Unsupervised Machine Learning - Clustering Stock Portfolio

Main objective of analysis

- Business initiative

As a long - term investor, I would like to dive into the massive stock data and try to build profitable portfolios through unsupervised machine learning methods.

- Expect outcome

The analysis will create and analyze several unsupervised machine learning models with different hyperparameters such as Kmeans, MeanShift, Hierarchical Cluster Analysis (HCA), explain the insights and to find the best model that could fulfill the business initiative.

Brief description of the data set and a summary of its attributes

As the data source, the Year - 2018 U.S public trading companies' financial indicator data is to be utilized, and it is from:

https://www.kaggle.com/cnic92/200-financial-indicators-of-us-stocks-20142018

Data load, pre - processing and exploration

The data source file contains 4392 records and 225 columns. The information is summarized as following:

```
path = './data/200+ Financial Indicators of US stocks (2014-2018)/'
df1 = pd.read_csv(path + '2018_Financial_Data.csv')
```

```
# Check each file's schema
print(df1.shape)
```

```
(4392, 225)
```

And there are two columns with object data types, which are "Unnamed: 0" and "Sector". Here I rename the first one to "Stock", drop off the second one, add a new column "Data Year" with value 2018 and rename column "2019 PRICE VAR [%]" to "Next Year VAR%".

```python
# Rename the first column
df1.rename(columns={'Unnamed: 0': 'Stock'}, inplace=True)
```

```python
df1_backup = df1.copy()
df1.drop(columns=['Sector'], inplace=True)
print(df1.shape)
```

```
(4392, 224)
```

```python
# Add Year column
df1['Data Year'] = 2018
# Rename Price VAR column
df1.rename(columns={'2019 PRICE VAR [%]':'Next Year VAR%'}, inplace=True)
```

```python
print('2018_Financial_Data.csv shape: {}'.format(df1.shape))
```

```
2018_Financial_Data.csv shape: (4392, 225)
```

	Data Year	Stock	Class	Next Year VAR%	Revenue	Revenue Growth	Cost of Revenue	Gross Profit	R&D Expenses	SG&A Expense	...	10Y Dividend per Share Growth (per Share)	5Y Dividend per Share Growth (per Share)	3Y Dividend per Share Growth (per Share)	Re
0	2018	CMCSA	1	32.794573	9.450700e+10	0.1115	0.000000e+00	9.450700e+10	0.000000e+00	6.482200e+10	...	0.2558	0.1865	0.2348	
1	2018	KMI	1	40.588068	1.414400e+10	0.0320	7.288000e+09	6.856000e+09	0.000000e+00	6.010000e+08	...	0.0000	-0.1421	-0.2785	
2	2018	INTC	1	30.295514	7.084800e+10	0.1289	2.711100e+10	4.373700e+10	1.354300e+10	6.750000e+09	...	0.0815	0.0592	0.0772	
3	2018	MU	1	64.213737	3.039100e+10	0.4955	1.250000e+10	1.789100e+10	2.141000e+09	8.130000e+08	...	0.0000	0.0000	0.0000	
4	2018	GE	1	44.757840	1.216150e+11	0.0285	9.546100e+10	2.615400e+10	0.000000e+00	1.811100e+10	...	-0.1139	-0.1408	-0.2619	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4387	2018	YRIV	0	-90.962099	0.000000e+00	0.0000	0.000000e+00	0.000000e+00	0.000000e+00	3.755251e+06	...	NaN	NaN	0.0000	
4388	2018	YTEN	0	-77.922077	5.560000e+05	-0.4110	0.000000e+00	5.560000e+05	4.759000e+06	5.071000e+06	...	0.0000	0.0000	0.0000	
4389	2018	ZKIN	0	-17.834400	5.488438e+07	0.2210	3.659379e+07	1.829059e+07	1.652633e+06	7.020320e+06	...	NaN	NaN	0.0000	
4390	2018	ZOM	0	-73.520000	0.000000e+00	0.0000	0.000000e+00	0.000000e+00	1.031715e+07	4.521349e+06	...	NaN	NaN	NaN	
4391	2018	ZYME	1	209.462222	5.301900e+07	0.0243	0.000000e+00	5.301900e+07	5.668400e+07	2.945700e+07	...	NaN	NaN	0.0000	

4392 rows × 225 columns

Data screening and variable selection

17 indicators are selected as the most common and representative financial indicators:

1. Profit Margin
2. Net Profit Margin
3. Operating Profit Margin
4. EPS Diluted
5. Return on Assets (ROA)
6. Return on Equity (ROE)
7. Price to Free Cash Flows Ratio
8. Price Earnings Ratio
9. Price Earnings to Growth Ratio
10. Asset Turnover
11. Current Ratio
12. Quick Ratio
13. Debt Equity Ratio
14. Interest Coverage
15. Receivables Turnover

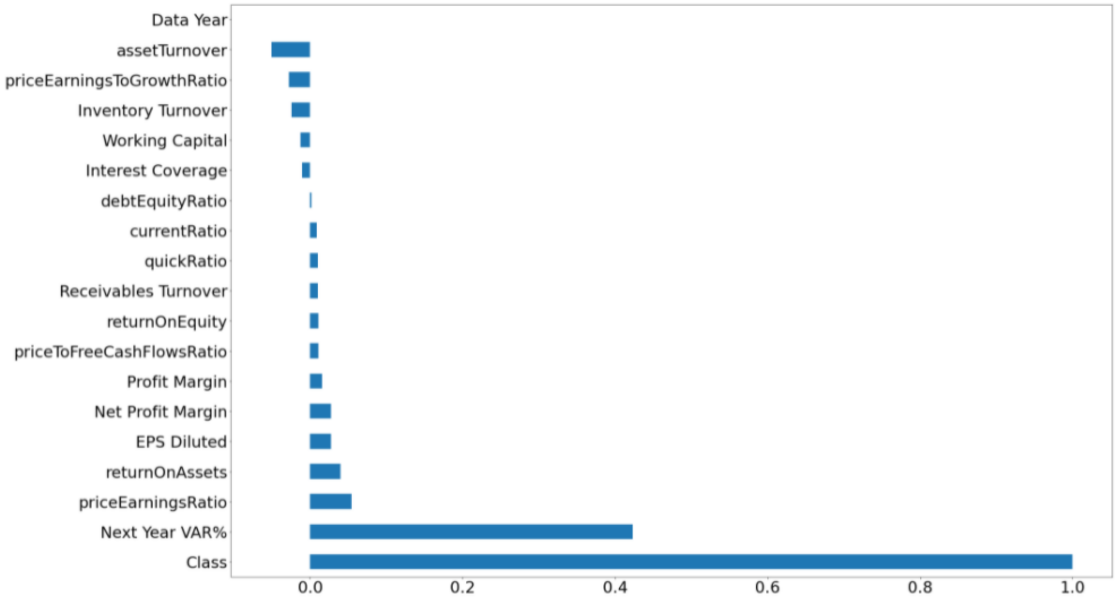
16. Invertory Turnover
17. Working Capital

The data set schema is described as following:

Column	Type	Description
Data Year	int64	
Stock	object	Stock code
Class	int64	The target variable (dependent variable) that the model is going to predict. “0” means not worth buying and “1” is worth buying.
Next Year VAR%	float64	This is the next - year stock price variation.
Profit Margin	float64	Profit (% of revenue)
Net Profit Margin	float64	The profit amount over company revenue.
EPS	float64	Earning per share
returnOnAssets	float64	The profitability of the company’s assets.
returnOnEquity	float64	The profitability of the company’s equity.
priceToFreeCashFlowsRatio	float64	The relationship between stock share price and company’s free cash.
priceEarningsRatio	float64	P/E ratio.
priceEarningsToGrowthRatio	float64	PEG ratio. It is similar to P/E and takes company growth into consideration.
assetTurnover	float64	How much revenue is generated from company owned assets.
currentRatio	float64	This ratio indicates the cash amount that a company owns in hand.
quickRatio	float64	How fast to turn assets into cash.
debtEquityRatio	float64	Company debt and equity percentage.

Interest Coverage	float64	How much interest expense that a company has to pay per year.
Receivables Turnover	float64	How fast (per year) a company can collect money from the goods sold.
Inventory Turnover	float64	The frequency a company can “sell and refresh” its inventory per year. The higher the better.
Working Capital	float64	The difference between company current assets and current liabilities. The higher the better.

