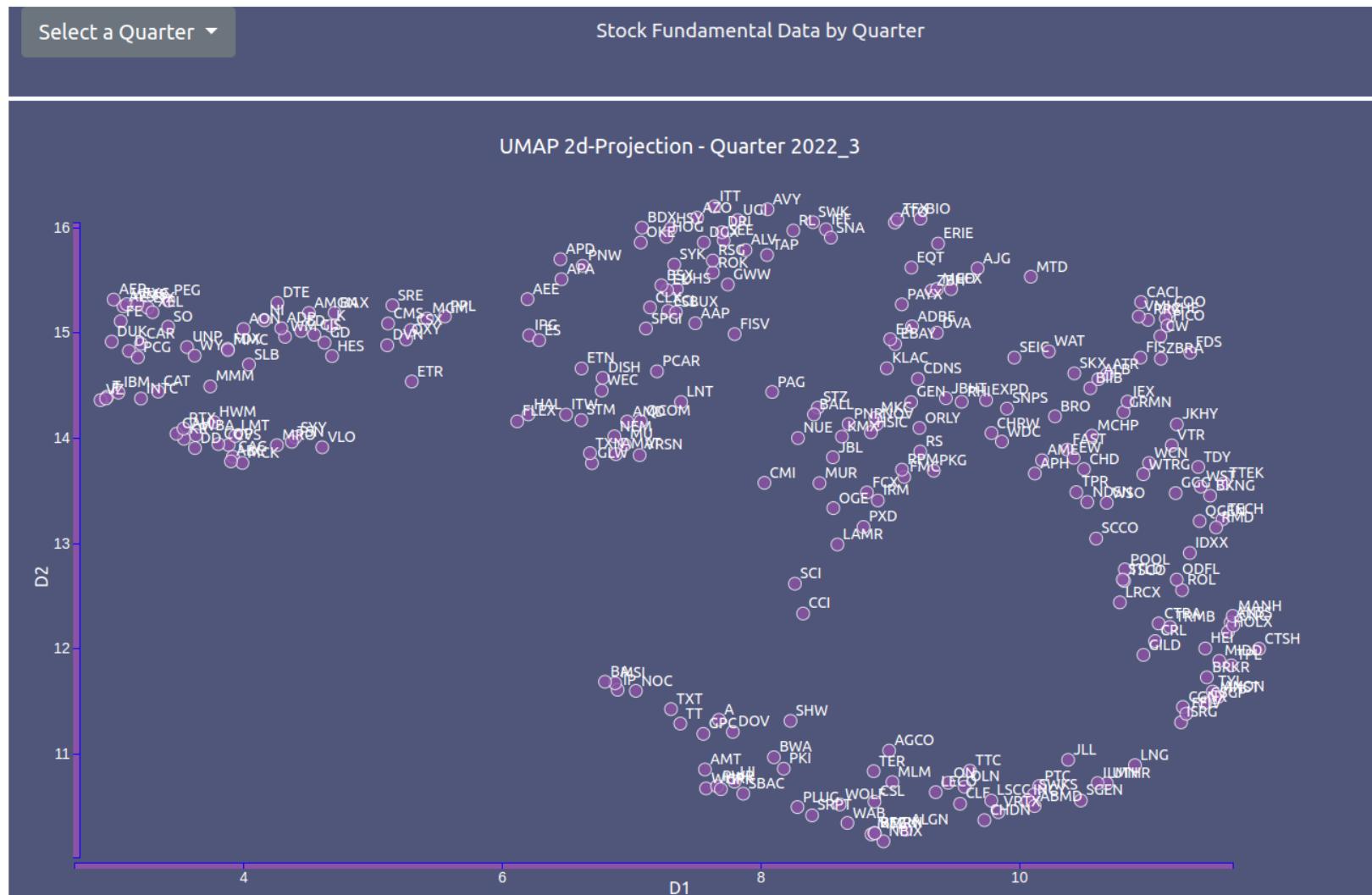
sns.scatterplot(x ='pca_D1',
                y='pca_D2',
                data = scaled_df[scaled_df['key'] =='2002_1'],
                ax = axs[2],
                legend= True,
                )
axs[2].set_xlabel('Dimension1')
axs[2].set_ylabel('Dimension2')
axs[2].set_title("PCA Dimension2 as a function of Dimension1 for 2002_Q1", fontsize=16,fontweight='b

plt.show()
```



UMAP seems to create better groups for our 278 stocks using fundamental data features. We will use our data reduced to 2 dimension by UMAP for visualization of our clusters. We have 83 quarters in total. Results may differ for each quarter.

UMAP Visualization for 2022-Quarter 3



5. Unsupervised learning - Clustering Models

In this section we trained several clustering models.

Our approach

Train a model on the first quarter data. Fine tune it to come up with the best parameters for the models. Select the model, and train each quarterly data using the same parameters. Since this is time series, and we want to cluster our data for each quarter based on the fundamentals, to find the best cluster for our portfolio, we have to train quarterly data separately.

Models Tried

1. Kmeans Clustering
2. Hierarchical Clustering
3. DBSCAN
4. Gaussian Mixture Model

Model Selection Methodology We used several methods for our model selection. For K-Means we used elbow method, and silhouette score graphs. For Hierarchical clustering we used silhouette score chart. For GMM we used BIC, AIC and Log likelihood score to select our models.

DBSCAN resulted in giving us only two cluster. We did not select the model.

5.1 KMeans Clustering

5.1.1 Helper method

```
In [42]: def plot_helper(min_k, max_k, y_list, y_label, title):  
    """  
        Helper function to plot Elbow plot, average silhouette.  
        :param max_k: max clusters  
        :param y_list: values to be plotted on y axis  
        :param ylabel = label of y axis  
        :title: title of the plot  
    """  
    fig, ax = plt.subplots(figsize=(14,6))  
    ax.plot(range(min_k,(max_k)), y_list, 'o--')  
    ax.set_xlabel('Number of clusters $k$')  
    ax.set_ylabel(y_label)  
    ax.set_xticks(list(range(min_k,max_k)))  
    ax.set_title(title)  
    plt.show()
```

```
In [43]: # subset our data to 2002_1 i.e. 2002 quarter 1 data to find the best cluster parameters.
#exclude features from the data frame that have features for visualization, and data analysis. i.e. P
#stock, and quarter.
exclude_features_for_training = -8
scaled_2002_q1 = scaled_df[scaled_df['key'] == '2002_1']
scaled_2002_q1.iloc[:, :exclude_features_for_training]
```

Out[43]:

	bal_totalAssets	bal_investments	bal_inventory	bal_totalLiabilities	bal_equity	bal_debt	inc_opex	inc_opinc	inc_netinc	inc_g
0	-0.059803	-0.113510	0.807935	-0.108007	0.087025	-0.159333	-0.301192	-0.052917	0.009751	
83	-0.436607	-0.174458	-0.126406	-0.345833	-0.566396	-0.282938	-0.333741	-0.181527	-0.106709	
166	1.702950	-0.175277	-0.352391	0.788463	3.652767	0.922843	0.243575	0.138337	0.043386	
249	0.100951	-0.175277	-0.267882	0.053569	0.160714	0.228835	0.036632	0.769361	0.416170	
332	1.289773	0.040361	1.093685	1.823042	-0.200428	2.317625	1.209765	2.655561	2.697719	
...
22659	0.947727	0.089969	-0.254441	0.922063	0.874507	1.161202	-0.042873	0.247470	0.184103	
22742	0.018363	-0.162121	0.973100	-0.027705	0.147175	-0.057799	0.000311	1.553663	1.692715	
22825	0.529986	-0.175277	1.323804	0.188791	1.283434	0.176041	-0.066839	0.486488	0.448403	
22908	6.468810	13.609920	4.272194	7.320910	3.356623	8.634956	1.513326	0.186437	-1.464567	
22991	-0.137057	-0.160217	-0.296761	-0.193181	0.038299	-0.083419	-0.111380	0.058171	0.053196	

278 rows × 17 columns

5.1.2 Train Kmeans with min cluster 3, and max cluster 20

```
In [44]: kmeans_model_list = []

max_k = 21
min_k =3
for i in range(min_k,max_k,1):
    km = KMeans(n_clusters=i, init='random', n_init=20, max_iter=1000, \
                random_state=random_state).fit(scaled_2002_q1.iloc[:, :exclude_features_for_training])

