# DATA 3300

## ICE - Clustering Analysis

# Name: Cabot Steward

## Q1

**Load the required packages and then import the dataset**

In [3]:
```
!pip install kneed
```

```
Collecting kneed
  Downloading kneed-0.8.5-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: numpy>=1.14.2 in c:\users\tucke\anaconda3\lib\site-pa
ckages (from kneed) (1.26.4)
Requirement already satisfied: scipy>=1.0.0 in c:\users\tucke\anaconda3\lib\site-pac
kages (from kneed) (1.13.1)
Downloading kneed-0.8.5-py3-none-any.whl (10 kB)
Installing collected packages: kneed
Successfully installed kneed-0.8.5
```

In [4]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE

import pandas as pd
import numpy as np
from kneed import KneeLocator
import sklearn.cluster
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

In [6]:
```
# replace with code to read in the dataframe
# replace with code to display a heading
df = pd.read_csv("sandp500 (4).csv")
df.head()
```

Out[6]:

| | Symbol | Name | Sector | Price | Dividend Yield | Price/Earnings | Earnings/Share | B Va |
|---|---|---|---|---|---|---|---|---|
| **0** | A | Agilent Technologies Inc | Health Care | 51.21 | 1.02 | 32.85 | 1.56 | 1: |
| **1** | AAL | American Airlines Group | Industrials | 44.84 | 0.85 | 9.32 | 4.81 | |
| **2** | AAP | Advance Auto Parts | Consumer Discretionary | 151.99 | 0.15 | 24.51 | 6.20 | 3! |
| **3** | AAPL | Apple Inc. | Information Technology | 139.52 | 1.63 | 16.75 | 8.33 | 2! |
| **4** | ABBV | AbbVie | Health Care | 63.69 | 4.04 | 17.55 | 3.63 | |

### 1A

**Before running the actual clustering analysis, you're going to exclude the Symbol and Name variables. Explain why this is important to do:**

These are both unique categorical identifiers. They dont carry any meaningful values

### 1B

**Also there would be a problem with keeping all three of Price, 52 Week High, and 52 Week Low involved when the clustering analysis is run. Explain why this is the case.**

high and low are bounds that were established, it can influence their clustering

In [7]:

```
df = df.drop(['Symbol', 'Name', 'Sector', '52 week low', '52 week high'], axis=1)
df = df.dropna()
df.head()
# replace with code to drop missing values
# replace with code to display a heading
```

Out[7]:

| | Price | Dividend Yield | Price/Earnings | Earnings/Share | Book Value | Market Cap | EBITDA | Price/Sales | F |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 51.21 | 1.02 | 32.85 | 1.56 | 13.35 | 16.49 | 0.942 | 3.89 | |
| 1 | 44.84 | 0.85 | 9.32 | 4.81 | 7.46 | 22.61 | 7.830 | 0.57 | |
| 2 | 151.99 | 0.15 | 24.51 | 6.20 | 39.66 | 11.18 | 1.120 | 1.19 | |
| 3 | 139.52 | 1.63 | 16.75 | 8.33 | 25.19 | 732.00 | 69.750 | 3.35 | |
| 4 | 63.69 | 4.04 | 17.55 | 3.63 | 2.91 | 101.52 | 10.950 | 3.95 | |

## Q2

**Run a clustering analysis in Python to find three groups of similar stocks.**

**Your analysis should address each of the following considerations (pre-processing)**

1. **It should remove variables not to be included in the analysis**
2. **It should standardize all variables using the Z-transformation method**
3. **It should utilize the K-means clustering method**

In [9]:
```
features = df[["Price", "Dividend Yield", "Price/Earnings", "Earnings/Share", "Book
#selects features for inclusion in analysis
```

In [10]:
```
scaler = StandardScaler() #uses the standard scaler function
scaled_features = scaler.fit_transform(features) #students should add a description
scaled_features
```