The correlation of the key financial is as following:



Data cleaning and feature engineering

- Deal with NA value
  Through a quick check it seems NA values are spread among the columns.

```
indicator_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4392 entries, 0 to 4391
Data columns (total 20 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Data Year                     4392 non-null   int64  
 1   Stock                         4392 non-null   object 
 2   Class                         4392 non-null   int64  
 3   Next Year VAR%                4392 non-null   float64
 4   Profit Margin                 4086 non-null   float64
 5   Net Profit Margin             4216 non-null   float64
 6   EPS Diluted                   4329 non-null   float64
 7   returnOnAssets                3286 non-null   float64
 8   returnOnEquity                4136 non-null   float64
 9   priceToFreeCashFlowsRatio     4139 non-null   float64
 10  priceEarningsRatio            4140 non-null   float64
 11  priceEarningsToGrowthRatio    2734 non-null   float64
 12  assetTurnover                 4162 non-null   float64
 13  currentRatio                  4141 non-null   float64
 14  quickRatio                    4143 non-null   float64
 15  debtEquityRatio               4141 non-null   float64
 16  Interest Coverage             4146 non-null   float64
 17  Receivables Turnover          4260 non-null   float64
 18  Inventory Turnover            4153 non-null   float64
 19  Working Capital               3289 non-null   float64
dtypes: float64(17), int64(2), object(1)
memory usage: 686.4+ KB
```

For this analysis I would like to drop the data with NA values, since at this moment it is unclear how to reasonably make up the missing financial indicators without jeopardizing the data quality.

```python
def get_na_columns(df):
    columns = df.columns
    columns_has_na = []
    for column in columns:
        has_na = df[column].isna().any().sum()
        if(has_na>0):
            columns_has_na.append(column)
    return columns_has_na
```

```python
# Deal with NA data
# Focus on financial indicator dataset.
# To maintain model quality, I drop the records with NA values instead of filling 0.
indicator_data = indicator_data.dropna()
indicator_data.shape
```

```
(1955, 20)
```

```python
# Verify if there is any na exists.
s = get_na_columns(indicator_data)
len(s)
```

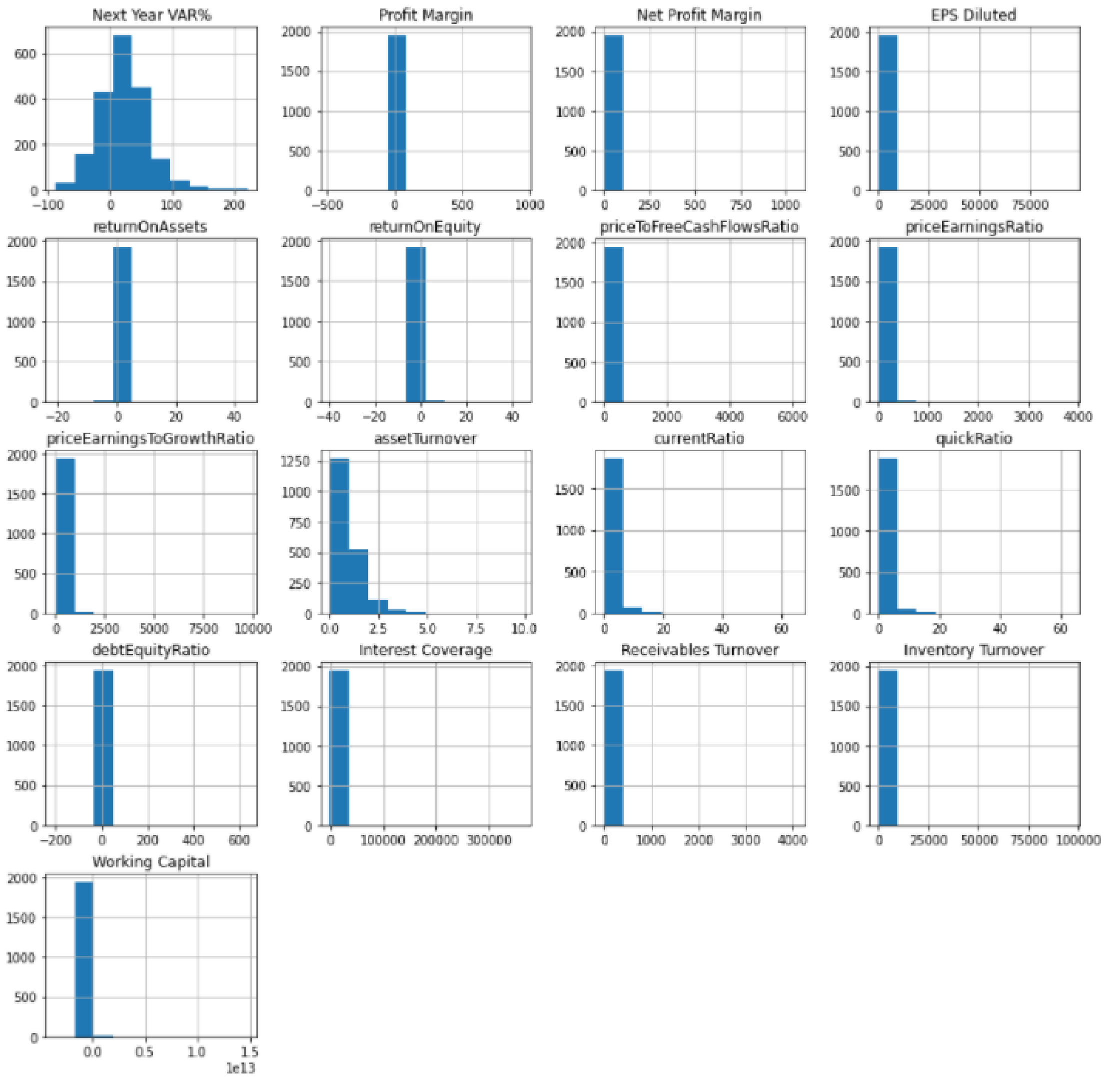
```
0
```

- Normalization, standardization and outlier process
  There are 1955 records after dropping NA values. The next task is to see whether there are outliers in the data set.

  Firstly I examine the each column's skewness and realize that the data is far away from normal distribution:

```
# Check skew
indicator_data.skew()
```

```
Data Year                      0.000000
Class                         -1.010378
Next Year VAR%                 0.473871
Profit Margin                 24.852242
Net Profit Margin             40.057755
EPS Diluted                   44.215133
returnOnAssets                13.883660
returnOnEquity                 3.714219
priceToFreeCashFlowsRatio     26.207001
priceEarningsRatio            16.430980
priceEarningsToGrowthRatio    21.740647
assetTurnover                  3.352897
currentRatio                   9.505593
quickRatio                    10.078875
debtEquityRatio               26.818857
Interest Coverage             43.961027
Receivables Turnover          21.869920
Inventory Turnover            43.805785
Working Capital               29.608817
dtype: float64
```