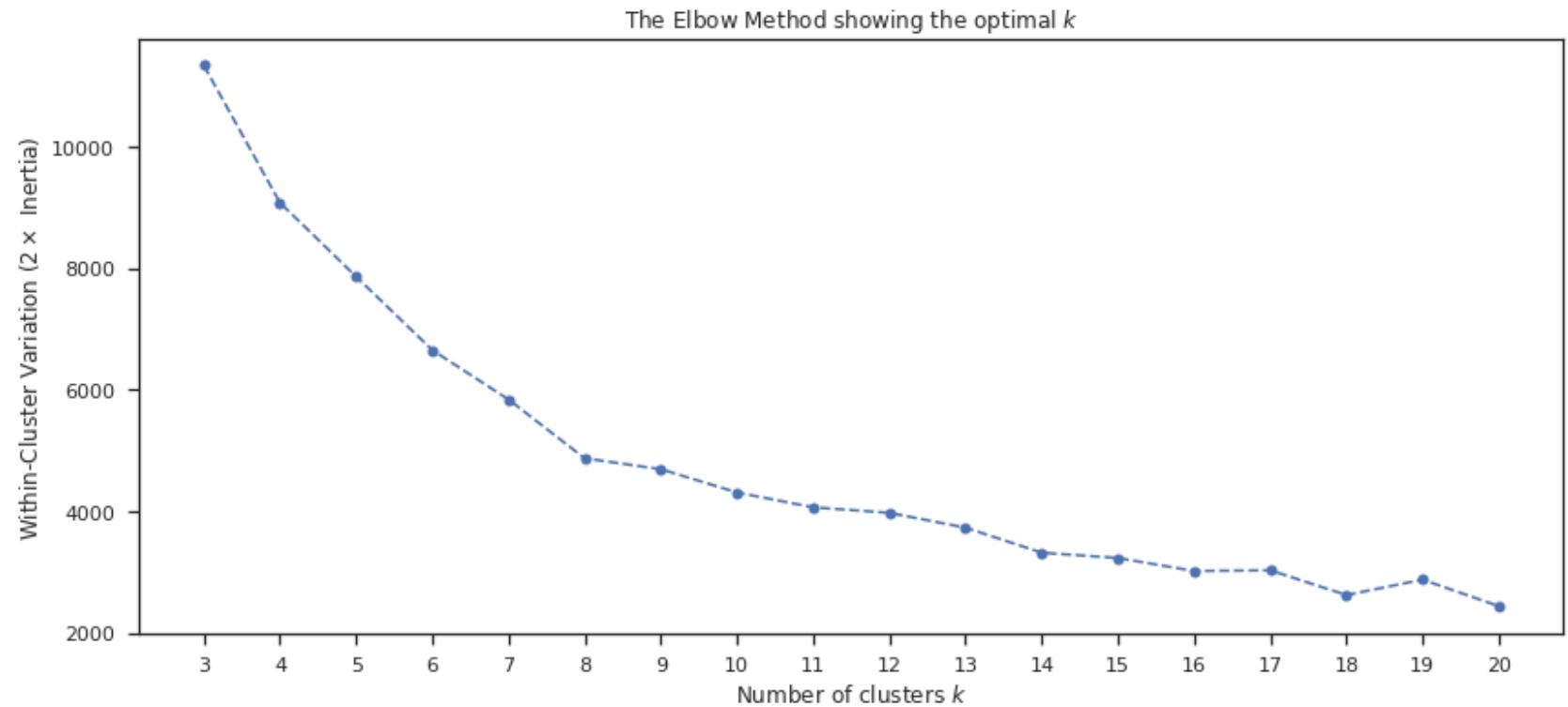
    kmeans_model_list.append(km)
```

5.1.3 Inertia,Elbow method, and silhouette score

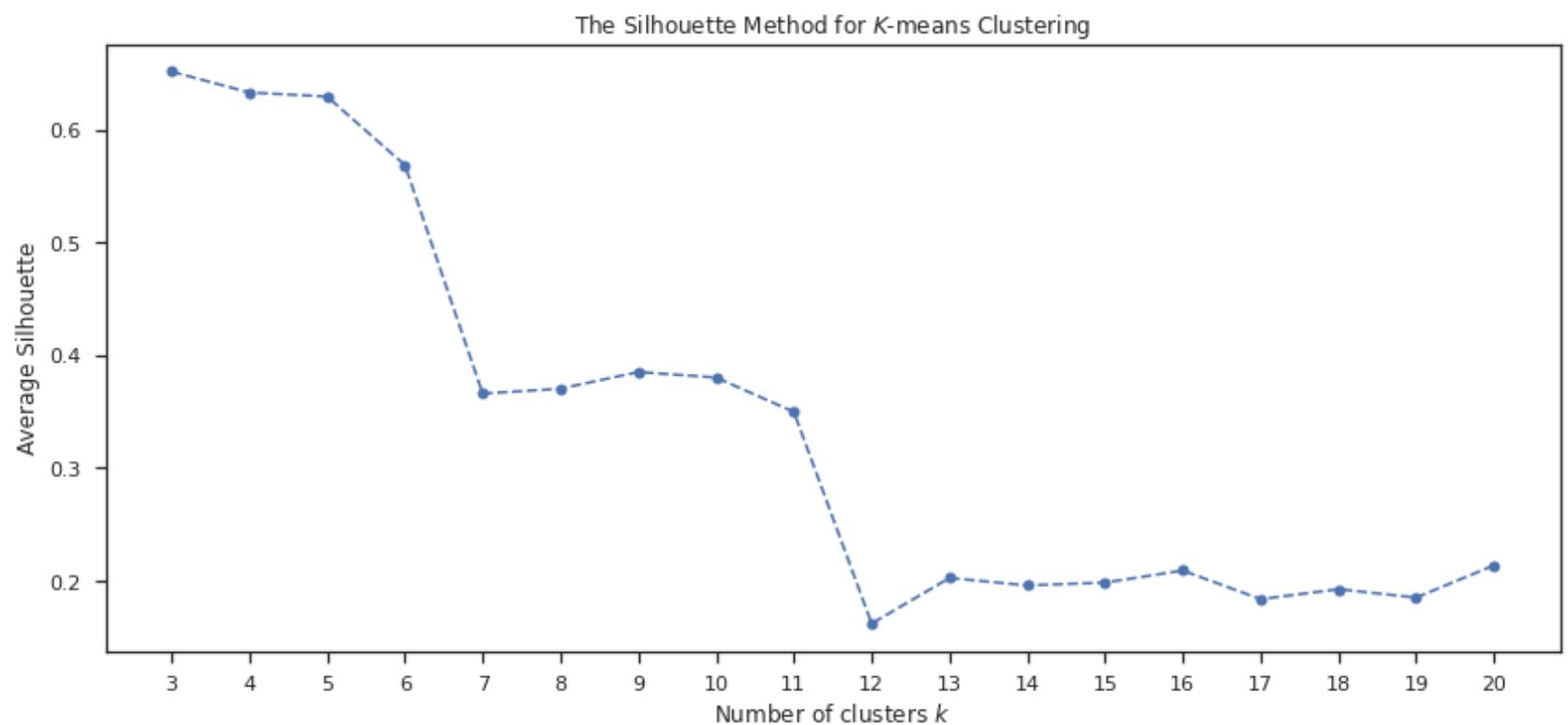
```
In [45]: kmeans_inertia_list = []
kmeans_silhouette_list = []

for kmeans in kmeans_model_list:
    inertia = 2*kmeans.inertia_
    sil_score = silhouette_score(scaled_2002_q1.iloc[:, :exclude_features_for_training], kmeans.labels_)
    kmeans_inertia_list.append(inertia)
    kmeans_silhouette_list.append(sil_score)
```

```
In [46]: plot_helper(min_k,max_k,kmeans_inertia_list, r'Within-Cluster Variation ($2\times$ Inertia)',\n                 'The Elbow Method showing the optimal $k$')
```

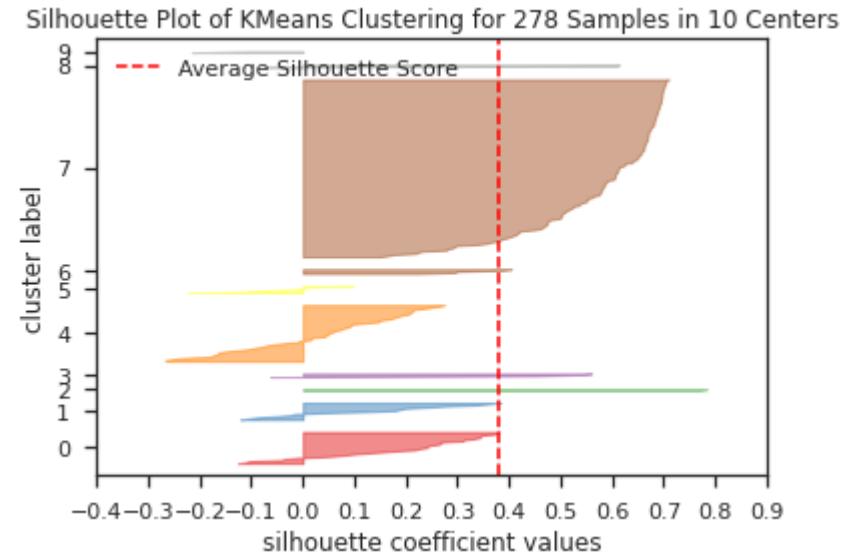


```
In [47]: plot_helper(min_k,max_k,kmeans_silhouette_list,"Average Silhouette",\n                  'The Silhouette Method for $K$-means Clustering')
```



```
In [48]: visualizer = SilhouetteVisualizer(kmeans_model_list[7], is_fitted=True)
visualizer.fit(scaled_2002_q1.iloc[:, :exclude_features_for_training])
visualizer.show()
plt.show()
```

```
/home/riku/anaconda3/envs/ml_final/lib/python3.9/site-packages/sklearn/base.py:409: UserWarning: X
does not have valid feature names, but KMeans was fitted with feature names
warnings.warn(
```



```
In [49]: print("distribution of stocks with 8 clusters")
pd.DataFrame(kmeans_model_list[5].labels_).value_counts()
```

distribution of stocks with 8 clusters

```
Out[49]: 1    170
2     69
7     20
6      8
4      5
0      3
3      2
5      1
dtype: int64
```

```
In [50]: print("distribution of stocks with 9 clusters")
pd.DataFrame(kmeans_model_list[6].labels_).value_counts()
```

distribution of stocks with 9 clusters

```
Out[50]: 6    165
1     61
2     20
8     13
3      6
7      5
0      3
4      3
5      2
dtype: int64
```

```
In [51]: print("distribution of stocks with 10 clusters")
pd.DataFrame(kmeans_model_list[7].labels_).value_counts()
```

distribution of stocks with 10 clusters

```
Out[51]: 7    159
4     51
0     29
1     16
5      7
6      5
3      4
8      3
2      2
9      2
dtype: int64
```

Elbow method suggests clusters 8 to 10 are appropriate. Silhouette plot suggest 6 clusters are appropriate. We experimented with this range. Our final choice was 10 clusters for Kmeans based on the distributions, and the charts we looked at using UMAP

Visualization using d3.js for our selected model for 2002 Quarter 1



5.1.4 - Train each quarter data with selected model parameters.

Store each quarter results in one dataframe for visualization and evaluation purposes Calculate the performance of each cluster based on sharpe ratio. The performance metric is used for evaluation of our clusters for each quarter

```
In [52]: scaled_df_final = pd.DataFrame()

selected_cluster =10
for key in scaled_df['key'].unique().tolist():
    temp = scaled_df[scaled_df['key'] == key].copy()
    km = KMeans(n_clusters=selected_cluster, init='random', n_init=20, max_iter=1000, \
                random_state=random_state).fit(temp.iloc[:, :exclude_features_for_training])

    temp['kmeans_cluster'] = km.labels_

    scaled_df_final = pd.concat([scaled_df_final, temp], axis=0)
```

```
In [53]: print("We see for 2002_1 kmeans assigned cluster for stocks TXT, LII, FIS\n")
scaled_df_final[['key','stock','kmeans_cluster']].head(3)
```

We see for 2002_1 kmeans assigned cluster for stocks TXT, LII, FIS

Out[53]:

	key	stock	kmeans_cluster
0	2002_1	TXT	7
83	2002_1	LII	7
166	2002_1	FIS	0

```
In [54]: #Calculate performance metric for each of the clusters for kmeans, and store it in df
performance_df = calculate_performance_cluster_by_quarter(scaled_df_final, 'kmeans_cluster', 'kmeans'
```

```
In [55]: print("Showing the performance of 2002 Qt 1. as an example. ")  
  
performance_df[performance_df['key'] =='2002_1'].sort_values(by='quarter_sharpe', ascending=False)  
  