Out[10]:
```
array([[-0.3846388 , -0.65969446,  0.07617064, ..., -0.44080382,
         0.21163671, -0.17976872],
       [-0.43859869, -0.78805688, -0.53396148, ...,  0.64328762,
        -0.95969278, -0.03094617],
       [ 0.46906269, -1.31660801, -0.14008529, ..., -0.41278868,
        -0.74095052, -0.17643189],
       ...,
       [ 0.17325741, -0.8182598 ,  1.23472156, ..., -0.14837613,
        -0.07766756, -0.27386728],
       [-0.43487148, -0.89376711, -0.18572203, ..., -0.58906371,
         0.25397392, -0.3472775 ],
       [-0.36888284, -0.83336126,  0.05827896, ..., -0.32150339,
         0.72321133,  0.7378591 ]])
```

In [11]:
```
# add comments where you see ... to explain each kmeans parameter
kmeans = KMeans(        #creates an object called kmeans, set equal to function KMe
    init="random",      # init does...
    n_clusters=3,       # n_clusters does...
    n_init=10,          # number of times for different initializations is 10
    max_iter=300,       # number of iterations to update centroids is 300
```

```
    random_state=42     # allows these methods to be reproduced
)
```

In [13]:
```
kmeans.fit(scaled_features) #fit kmeans model to scaled data
# replace with code to find SSE value or inertia value of the model
kmeans.inertia_
```

```
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
```

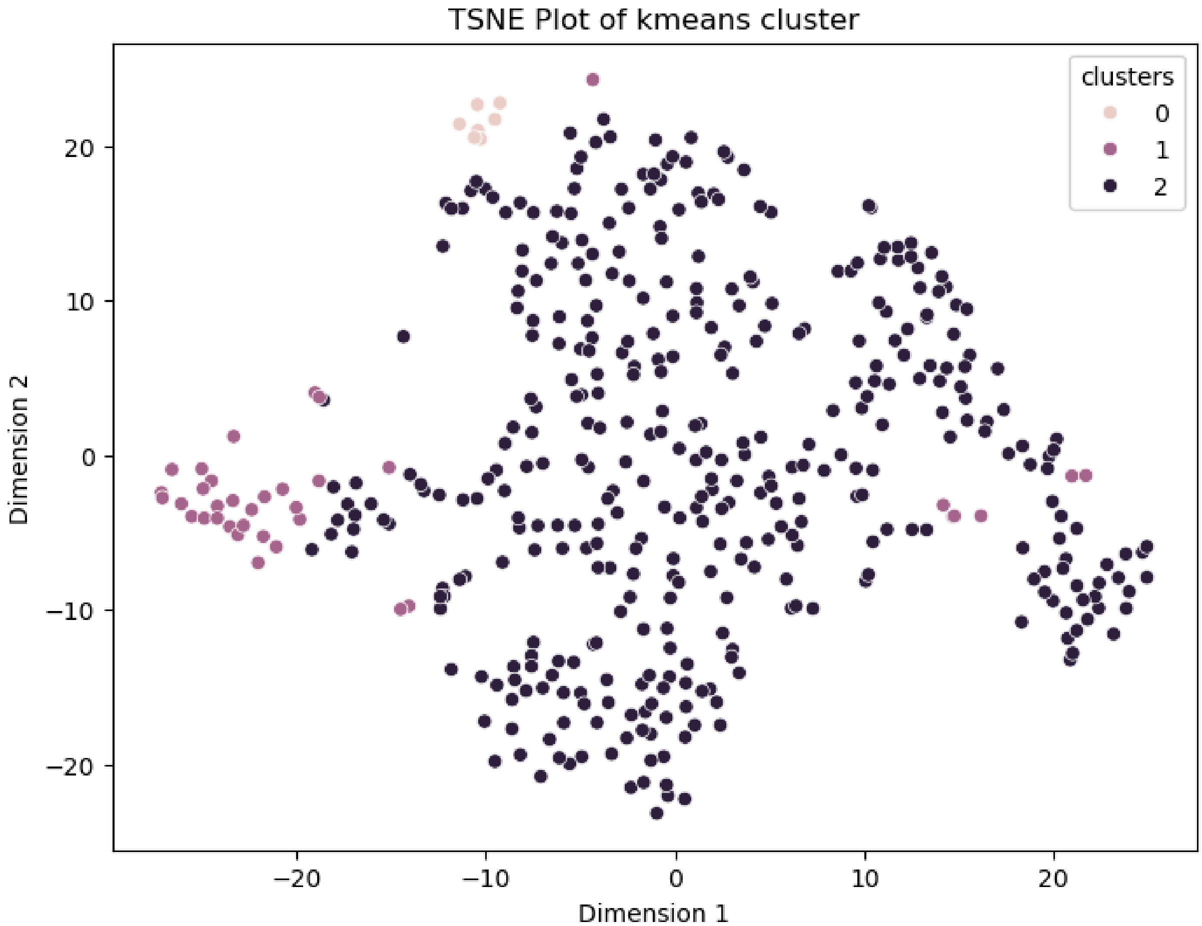
Out[13]: 2796.0618193387277

In [17]:
```
# Set up the TSNE model to visualize clusters
model = TSNE(n_components= 2, random_state= 42)

# Reduce dimensionality and obtain the transformed data
transformed_data = model.fit_transform(scaled_features)

# Extract the cluster labels
cluster_labels = kmeans.labels_

# Create a scatter plot with color-coded clusters
plt.figure(figsize=(8,6))
sns.scatterplot(x=transformed_data[:, 0], y=transformed_data[:, 1], hue=cluster_lab
plt.title('TSNE Plot of kmeans cluster')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.legend(title='clusters')
plt.show()
```




**Using the TSNE visualization, how can we describe the distribution of observations into clusters? Does three clusters appear to be sufficient?**

three clusters do not appear to be sufficient, because we have a dominant edge.

## Q3

**Use the Elbow rule to determine the optimal number of clusters between 1-11 by:**

1. **Running k-means iteratively on k of size 1-11**
2. **Plotting the SSE curve by k size**
3. **Using the Knee Locator method**

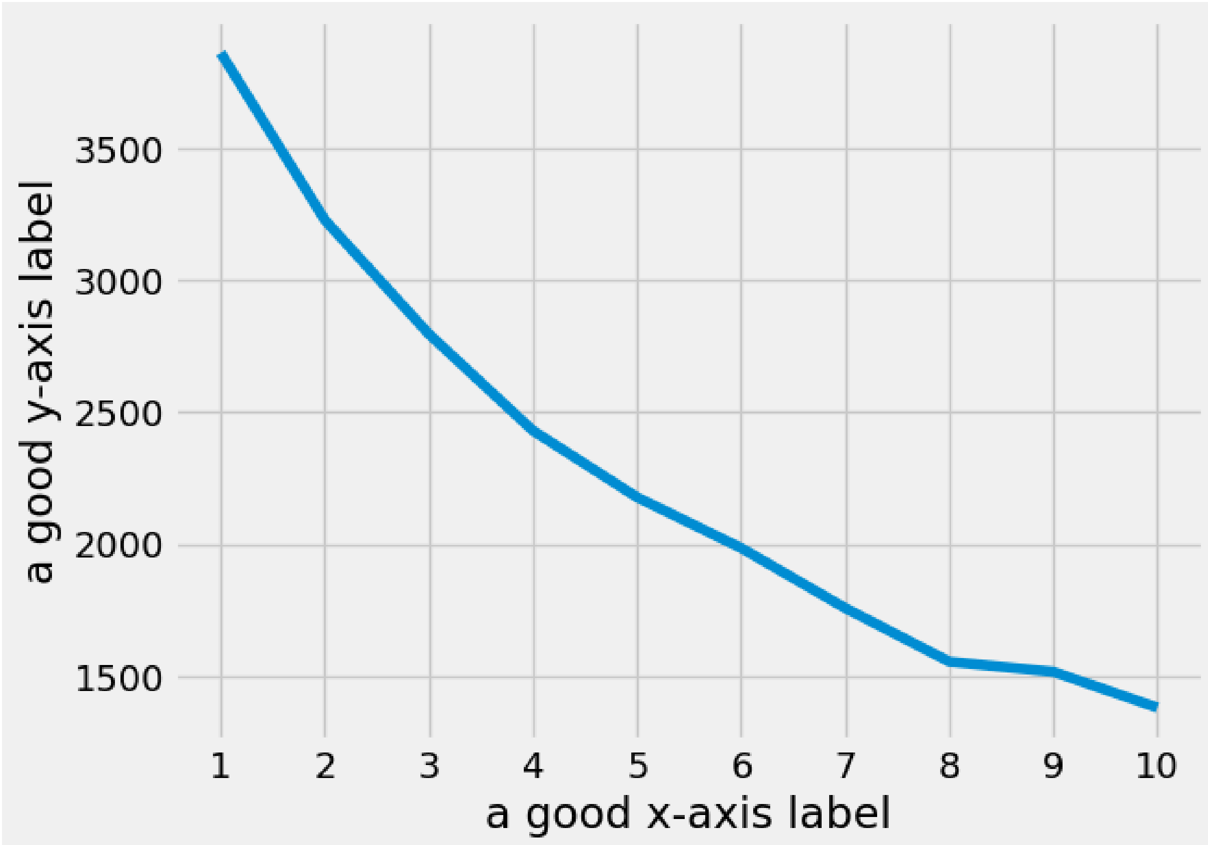
In [20]:
```
kmeans_kwargs = {
    "init": "random",
    "n_init": 10,
    "max_iter": 300,
    "random_state": 42,
} #creates a kmeans initialization dictionary


# replace with code to create empty list for SSE values
sse = []
for k in range(1,11): # replace a and b with the range of values for the number of
```

```
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(scaled_features)
    sse.append(kmeans.inertia_)
```

```
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
```

In [21]:
```
plt.style.use("fivethirtyeight")
plt.plot(range(1,11), sse)
plt.xticks(range(1,11))
plt.xlabel("a good x-axis label") #replace with code to label x-axis
plt.ylabel("a good y-axis label") #replace with code to label y-axis
plt.show() #creates elbow plot
```




In [24]:
```
kl = KneeLocator(
    range(1,11), sse, curve="convex", direction="decreasing"                #
)

# replace with code to locate knee/elbow in the plot
kl.elbow
```

Out[24]: 4

## Q4

**Rerun kmeans with the optimal number of clusters and report its SSE value. Is it less than when k was set to 3? Why would this occur?**

In [25]:
```
# copy down the initial kmeans parameters, replace with code to run 5 clusters
kmeans = KMeans(        #creates an object called kmeans, set equal to function KMe
    init="random",      # init does...
    n_clusters=4,       # n_clusters does...
    n_init=10,          # number of times for different initializations is 10
    max_iter=300,       # number of iterations to update centroids is 300
```

```
    random_state=42     # allows these methods to be reproduced
)
```

In [26]:
```
kmeans.fit(scaled_features)
kmeans.inertia_
```

```
c:\Users\tucke\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
```

Out[26]: 2430.0379910652787

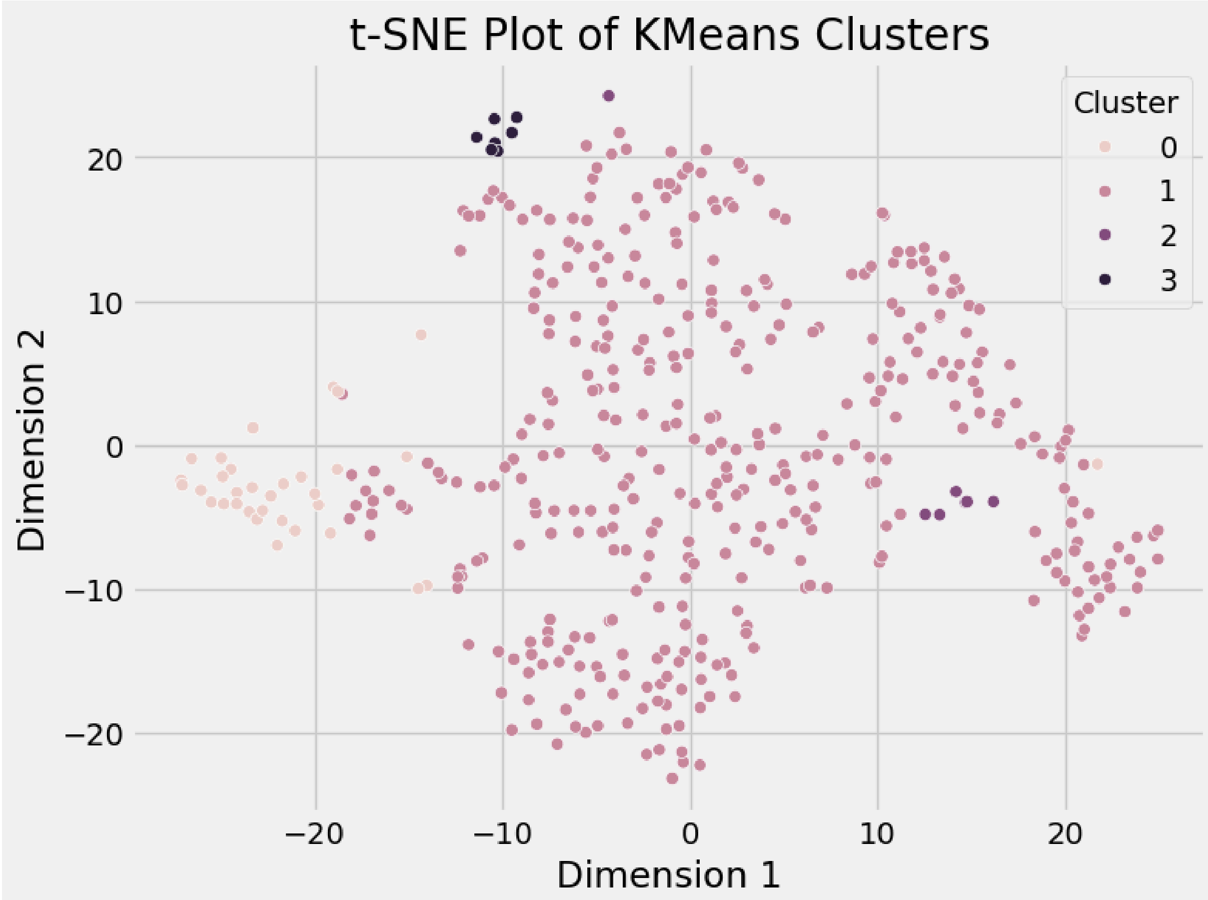
the inertia will always decrease when adding additional clusters because you have fewer observations per cluster therefor less.

In [27]:
```
# Set up the TSNE model
model = TSNE(n_components=2, random_state=42)

# Reduce dimensionality and obtain the transformed data
transformed_data = model.fit_transform(scaled_features)

# Extract the cluster labels
cluster_labels = kmeans.labels_

# Create a scatter plot with color-coded clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x=transformed_data[:, 0], y=transformed_data[:, 1], hue=cluster_lab
plt.title('t-SNE Plot of KMeans Clusters')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.legend(title='Cluster')
plt.show()
```




**What are cluster plots like the TNSE useful for, and what should they not be used for?**

the cluster plot makes visually representative clusters whcih can aid in decision making.

## Q5

**Generate a centroid table using the cluster_centers_ feature from kmeans, convert the array into a dataframe. Which cluster of stocks are the highest price on average, how do you know?**

In [32]:
```
# replace with code to save cluster centroids to an object called 'centroids'
centroids = kmeans.cluster_centers_
centroid_table = pd.DataFrame(centroids,
                             columns = ["Price", "Dividend Yield", "Price/Earnings
                                        "Earnings/Share", "Book Value", "Market Ca
                                        "EBITDA", "Price/Sales", "Price/Book"], #L
                             index =['Cluster_0', 'Cluster_1', 'Cluster_2', 'Clust

# replace with code to display table
centroid_table
```

Out[32]:

| | Price | Dividend Yield | Price/Earnings | Earnings/Share | Book Value | Market Cap | EBIT |
|---|---|---|---|---|---|---|---|
| **Cluster_0** | -0.034250 | 0.635141 | -0.204572 | 0.212148 | -0.344659 | 2.329804 | 2.394 |
| **Cluster_1** | -0.112840 | -0.019523 | -0.100159 | -0.103349 | -0.072503 | -0.236930 | -0.215 |
| **Cluster_2** | 1.671767 | -1.009185 | 6.567219 | -0.710167 | -0.052085 | 0.556646 | -0.237 |
| **Cluster_3** | 4.647560 | -0.919655 | -0.136974 | 5.349959 | 5.633525 | 1.389623 | 0.687 |

cluster 4.

## Q6

**Now create a new centroid table after de-normalizing the centroid values. Why is it important to de-normalize your centroids after the fact?**

In [36]:

```
# replace with code to de-normalize centroid values from centroids object, called u
unscaled = scaler.inverse_transform
unscaled_centroid_table = pd.DataFrame(unscaled,
                            columns = df.columns,
                            index =['Cluster_0', 'Cluster_1', 'Cluster_2', 'Clust

unscaled_centroid_table #produces a non-scaled centroid table
```

Out[36]:

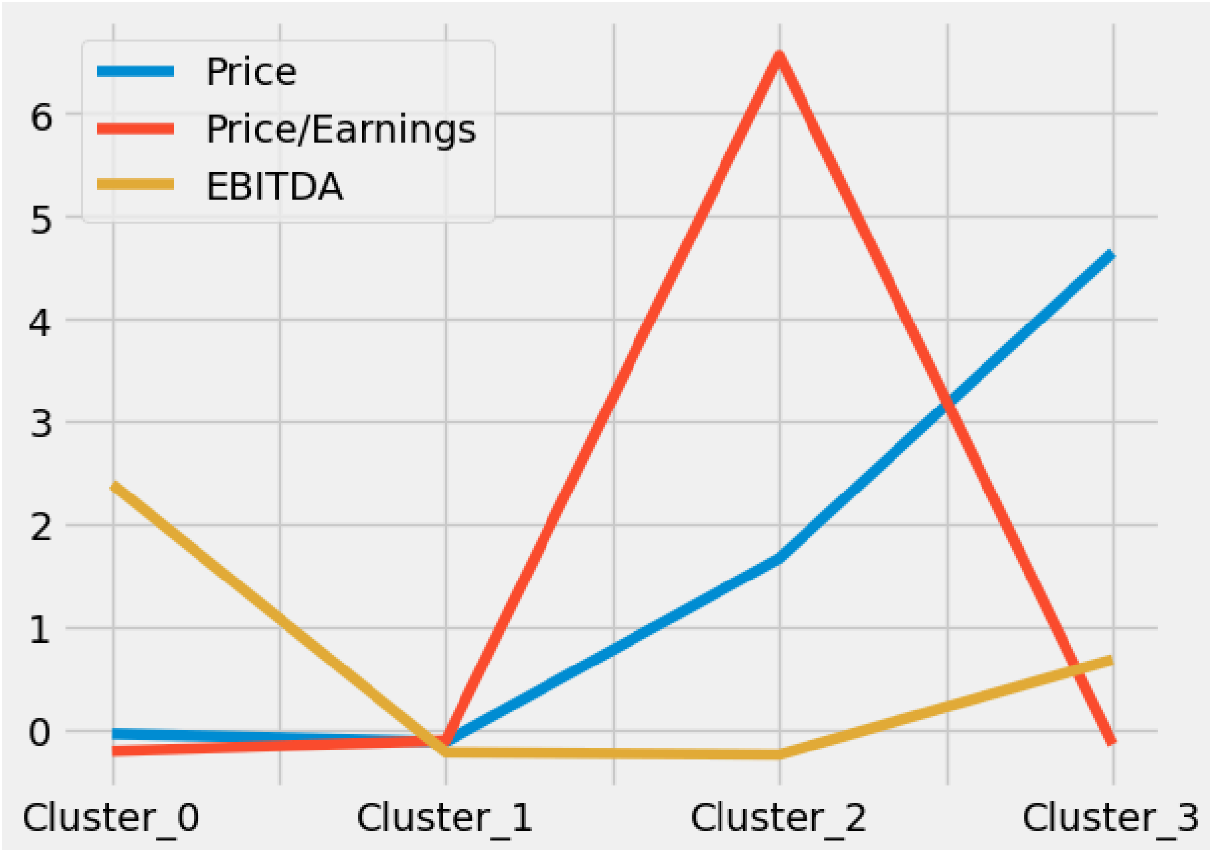
| | Price | Dividend Yield | |
|---|---|---|---|
| **Cluster_0** | <bound method StandardScaler.inverse_transform... | <bound method StandardScaler.inverse_transform... | StandardScaler.in |
| **Cluster_1** | <bound method StandardScaler.inverse_transform... | <bound method StandardScaler.inverse_transform... | StandardScaler.in |
| **Cluster_2** | <bound method StandardScaler.inverse_transform... | <bound method StandardScaler.inverse_transform... | StandardScaler.in |
| **Cluster_3** | <bound method StandardScaler.inverse_transform... | <bound method StandardScaler.inverse_transform... | StandardScaler.in |
| **Cluster_4** | <bound method StandardScaler.inverse_transform... | <bound method StandardScaler.inverse_transform... | StandardScaler.in |

denormalizing is important because it can allow us to gain a different insight

## Q7

**Explore plotting the centroid values to examine differences between clusters across the different variables of interest.**

In [38]:
```
centroid_table.plot(kind = 'line', y = ['Price', 'Price/Earnings','EBITDA' ]) #repl
plt.show()
```




Then come up with a brief descriptive title for each of the 5 clusters of stocks:

1. **Cluser_0:** good earnings before aver price after
2. **Cluster_1:** poor performance
3. **Cluster_2:** really good price/earnings
4. **Cluster_3:** good price lower price/earnings
5. **Cluster_4** this one broke

## Q8

**Based on your plot and centroid tables, describe which cluster of stocks you'd recommend and why. Create a visualization that supports your recommendation (pull from the data understanding module!). Briefly describe what the viz is showing and why it's relevant.**

In [40]:
```
unscaled_centroid_table.plot(kind = 'bar', y = ['Price', 'Market Cap']) #replace wi
plt.title("an informative title") # replace with code to label viz
plt.ylabel("an informative y-axis label") # replace with code to label y-axis
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 1
----> 1 unscaled_centroid_table.plot(kind = 'bar', y = ['Price', 'Market Cap']) #rep
lace with code for column names
      2 plt.title("an informative title") # replace with code to label viz
      3 plt.ylabel("an informative y-axis label") # replace with code to label y-axi
s

File c:\Users\tucke\anaconda3\Lib\site-packages\pandas\plotting\_core.py:1030, in Pl
otAccessor.__call__(self, *args, **kwargs)
   1027             label_name = label_kw or data.columns
   1028             data.columns = label_name
-> 1030 return plot_backend.plot(data, kind=kind, **kwargs)

File c:\Users\tucke\anaconda3\Lib\site-packages\pandas\plotting\_matplotlib\__init_
_.py:71, in plot(data, kind, **kwargs)
     69         kwargs["ax"] = getattr(ax, "left_ax", ax)
     70 plot_obj = PLOT_CLASSES[kind](data, **kwargs)
---> 71 plot_obj.generate()
     72 plot_obj.draw()
     73 return plot_obj.result

File c:\Users\tucke\anaconda3\Lib\site-packages\pandas\plotting\_matplotlib\core.py:
499, in MPLPlot.generate(self)
    497 @final
    498 def generate(self) -> None:
--> 499     self._compute_plot_data()
    500     fig = self.fig
    501     self._make_plot(fig)

File c:\Users\tucke\anaconda3\Lib\site-packages\pandas\plotting\_matplotlib\core.py:
698, in MPLPlot._compute_plot_data(self)
    696 # no non-numeric frames or series allowed
    697 if is_empty:
--> 698     raise TypeError("no numeric data to plot")
    700 self.data = numeric_data.apply(type(self)._convert_to_ndarray)

TypeError: no numeric data to plot
```

I would recommend cluster 1, this is due to its good performance.