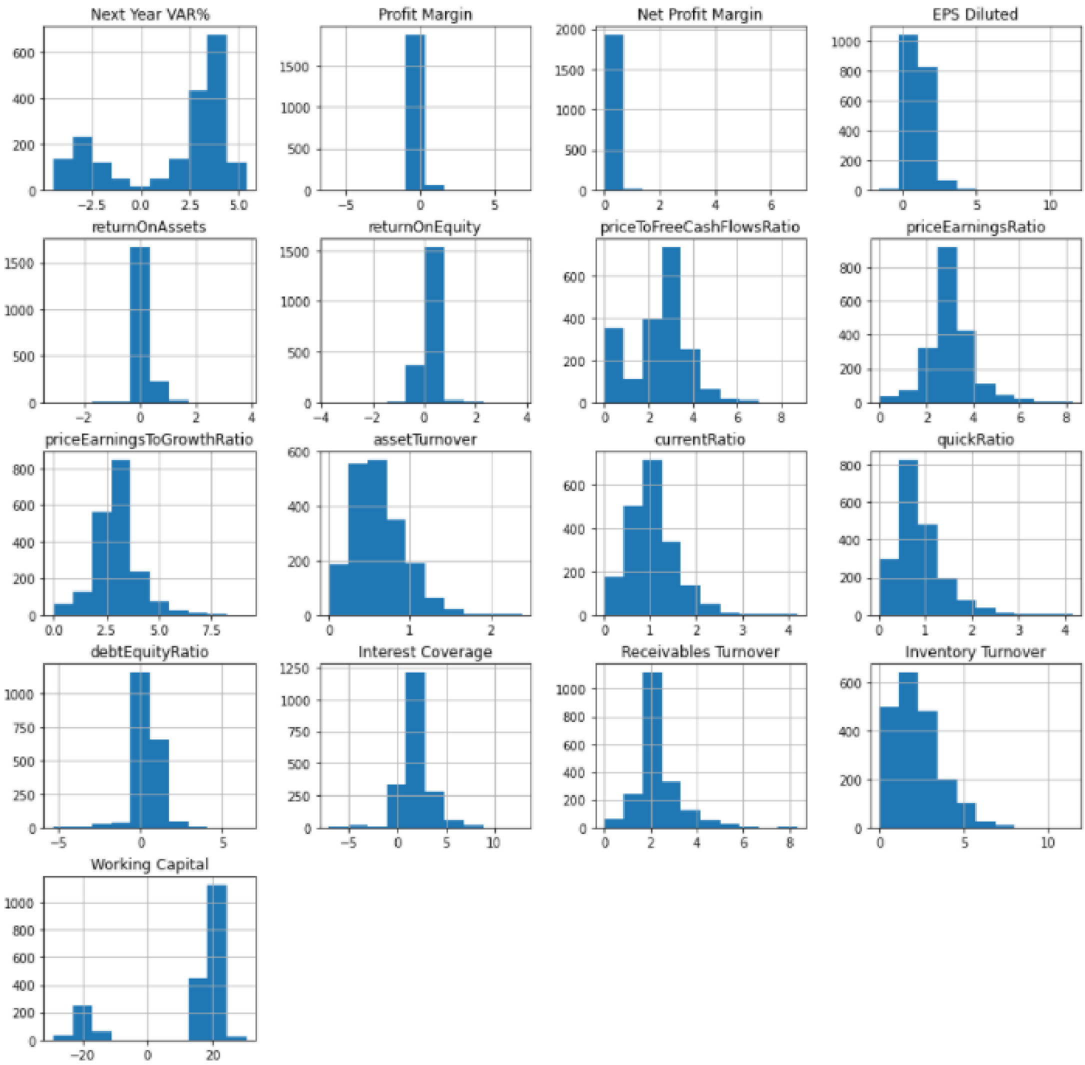


Therefore I use log1p to normalize the skewness:

```
# Fix the skew and normalize the data with log
for c in indicator_list[3:]:
    arr_values = indicator_data[c].values
    logarr = []
#     print(arr_values)
    for value in arr_values:
        if (value < 0):
            value = np.log1p(np.abs(value))*-1
        else:
            value = np.log1p(value)
        logarr.append(value)
    indicator_data[c] = logarr

indicator_data.skew()
```

```
Data Year                      0.000000
Class                         -1.010378
Next Year VAR%                -0.887898
Profit Margin                  6.840723
Net Profit Margin             16.189745
EPS Diluted                    1.867669
returnOnAssets                -0.312514
returnOnEquity                -0.423811
priceToFreeCashFlowsRatio     -0.281104
priceEarningsRatio             0.513274
priceEarningsToGrowthRatio     0.721049
assetTurnover                  0.923419
currentRatio                   1.054312
quickRatio                     1.647107
debtEquityRatio               -1.018057
Interest Coverage             -0.189418
Receivables Turnover           1.389525
Inventory Turnover             0.517996
Working Capital               -1.644021
dtype: float64
```

Now the data is distributed more normally after normalization, however it still contains some outliers which makes the plot seem biased.



The next step is to standardize the data and use the IQR method to identify and remove the outliers. Data standardization is crucial to unsupervised analysis which is mostly based on the “distance” among data points.

```python
def check_IQR(df, column):
    scale = 1.5
    describe_df = df[column].describe()
    Q1 = describe_df['25%']
    Q3 = describe_df['75%']
    IQR = Q3 - Q1
    lower_bound = Q1 - scale*IQR
    upper_bound = Q3 + scale*IQR
    return df[column].loc[(df[column]<lower_bound)|(df[column]>upper_bound)]
```

```python
# Use 1.5 IQR to isolate and drop outliers
tmp_index_list=[]
for check_column in feature_column:
    column_outlier_df = check_IQR(indicator_data, check_column)
    print('Column {} has {} outlier data'.format(check_column, column_outlier_df.shape[0]))
#     add index to temp list
    tmp_index_list.extend(column_outlier_df.index.values.tolist())

outlier_index = set(tmp_index_list)
# Try to drop the outliers
indicator_data.drop(index=outlier_index, inplace=True)
indicator_data.shape
indicator_data['Data Year'].value_counts()
```

```
Column Next Year VAR% has 0 outlier data
Column Profit Margin has 118 outlier data
Column Net Profit Margin has 120 outlier data
Column EPS Diluted has 19 outlier data
Column returnOnAssets has 164 outlier data
Column returnOnEquity has 186 outlier data
Column priceToFreeCashFlowsRatio has 28 outlier data
Column priceEarningsRatio has 143 outlier data
Column priceEarningsToGrowthRatio has 158 outlier data
Column assetTurnover has 29 outlier data
Column currentRatio has 64 outlier data
Column quickRatio has 84 outlier data
Column debtEquityRatio has 129 outlier data
Column Interest Coverage has 129 outlier data
Column Receivables Turnover has 207 outlier data
Column Inventory Turnover has 42 outlier data
Column Working Capital has 360 outlier data
```