Showing the performance of 2002 Qt 1. as an example.
```

Out[55]:

	key	cluster	quarter_return	quarter_vol	quarter_sharpe	model
7	2002_1	9	0.181148	0.208810	0.867524	kmeans
8	2002_1	3	0.201540	0.349963	0.575890	kmeans
2	2002_1	4	1.998277	3.857358	0.518043	kmeans
5	2002_1	5	0.235484	0.517667	0.454894	kmeans
3	2002_1	1	0.572772	1.424710	0.402027	kmeans
0	2002_1	7	3.611904	10.972934	0.329165	kmeans
4	2002_1	6	0.098935	0.531049	0.186301	kmeans
1	2002_1	0	0.197640	2.347955	0.084175	kmeans
6	2002_1	8	-0.062044	0.539330	-0.115038	kmeans
9	2002_1	2	-0.045945	0.200501	-0.229151	kmeans

5.2 Hierrarchical Clustering

5.2.1 - Train models with n_cluster = 3 to 30

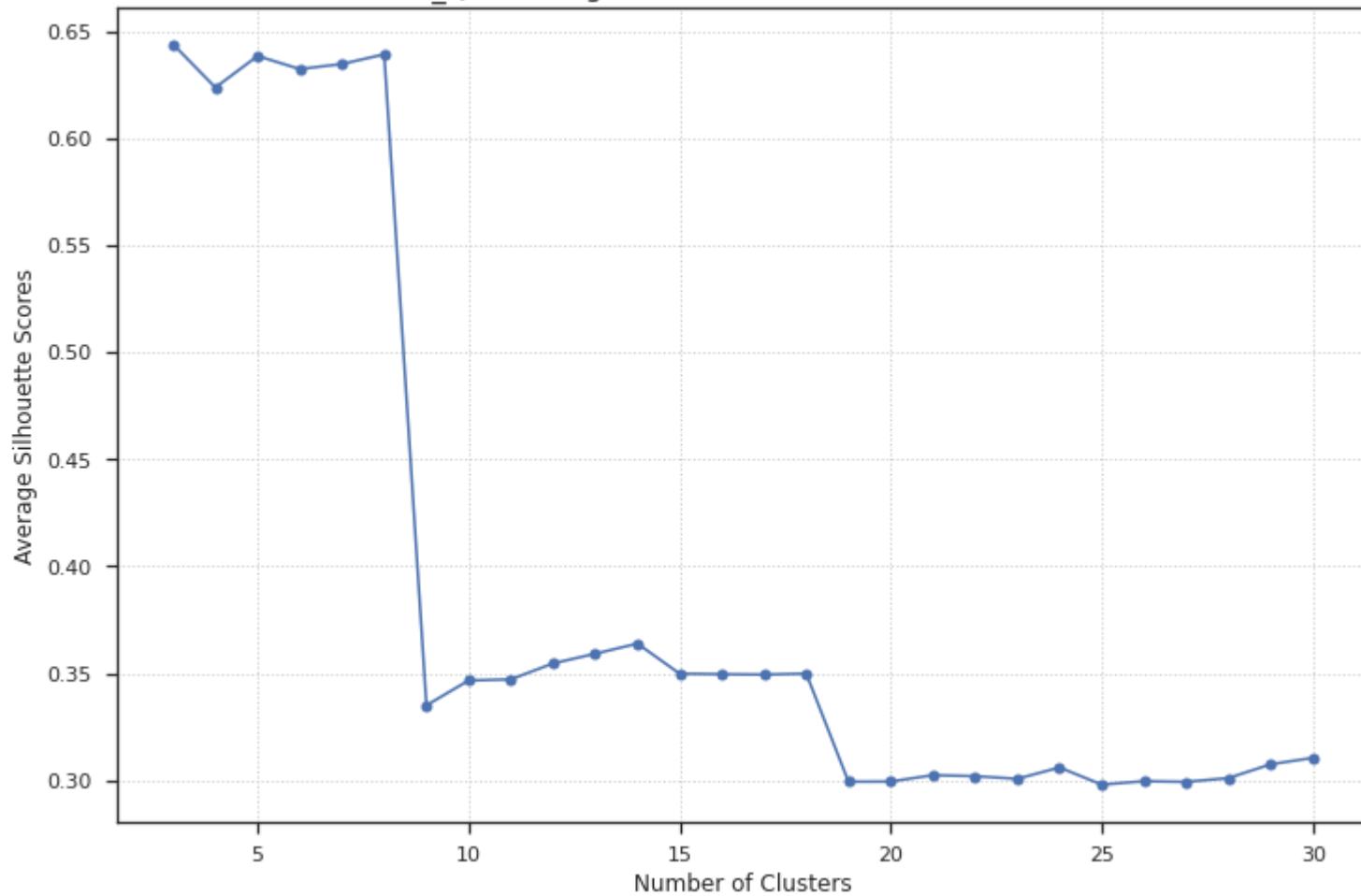
```
In [56]: hc_model_list = []  
  
for i in range(3, 31, 1):  
    hc_ = AgglomerativeClustering(n_clusters=i, metric='euclidean',  
                                    linkage='ward').fit\  
    (scaled_2002_q1.iloc[:, :exclude_features_for_training])  
    hc_model_list.append(hc_)
```

```
In [57]: hc_silhouette_list = []
hc_ncluster_list = []
for hc in hc_model_list:
    sil_score = silhouette_score(scaled_2002_q1.iloc[:, :exclude_features_for_training], \
                                  hc.labels_)
    hc_silhouette_list.append(sil_score)
    hc_ncluster_list.append(hc.n_clusters_)

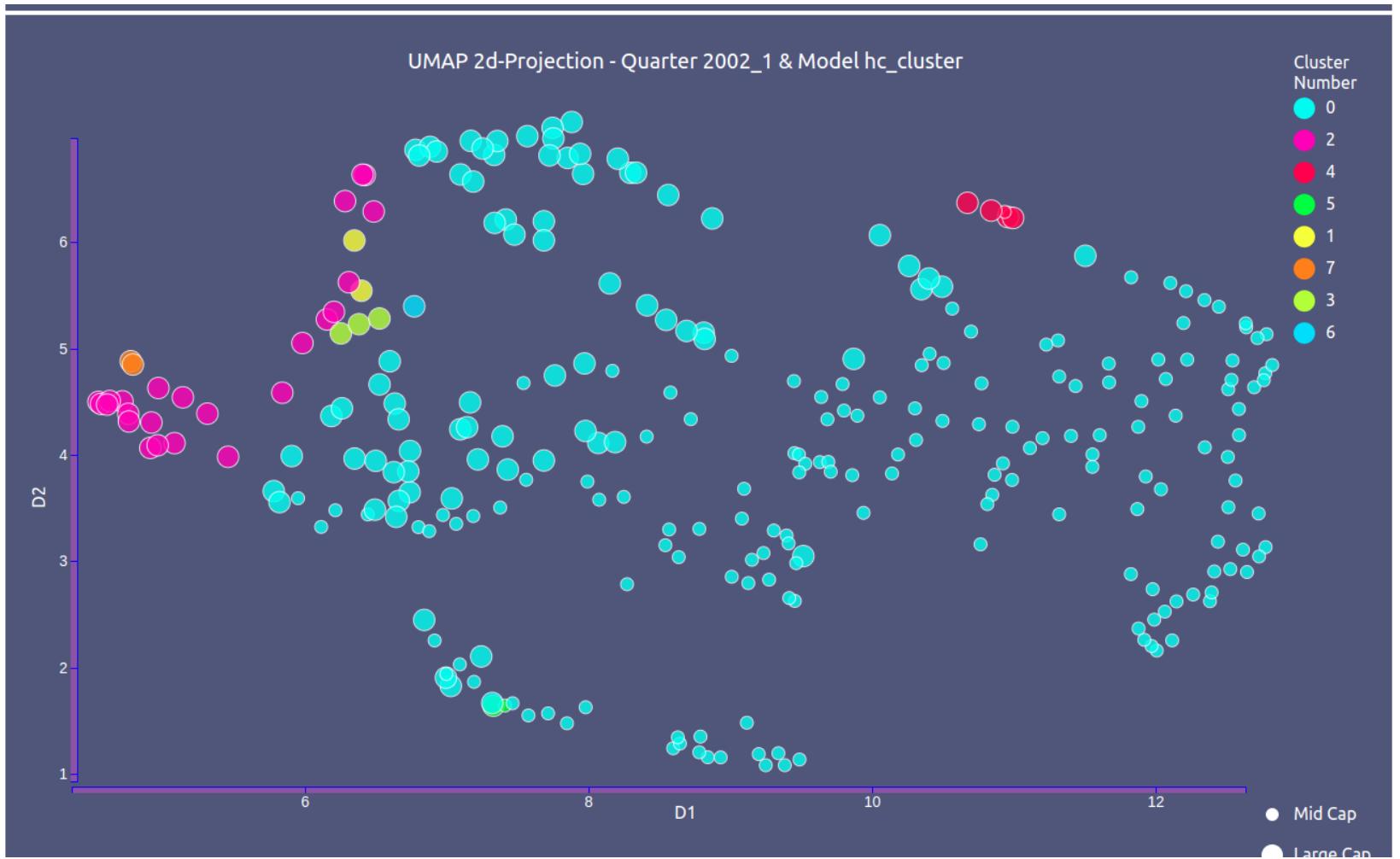
fig, ax = plt.subplots(1,1, figsize=(12, 8))
ax.plot(hc_ncluster_list, hc_silhouette_list, marker='o')
ax.set_xlabel('Number of Clusters')
ax.set_ylabel('Average Silhouette Scores')
ax.set_title('2002_Q1: Average Silhouette Scores Across K Clusters', fontsize=15)
ax.grid(ls=':')

plt.show()
```

2002_Q1: Average Silhouette Scores Across K Clusters



Based on silhouette score, and plot we select n_cluster =8 for our model



5.2.3 - Train each quarter data with selected model parameters.

Store each quarter results in one dataframe for visualization and evaluation purposes Calculate the performance of each cluster based on sharpe ratio. The performance metric is used for evaluation of our clusters for each quarter

```
In [58]: hc_temp = pd.DataFrame()

selected_cluster =8
for key in scaled_df['key'].unique().tolist():
    temp = scaled_df[scaled_df['key'] == key].copy()
    hc_ = AgglomerativeClustering(n_clusters=selected_cluster, metric='euclidean', \
                                    linkage='ward').fit(temp.iloc[:, :exclude_features_for_training])

    temp['hc_cluster'] = hc_.labels_

    hc_temp = pd.concat([hc_temp, temp], axis=0)
```

```
In [59]: scaled_df_final['hc_cluster'] = hc_temp['hc_cluster']
```

```
In [60]: print("We see for 2002_1 hc cluster assigned cluster for stocks TXT, LII, FIS\n")
scaled_df_final[['key','stock','hc_cluster']].head(3)
```

We see for 2002_1 hc cluster assigned cluster for stocks TXT, LII, FIS

Out[60]:

	key	stock	hc_cluster
0	2002_1	TXT	0
83	2002_1	LII	0
166	2002_1	FIS	0

```
In [61]: #Calculate performance metric for each of the clusters for kmeans, and store it in df
performance_df_temp = calculate_performance_cluster_by_quarter(scaled_df_final, 'hc_cluster', 'hc')
```

```
In [62]: t('*100 + '\nHierarchical Clustering Performance Example: Clusters Ranked by Sharpe Ratio - 2022 Q3\nperformance_df_temp[performance_df_temp['key'] =='2022_3'].sort_values(by='quarter_sharpe', ascending=False)\n\n=====\n=\nHierarchical Clustering Performance Example: Clusters Ranked by Sharpe Ratio - 2022 Q3\n=====\n=
```

Out[62]:

	key	cluster	quarter_return	quarter_vol	quarter_sharpe	model
663	2022_3	7	0.006798	0.216131	0.031451	hc
657	2022_3	4	-3.046241	19.447249	-0.156641	hc
656	2022_3	0	-0.248133	1.457766	-0.170214	hc

```
In [63]: #Concat the performance result of Hierarchical clustering to our performance df with existing Kmeans
```

```
In [64]: performance_df= pd.concat([performance_df.copy(), performance_df_temp.copy()], axis=0, ignore_index=True)
```

5.3 Gaussian Mixture Model (GMM)

5.3.1 - Train models with n_cluster = 3 to 30

```
In [65]: gmm_model_list = []\nfor i in range(3, 31, 1):\n    gmm_ = GMM(n_components=i, covariance_type='full', \n               random_state=random_state).fit(scaled_2002_q1.iloc[:, :exclude_features_for_training])\n    gmm_model_list.append(gmm_)
```

In [66]: *#calculate bic, aic, and log likelihood score of each model and store it in to a table*

```
bic_list = [model.bic(scaled_2002_q1.iloc[:, :exclude_features_for_training]) for model in gmm_model_
aic_list = [model.aic(scaled_2002_q1.iloc[:, :exclude_features_for_training]) for model in gmm_model_
score_list = [model.score(scaled_2002_q1.iloc[:, :exclude_features_for_training]) for model in gmm_mo
df_results_gmm = pd.DataFrame({
    'bic':bic_list,
    'aic':aic_list,
    'score': score_list,
    'components':np.arange(3,31,1)
})
```

In [67]: df_results_gmm

Out[67]:

	bic	aic	score	components
0	-3498.805448	-5356.147458	11.475085	3
1	-7104.211680	-9581.876901	19.690426	4
2	-6551.024440	-9649.012871	20.426282	5
3	-6161.765987	-9880.077629	21.456974	6
4	-5431.298211	-9769.933063	21.873980	7
5	-5116.663244	-10075.621307	23.038887	8
6	-5505.433175	-11084.714448	25.468911	9
7	-4643.258563	-10842.863046	25.649034	10
8	-4753.516231	-11573.443925	27.578137	11
9	-4341.732830	-11781.983734	28.568316	12
10	-3574.042480	-11634.616595	28.918375	13
11	-3340.427176	-12021.324501	30.229001	14
12	-2683.266380	-11984.486916	30.777854	15
13	-1992.282985	-11913.826731	31.265875	16
14	-1155.501107	-11697.368064	31.491669	17
15	-533.329426	-11695.519593	32.103453	18
16	334.571415	-11447.941962	32.273277	19
17	815.078923	-11587.757664	33.139852	20
18	707.381083	-12315.778715	35.064350	21
19	965.094384	-12678.388625	36.331634	22
20	1181.330535	-13082.475684	37.673517	23
21	1423.848532	-13460.280898	38.968131	24
22	1728.714424	-13775.738216	40.150608	25
23	2600.729316	-13524.046534	40.313033	26
24	3083.469963	-13661.629098	41.175592	27

	bic	aic	score	components
25	4014.789113	-13350.633158	41.231355	28
26	4916.241179	-13069.504302	41.340835	29
27	5469.943174	-13136.125518	42.075765	30

```
In [68]: fig, ax = plt.subplots(1, 3, figsize=(24,10))
axs = ax.ravel()

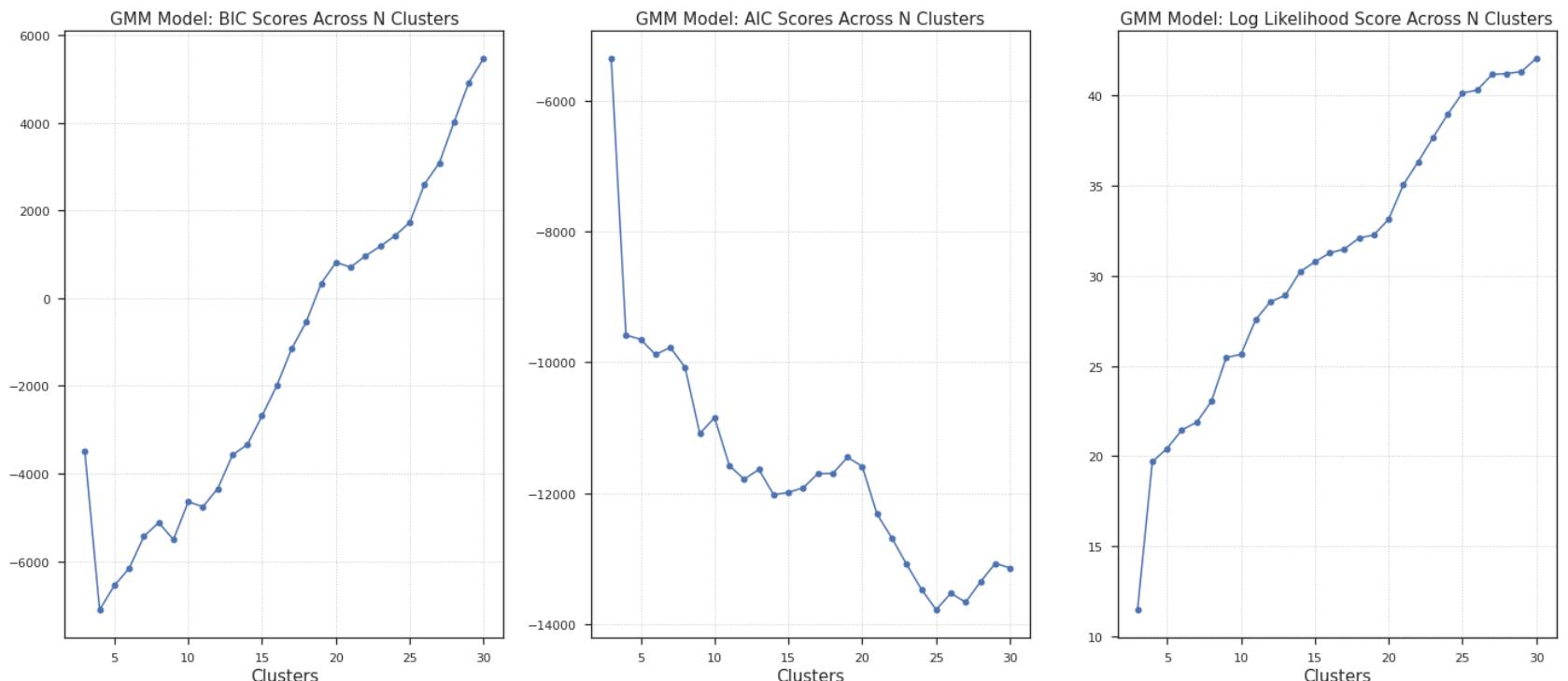
axs[0].plot(np.arange(3,31, 1), df_results_gmm['bic'] , label='BIC', marker='o')
axs[1].plot(np.arange(3,31, 1), df_results_gmm['aic'], label='AIC', marker='o')
axs[2].plot(np.arange(3,31, 1), df_results_gmm['score'], label='log likelihood', marker='o')

axs[0].set_title('GMM Model: BIC Scores Across N Clusters', fontsize=15)
axs[0].set_xlabel('Clusters', fontsize=15)
axs[0].grid(ls=':')

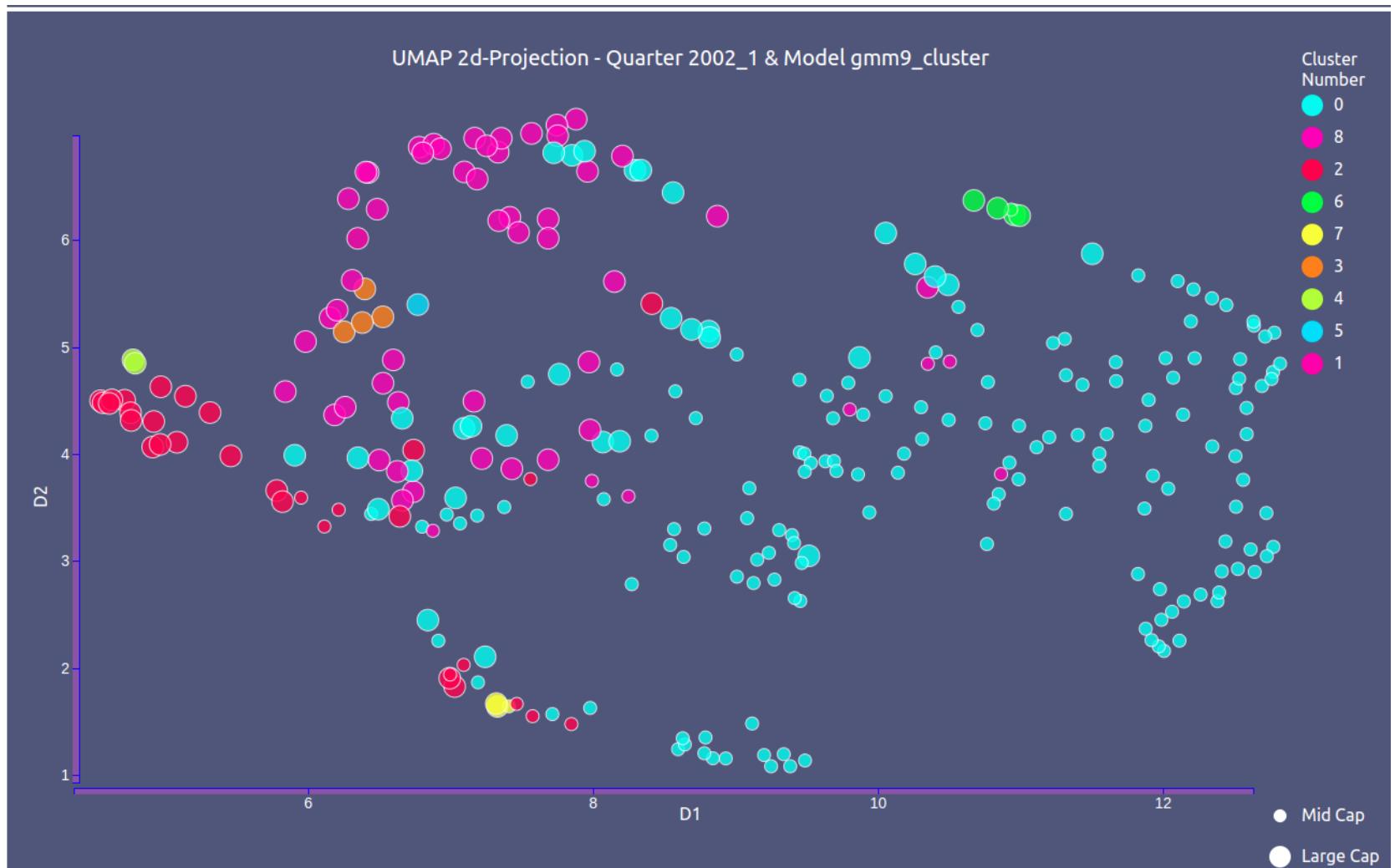
axs[1].set_title('GMM Model: AIC Scores Across N Clusters', fontsize=15)
axs[1].set_xlabel('Clusters', fontsize=15)
axs[1].grid(ls=':')