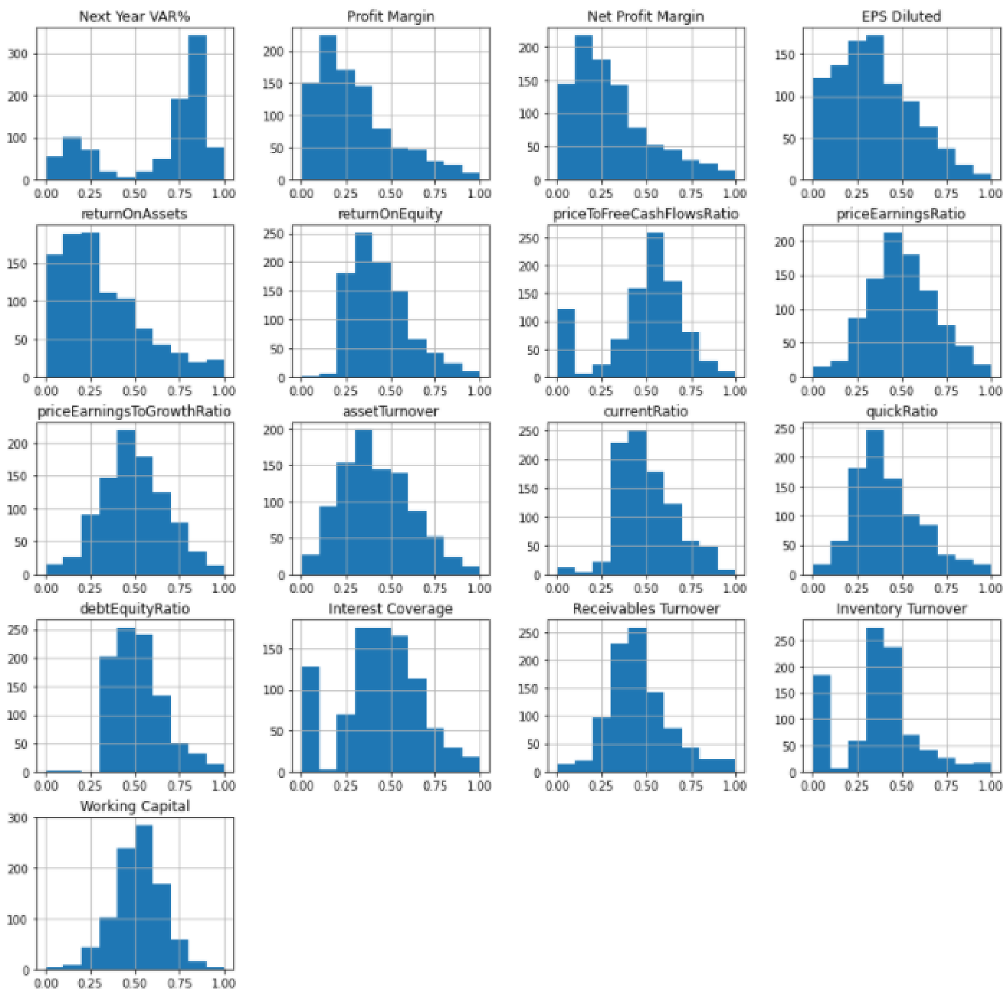
```python
# Scale the rest features
scaler = MinMaxScaler()
# scaler = RobustScaler()
example_standardized = scaler.fit_transform(indicator_data[feature_column])
# example_standardized.shape
example_standardized_df = pd.DataFrame(data=example_standardized, columns=feature_column)
```

```python
example_standardized_df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Next Year VAR%	930.0	0.640871	0.293331	0.0	0.331227	0.782522	0.852985	1.0
Profit Margin	930.0	0.298781	0.215685	0.0	0.134357	0.244524	0.402468	1.0
Net Profit Margin	930.0	0.305080	0.218714	0.0	0.137908	0.251199	0.410970	1.0
EPS Diluted	930.0	0.347180	0.209258	0.0	0.182384	0.325119	0.488720	1.0
returnOnAssets	930.0	0.313706	0.225161	0.0	0.138942	0.266422	0.435977	1.0
returnOnEquity	930.0	0.439454	0.160812	0.0	0.319454	0.412960	0.536053	1.0
priceToFreeCashFlowsRatio	930.0	0.484845	0.229223	0.0	0.407537	0.537151	0.627423	1.0
priceEarningsRatio	930.0	0.503841	0.185704	0.0	0.382817	0.493239	0.627757	1.0
priceEarningsToGrowthRatio	930.0	0.494487	0.181037	0.0	0.377833	0.488408	0.611442	1.0
assetTurnover	930.0	0.420806	0.197061	0.0	0.271066	0.397218	0.555919	1.0
currentRatio	930.0	0.504186	0.160041	0.0	0.387037	0.478160	0.601739	1.0
quickRatio	930.0	0.419315	0.186930	0.0	0.291430	0.380937	0.530224	1.0
debtEquityRatio	930.0	0.520414	0.139682	0.0	0.411714	0.502105	0.599203	1.0
Interest Coverage	930.0	0.445009	0.212951	0.0	0.321138	0.445030	0.580564	1.0
Receivables Turnover	930.0	0.459951	0.171466	0.0	0.355638	0.435920	0.544023	1.0
Inventory Turnover	930.0	0.356472	0.223746	0.0	0.280761	0.379642	0.459917	1.0
Working Capital	930.0	0.519868	0.138902	0.0	0.436255	0.525998	0.609192	1.0

After scaling the data set and removing the outlier data points identified by 1.5 IQR, the data points are much more normally distributed:

	count	mean	std	min	25%	50%	75%	max
Next Year VAR%	930.0	0.640871	0.293331	0.0	0.331227	0.782522	0.852985	1.0
Profit Margin	930.0	0.298781	0.215685	0.0	0.134357	0.244524	0.402468	1.0
Net Profit Margin	930.0	0.305080	0.218714	0.0	0.137908	0.251199	0.410970	1.0
EPS Diluted	930.0	0.347180	0.209258	0.0	0.182384	0.325119	0.488720	1.0
returnOnAssets	930.0	0.313706	0.225161	0.0	0.138942	0.266422	0.435977	1.0
returnOnEquity	930.0	0.439454	0.160812	0.0	0.319454	0.412960	0.536053	1.0
priceToFreeCashFlowsRatio	930.0	0.484845	0.229223	0.0	0.407537	0.537151	0.627423	1.0
priceEarningsRatio	930.0	0.503841	0.185704	0.0	0.382817	0.493239	0.627757	1.0
priceEarningsToGrowthRatio	930.0	0.494487	0.181037	0.0	0.377833	0.488408	0.611442	1.0
assetTurnover	930.0	0.420806	0.197061	0.0	0.271066	0.397218	0.555919	1.0
currentRatio	930.0	0.504186	0.160041	0.0	0.387037	0.478160	0.601739	1.0
quickRatio	930.0	0.419315	0.186930	0.0	0.291430	0.380937	0.530224	1.0
debtEquityRatio	930.0	0.520414	0.139682	0.0	0.411714	0.502105	0.599203	1.0
Interest Coverage	930.0	0.445009	0.212951	0.0	0.321138	0.445030	0.580564	1.0
Receivables Turnover	930.0	0.459951	0.171466	0.0	0.355638	0.435920	0.544023	1.0
Inventory Turnover	930.0	0.356472	0.223746	0.0	0.280761	0.379642	0.459917	1.0
Working Capital	930.0	0.519868	0.138902	0.0	0.436255	0.525998	0.609192	1.0



Evaluation of unsupervised machine learning models

Here to check again the data set that is going to be analyzed:

```
indicator_data.shape
a = indicator_data[['Data Year', 'Stock', 'Class']]
# a.dropna(inplace=True)
a.reset_index(inplace=True, drop=True)
a
example_standardized_df
indicator_data = pd.concat([a, example_standardized_df], axis=1)
indicator_data
```

	Data Year	Stock	Class	Next Year VAR%	Profit Margin	Net Profit Margin	EPS Diluted	returnOnAssets	returnOnEquity	priceToFreeCashFlowsRatio	priceEarningsRatio	priceEarning:
0	2018	MSFT	1	0.878362	0.554393	0.553837	0.379612	0.214901	0.555854	0.611313	0.745893	
1	2018	F	1	0.795965	0.087865	0.084585	0.213833	0.128893	0.377558	0.313765	0.280434	
2	2018	AMD	1	0.969817	0.199306	0.196808	0.086730	0.329692	0.667741	0.000000	0.788942	
3	2018	VALE	1	0.506802	0.683996	0.681851	0.278028	0.238926	0.477132	0.407489	0.330187	
4	2018	ORCL	1	0.767768	0.344427	0.342805	0.201235	0.110669	0.329729	0.516416	0.785880	
...	...	...	...	...	...	...	...	...	...	...	...	
925	2018	USDP	1	0.681243	0.646911	0.646663	0.186239	0.258553	0.729641	0.407055	0.410228	
926	2018	WHLM	0	0.110906	0.040824	0.037770	0.042899	0.235666	0.241228	0.444986	0.678193	
927	2018	WVVI	0	0.315567	0.304027	0.461297	0.099342	0.099074	0.295843	0.000000	0.494731	
928	2018	XELB	1	0.801902	0.118920	0.114976	0.012318	0.190590	0.196362	0.307528	0.497779	
929	2018	ZKIN	0	0.158609	0.477388	0.475511	0.134586	0.407445	0.536934	0.000000	0.208201	

930 rows × 20 columns

The dimension of the managed data set is 930 records with 20 columns.

1. Define the independent variables

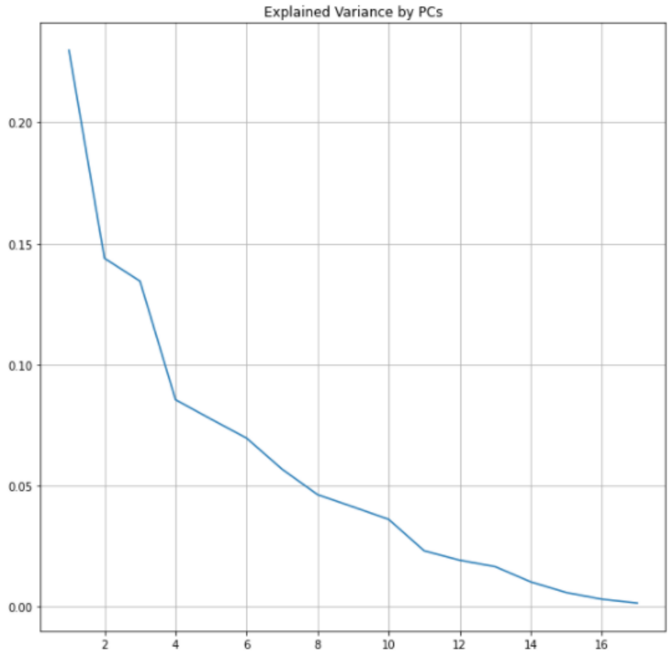
Among these columns, Data Year, Stock and Class are for reference usage and the rest 17 columns are used to train the unsupervised models.

2. Principal components analysis (PCA)

In order to further scale down the dimension and model complexity, I would use PCA to streamline and extract the most important attributes and create new features to train the models. I loop out the whole features to run PCA and pick up the best number of components based on the respective explained variances (elbow method).