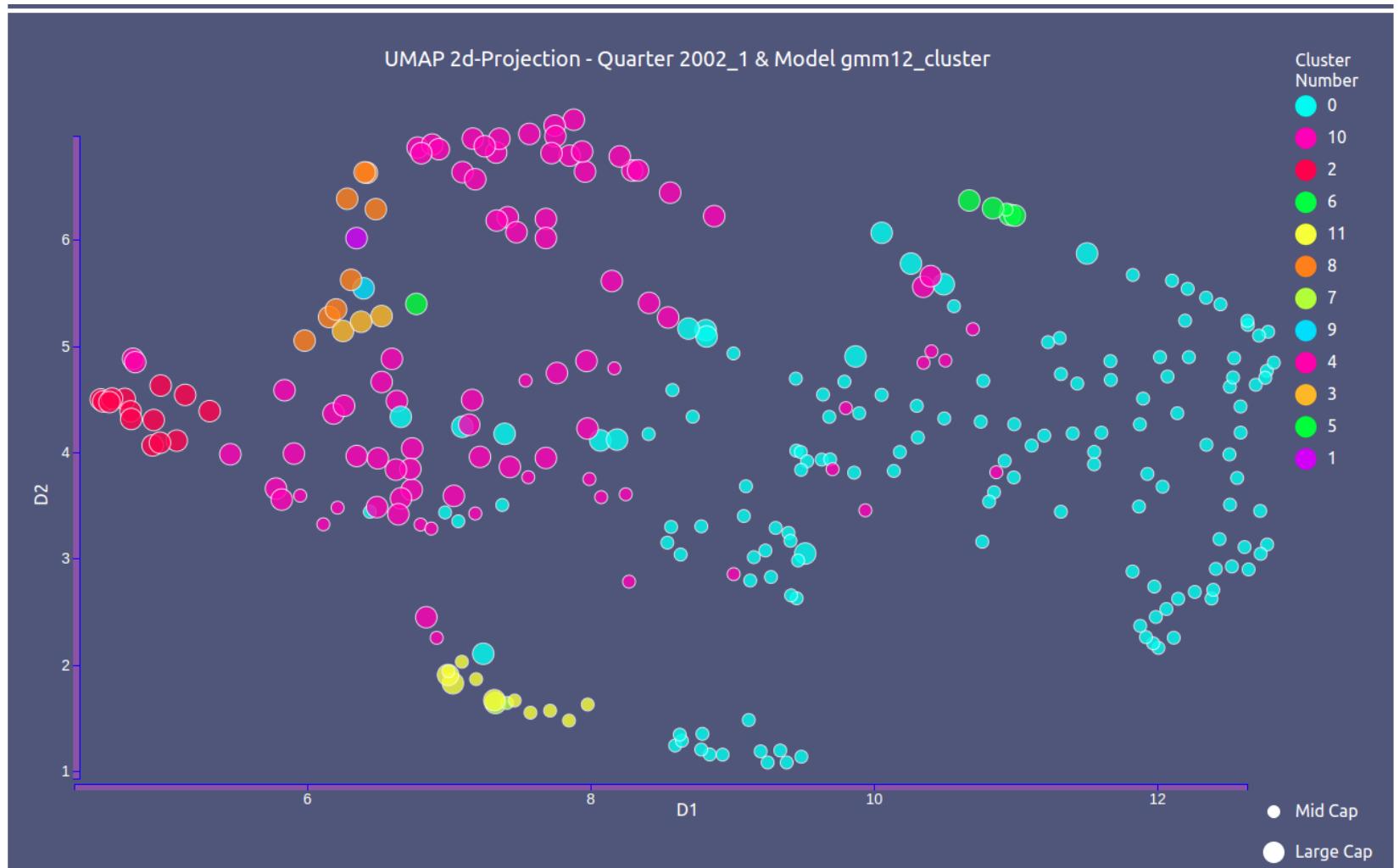
axs[2].set_title('GMM Model: Log Likelihood Score Across N Clusters', fontsize=15)
axs[2].set_xlabel('Clusters', fontsize=15)
axs[2].grid(ls=':')

plt.show()
```



Based on BIC we select GMM model with n_components =9 We also choose a GMM model with n_components =12 based on the Log Likelihood score of the models





5.3.3 Train each quarter data using GMM for our 2 selection, one with `n_components =9`, and 1 with `n_components =12`

```
In [69]: gmm_temp = pd.DataFrame()

for key in scaled_df['key'].unique().tolist():
    temp = scaled_df[scaled_df['key'] == key].copy()
    gmm9_ = GMM(n_components=9, covariance_type='full', \
                random_state= random_state).fit(temp.iloc[:, :exclude_features_for_training])

    gmm12_ = GMM(n_components=12, covariance_type='full', \
                  random_state= random_state).fit(temp.iloc[:, :exclude_features_for_training])

    gmm9_pred = gmm9_.predict(temp.iloc[:, :exclude_features_for_training])
    gmm12_pred = gmm12_.predict(temp.iloc[:, :exclude_features_for_training])

    temp['gmm9'] = gmm9_pred
    temp['gmm12'] = gmm12_pred
    gmm_temp = pd.concat([gmm_temp, temp], axis=0)
```

```
In [70]: #store the cluster results for each model in our dataframe
scaled_df_final['gmm9_cluster'] = gmm_temp['gmm9'].copy()
scaled_df_final['gmm12_cluster'] = gmm_temp['gmm12'].copy()
```

```
In [71]: print("We see for 2002_1 gmm9 cluster assigned cluster for stocks TXT, LII, FIS\n")
scaled_df_final[['key','stock','gmm9_cluster']].head(3)
```

We see for 2002_1 gmm9 cluster assigned cluster for stocks TXT, LII, FIS

Out[71]:

	key	stock	gmm9_cluster
0	2002_1	TXT	0
83	2002_1	LII	0
166	2002_1	FIS	8

```
In [72]: print("We see for 2002_1 gmm12 cluster assigned cluster for stocks TXT, LII, FIS\n")  
scaled_df_final[['key','stock','gmm12_cluster']].head(3)
```

We see for 2002_1 gmm12 cluster assigned cluster for stocks TXT, LII, FIS

Out[72]:

	key	stock	gmm12_cluster
0	2002_1	TXT	0
83	2002_1	LII	0
166	2002_1	FIS	10

```
In [73]: performance_df_temp = calculate_performance_cluster_by_quarter(scaled_df_final, 'gmm9_cluster', 'gmm9'  
performance_df = pd.concat([performance_df.copy(), performance_df_temp.copy()], axis=0, ignore_index=True)
```

```
In [74]: performance_df_temp = calculate_performance_cluster_by_quarter(scaled_df_final, 'gmm12_cluster', 'gmm12'  
performance_df = pd.concat([performance_df.copy(), performance_df_temp.copy()], axis=0, ignore_index=True)
```

```
In [75]: print('*'*100 + '\nGMM9 Clustering Performance Example: Clusters Ranked by Sharpe Ratio - 2022 Q3\n')  
performance_df[(performance_df['key'] == '2022_3') & (performance_df['model'] == 'gmm9')].sort_values(by='quarter_sharpe', ascending=False)
```

=====

=

GMM9 Clustering Performance Example: Clusters Ranked by Sharpe Ratio - 2022 Q3

=====

=

Out[75]:

	key	cluster	quarter_return	quarter_vol	quarter_sharpe	model
2240	2022_3	1	0.006798	0.216131	0.031451	gmm9
2232	2022_3	5	-0.039903	2.543013	-0.015691	gmm9
2233	2022_3	0	-2.083281	14.2222908	-0.146474	gmm9

```
In [76]: print('*'*100 + '\nGMM12 Clustering Performance Example: Clusters Ranked by Sharpe Ratio - 2022 Q3\nperformance_df[(performance_df['key'] =='2022_3')\n& (performance_df['model'] =='gmm12')].sort_values(by='quarter_sharpe', ascending=False)\n\n======\n=\nGMM12 Clustering Performance Example: Clusters Ranked by Sharpe Ratio - 2022 Q3\n======\n=
```

Out[76]:

	key	cluster	quarter_return	quarter_vol	quarter_sharpe	model
3236	2022_3	1	0.006798	0.216131	0.031451	gmm12
3233	2022_3	10	0.000471	0.184781	0.002549	gmm12
3225	2022_3	5	-0.071978	0.908825	-0.079199	gmm12

5.4 DBSCAN

```
In [78]: from itertools import product
##### 2002_Q1: HDBSCAN across different # clusters -----
dbscan_2002_1 = {}
dbscan_2002_1_len = {}
dbs_eps = np.arange(1, 4, 0.5)
dbs_min_samp = np.arange(5, 50, 5)
for (eps, sample) in list(product(dbs_eps, dbs_min_samp)):
    dbscan_ = DBSCAN(eps=eps, min_samples=sample)
    dbscan_fitpred = dbscan_.fit_predict(by_quarter_dict_clean['2002_1'])
    dbscan_2002_1[(eps, sample)] = dbscan_fitpred
    dbscan_2002_1_len[(eps, sample)] = len(np.unique(dbscan_fitpred))

#####
##### 2002_Q1: Compute Silhouette Scores across n clusters -----
dbscan_2002_1_sil_scores = {}
for k, v in dbscan_2002_1.items():
    sil_score_ = silhouette_score(by_quarter_dict_clean['2002_1'], v, metric='cosine')
    dbscan_2002_1_sil_scores[k] = sil_score_
```

In [79]: *##### DBSCAN Statistics Result #####*

```
dbscan_df = pd.DataFrame({  
    'eps_minsample': dbscan_2002_1.keys(),  
    'sil_score': dbscan_2002_1_sil_scores.values(),  
    'n_clusters': dbscan_2002_1_len.values()  
})  
  
dbscan_df
```

Out[79]:

	eps_minsample	sil_score	n_clusters
0	(1.0, 5)	0.341574	2
1	(1.0, 10)	0.403980	2
2	(1.0, 15)	0.422588	2
3	(1.0, 20)	0.438400	2
4	(1.0, 25)	0.453115	2
5	(1.0, 30)	0.448195	2
6	(1.0, 35)	0.446382	2
7	(1.0, 40)	0.441305	2
8	(1.0, 45)	0.440550	2
9	(1.5, 5)	0.206830	2
10	(1.5, 10)	0.210811	2
11	(1.5, 15)	0.224419	2
12	(1.5, 20)	0.263209	2
13	(1.5, 25)	0.280375	2
14	(1.5, 30)	0.283537	2
15	(1.5, 35)	0.283537	2
16	(1.5, 40)	0.289628	2
17	(1.5, 45)	0.288778	2
18	(2.0, 5)	0.158022	2
19	(2.0, 10)	0.173107	2

	eps_minsample	sil_score	n_clusters
20	(2.0, 15)	0.177803	2
21	(2.0, 20)	0.197666	2
22	(2.0, 25)	0.190043	2
23	(2.0, 30)	0.193312	2
24	(2.0, 35)	0.196659	2
25	(2.0, 40)	0.196659	2
26	(2.0, 45)	0.200920	2
27	(2.5, 5)	0.139931	2
28	(2.5, 10)	0.141142	2
29	(2.5, 15)	0.149863	2
30	(2.5, 20)	0.148144	2
31	(2.5, 25)	0.148144	2
32	(2.5, 30)	0.148144	2
33	(2.5, 35)	0.148144	2
34	(2.5, 40)	0.159754	2
35	(2.5, 45)	0.159754	2
36	(3.0, 5)	0.127169	2
37	(3.0, 10)	0.125518	2
38	(3.0, 15)	0.129647	2
39	(3.0, 20)	0.135445	2
40	(3.0, 25)	0.145142	2
41	(3.0, 30)	0.145142	2
42	(3.0, 35)	0.145142	2
43	(3.0, 40)	0.145576	2
44	(3.0, 45)	0.145576	2
45	(3.5, 5)	0.127513	2
46	(3.5, 10)	0.127082	2

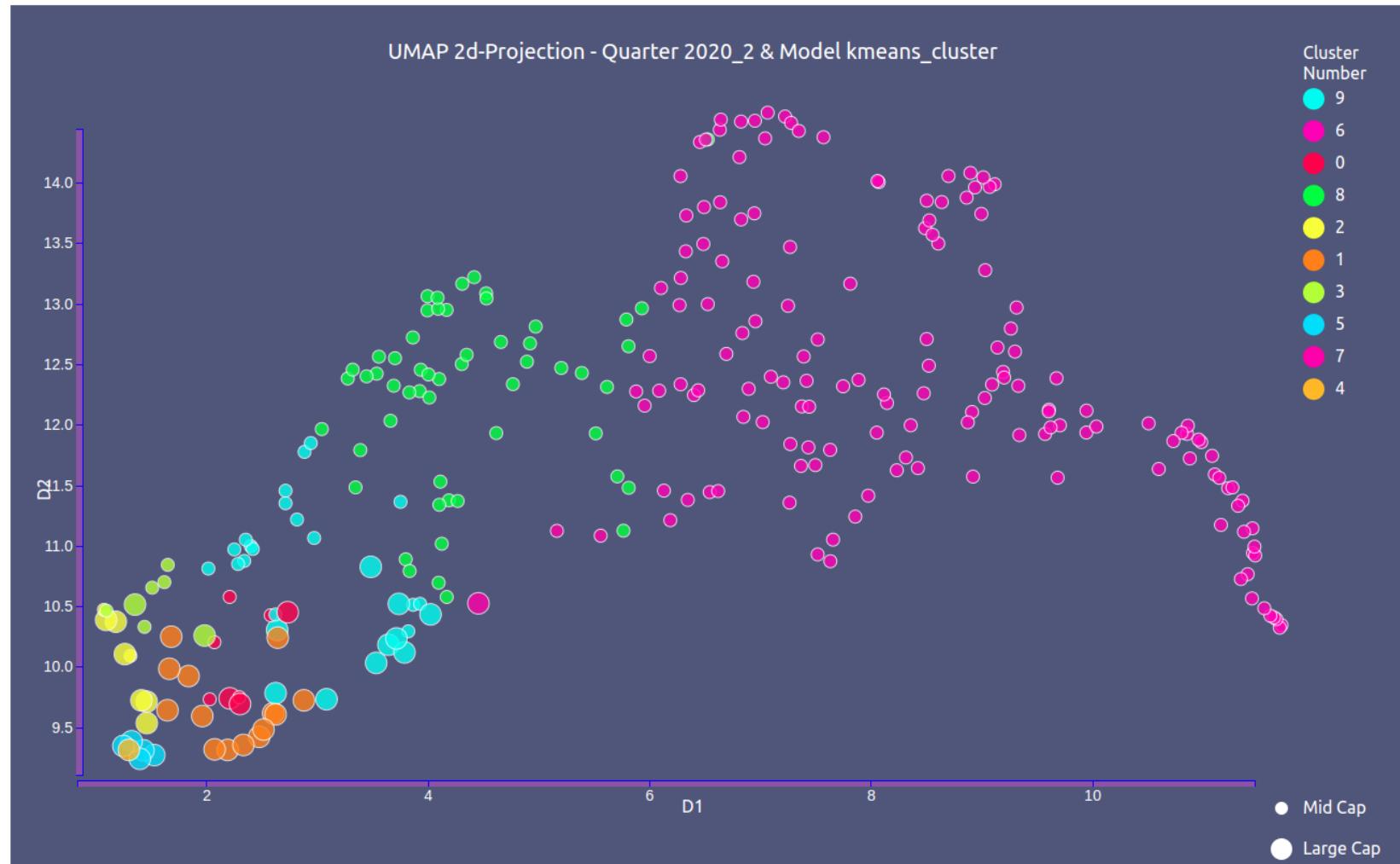
	eps_minsample	sil_score	n_clusters
47	(3.5, 15)	0.129225	2
48	(3.5, 20)	0.129225	2
49	(3.5, 25)	0.129072	2
50	(3.5, 30)	0.129072	2
51	(3.5, 35)	0.129072	2
52	(3.5, 40)	0.127636	2
53	(3.5, 45)	0.125943	2

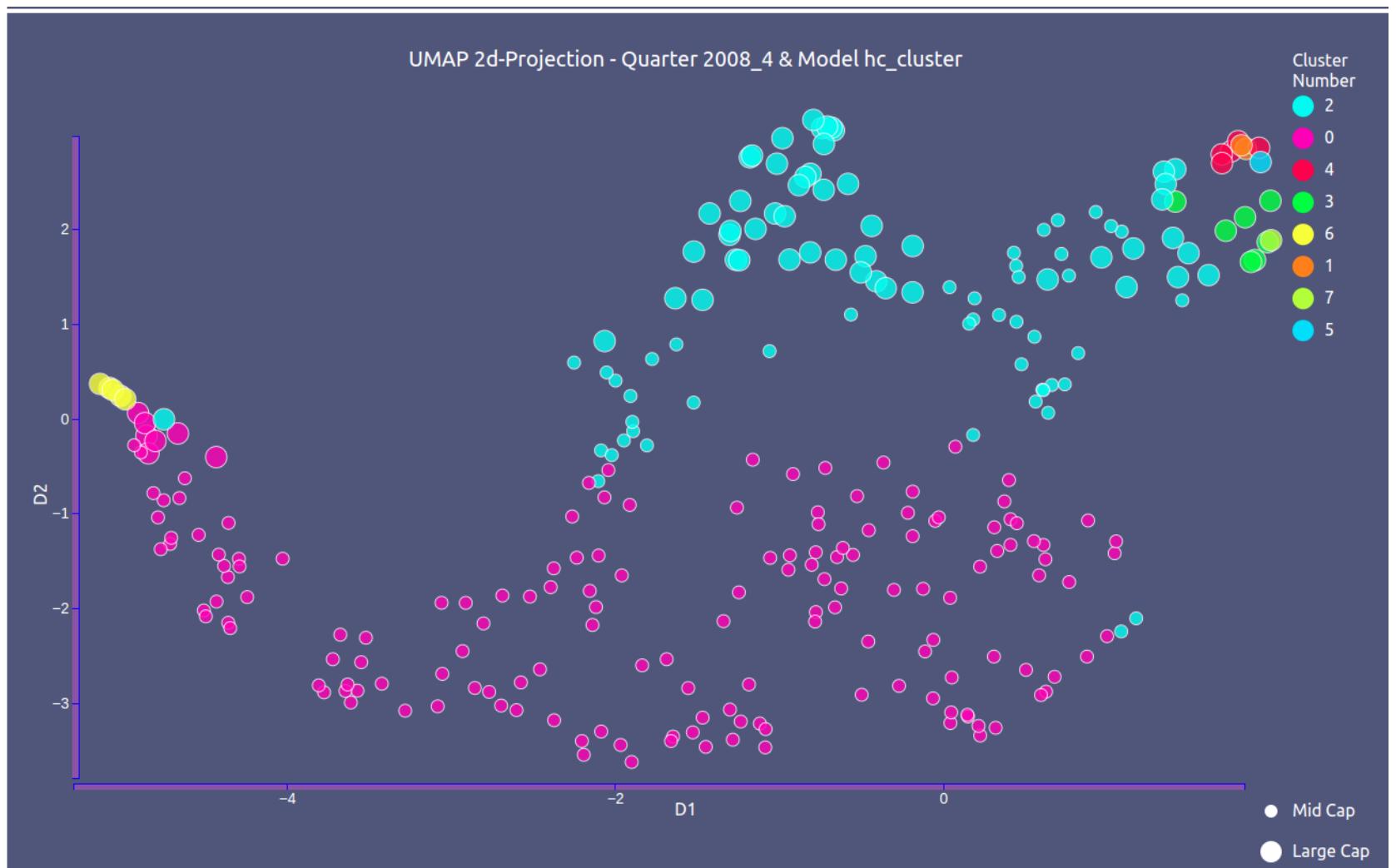
We did not select this model for our clustering. For each of the parameter trained, this model only gave 2 clusters. However, based on our experience from previous clustering, we have seen that cluster range for 8 to 12 are more appropriate

6. Visualizations -

As part of the project, we needed to visualize the clusters, across quarters to see how our clusters work along with performance metrics. For this part we created a tool using d3.js, javascript and html. We used the visualization tool throughout our model selection, and for verification of results. The tool allows us to visualize clusters for all 83 quarters along with the clustering algorithm. We can also see which stocks belong to which clusters. There can be more improvement to the tool. We used it along with pandas filtering to check our results. The tool can be found at : <https://dashing-crisp-38c329.netlify.app/> (<https://dashing-crisp-38c329.netlify.app/>)

```
In [119]: ### Example visualization during Covid 2020 Q2 - Kmeans and Financial Crisis - 2008-4 - Hierarchica
```





In []:

7. Performance

In this section we evaluate each of our cluster models performance against our benchmark S&P 500. For each Quarter we look at performance of each cluster, using sharpe ratio. "The Sharpe ratio compares the return of an investment with its risk. It's a mathematical expression of the insight that excess returns over a period of time may signify more volatility and risk" (<https://www.investopedia.com/terms/s/sharperatio.asp>) (<https://www.investopedia.com/terms/s/sharperatio.asp>)

We compare the top clusters sharpe ratio with the benchmark S&P 500 sharpe ratio for that quarter.

7.1 Helper code for generating reports

```
In [80]: #Add Sp500 data to our cluster performance df
sp500_sharpe_list = []
for i in range(performance_df.shape[0]):
    key = performance_df['key'].iloc[i]
    sp500_sharpe_list.append(sp500_performance_df.loc[key][0])

performance_df['sp500_quarter_sharpe'] = sp500_sharpe_list
```

```
In [81]: performance_df.head()
```

Out[81]:

	key	cluster	quarter_return	quarter_vol	quarter_sharpe	model	sp500_quarter_sharpe
0	2002_1	7	3.611904	10.972934	0.329165	kmeans	-0.039426
1	2002_1	0	0.197640	2.347955	0.084175	kmeans	-0.039426
2	2002_1	4	1.998277	3.857358	0.518043	kmeans	-0.039426
3	2002_1	1	0.572772	1.424710	0.402027	kmeans	-0.039426
4	2002_1	6	0.098935	0.531049	0.186301	kmeans	-0.039426

```
In [82]: #Create a dataframe to report the top3 cluster performer for each cluster for each quarter. The metric  
# is sharpe ratio
```

```
rank1_list = []  
rank2_list = []  
rank3_list = []  
beat_list = []  
sp500_sharpe_list = []  
key_list = []  
model_name_list =[]  
#perf_comparison_all_df = pd.DataFrame  
  
for model in performance_df['model'].unique().tolist():  
    temp = performance_df[performance_df['model'] == model]  
  
    for key in temp['key'].unique().tolist():  
        temp2 = temp[temp['key'] == key].sort_values(by='quarter_sharpe', ascending=False).iloc[:3]  
        rank1_list.append(temp2['quarter_sharpe'].iloc[0])  
        rank2_list.append(temp2['quarter_sharpe'].iloc[1])  
        rank3_list.append(temp2['quarter_sharpe'].iloc[2])  
        sp500_sharpe_list.append(temp2['sp500_quarter_sharpe'].iloc[0])  
  
        if temp2['quarter_sharpe'].iloc[0] > temp2['sp500_quarter_sharpe'].iloc[0]:  
            beat = "Beat"  
        else:  
            beat ="Fail"  
  
        beat_list.append(beat)  
  
        key_list.append(key)  
        model_name_list.append(model)
```

```
In [83]: perf_comparison_all_models = pd.DataFrame({'Quarter' : key_list,
                                                 '1st': rank1_list,
                                                 '2nd': rank2_list,
                                                 '3rd':rank3_list,
                                                 'S&P500':sp500_sharpe_list,
                                                 'Best Portfolio-Beat/Fail':beat_list,
                                                 'model':model_name_list
                                               })
```

```
In [84]: perf_comparison_all_models.head()
```

Out[84]:

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail	model
0	2002_1	0.867524	0.575890	0.518043	-0.039426	Beat	kmeans
1	2002_2	0.352154	-0.010013	-0.197571	-0.735674	Beat	kmeans
2	2002_3	-0.343881	-0.486416	-0.512664	-0.538130	Beat	kmeans
3	2002_4	0.447227	0.382396	0.298768	0.276784	Beat	kmeans
4	2003_1	0.385653	0.041056	-0.096267	-0.157044	Beat	kmeans