```
# Using Principle Components to define scale down the variables
# Determine the number of PC
PCA_test = PCA(n_components=len(feature_column))
PCA_test.fit(X)
plt.figure(figsize=(10,10))
plt.grid()
plt.title('Explained Variance by PCs')
plt.plot(range(1,len(feature_column)+1), PCA_test.explained_variance_ratio_)
```

```
[<matplotlib.lines.Line2D at 0x2e781e44550>]
```



According to the elbow plot I would take two principal components to train different unsupervised models.

3. Model verification

In the following model verification, I would like to go through several unsupervised machine learning approaches and evaluate the model effectiveness.

a. KMeans

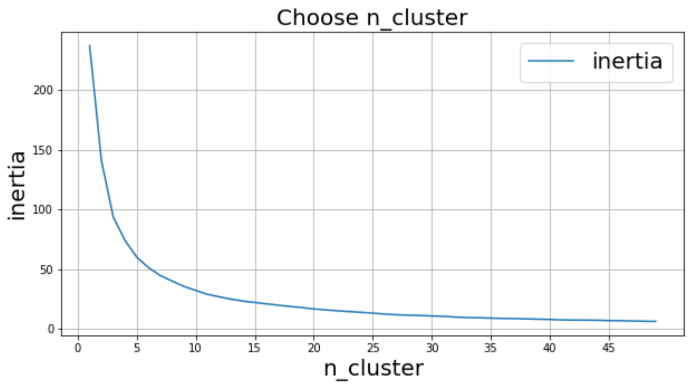
First of all I would create and train a KMeans model to see how it clusters the data set.

A good question is how to determine the optimal hyperparameter “n_cluster”. The quick answer is to fit a number of KMeans models with different “n_cluster” values and check the inertia value respectively through the elbow method.

```python
# Here we start with Kmeans to cluster the stock data
n_clusters = range(1,50)
k_df = pd.DataFrame()
for n in n_clusters:
    kmeans = KMeans(n_clusters=n, random_state=0).fit(PCA_X)
    inertia = kmeans.inertia_
    s = pd.Series({'n_cluster':n,
                   'inertia': inertia})
    k_df = k_df.append(s, ignore_index=True)
ax = k_df.plot(kind='line', x='n_cluster', y='inertia', figsize=(10,5), grid=True)
ax.set_title('Choose n_cluster', fontsize=20)
ax.set_xlabel('n_cluster', fontsize=20)
ax.set_ylabel('inertia', fontsize=20)
ax.set_xticks(range(0,50,5))
ax.legend(fontsize=20)
```

```
C:\Users\mikee\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:882: UserWarning: KMeans is known to have a memory leak o
n Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=4.
  f"KMeans is known to have a memory leak on Windows "
```

```
<matplotlib.legend.Legend at 0x2e78e35b470>
```

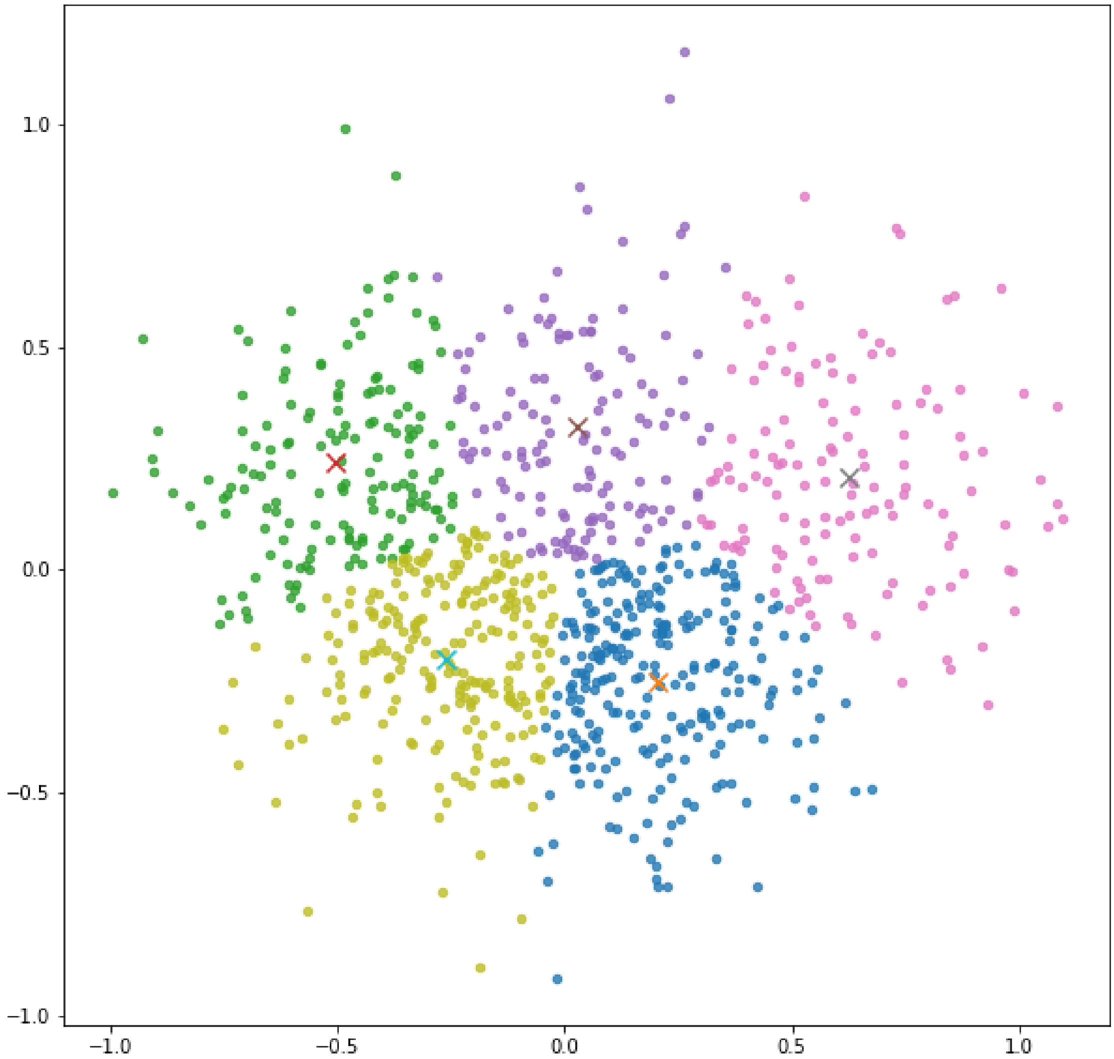


From the elbow plot the slope of the curve decreases at n_cluster equals to 5, which indicates we may set 5 clusters to train the KMeans model.

```python
# Set n_cluster = 5 according to elbow method and fit PCA X
kmeans = KMeans(n_clusters=5, random_state=0).fit(PCA_X)
inertia = kmeans.inertia_
print(inertia)
PCA_label_array = kmeans.predict(PCA_X)

for c in set(PCA_label_array):
    print('Cluster {} has {} data points'.format(c, PCA_X[kmeans.labels_ == c].shape[0]))

display_cluster(PCA_X, kmeans, num_clusters=set(PCA_label_array))

tmp_df = indicator_data.copy()
# tmp_df['kmeans cluster'] = label_array
tmp_df['kmeans cluster'] = PCA_label_array

# tmp_column = tmp_df.columns.to_list()
# tmp_column1 = tmp_column[0:3]
# tmp_column1.append(tmp_column[-1])
# tmp_column1.extend(tmp_column[3:-1])
# tmp_df = tmp_df[tmp_column1]

report_df = cluster_report(tmp_df)
report_df.set_index('cluster', inplace=True, drop=True)
report_df.plot(kind='barh', figsize=(10,10), title='Portfolio Cluster')
```

```
59.96630803620097
Cluster 0 has 258 data points
Cluster 1 has 161 data points
Cluster 2 has 139 data points
Cluster 3 has 138 data points
Cluster 4 has 234 data points
```

The clustering effect is as following:



From the scatter plot each cluster is clearly grouped with distinct boundaries to others. The distribution of data points in each cluster is as following:

Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4
258	161	139	138	234

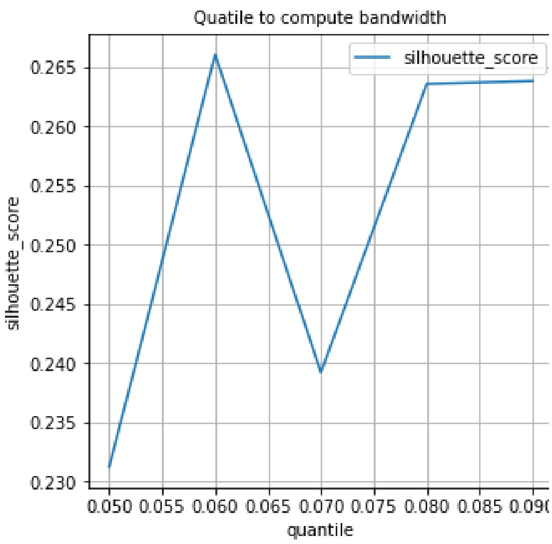
b. MeanShift

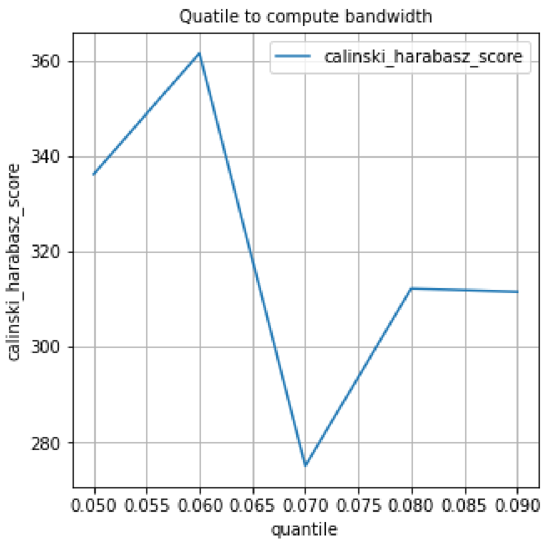
The next step is to train the MeanShift model. Before creating and training the model, it is important to choose the bandwidth hyperparameter with proper value.

Since the data set is distributed densely, and at least 3 to 4 clusters are desired, it seems that the bandwidth value would be set to a smaller one in order to create more meaningful clusters. Therefore I would limit the quantile values between 0.05 and 0.09 and choose the one that performs better in terms of silhouette_score and calinski_harabasz_score.

The computing result is as following:

	quantile	bandwidth	silhouette_score	calinski_harabasz_score
0	0.05	0.183865	0.231248	336.130076
1	0.06	0.200607	0.266052	361.491564
2	0.07	0.217754	0.239212	275.107609
3	0.08	0.232078	0.263564	312.239970
4	0.09	0.245833	0.263810	311.549997





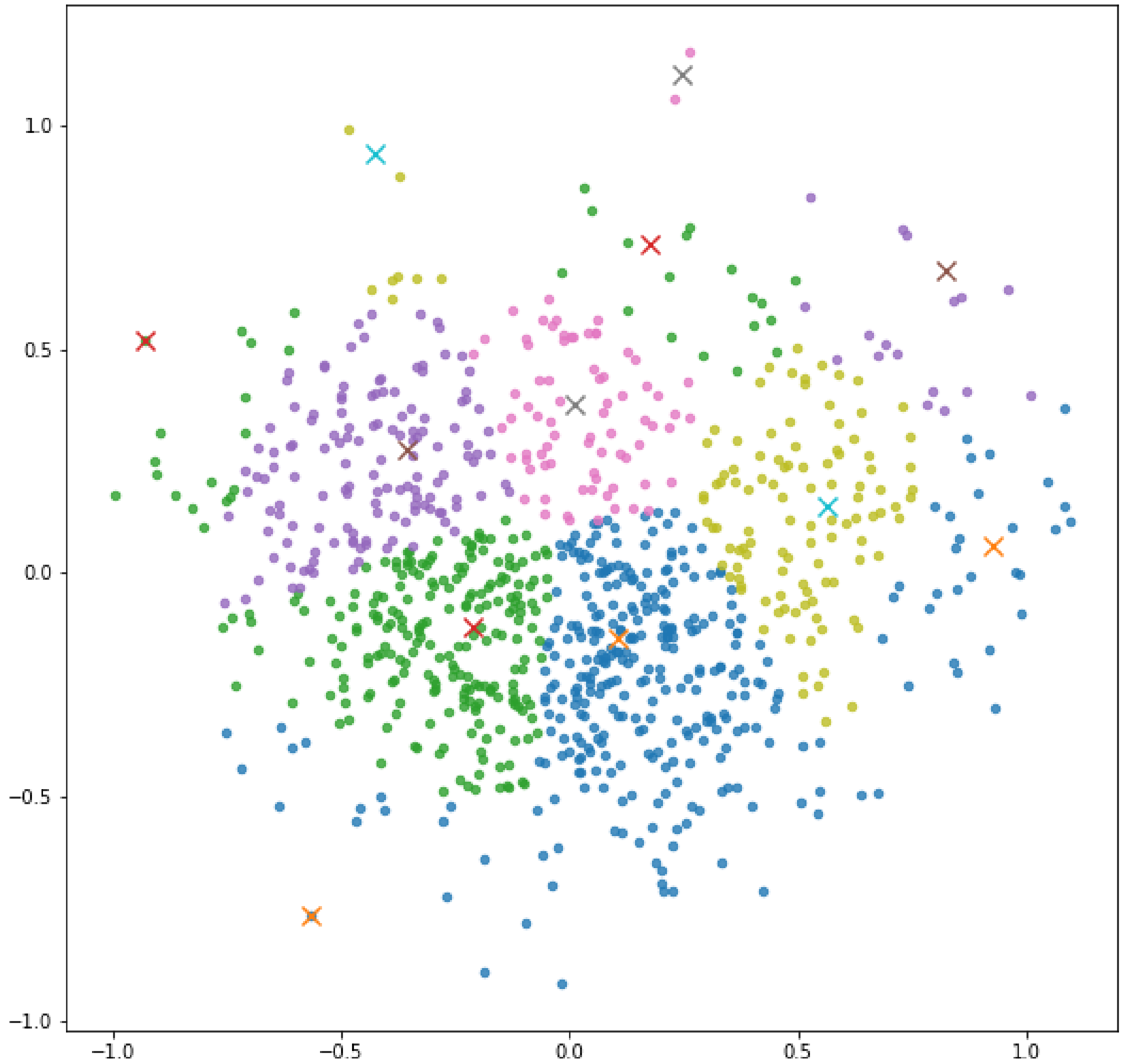
It is obvious that when the quantile is 0.06, the computed bandwidth can generate better scores. So as a consequence I would take it to train the model.

```
#estimate bandwidth:: Loop in
bandwidth = estimate_bandwidth(PCA_X, quantile=0.06)
#Mean Shift method
model = MeanShift(bandwidth = bandwidth, bin_seeding = True)
model.fit(PCA_X)
labels = model.predict(PCA_X)
cluster_num = len(set(labels))
for c in set(labels):
    print('Cluster {} has {} data points'.format(c, PCA_X[model.labels_ == c].shape[0]))
display_cluster(PCA_X, model, num_clusters=set(labels))

tmp_df = indicator_data.copy()
tmp_df['mean shift cluster'] = labels
report_df = cluster_report(tmp_df, n_cluster=range(0, cluster_num), key_column='mean shift cluster')
report_df.set_index('cluster', inplace=True, drop=True)
report_df.plot(kind='barh', figsize=(10,10))
```

```
Cluster 0 has 276 data points
Cluster 1 has 221 data points
Cluster 2 has 138 data points
Cluster 3 has 75 data points
Cluster 4 has 107 data points
Cluster 5 has 31 data points
Cluster 6 has 18 data points
Cluster 7 has 17 data points
Cluster 8 has 2 data points
Cluster 9 has 8 data points
Cluster 10 has 19 data points
Cluster 11 has 18 data points
```

Cluster	Data points
0	276
1	221
2	138
3	75
4	107
5	31
6	18
7	17
8	2
9	8
10	19
11	18

With the selected bandwidth setting, the MeanShfit model provides more clusters than KMeans does, and the cluster distribution is as following:



c. Hierarchical Clustering Analysis (HCA) with different hyperparameters.

Then the next clustering approach is HCA. It provides a bottom - up grouping mechanism to establish the final clusters. Similar to MeanShift, the HCA requires hyperparameters to conduct the clustering. Therefore I would evaluate the hyperparameter combination with silhouette_score and calinski_harabasz_score and pick up the best one.

The key hyperparameters for HCA are n_clusters and linkage. I would look up n_clusters in a range from **2 to 10** and linkage in **ward, single, complete and average**.

```python
# Find out the optimal cluster number for HCA
linkages = ['ward', 'single', 'complete', 'average']
metrics_df = pd.DataFrame()
for cluster in range(2,11):
    for linkage in linkages:
        agglo_ = AgglomerativeClustering(n_clusters=cluster, linkage=linkage, compute_full_tree=True)
        agglo_.fit(PCA_X)
        agglo_labels = agglo_.fit_predict(PCA_X)
        sil_score = silhouette_score(PCA_X, agglo_labels)
        calin_score = calinski_harabasz_score(PCA_X, agglo_labels)
        s = pd.Series({'n_cluster':cluster,
                       'linkage': linkage,
                       'silhouette_score':sil_score,
                      'calinski_harabasz_score':calin_score})
        metrics_df = metrics_df.append(s, ignore_index=True)
#         plot_hca(agglo_, linkage)
metrics_df = metrics_df[['n_cluster','linkage', 'silhouette_score', 'calinski_harabasz_score']]
metrics_df.sort_values(by=['silhouette_score', 'calinski_harabasz_score'], ascending=False)
```

	n_cluster	linkage	silhouette_score	calinski_harabasz_score
1	2.0	single	0.444174	10.291476
3	2.0	average	0.376252	416.801905
0	2.0	ward	0.347690	585.370683
4	3.0	ward	0.321414	564.159954
8	4.0	ward	0.320120	581.441852
7	3.0	average	0.313038	227.776467
12	5.0	ward	0.306366	555.824249
11	4.0	average	0.286975	354.321542
28	9.0	ward	0.283918	558.268383
24	8.0	ward	0.280083	554.616150
16	6.0	ward	0.279740	567.024228
20	7.0	ward	0.279168	553.050537
32	10.0	ward	0.276153	551.461081
22	7.0	complete	0.275764	532.278947
26	8.0	complete	0.272803	499.601301
23	7.0	average	0.270766	390.154917
18	6.0	complete	0.257842	451.087642
15	5.0	average	0.256988	327.737404
27	8.0	average	0.253421	351.600722
19	6.0	average	0.251540	264.810829
30	9.0	complete	0.250727	459.679290
31	9.0	average	0.240817	338.824774
5	3.0	single	0.239648	6.948376
14	5.0	complete	0.236963	429.619937
34	10.0	complete	0.235984	440.795060
35	10.0	average	0.231140	302.389189
6	3.0	complete	0.224472	338.476101
2	2.0	complete	0.218983	286.736601
10	4.0	complete	0.199472	334.413460
9	4.0	single	0.190306	7.519156
13	5.0	single	0.158758	6.808510
17	6.0	single	0.098294	6.261230
21	7.0	single	0.091730	6.787451
25	8.0	single	0.084601	7.983137
29	9.0	single	0.033359	7.430796
33	10.0	single	0.022884	6.603844

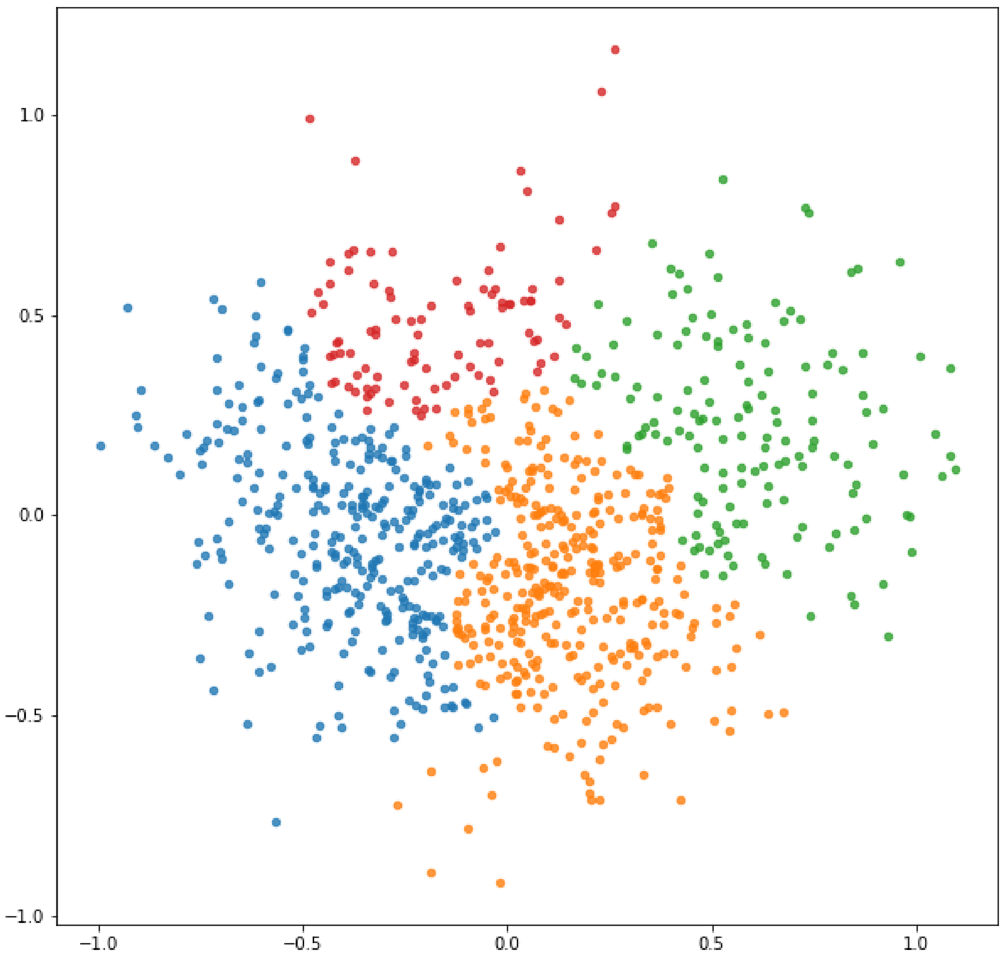
Through the looping results, the combination of **n_clusters=2 and linkage=ward** and **n_clusters=4 and linkage=ward** would generate the best calinski_harabasz_score and good silhouette_score. In this case since I want to group the stock data as investment portfolios, therefore the combination of **n_clusters=4 and linkage=ward** is selected to provide 4 clusters instead of 2 even though it has slightly lower scoring.

```
agglo_ = AgglomerativeClustering(n_clusters=4, linkage='ward', compute_full_tree=True)
agglo_.fit(PCA_X)
agglo_labels = agglo_.fit_predict(PCA_X)
for c in set(agglo_labels):
    print('Cluster {} has {} data points'.format(c, PCA_X[agglo_.labels_ == c].shape[0]))
display_cluster(PCA_X, agglo_, num_clusters=set(agglo_labels))

tmp_df = indicator_data.copy()
tmp_df['HCA cluster'] = agglo_labels
report_df = cluster_report(tmp_df, n_cluster=range(0,4), key_column='HCA cluster')
report_df.set_index('cluster', inplace=True, drop=True)
report_df.plot(kind='barh', figsize=(10, 10), title='Portfolio Cluster')

c_df = c_df.append(add_result('HCA', agglo_labels), ignore_index=True)
```

```
Cluster 0 has 327 data points
Cluster 1 has 358 data points
Cluster 2 has 149 data points
Cluster 3 has 96 data points
```

Cluster	Data points
0	327
1	358
2	149
3	96

The clustering effect is as following:



4. Model selection

In the model verification section, I have trained three unsupervised models and each of them has grouped data points into clusters. To evaluate the effectiveness of the clustering, I have also collected the silhouette_score and calinski_harabasz_score from each trained model and summarized as following:

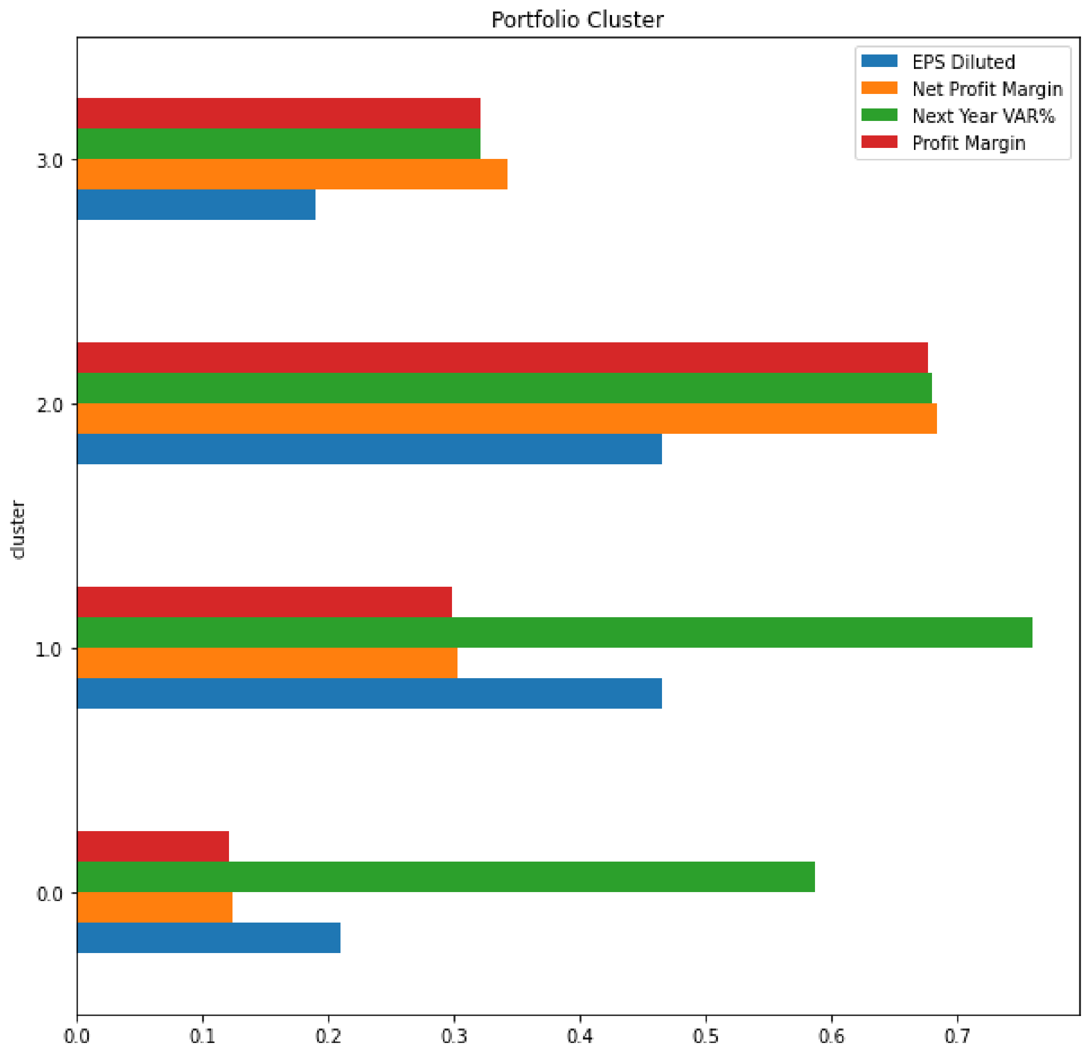
	model	silhouette_score	calinski_harabasz_score
0	KMeans	0.174772	392.754973
1	MeanShift	0.266053	361.496133
2	HCA	0.320115	581.433660

The HCA model with **n_clusters=4 and linkage=ward** hyperparameters provides the best efficient clustering effects based on the highest scores. It tells that the HCA model has the best dispersion ratio within and between clusters.

Key Findings and Insights, which walks your reader through the main drivers of your model and insights from your data derived from your classification model.

As mentioned in the beginning of this analysis, the objective is to find out the most profitable stock portfolio through unsupervised learning. Now let's examine the **average** values of key financial indicators within each cluster (portfolio) generated by the selected HCA model.

By default I will pick up four indicators which are Next Year VAR%, EPS Diluted, Profit Margin, and Net Profit Margin respectively to simplify the histogram.



From the histogram it is said that cluster 1 and cluster 2 have higher next year price variation (2019 price variation) and EPS than cluster 3 and cluster 4. So these two can be assumed as good candidate portfolios for the stock investors to collect target stocks with the indicators that perform above the average.

```python
# Real data mapping
agglo_labels.shape

indicator_data['HCA Cluster'] = agglo_labels
indicator_data.shape

indicator_data.loc[(indicator_data['HCA Cluster']==1) |
                   (indicator_data['HCA Cluster']==2)].sort_values(by=['Next Year VAR%',
                                                                      'EPS Diluted',
                                                                      'Profit Margin',
                                                                      'Net Profit Margin'], ascending=False)
```

	Data Year	Stock	Class	Next Year VAR%	Profit Margin	Net Profit Margin	EPS Diluted	returnOnAssets	returnOnEquity	priceToFreeCashFlowsRatio	...	priceEarningsToGrowthRatio
638	2018	SEAC	1	1.000000	0.616309	0.615959	0.101809	0.426676	0.479847	0.447135	...	0.303284
364	2018	UCTT	1	0.987288	0.126646	0.125442	0.217348	0.151305	0.342320	0.581314	...	0.299696
404	2018	SEDG	1	0.982793	0.509070	0.509418	0.435446	0.412270	0.605489	0.465407	...	0.387576
612	2018	VEC	1	0.961560	0.107302	0.102925	0.471186	0.604325	0.483461	0.419306	...	0.245849
521	2018	BLD	1	0.957627	0.218209	0.213885	0.523240	0.554731	0.421531	0.509996	...	0.373877
...	...	...	...	...	...	...	...	...	...	...	...	...
16	2018	SWN	0	0.092318	0.516076	0.514690	0.215595	0.215267	0.602421	0.153677	...	0.104672
20	2018	X	0	0.085194	0.300334	0.296708	0.664546	0.320326	0.666253	0.000000	...	0.056391
906	2018	NEWA	0	0.072538	0.564779	0.564604	0.166511	0.464115	0.668568	0.000000	...	0.293660
200	2018	HCLP	0	0.054181	0.599207	0.599144	0.292343	0.225674	0.503047	0.277690	...	0.023465
192	2018	QRTEA	0	0.043152	0.248269	0.246319	0.296522	0.940105	0.489590	0.433930	...	0.325535

507 rows × 21 columns

Let’s pick up some of the stock price charts (from January 2020 to July 2021) to see if this assumption sounds reasonable at this moment.
(Quote source: Yahoo Finance)

- **Cluster 1**
  - Ultra Clean Holdings, Inc. (UTCC)

- Vectrus, Inc. (VEC)

- TopBuild Corp. (BLD)

- **Cluster 2**
    - SolarEdge Technologies, Inc (SEDG)

    - Luna Innovations Incorporated (LUNA)

- Teradyne, Inc. (TER)

All the samples from cluster 1 and cluster 2 performed great in 2020 whole year and 2021 up to date. Therefore I think the HCA model has the potential and is worth further evaluating.

Possible flaws in the models

1. The analysis is purely built on company financial and operational performance data. The macro economic indicators such as GDP, interest rate and exchange rate are not taken into consideration. Also, the respective industry growth indicators are not included.
2. Dropped data sets due to NA values may dilute or jeopardize the model training and model evaluation.
3. Only focus on a single - year data source.

Suggestions for next steps

The follow up steps for this analysis can be:

1. Sampling the clustered real data to see if the suggested portfolio matches actual performance.
2. Accumulate multiple - year stock data and redo the analysis to see if HCA is still the first choice with larger data volume.
3. Try to fill up the NA values from other data sources so that less data points would be sacrificed.
4. In the long term to collect industry growth data and macro economic data and integrate with company performance data to refine the analysis.