```
In [85]: # data frame for comparing all models

### Creates dataframe to compare each models top performer for each quarter

hc_list = []
kmeans_list = []
gmm9_list = []
gmm12_list = []
best_model_list = []

for model in perf_comparison_all_models['model'].unique().tolist():
    temp = perf_comparison_all_models[perf_comparison_all_models['model'] == model]

    for i in range(temp.shape[0]):
        if model == 'kmeans':
            kmeans_list.append(temp['1st'].iloc[i])

        elif model == 'hc':
            hc_list.append(temp['1st'].iloc[i])

        elif model == 'gmm9':
            gmm9_list.append(temp['1st'].iloc[i])

        elif model == 'gmm12':
            gmm12_list.append(temp['1st'].iloc[i])

for i in range(len(kmeans_list)):
    best_idx = np.argmax(np.array([kmeans_list[i], hc_list[i], gmm9_list[i], gmm12_list[i]]))
    if best_idx==0:
        best_model_list.append('kmeans')
    elif best_idx==1:
        best_model_list.append('hc')
    elif best_idx==2:
        best_model_list.append('gmm9')
    elif best_idx==3:
        best_model_list.append('gmm12')
```

```
best_performing_df = pd.DataFrame({'Quarter': np.unique(perf_comparison_all_models['Quarter']),
                                    'KMeans': kmeans_list,
                                    'Hierarchical':hc_list,
                                    'GMM(9 clusters)':gmm9_list,
                                    'GMM(12 clusters)':gmm12_list,
                                    'Best Performing Model':best_model_list
                                   })
```

In [86]: `best_performing_df.head(2)`

Out[86]:

	Quarter	KMeans	Hierarchical	GMM(9 clusters)	GMM(12 clusters)	Best Performing Model
0	2002_1	0.867524	0.822533	0.822533	0.822533	kmeans
1	2002_2	0.352154	-0.197571	0.010368	0.010368	kmeans

7.2 Hierarchical Clustering Performance

In [87]: ##### Hierarchical Performance: Top 3 Clustered Portfolios Across Quarters #####

```
shape = perf_comparison_all_models[(perf_comparison_all_models['model'] == 'hc')\n        & (perf_comparison_all_models[\n            'Best Portfolio-Beat/Fail'] == 'Beat')].iloc[:, :-1].shape[0]\n\nstr_output = f"Hierarchical Clustering : \n\nTop 3 clustered portfolio performance where top cluster\nstr_output=str_output+f" based on Sharpe Ratio between 2002-Q1 & 2022-Q3.\nstr_output = str_output+f"\n\nOur top cluster beat S&P {shape}/83 times\"\nprint('*'*110 + '\n' + str_output+'\n' +'*'*110 + '\n')\n\nperf_comparison_all_models[(perf_comparison_all_models['model'] == 'hc')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] == 'Beat')].iloc[
```

=====
=====
Hierarchical Clustering :

Top 3 clustered portfolio performance where top cluster beat S&P500 based on Sharpe Ratio between 2002-Q1 & 2022-Q3.

Our top cluster beat S&P 78/83 times
=====

Out[87]:

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
83	2002_1	0.822533	0.523622	0.349990	-0.039426	Beat
84	2002_2	-0.197571	-0.221862	-0.297719	-0.735674	Beat
85	2002_3	-0.343881	-0.486416	-0.526051	-0.538130	Beat
86	2002_4	0.795233	0.392234	0.266476	0.276784	Beat
87	2003_1	-0.143903	-0.156968	-0.168911	-0.157044	Beat

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
...
161	2021_3	0.163878	0.007454	-0.069299	0.020515	Beat
162	2021_4	0.892406	0.848759	0.765501	0.716955	Beat
163	2022_1	0.618255	0.476362	0.434672	-0.239911	Beat
164	2022_2	0.590578	0.034627	-0.263848	-0.641089	Beat
165	2022_3	0.031451	-0.156641	-0.170214	-0.182182	Beat

78 rows × 6 columns

```
In [88]: shape = perf_comparison_all_models[(perf_comparison_all_models['model'] =='hc')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] =='Fail')]\n        .iloc[:, :-1].shape[0]\n\nstr_output = f"Hierarchical Clustering :\n\nTop 3 clustered portfolio performance where top cluster\nstr_output= str_output+f" to beat S&P500 based on Sharpe Ratio between 2002-Q1 & 2022-Q3."\nstr_output = str_output +f"\n\nOur top cluster failed to beat S&P {shape}/83 times"\nprint('*'*110 + '\n' + str_output+'\n' +'*'*110 + '\n')\n\nperf_comparison_all_models[(perf_comparison_all_models['model'] =='hc')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] =='Fail')].iloc[
```

```
=====\n=====\\nHierarchical Clustering :\n\nTop 3 clustered portfolio performance where top cluster Failed to beat S&P500 based on Sharpe Ratio\nbetween 2002-Q1 & 2022-Q3.\n\nOur top cluster failed to beat S&P 5/83 times\n=====\\n=====
```

Out[88]:

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
123	2012_1	1.167031	1.114425	0.971874	1.206777	Fail
125	2012_3	0.427363	0.406430	0.385870	0.487914	Fail
149	2018_3	0.806466	0.777989	0.709346	0.965283	Fail
154	2019_4	0.794304	0.669718	0.663787	0.856314	Fail
160	2021_2	0.566577	0.434089	0.402378	0.704101	Fail

7.3 Kmeans Clustering Performance

```
In [89]: #####----- Kmean Performance: Top 3 Clustered Portfolios Across Quarters -----#####
shape = perf_comparison_all_models[(perf_comparison_all_models['model'] == 'kmeans')\n
                                    & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] == 'Beat')\n
                                    ].iloc[:, :-1].shape[0]

str_output = f"Kmeans Clustering : \n\nTop 3 clustered portfolio performance where top cluster beat S&P500 based on Sharpe Ratio between 2002-Q1 & 2022-Q3."
str_output += f"\n\nOur top cluster beat S&P {shape}/83 times"
print('*'*110 + '\n' + str_output + '\n' + '*'*110 + '\n')

perf_comparison_all_models[(perf_comparison_all_models['model'] == 'kmeans')\n
                            & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] == 'Beat')].iloc[
```

```
=====
=====
Kmeans Clustering :

Top 3 clustered portfolio performance where top cluster beat S&P500 based on Sharpe Ratio between 2002-Q1 & 2022-Q3.

Our top cluster beat S&P 78/83 times
=====
```

Out[89]:

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
0	2002_1	0.867524	0.575890	0.518043	-0.039426	Beat
1	2002_2	0.352154	-0.010013	-0.197571	-0.735674	Beat
2	2002_3	-0.343881	-0.486416	-0.512664	-0.538130	Beat
3	2002_4	0.447227	0.382396	0.298768	0.276784	Beat

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
4	2003_1	0.385653	0.041056	-0.096267	-0.157044	Beat
...
78	2021_3	0.046722	0.016411	0.002194	0.020515	Beat
79	2021_4	0.893857	0.892406	0.838055	0.716955	Beat
80	2022_1	0.588966	0.452037	0.434672	-0.239911	Beat
81	2022_2	0.613154	0.244463	-0.263848	-0.641089	Beat
82	2022_3	0.064978	0.031451	-0.150180	-0.182182	Beat

78 rows × 6 columns

```
In [90]: #####----- Kmean Performance: Top 3 Clustered Portfolios Across Quarters -----#####
shape = perf_comparison_all_models[(perf_comparison_all_models['model'] == 'kmeans')\n
                                    & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] == 'Fail')]\n
                                    .iloc[:, :-1].shape[0]

str_output = f"Kmeans Clustering : \n\nTop 3 clustered portfolio performance where"
str_output+=f" top cluster Failed to beat S&P500 based on Sharpe Ratio between 2002-Q1 &\n"
str_output = str_output+f"\n\nOur top cluster Failed to beat S&P {shape}/83 times"
print('*'*110 + '\n' + str_output+'\n' +'*'*110 + '\n')

perf_comparison_all_models[(perf_comparison_all_models['model'] == 'kmeans')\n
                            & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] == 'Fail')].iloc[

=====
=====

Kmeans Clustering :

Top 3 clustered portfolio performance where top cluster Failed to beat S&P500 based on Sharpe Ratio
between 2002-Q1 & 2022-Q3.

Our top cluster Failed to beat S&P 5/83 times
=====
```

Out[90]:

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
42	2012_3	0.482821	0.459996	0.404464	0.487914	Fail
66	2018_3	0.849734	0.845839	0.811815	0.965283	Fail
71	2019_4	0.804455	0.711618	0.670438	0.856314	Fail
72	2020_1	-0.401108	-0.422149	-0.443080	-0.394457	Fail
77	2021_2	0.579843	0.491961	0.434089	0.704101	Fail

7.4 Generalized Mixture Model with 9 clusters

```
In [91]: ##### GMM with 9 components Performance: Top 3 Clustered Portfolios Across Quarters #####
```

```
shape = perf_comparison_all_models[(perf_comparison_all_models['model'] == 'gmm9')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] == 'Beat')].iloc[0]\n\nstr_output = f"Gaussian Mixture Model Clustering (9 clusters) : \n\nTop 3 clustered portfolio performance where top cluster beat S&P500"\nstr_output=str_output+f" based on Sharpe Ratio between 2002-Q1 & 2022-Q3."\nstr_output = str_output+f"\n\nOur top cluster beat S&P {shape}/83 times"\nprint('*'*110 + '\n' + str_output+'\n' +'*'*110 + '\n')\n\nperf_comparison_all_models[(perf_comparison_all_models['model'] == 'gmm9')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] == 'Beat')].iloc[0]
```

```
=====
```

```
=====
```

```
Gaussian Mixture Model Clustering (9 clusters) :
```

```
Top 3 clustered portfolio performance where top cluster beat S&P500 based on Sharpe Ratio between 2002-Q1 & 2022-Q3.
```

```
Our top cluster beat S&P 80/83 times
```

```
=====
```

```
=====
```

```
Out[91]:
```

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
166	2002_1	0.822533	0.429173	0.424221	-0.039426	Beat
167	2002_2	0.010368	-0.197571	-0.297719	-0.735674	Beat
168	2002_3	-0.343881	-0.526051	-0.527800	-0.538130	Beat
169	2002_4	0.795233	0.326619	0.277970	0.276784	Beat
170	2003_1	0.238750	-0.098212	-0.197504	-0.157044	Beat
...
244	2021_3	0.332238	-0.024580	-0.055581	0.020515	Beat

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
245	2021_4	0.892406	0.752502	0.740423	0.716955	Beat
246	2022_1	0.551427	0.434672	0.312881	-0.239911	Beat
247	2022_2	0.590578	0.034627	-0.263848	-0.641089	Beat
248	2022_3	0.031451	-0.015691	-0.146474	-0.182182	Beat

80 rows × 6 columns

```
In [92]: shape = perf_comparison_all_models[(perf_comparison_all_models['model'] =='gmm9')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] =='Fail')]\n        .iloc[:, :-1].shape[0]\n\nstr_output = f"Generalized Mixture Model Clustering (9 clusters) : \n\nTop 3 clustered portfolio "\nstr_output = str_output+f"performance where top cluster Failed to beat S&P500"\nstr_output=str_output+f" based on Sharpe Ratio between 2002-Q1 & 2022-Q3."\nstr_output = str_output+f"\n\nOur top cluster Failed to beat S&P {shape}/83 times"\nprint('=*110 + '\n' + str_output+'\n' +'*110 + '\n')\n\nperf_comparison_all_models[(perf_comparison_all_models['model'] =='gmm9')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] =='Fail')].iloc[
```

```
=====\n=====
```

Generalized Mixture Model Clustering (9 clusters) :

Top 3 clustered portfolio performance where top cluster Failed to beat S&P500 based on Sharpe Ratio between 2002-Q1 & 2022-Q3.

Our top cluster Failed to beat S&P 3/83 times

```
=====
```

Out[92]:

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
208	2012_3	0.462373	0.461569	0.360660	0.487914	Fail
237	2019_4	0.775173	0.724016	0.663787	0.856314	Fail
243	2021_2	0.558702	0.471506	0.434089	0.704101	Fail

7.5 Generalized Mixture Model with 12 clusters

```
In [93]: #####---- GMM with 12 components Performance: Top 3 Clustered Portfolios Across Quarters ----#####
```

```
shape = perf_comparison_all_models[(perf_comparison_all_models['model'] =='gmm12')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] =='Beat')]\n        .iloc[:, :-1].shape[0]\n\nstr_output = f"Generalized Mixture Model Clustering (12 clusters) : \n\nTop 3 clustered portfolio "\nstr_output = str_output+f"performance where top cluster beat S&P500"\nstr_output=str_output+f" based on Sharpe Ratio between 2002-Q1 & 2022-Q3."\nstr_output = str_output+f"\n\nOur top cluster beat S&P {shape}/83 times"\nprint('=*110 + '\n' + str_output+'\n' +'*110 + '\n')
```

```
perf_comparison_all_models[(perf_comparison_all_models['model'] =='gmm12')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] =='Beat')].iloc[
```

```
======\n======\nGeneralized Mixture Model Clustering (12 clusters) :\n\nTop 3 clustered portfolio performance where top cluster beat S&P500 based on Sharpe Ratio between 2\n002-Q1 & 2022-Q3.\n\nOur top cluster beat S&P 78/83 times\n======\n======\n
```

```
Out[93]:
```

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
249	2002_1	0.822533	0.560969	0.488821	-0.039426	Beat
250	2002_2	0.010368	-0.067586	-0.197571	-0.735674	Beat
251	2002_3	-0.323336	-0.343881	-0.395489	-0.538130	Beat
252	2002_4	0.795233	0.524319	0.342196	0.276784	Beat
253	2003_1	0.108710	0.048690	0.012341	-0.157044	Beat

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
327	2021_3	0.332238	-0.013485	-0.055029	0.020515	Beat
328	2021_4	0.892406	0.815558	0.761384	0.716955	Beat
329	2022_1	0.579989	0.539062	0.434672	-0.239911	Beat
330	2022_2	0.613154	0.352033	0.034627	-0.641089	Beat
331	2022_3	0.031451	0.002549	-0.079199	-0.182182	Beat

78 rows × 6 columns

```
In [94]: shape = perf_comparison_all_models[(perf_comparison_all_models['model'] =='gmm12')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] =='Fail')].iloc[0]\n\n        str_output = f"Generalized Mixture Model Clustering (9 clusters) : \n\nTop 3 clustered portfolio " \n        str_output = str_output+f"performance where top cluster Failed to beat S&P500"\n        str_output=str_output+f" based on Sharpe Ratio between 2002-Q1 & 2022-Q3."\n        str_output = str_output+f"\n\nOur top cluster Failed to beat S&P {shape}/83 times"\n        print('=*110 + '\n' + str_output+'\n' +'*110 + '\n')\n\nperf_comparison_all_models[(perf_comparison_all_models['model'] =='gmm12')\n        & (perf_comparison_all_models['Best Portfolio-Beat/Fail'] =='Fail')].iloc[0]
```

```
=====\n===== \nGeneralized Mixture Model Clustering (9 clusters) :\n\nTop 3 clustered portfolio performance where top cluster Failed to beat S&P500 based on Sharpe Ratio\nbetween 2002-Q1 & 2022-Q3.\n\nOur top cluster Failed to beat S&P 5/83 times\n===== \n=====
```

Out[94]:

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail
291	2012_3	0.461569	0.424961	0.404115	0.487914	Fail
312	2017_4	0.954299	0.931779	0.927617	1.070671	Fail
315	2018_3	0.806466	0.806155	0.780632	0.965283	Fail
320	2019_4	0.771634	0.741511	0.681409	0.856314	Fail
326	2021_2	0.553083	0.542739	0.445932	0.704101	Fail

7.6 Comparison of Models By Quarter

```
In [95]: print("Model comparison for Each Quarter from 2002 Q1 to 2006 Q4")
best_performing_df.head(20)
```

Model comparison for Each Quarter from 2002 Q1 to 2006 Q4

Out[95]:

	Quarter	KMeans	Hierarchical	GMM(9 clusters)	GMM(12 clusters)	Best Performing Model
0	2002_1	0.867524	0.822533	0.822533	0.822533	kmeans
1	2002_2	0.352154	-0.197571	0.010368	0.010368	kmeans
2	2002_3	-0.343881	-0.343881	-0.343881	-0.323336	gmm12
3	2002_4	0.447227	0.795233	0.795233	0.795233	hc
4	2003_1	0.385653	-0.143903	0.238750	0.108710	kmeans
5	2003_2	1.290325	1.299985	1.307216	1.636791	gmm12
6	2003_3	0.719236	0.567762	0.643372	0.672054	kmeans
7	2003_4	1.069879	1.181842	1.181842	1.181842	hc
8	2004_1	0.528640	0.528640	0.462824	0.477352	kmeans
9	2004_2	1.055009	1.055009	1.055009	1.055009	kmeans
10	2004_3	0.505708	0.562852	0.529345	0.562852	hc
11	2004_4	1.071007	1.053235	1.065487	1.049002	kmeans
12	2005_1	0.517124	0.590147	0.595530	0.749469	gmm12
13	2005_2	0.819335	0.828967	0.819335	0.819335	hc
14	2005_3	0.908724	0.761603	0.992189	0.992189	gmm9
15	2005_4	0.665748	0.269646	0.233891	0.233891	kmeans
16	2006_1	1.143026	0.990914	1.129321	1.198128	gmm12
17	2006_2	0.210455	0.261593	0.306820	0.306820	gmm9
18	2006_3	0.949063	0.949063	0.949063	0.939382	kmeans
19	2006_4	1.005012	0.852899	0.880955	0.874086	kmeans

```
In [96]: financial_crisis_quarter =['2008_1','2008_2','2008_3','2008_4','2009_1', '2009_2','2009_3']
```

```
In [97]: print("Model comparison for Each Quarter during financial crisis")

for quarter in financial_crisis_quarter:
    display(perf_comparison_all_models[perf_comparison_all_models['Quarter'] == quarter])
```

Model comparison for Each Quarter during financial crisis

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail	model
24	2008_1	-0.001144	-0.149104	-0.246590	-0.441508	Beat	kmeans
107	2008_1	-0.047843	-0.175101	-0.281974	-0.441508	Beat	hc
190	2008_1	-0.047843	-0.110409	-0.359408	-0.441508	Beat	gmm9
273	2008_1	-0.006630	-0.047843	-0.212640	-0.441508	Beat	gmm12

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail	model
25	2008_2	0.810241	0.297599	0.124616	-0.180974	Beat	kmeans
108	2008_2	0.939847	0.165163	0.062405	-0.180974	Beat	hc
191	2008_2	0.728298	0.140814	-0.000251	-0.180974	Beat	gmm9
274	2008_2	0.734215	0.328321	0.303998	-0.180974	Beat	gmm12

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail	model
26	2008_3	-0.021507	-0.131127	-0.218100	-0.254455	Beat	kmeans
109	2008_3	0.082203	-0.121425	-0.242919	-0.254455	Beat	hc
192	2008_3	0.082203	0.000284	-0.121425	-0.254455	Beat	gmm9
275	2008_3	0.092857	0.084874	0.082203	-0.254455	Beat	gmm12

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio-Beat/Fail	model
27	2008_4	-0.202312	-0.274977	-0.300878	-0.373778	Beat	kmeans
110	2008_4	0.078286	-0.320004	-0.345618	-0.373778	Beat	hc
193	2008_4	0.101863	0.050227	-0.301248	-0.373778	Beat	gmm9
276	2008_4	0.101863	0.050227	-0.292684	-0.373778	Beat	gmm12

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio	Beat/Fail	model
28	2009_1	0.266964	0.137746	-0.045440	-0.305279		Beat	kmeans
111	2009_1	0.181255	0.074745	-0.045440	-0.305279		Beat	hc
194	2009_1	0.181255	0.074745	-0.045440	-0.305279		Beat	gmm9
277	2009_1	0.181255	0.123164	0.074745	-0.305279		Beat	gmm12

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio	Beat/Fail	model
29	2009_2	0.824508	0.822212	0.634409	0.538494		Beat	kmeans
112	2009_2	0.952250	0.680741	0.639218	0.538494		Beat	hc
195	2009_2	0.952250	0.667044	0.642985	0.538494		Beat	gmm9
278	2009_2	0.952250	0.692944	0.649746	0.538494		Beat	gmm12

	Quarter	1st	2nd	3rd	S&P500	Best Portfolio	Beat/Fail	model
30	2009_3	1.212918	1.090723	0.992468	0.791834		Beat	kmeans
113	2009_3	1.036822	0.891033	0.841848	0.791834		Beat	hc
196	2009_3	1.212918	0.905071	0.818703	0.791834		Beat	gmm9
279	2009_3	1.141163	1.058888	0.924535	0.791834		Beat	gmm12

As seen above, our best cluster managed to do better than S&P 500 during the financial crisis. However, since clustering we cannot forecast , we have no way to know which cluster is the best. For this we need a supervised approach to look at each clusters stock, and pick the best cluster based on that.

7.7 - Best performing model

```
In [98]: best_performing_df['Best Performing Model'].value_counts()
```

```
Out[98]: kmeans    38  
hc        17  
gmm12     14  
gmm9      14  
Name: Best Performing Model, dtype: int64
```

Kmeans had the highest sharpe ratio for its top cluster on each quarter, followed by Hierarchical, with a tie on GMM9, and GMM12

8. Conclusion for Clustering

We used fundamental data of 278 companies, and picked a model with best params based on the first quarter data in our dataset. We then used this model parameter to train our data for all quarters. Each quarter produced different clustering for our 278 stocks. We calculated the sharpe ratio for each cluster stocks for each quarter.

We compared each quarters sharpe ratio for different models against our benchmark S&P 500. We found that our best performing cluster for each algorithm beat our benchmark S&P 500 at a minimum of 78 times for Kmeans, Hierarchical, and GMM12 and a maximum of 80 times for our GMM9 model.

We Looked at the performance among the models. Kmeans had the highest sharpe ratio for its top cluster, as compared to other models.

We also explored visualization using d3.js to create visualizations for each quarter for each algorithm. The visualization not only helped us understand clustering better, but also gave us confirmation as we proceeded in the project that our approach was correct, and the results were not erroneous

Since, this is an unsupervised we cannot forecast which cluster for each of the algorithm will perform best. We will need additional supervised approach to train models with each of the stock in each cluster, and then pick the stocks for our portfolio that has the highest return forecast.

Clustering approach does seems to be valuable in portfolio construction as seen from the results. We are essentially able to cluster stocks based on their fundamentals. Combininig cluster with Picking the cluster that supervised learning approach will be a reasonable approach in creating a portfolio.

As for picking the best model among Kmeans, Hierarchical Clustering, GMM with 9 cluster, and GMM with 9 cluster, we would use some intersection between these models to select the best stock cluster. If we had to pick one, we would pick Kmeans clustering, it is one of the simplest model, and as our results shows had the highest sharpe ration for 38 quarters, which is twice as more than other models.

Our sincere thanks Professor Henstock, Daniel, Lu, Christina and Debbie.

In []: