Unsupervised Learning: Trade&Ahead

Marks: 60

Context

The stock market has consistently proven to be a good place to invest in and save for the future. There are a lot of compelling reasons to invest in stocks. It can help in fighting inflation, create wealth, and also provides some tax benefits. Good steady returns on investments over a long period of time can also grow a lot more than seems possible. Also, thanks to the power of compound interest, the earlier one starts investing, the larger the corpus one can have for retirement. Overall, investing in stocks can help meet life's financial aspirations.

It is important to maintain a diversified portfolio when investing in stocks in order to maximise earnings under any market condition. Having a diversified portfolio tends to yield higher returns and face lower risk by tempering potential losses when the market is down. It is often easy to get lost in a sea of financial metrics to analyze while determining the worth of a stock, and doing the same for a multitude of stocks to identify the right picks for an individual can be a tedious task. By doing a cluster analysis, one can identify stocks that exhibit similar characteristics and ones which exhibit minimum correlation. This will help investors better analyze stocks across different market segments and help protect against risks that could make the portfolio vulnerable to losses.

Objective

Trade&Ahead is a financial consultancy firm who provide their customers with personalized investment strategies. They have hired you as a Data Scientist and provided you with data comprising stock price and some financial indicators for a few companies listed under the New York Stock Exchange. They have assigned you the tasks of analyzing the data, grouping the stocks based on the attributes provided, and sharing insights about the characteristics of each group.

Data Dictionary

- Ticker Symbol: An abbreviation used to uniquely identify publicly traded shares of a particular stock on a particular stock market
- Company: Name of the company
- GICS Sector: The specific economic sector assigned to a company by the Global Industry Classification Standard (GICS) that best defines its business operations
- GICS Sub Industry: The specific sub-industry group assigned to a company by the Global Industry Classification Standard (GICS) that best defines its business operations
- Current Price: Current stock price in dollars

- Price Change: Percentage change in the stock price in 13 weeks
- Volatility: Standard deviation of the stock price over the past 13 weeks
- ROE: A measure of financial performance calculated by dividing net income by shareholders' equity (shareholders' equity is equal to a company's assets minus its debt)
- Cash Ratio: The ratio of a company's total reserves of cash and cash equivalents to its total current liabilities
- Net Cash Flow: The difference between a company's cash inflows and outflows (in dollars)
- Net Income: Revenues minus expenses, interest, and taxes (in dollars)
- Earnings Per Share: Company's net profit divided by the number of common shares it has outstanding (in dollars)
- Estimated Shares Outstanding: Company's stock currently held by all its shareholders
- P/E Ratio: Ratio of the company's current stock price to the earnings per share
- P/B Ratio: Ratio of the company's stock price per share by its book value per share (book value of a company is the net difference between that company's total assets and total liabilities)

Importing necessary libraries and data

In [2]:
```
# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_theme(style='darkgrid')

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)

# to scale the data using z-score
from sklearn.preprocessing import StandardScaler

# to compute distances
from scipy.spatial.distance import cdist, pdist

# to perform k-means clustering and compute silhouette scores
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# to visualize the elbow curve and silhouette scores
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer

# to perform hierarchical clustering, compute cophenetic correlation, and create dendr
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, cophenet

# to suppress warnings
```

```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
# loading the dataset
data = pd.read_csv("stock_data.csv")
```

Data Overview

- Observations
- Sanity checks

In [5]:
```
data.shape
```

Out[5]:
```
(340, 15)
```

- There are 340 rows and 15 columns

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 340 entries, 0 to 339
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Ticker Symbol                340 non-null    object 
 1   Security                     340 non-null    object 
 2   GICS Sector                  340 non-null    object 
 3   GICS Sub Industry            340 non-null    object 
 4   Current Price                340 non-null    float64
 5   Price Change                 340 non-null    float64
 6   Volatility                   340 non-null    float64
 7   ROE                          340 non-null    int64  
 8   Cash Ratio                   340 non-null    int64  
 9   Net Cash Flow                340 non-null    int64  
 10  Net Income                   340 non-null    int64  
 11  Earnings Per Share           340 non-null    float64
 12  Estimated Shares Outstanding  340 non-null    float64
 13  P/E Ratio                    340 non-null    float64
 14  P/B Ratio                    340 non-null    float64
dtypes: float64(7), int64(4), object(4)
memory usage: 40.0+ KB
```

- There are 4 Object data types, 7 float data types and 4 integer data type columns.
- Ticker Symbol will be dropped
- All other object data types will be converted to category

In [7]:
```
data.head()
```

Out[7]:

	Ticker Symbol	Security	GICS Sector	GICS Sub Industry	Current Price	Price Change	Volatility	ROE	Cash Ratio	N
0	AAL	American Airlines Group	Industrials	Airlines	42.349998	9.999995	1.687151	135	51	-60
1	ABBV	AbbVie	Health Care	Pharmaceuticals	59.240002	8.339433	2.197887	130	77	5
2	ABT	Abbott Laboratories	Health Care	Health Care Equipment	44.910000	11.301121	1.273646	21	67	93
3	ADBE	Adobe Systems Inc	Information Technology	Application Software	93.940002	13.977195	1.357679	9	180	-24
4	ADI	Analog Devices, Inc.	Information Technology	Semiconductors	55.320000	-1.827858	1.701169	14	272	31

In [8]:
```
data.duplicated().sum()
```

Out[8]:
```
0
```

- There are no duplicate rows in the data set.

In [12]:
```
data.isnull().sum()
```

Out[12]:
```
Ticker Symbol                  0
Security                       0
GICS Sector                    0
GICS Sub Industry              0
Current Price                  0
Price Change                   0
Volatility                     0
ROE                            0
Cash Ratio                     0
Net Cash Flow                  0
Net Income                     0
Earnings Per Share             0
Estimated Shares Outstanding   0
P/E Ratio                      0
P/B Ratio                      0
dtype: int64
```

- There are no NULL values in any column.

In [10]:
```
data.describe(include='all').T
```

Out[10]:

	count	unique	top	freq	mean	std	min
Ticker Symbol	340	340	AAL	1	NaN	NaN	NaN
Security	340	340	American Airlines Group	1	NaN	NaN	NaN
GICS Sector	340	11	Industrials	53	NaN	NaN	NaN
GICS Sub Industry	340	104	Oil & Gas Exploration & Production	16	NaN	NaN	NaN
Current Price	340.0	NaN	NaN	NaN	80.862345	98.055086	4.5
Price Change	340.0	NaN	NaN	NaN	4.078194	12.006338	-47.129693
Volatility	340.0	NaN	NaN	NaN	1.525976	0.591798	0.733163
ROE	340.0	NaN	NaN	NaN	39.597059	96.547538	1.0
Cash Ratio	340.0	NaN	NaN	NaN	70.023529	90.421331	0.0
Net Cash Flow	340.0	NaN	NaN	NaN	55537620.588235	1946365312.175789	-11208000000.0
Net Income	340.0	NaN	NaN	NaN	1494384602.941176	3940150279.327937	-23528000000.0
Earnings Per Share	340.0	NaN	NaN	NaN	2.776662	6.587779	-61.2
Estimated Shares Outstanding	340.0	NaN	NaN	NaN	577028337.754029	845849595.417695	27672156.86
P/E Ratio	340.0	NaN	NaN	NaN	32.612563	44.348731	2.935451
P/B Ratio	340.0	NaN	NaN	NaN	-1.718249	13.966912	-76.119077

- Observing the data in multiple columns gives an early insight of outliers.

In [13]:
```
# copying the data to another variable to avoid any changes to original data
df = data.copy()
```

In [14]:
```
# convert all columns with dtype object into category
for col in df.columns[df.dtypes=='object']:
    df[col] = df[col].astype('category')
```

In [15]:
```
# dropping the ticker symbol column, as it does not provide any information
df.drop("Ticker Symbol", axis=1, inplace=True)
```

In [16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 340 entries, 0 to 339
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Security                      340 non-null    category
 1   GICS Sector                   340 non-null    category
 2   GICS Sub Industry             340 non-null    category
 3   Current Price                 340 non-null    float64
 4   Price Change                  340 non-null    float64
 5   Volatility                    340 non-null    float64
 6   ROE                           340 non-null    int64
 7   Cash Ratio                    340 non-null    int64
 8   Net Cash Flow                 340 non-null    int64
 9   Net Income                    340 non-null    int64
 10  Earnings Per Share            340 non-null    float64
 11  Estimated Shares Outstanding  340 non-null    float64
 12  P/E Ratio                     340 non-null    float64
 13  P/B Ratio                     340 non-null    float64
dtypes: category(3), float64(7), int64(4)
memory usage: 46.7 KB
```

- All the object data types have now been converted to category.

Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

Questions:

1. What does the distribution of stock prices look like?
2. The stocks of which economic sector have seen the maximum price increase on average?
3. How are the different variables correlated with each other?
4. Cash ratio provides a measure of a company's ability to cover its short-term obligations using only cash and cash equivalents. How does the average cash ratio vary across economic sectors?
5. P/E ratios can help determine the relative value of a company's shares as they signify the amount of money an investor is willing to invest in a single share of a company per dollar of its earnings. How does the P/E ratio vary, on average, across economic sectors?

In [17]:
```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
```

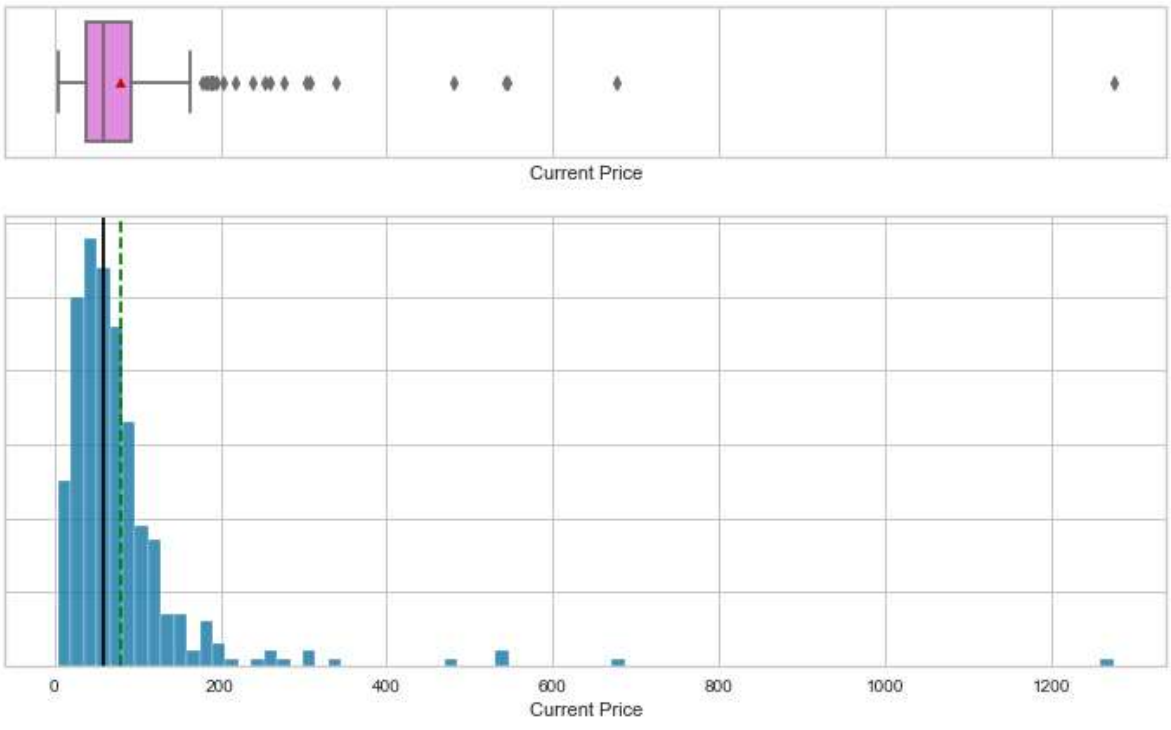
```
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [18]:

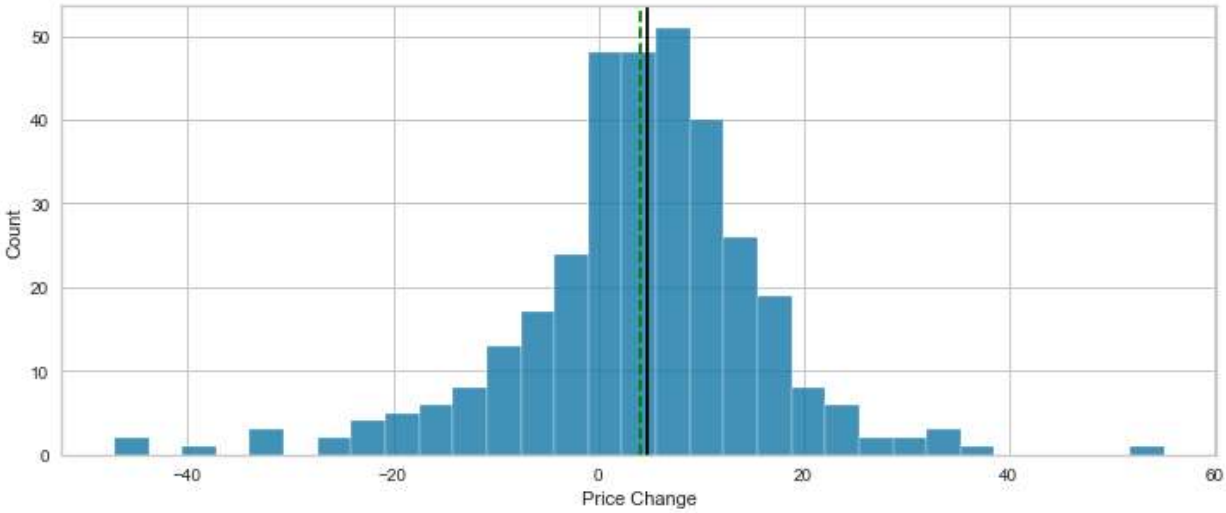
```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=12)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="viridis",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
```

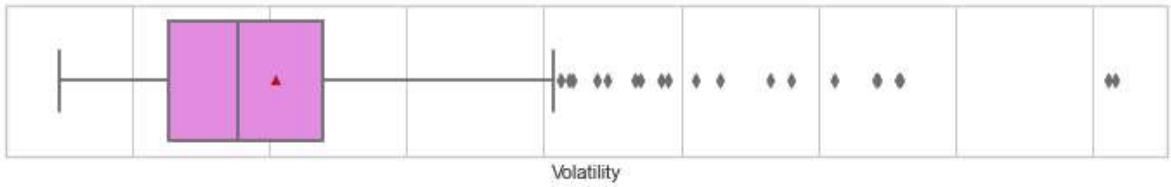
```
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

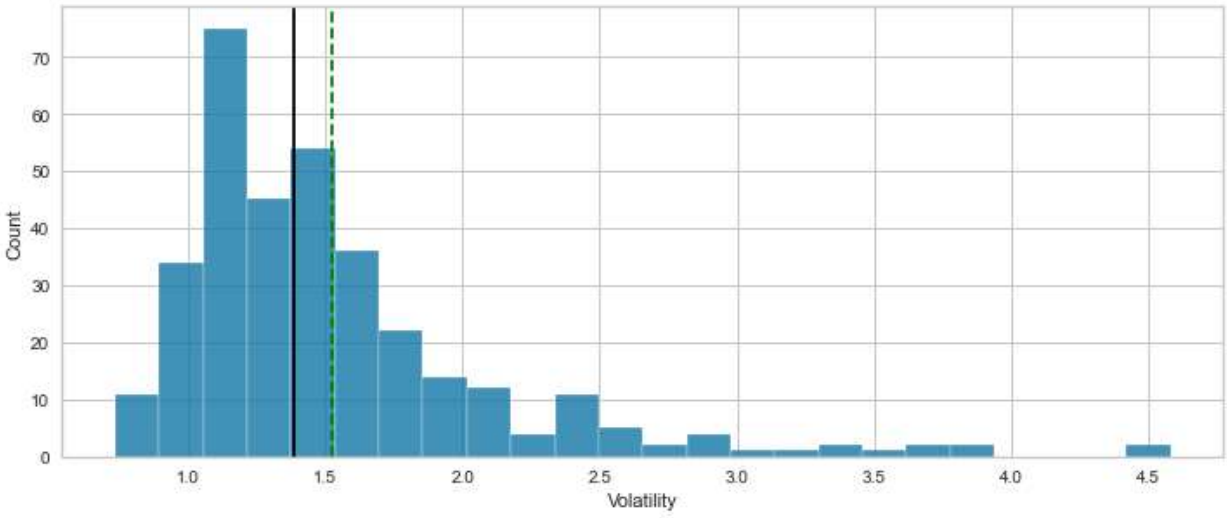
In [19]:
```
# selecting numerical columns
num_col = df.select_dtypes(include=np.number).columns.tolist()

for item in num_col:
    histogram_boxplot(df, item)
```



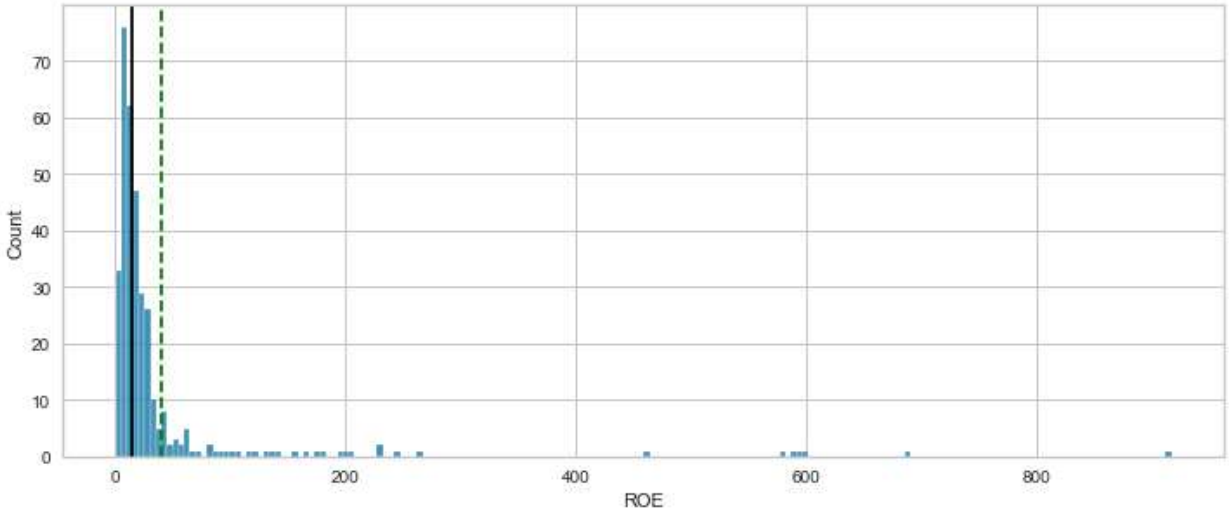

Price Change


Count
50
40
30
20
10
0
−40
−20
0
20
40
60
Price Change

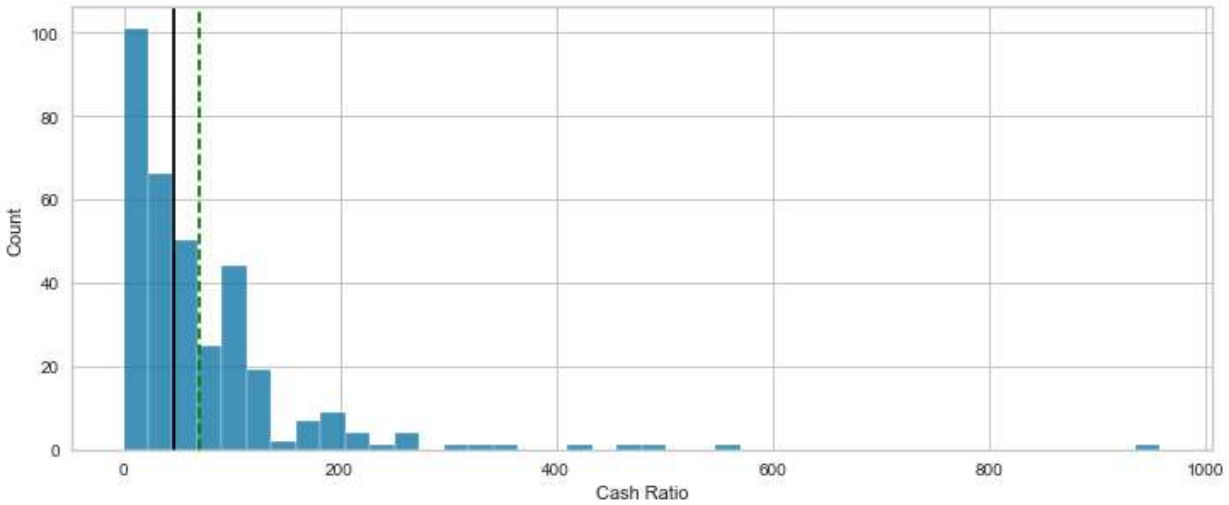

Volatility

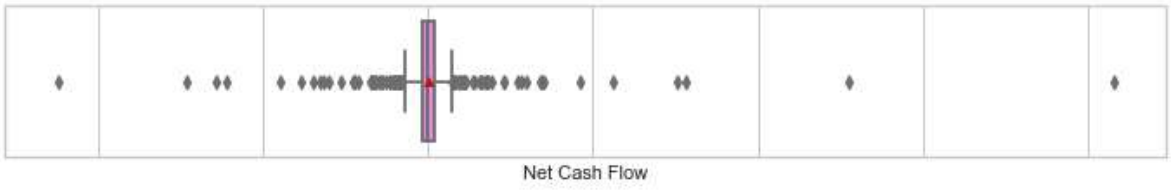

Count
70
60
50
40
30
20
10
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
Volatility

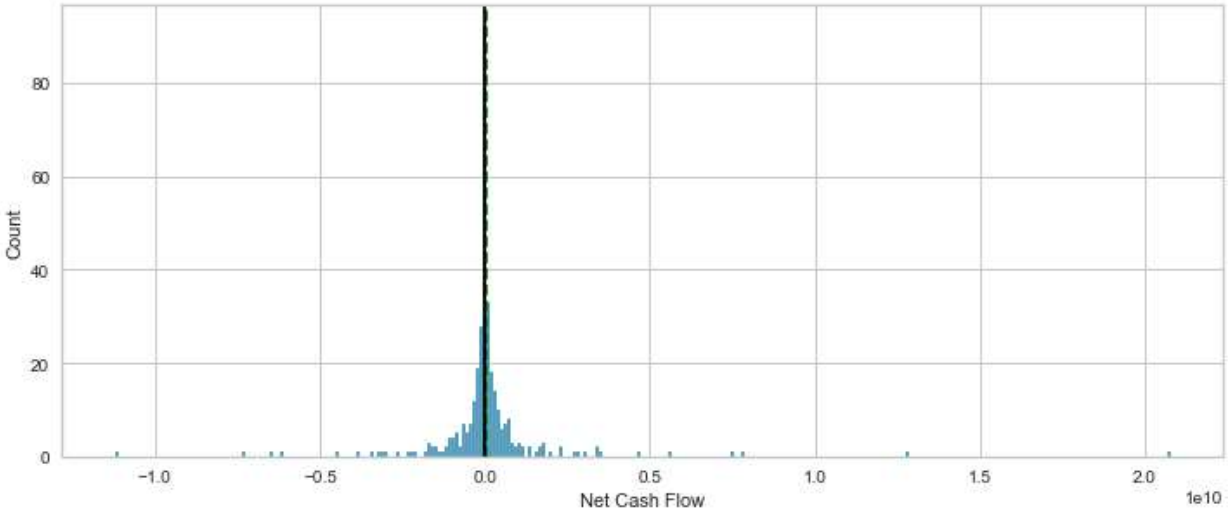

ROE

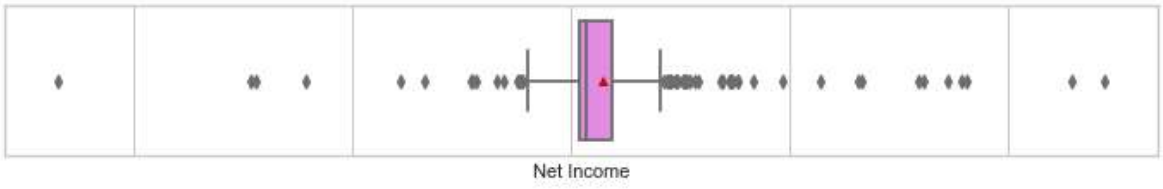

Count
70
60
50
40
30
20
10
0
0
200
400
600
800
ROE

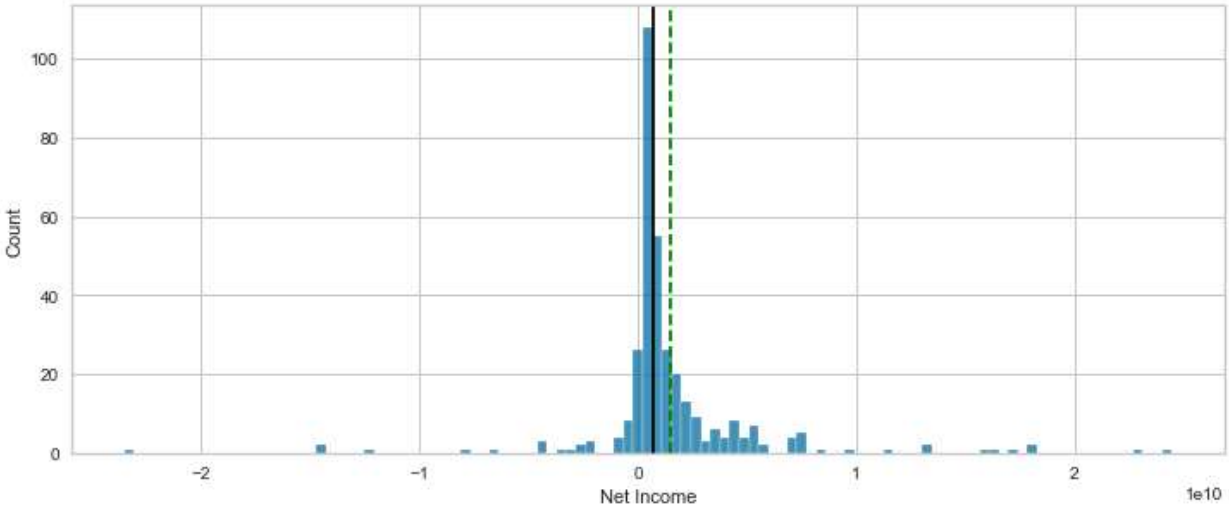

Cash Ratio

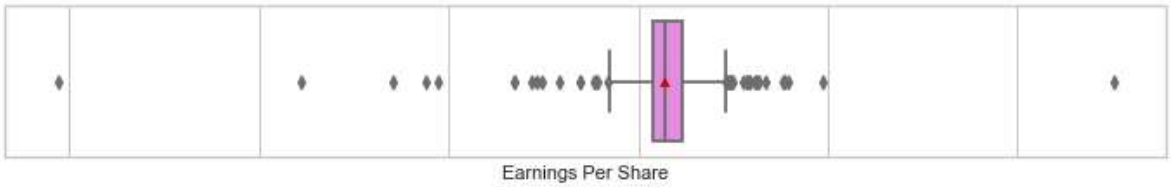

Count
100
80
60
40
20
0
0
200
400
600
800
1000
Cash Ratio

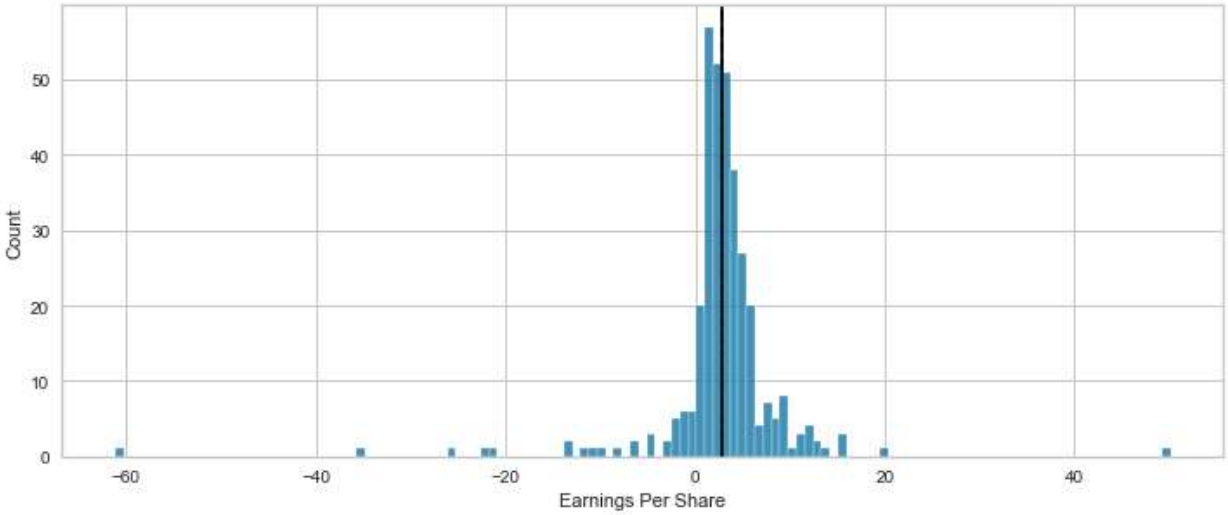

Net Cash Flow

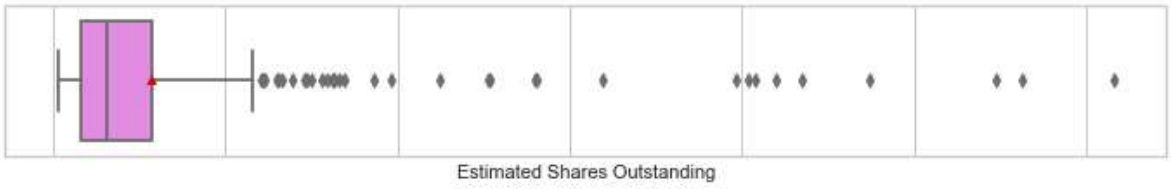

Count
0
20
40
60
80
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Net Cash Flow
1e10

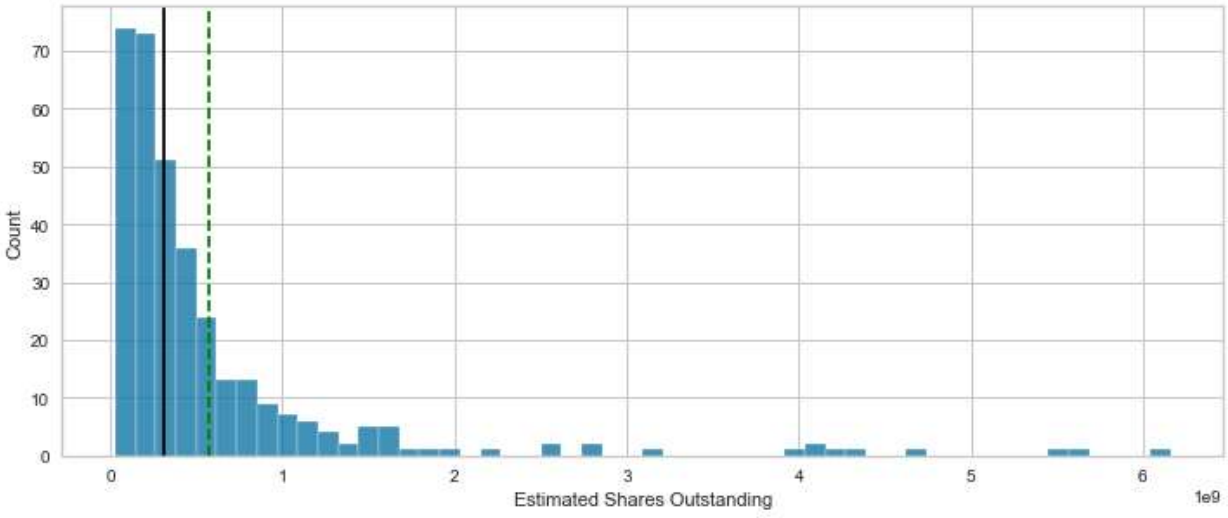

Net Income

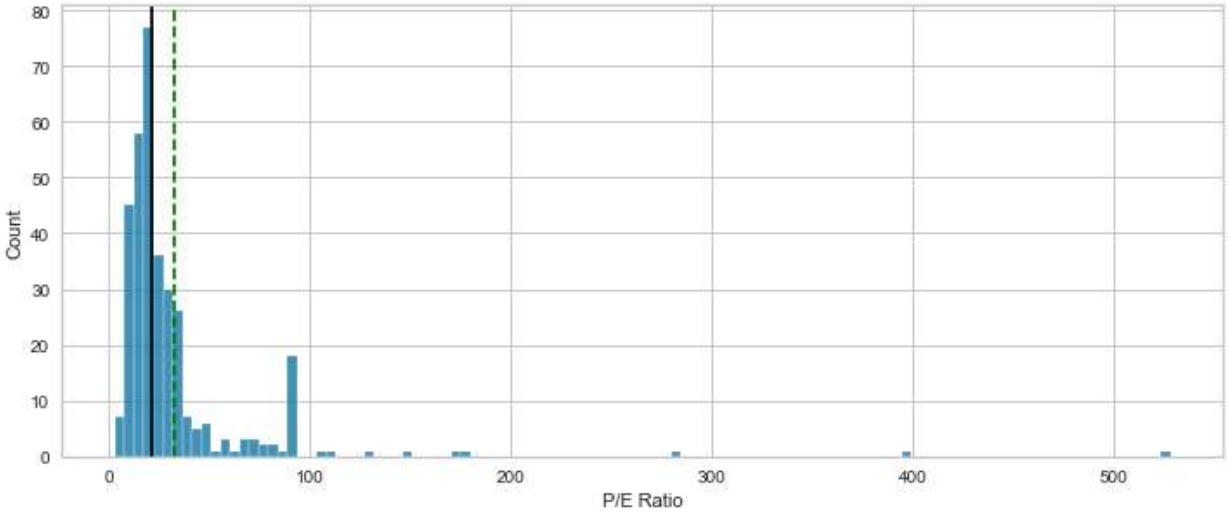

Count
0
20
40
60
80
100
−2
−1
0
1
2
Net Income
1e10

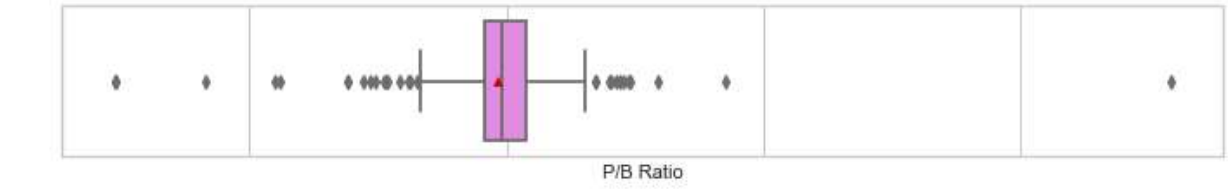

Earnings Per Share

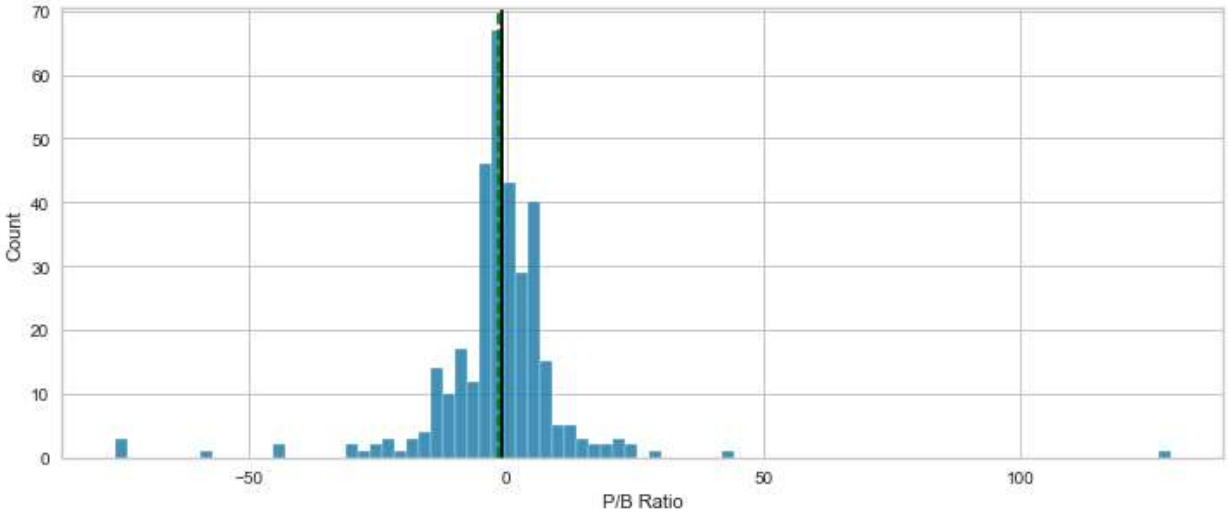

Count
50
40
30
20
10
0
−60
−40
−20
0
20
40
Earnings Per Share


Estimated Shares Outstanding


Count
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6
Estimated Shares Outstanding
1e9



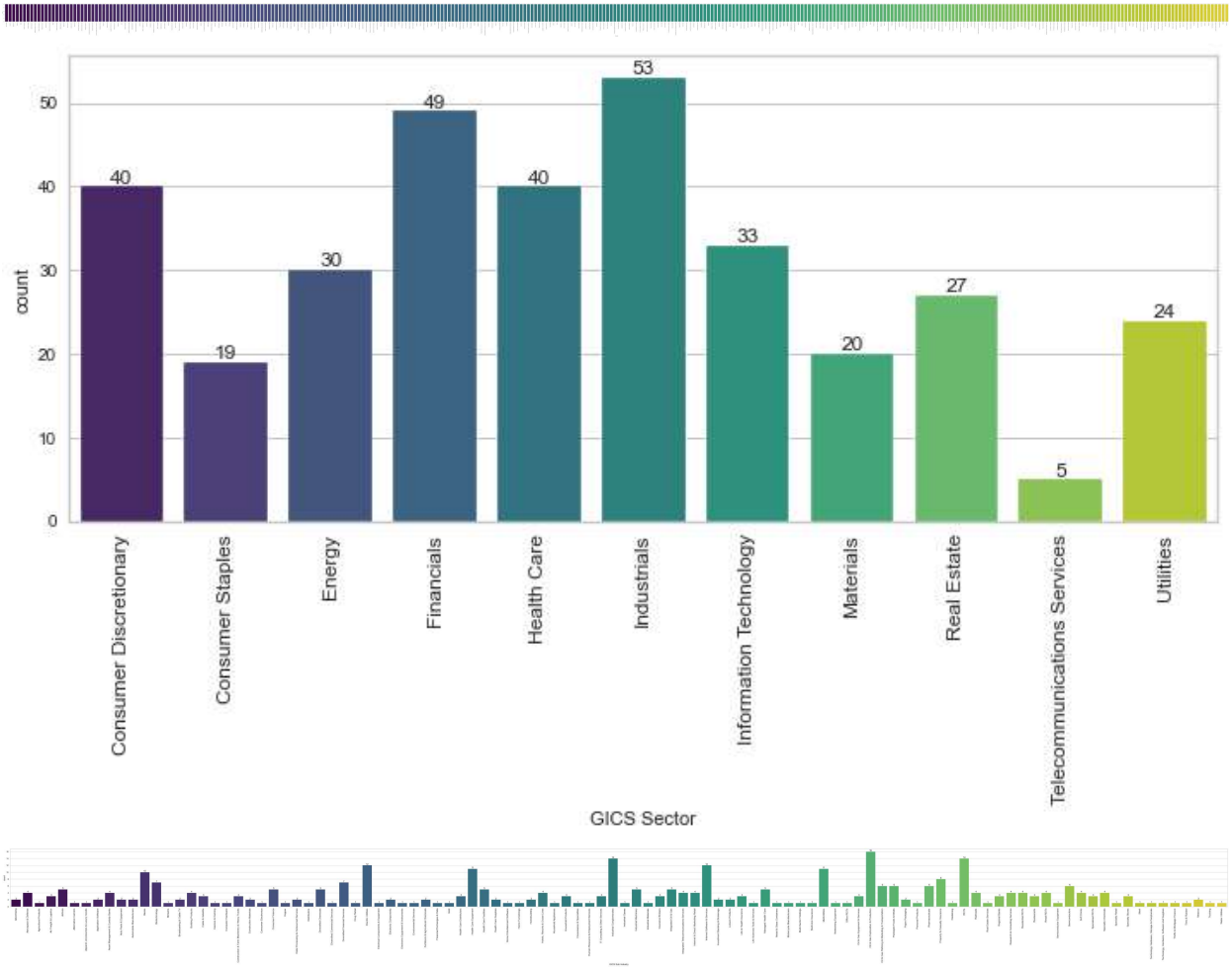




- Current Price
  - The distribution is right skewed with quite a few outliers.
- Price Change
  - The distribution appears to be normal with outliers in either direction.
- Volatility
  - The distibution is heavily skewed on the right.
- ROE
  - The distribution is heavily skewed on the right.
- Cash Ratio

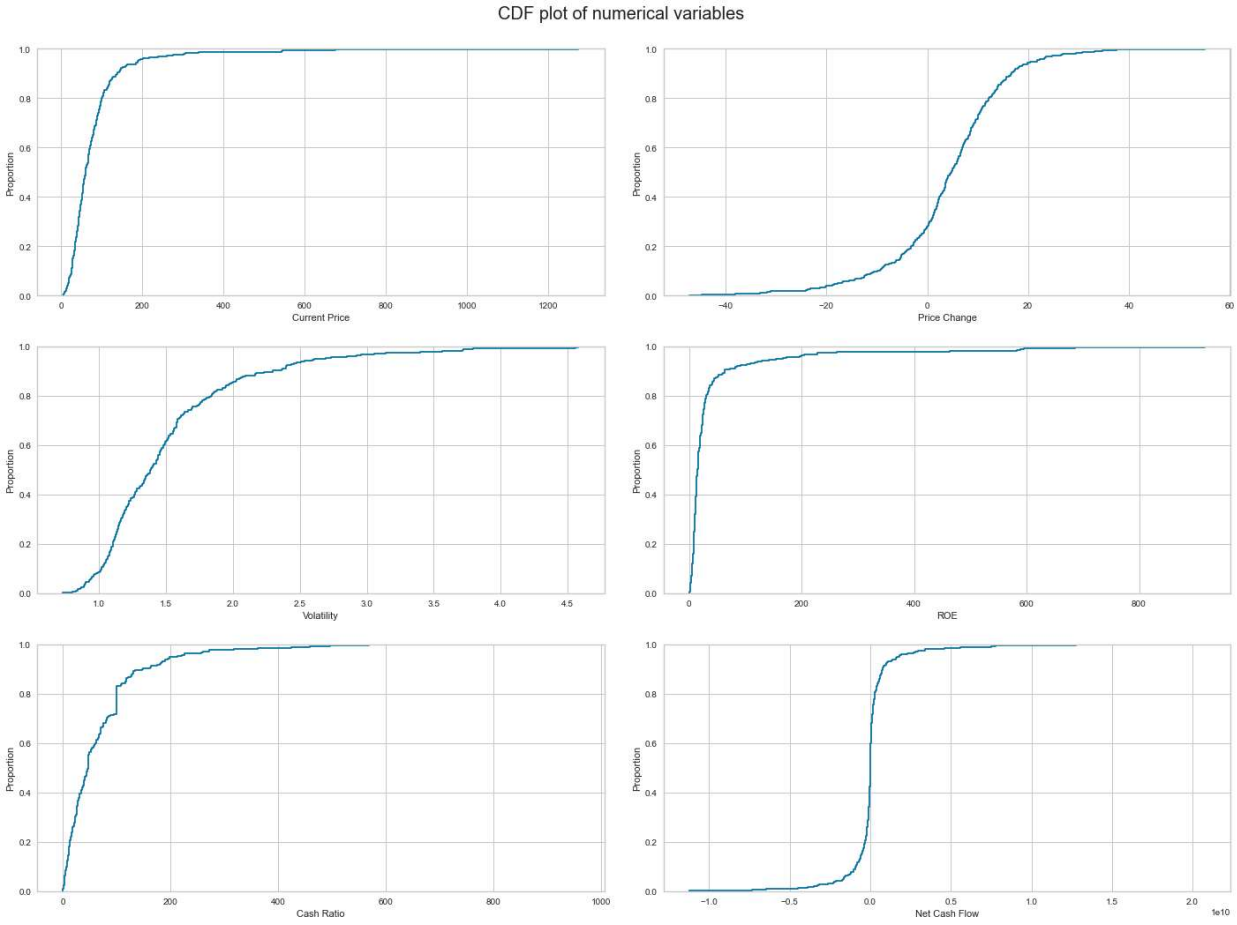
    - The distribution does not appear to be normal and there are quite a few outliers on the right.
- Net Cash Flow
    - The distribution appears to be normal with outliers in either direction
- Net Income
    - The distribution is skewed on the right with outliers in either direction
- Earnings Per Share
    - The distribution appears to be normal with outliers in either direction
- Estimated Shares Outstanding
    - The distribution is heavily skewed with outliers on the right.
- P/E Ratio
    - The distribution is heavily skewed with outliers on the right.
- P/B Ratio
    - The distribution appears to be normal with outliers in either direction.

In [20]:
```
# selecting categorical columns
cat_col = df.select_dtypes(include='category').columns.tolist()

for item in cat_col:
    labeled_barplot(df, item)
```



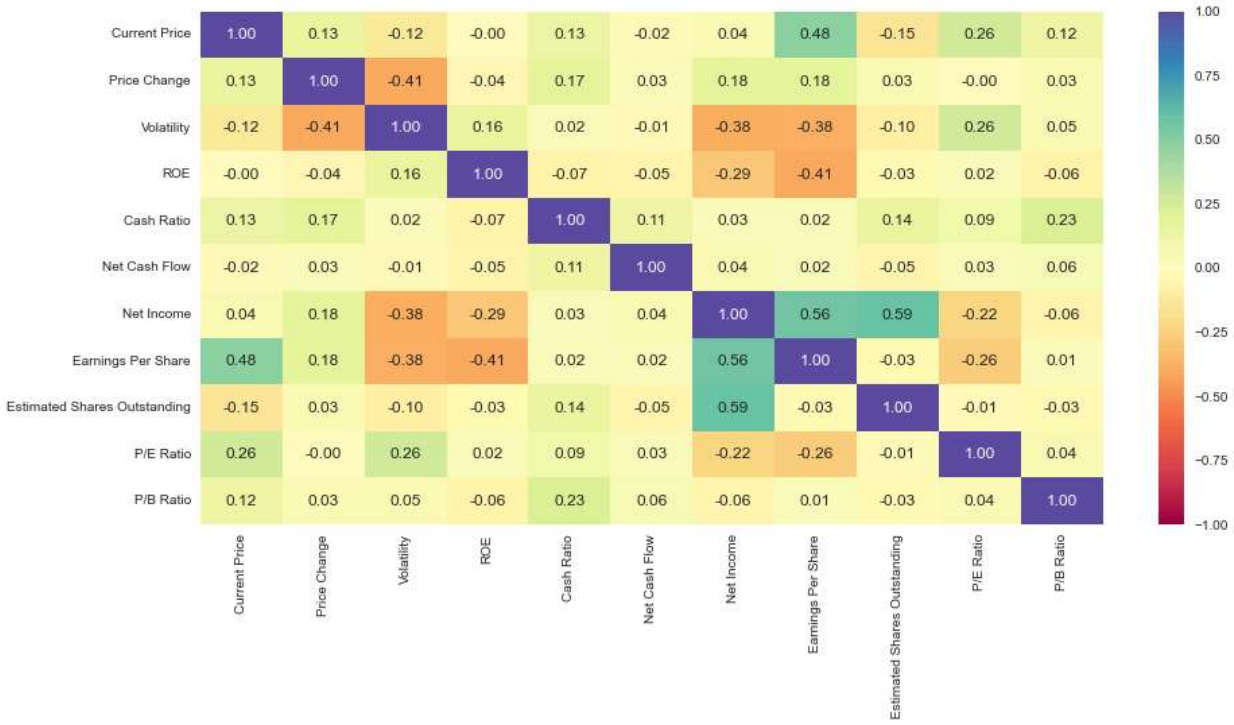
In [31]:
```
fig, axes = plt.subplots(3, 2, figsize=(20, 15))
fig.suptitle("CDF plot of numerical variables", fontsize=20)
```

```
counter = 0
for ii in range(3):
    sns.ecdfplot(ax=axes[ii][0], x=df[num_col[counter]])
    counter = counter + 1
    if counter != 10:
        sns.ecdfplot(ax=axes[ii][1], x=df[num_col[counter]])
        counter = counter + 1
    else:
        pass

fig.tight_layout(pad=2.0)
```



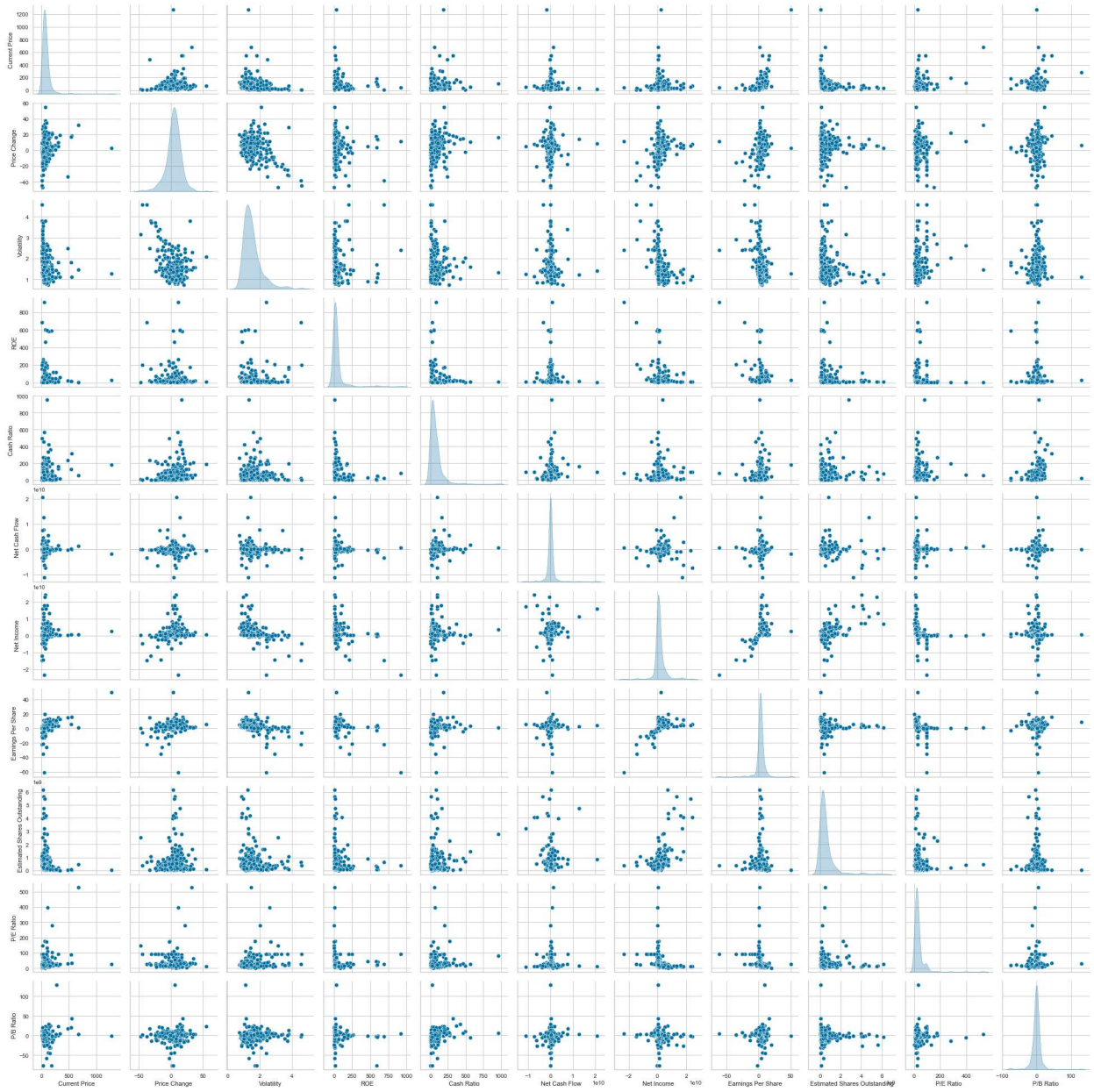
- A very high majority of shares have their current price below 800
- Major chunk of price fluctuation is between -40 to +40
- Volatility appears to be evenly distributed between 0.5 and 4
- A very high majority of shares has an ROE<600
- Cash Ratio below 600 is for a significantly high shares.
- Net cashflow varies between -1 and +1 for the most part.

In [24]:
```
plt.figure(figsize=(15, 7))
sns.heatmap(
    df.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



- We do not observe a very strong correlation (neither positive nor negative) between the numerical variables
- A slightly moderate correlation (both positive and negative) can be observed between few variables.

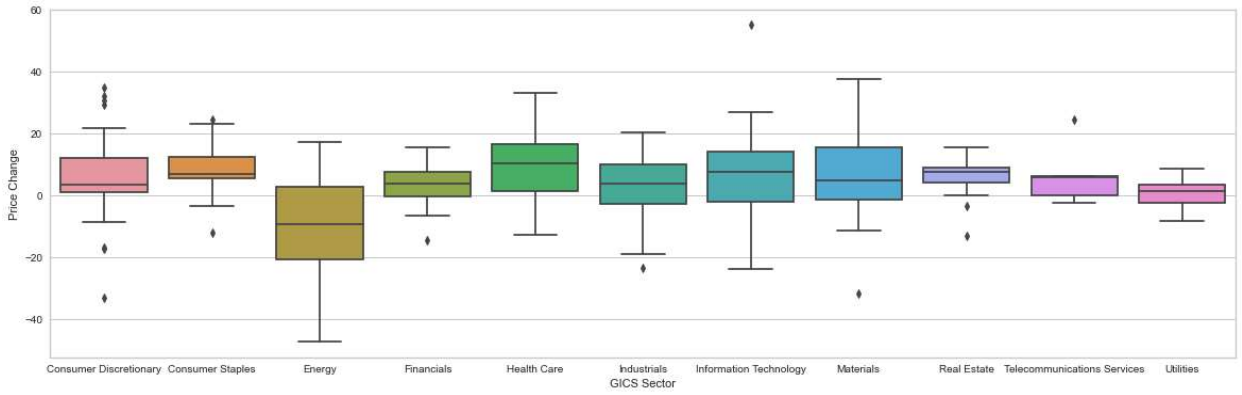
In [25]:
```
sns.pairplot(data=df, diag_kind="kde")
plt.show()
```



In [33]:
```
df.groupby('GICS Sector')['Price Change'].mean().sort_values()
```

Out[33]:
```
GICS Sector
Energy                         -10.228289
Utilities                        0.803657
Industrials                      2.833127
Financials                       3.865406
Materials                        5.589738
Consumer Discretionary           5.846093
Real Estate                      6.205548
Telecommunications Services      6.956980
Information Technology           7.217476
Consumer Staples                 8.684750
Health Care                      9.585652
Name: Price Change, dtype: float64
```

In [37]:
```
plt.figure(figsize=(20,6))
sns.boxplot(data = df, y = "Price Change", x = "GICS Sector");
```

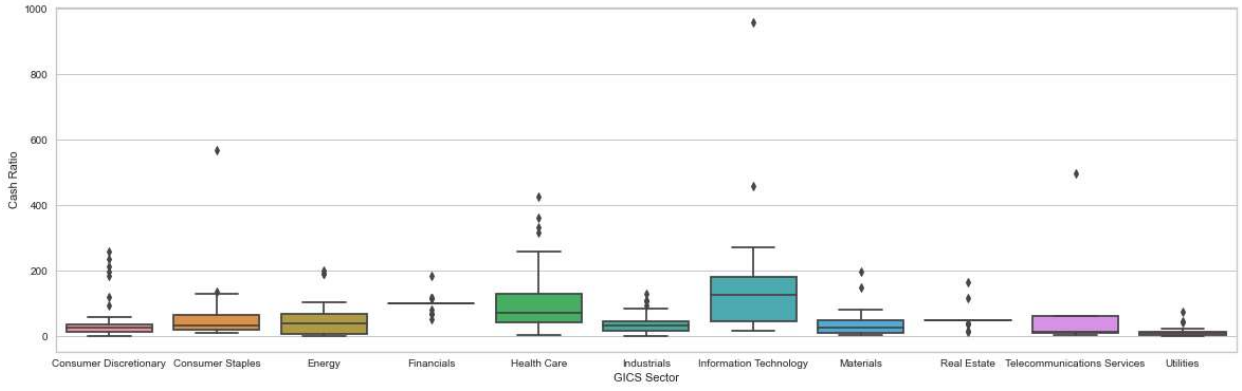


- Energy has seen the highest price change (negative).
- Health Care has seen the biggest uptick overall.

In [34]:
```
df.groupby('GICS Sector')['Cash Ratio'].mean().sort_values()
```

Out[34]:
```
GICS Sector
Utilities                       13.625000
Industrials                     36.188679
Materials                       41.700000
Consumer Discretionary          49.575000
Real Estate                     50.111111
Energy                          51.133333
Consumer Staples                70.947368
Financials                      98.591837
Health Care                    103.775000
Telecommunications Services    117.000000
Information Technology         149.818182
Name: Cash Ratio, dtype: float64
```

In [38]:
```
plt.figure(figsize=(20,6))
sns.boxplot(data = df, y = "Cash Ratio", x = "GICS Sector");
```

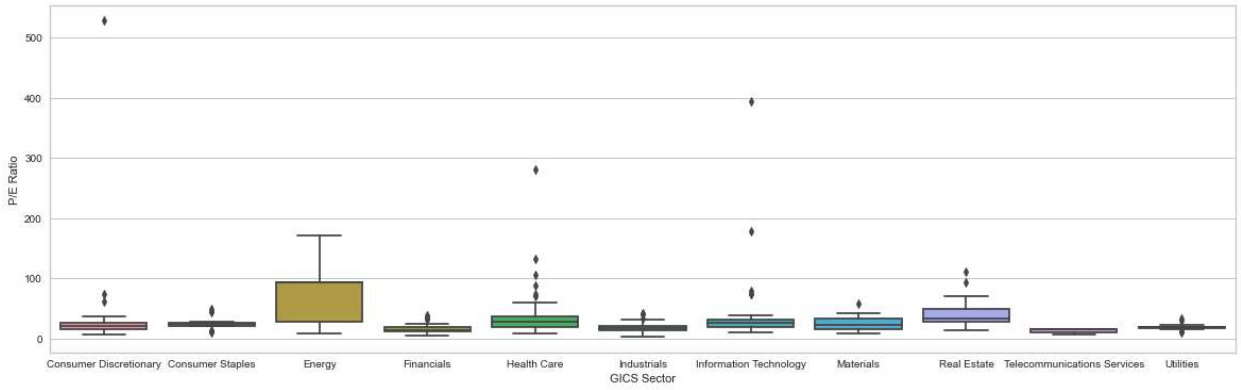


- Utilities sector has the lowest cash ratio among all sectors.
- Information Technology has the highest cash ratio among all sectors.

In [39]:
```
df.groupby('GICS Sector')['P/E Ratio'].mean().sort_values()
```

Out[39]:
```
GICS Sector
Telecommunications Services    12.222578
Financials                     16.023151
Industrials                    18.259380
Utilities                      18.719412
Materials                      24.585352
Consumer Staples               25.521195
Consumer Discretionary         35.211613
Health Care                    41.135272
Real Estate                    43.065585
Information Technology         43.782546
Energy                         72.897709
Name: P/E Ratio, dtype: float64
```

In [44]:
```
plt.figure(figsize=(20,6))
sns.boxplot(data = df, y = "P/E Ratio", x = "GICS Sector");
```



- Going by the mean, Telecommunications Services has one of the lowest P/E ratios in contrast to Energy, that has highest P/E ratio.

Data Preprocessing

- Duplicate value check
- Missing value treatment
- Outlier check
- Feature engineering (if needed)
- Any other preprocessing steps (if needed)

In [45]:
```
df.head()
```

Out[45]:

	Security	GICS Sector	GICS Sub Industry	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash Flow
0	American Airlines Group	Industrials	Airlines	42.349998	9.999995	1.687151	135	51	-604000000
1	AbbVie	Health Care	Pharmaceuticals	59.240002	8.339433	2.197887	130	77	51000000
2	Abbott Laboratories	Health Care	Health Care Equipment	44.910000	11.301121	1.273646	21	67	938000000
3	Adobe Systems Inc	Information Technology	Application Software	93.940002	13.977195	1.357679	9	180	-240840000
4	Analog Devices, Inc.	Information Technology	Semiconductors	55.320000	-1.827858	1.701169	14	272	315120000

Data Scaling

In [46]:
```
# variables used for clustering
num_col
```

Out[46]:
```
['Current Price',
 'Price Change',
 'Volatility',
 'ROE',
 'Cash Ratio',
 'Net Cash Flow',
 'Net Income',
 'Earnings Per Share',
 'Estimated Shares Outstanding',
 'P/E Ratio',
 'P/B Ratio']
```

In [47]:
```
# scaling the dataset before clustering
scaler = StandardScaler()
subset = df[num_col].copy()
subset_scaled = scaler.fit_transform(subset)
```

In [48]:
```
# creating a dataframe of the scaled columns
subset_scaled_df = pd.DataFrame(subset_scaled, columns=subset.columns)
```

In [49]:
```
subset_scaled_df.head()
```

Out[49]:

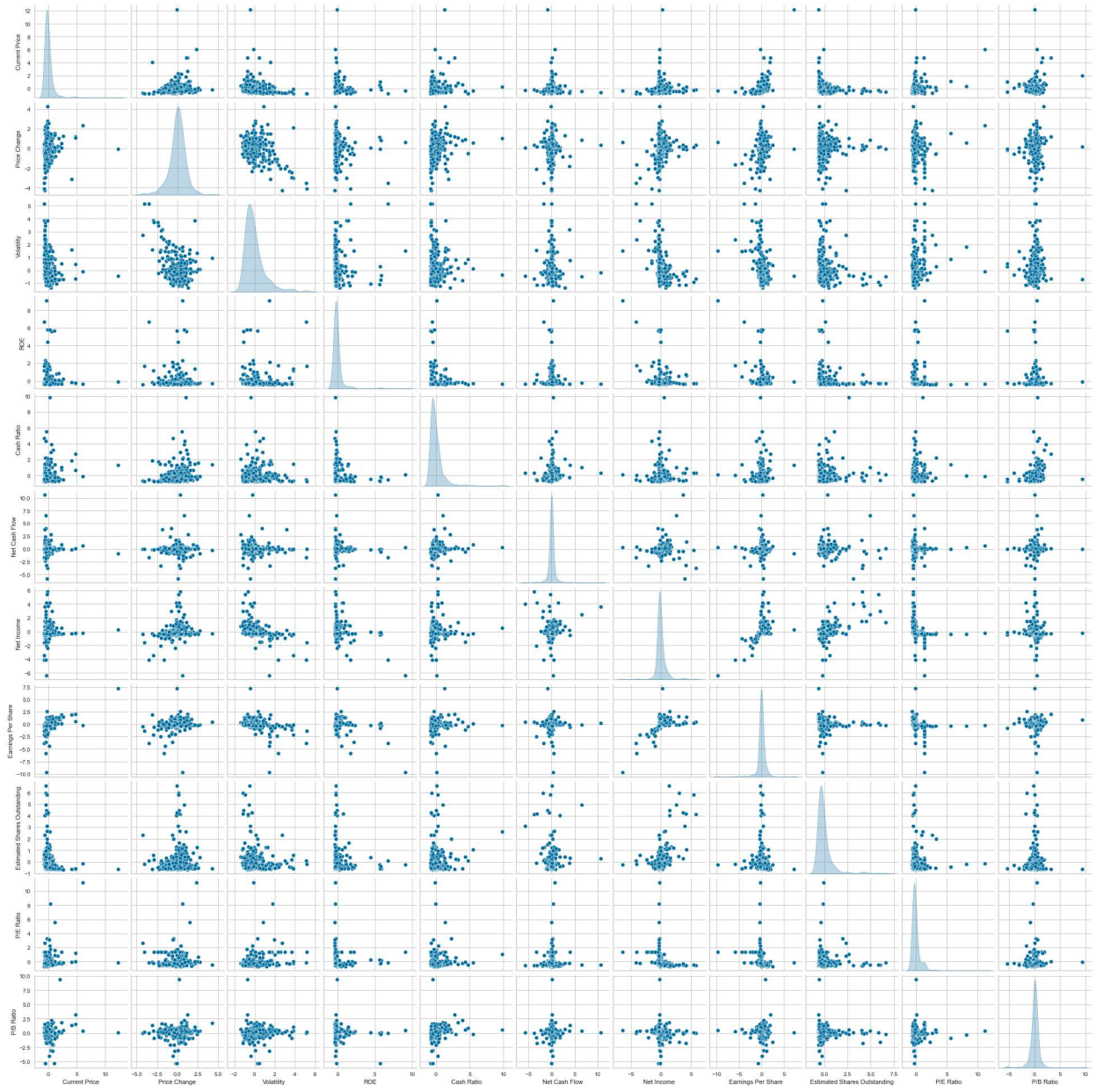
	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash Flow	Net Income	Earnings Per Share	Estimated Shares Outstanding
0	-0.393341	0.493950	0.272749	0.989601	-0.210698	-0.339355	1.554415	1.309399	0.107863
1	-0.220837	0.355439	1.137045	0.937737	0.077269	-0.002335	0.927628	0.056755	1.250274
2	-0.367195	0.602479	-0.427007	-0.192905	-0.033488	0.454058	0.744371	0.024831	1.098021
3	0.133567	0.825696	-0.284802	-0.317379	1.218059	-0.152497	-0.219816	-0.230563	-0.091622
4	-0.260874	-0.492636	0.296470	-0.265515	2.237018	0.133564	-0.202703	-0.374982	1.978399

- The data has been scaled. All numerical attributes have been appropriately scaled using standard scaler.

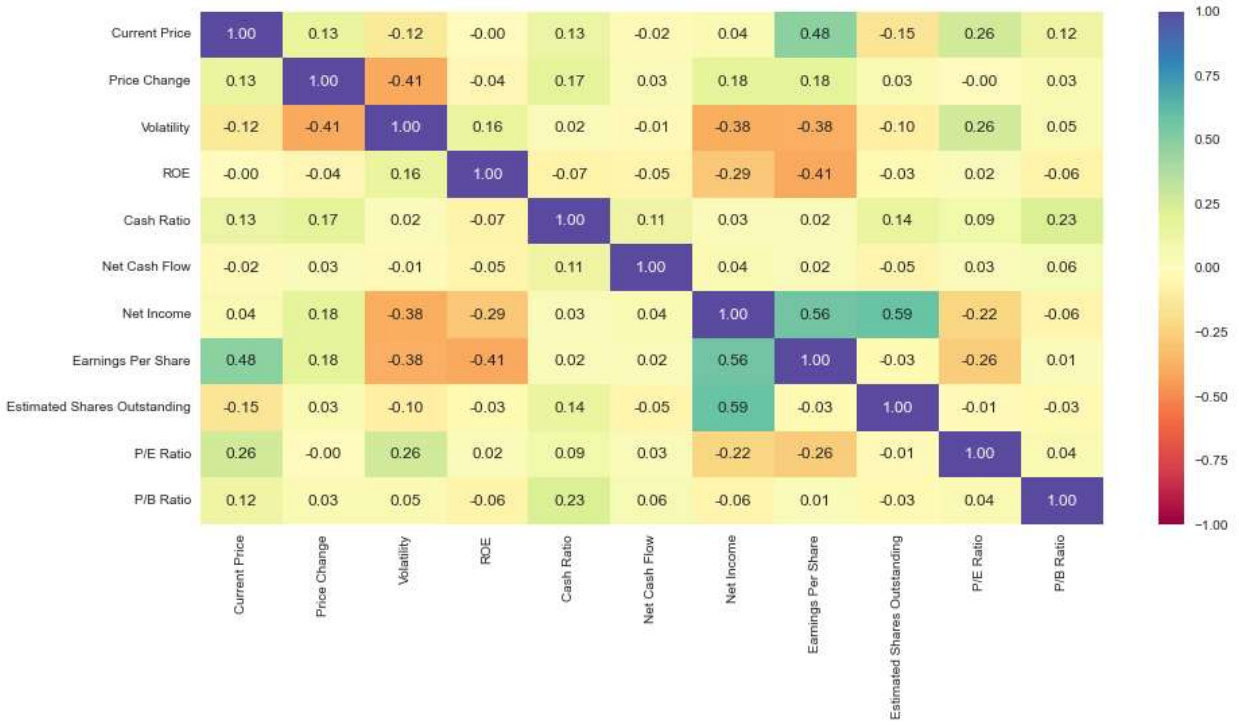
EDA

- It is a good idea to explore the data once again after manipulating it.

In [50]:
```
# Pair-plot analysis
sns.pairplot(subset_scaled_df ,diag_kind="kde");
```



In [51]:
```
plt.figure(figsize=(15, 7))
sns.heatmap(
    subset_scaled_df.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



- The correlation has stayed the same and we do no observe any significant change after scaling.

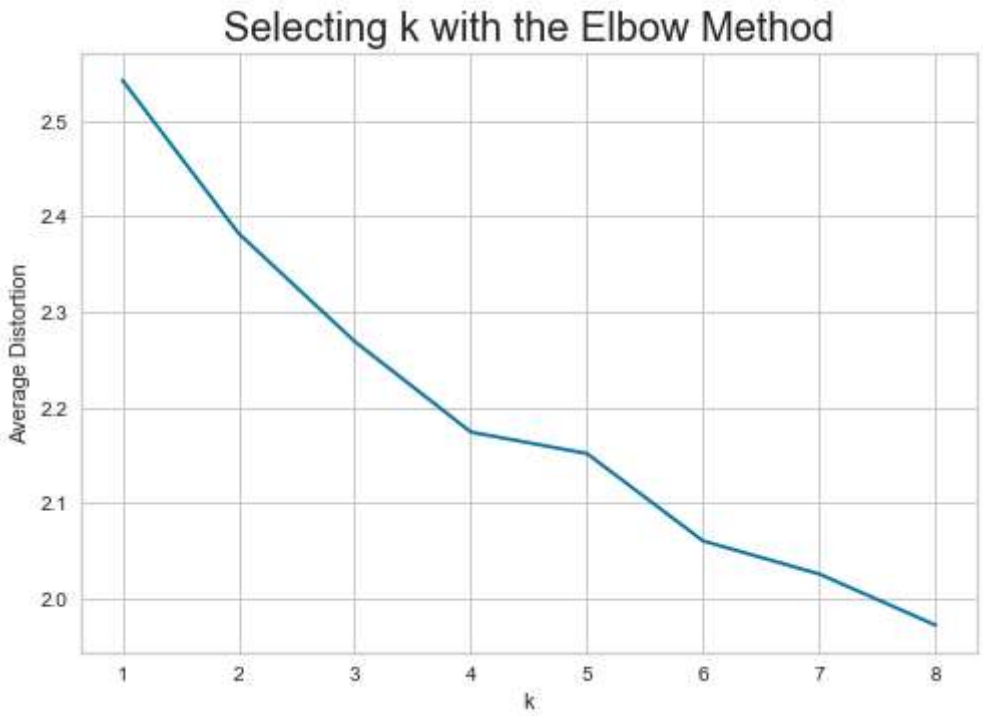
K-means Clustering

In [56]:

```
clusters = range(1, 9)
meanDistortions = []

for k in clusters:
    model = KMeans(n_clusters=k)
    model.fit(subset_scaled_df)
    prediction = model.predict(subset_scaled_df)
    distortion = (
        sum(
            np.min(cdist(subset_scaled_df, model.cluster_centers_, "euclidean"), axis=
        )
        / subset_scaled_df.shape[0]
    )

    meanDistortions.append(distortion)

    print("Number of Clusters:", k, "\tAverage Distortion:", distortion)

plt.plot(clusters, meanDistortions, "bx-")
plt.xlabel("k")
plt.ylabel("Average Distortion")
plt.title("Selecting k with the Elbow Method", fontsize=20)
```

```
Number of Clusters: 1   Average Distortion: 2.5425069919221697
Number of Clusters: 2   Average Distortion: 2.382318498894466
Number of Clusters: 3   Average Distortion: 2.2692367155390745
Number of Clusters: 4   Average Distortion: 2.1745559827866363
Number of Clusters: 5   Average Distortion: 2.152029187283101
Number of Clusters: 6   Average Distortion: 2.0606611536221404
Number of Clusters: 7   Average Distortion: 2.0261348814329385
Number of Clusters: 8   Average Distortion: 1.972721196089717
```

Out[56]: Text(0.5, 1.0, 'Selecting k with the Elbow Method')



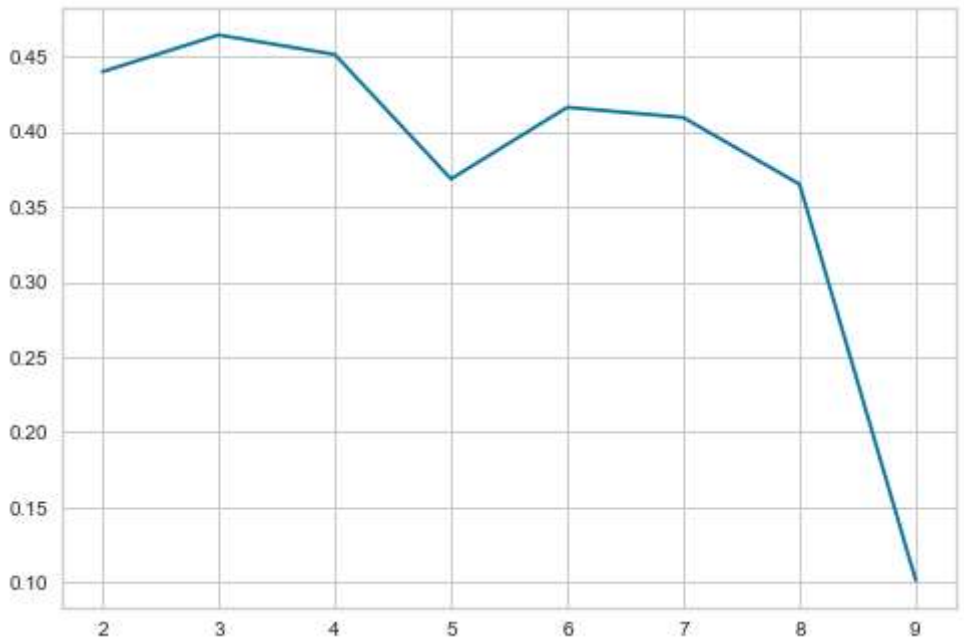
- The appropriate K appears to be 4 or 5

Let's check the silhouette scores

In [57]:
```
sil_score = []
cluster_list = list(range(2, 10))
for n_clusters in cluster_list:
    clusterer = KMeans(n_clusters=n_clusters)
    preds = clusterer.fit_predict((subset_scaled_df))
    # centers = clusterer.cluster_centers_
    score = silhouette_score(subset_scaled_df, preds)
    sil_score.append(score)
    print("For n_clusters = {}, silhouette score is {}".format(n_clusters, score))

plt.plot(cluster_list, sil_score)
```

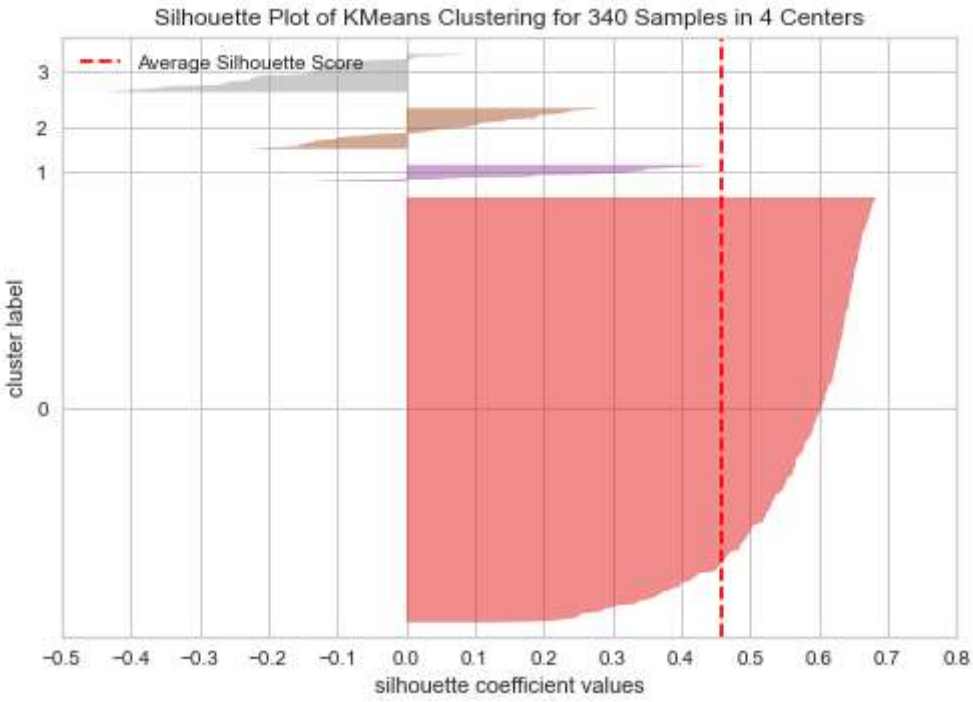
```
For n_clusters = 2, silhouette score is 0.43969639509980457
For n_clusters = 3, silhouette score is 0.4641340132225366
For n_clusters = 4, silhouette score is 0.451150358009578
For n_clusters = 5, silhouette score is 0.36841420223721716
For n_clusters = 6, silhouette score is 0.4160418984271558
For n_clusters = 7, silhouette score is 0.40923683991799303
For n_clusters = 8, silhouette score is 0.3649128072196648
For n_clusters = 9, silhouette score is 0.10171429153168896
```

Out[57]: [<matplotlib.lines.Line2D at 0x1e3cabddb50>]

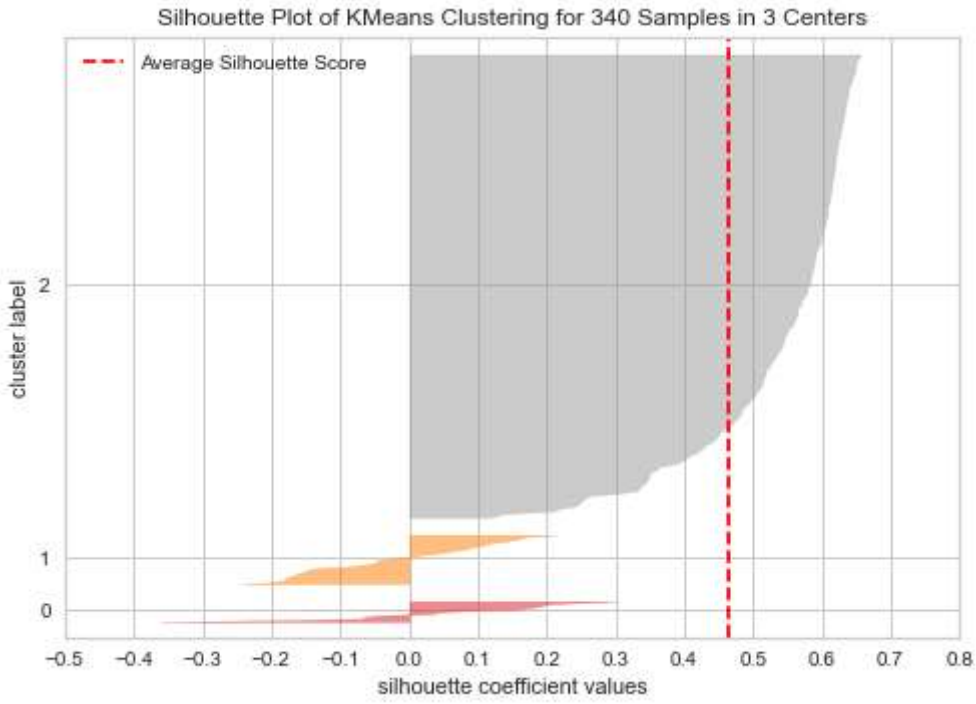


From the silhouette scores, it seems that 3 and 4 are good value of k.

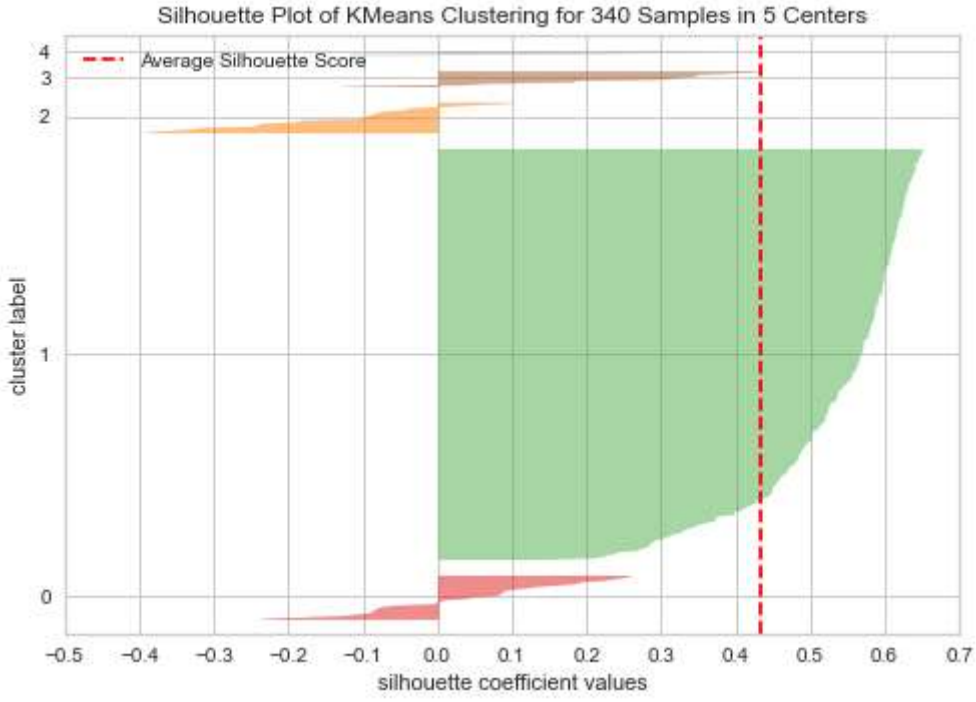
In [59]:
```
# finding optimal no. of clusters with silhouette coefficients
visualizer = SilhouetteVisualizer(KMeans(4, random_state=1))
visualizer.fit(subset_scaled_df)
visualizer.show();
```



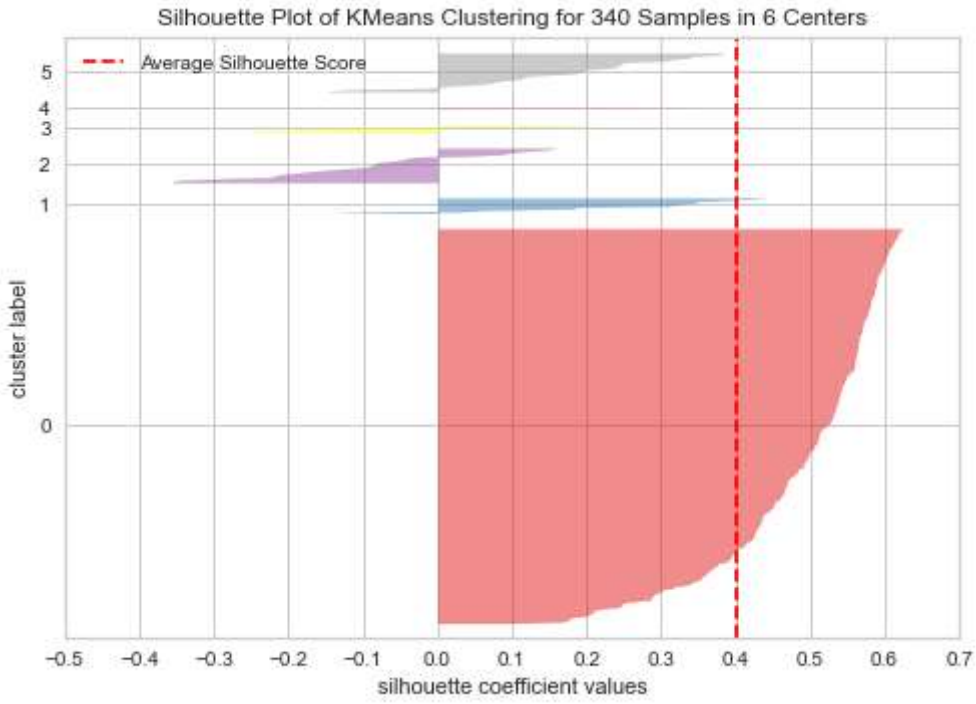
In [61]:
```
# finding optimal no. of clusters with silhouette coefficients
visualizer = SilhouetteVisualizer(KMeans(3, random_state=1))
visualizer.fit(subset_scaled_df)
visualizer.show();
```



In [62]:
```
# finding optimal no. of clusters with silhouette coefficients
visualizer = SilhouetteVisualizer(KMeans(5, random_state=1))
visualizer.fit(subset_scaled_df)
visualizer.show();
```



In [66]:
```
# finding optimal no. of clusters with silhouette coefficients
visualizer = SilhouetteVisualizer(KMeans(6, random_state=1))
visualizer.fit(subset_scaled_df)
visualizer.show();
```



Let's take 4 as the appropriate no. of clusters as the silhouette score is high enough

In [64]:
```
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(subset_scaled_df)
```

Out[64]:
```
KMeans(n_clusters=4, random_state=0)
```

In [65]:
```
# adding kmeans cluster labels to the original dataframe
df["K_means_segments"] = kmeans.labels_
```

Cluster Profiling

In [67]:
```
cluster_profile = df.groupby("K_means_segments").mean()
```

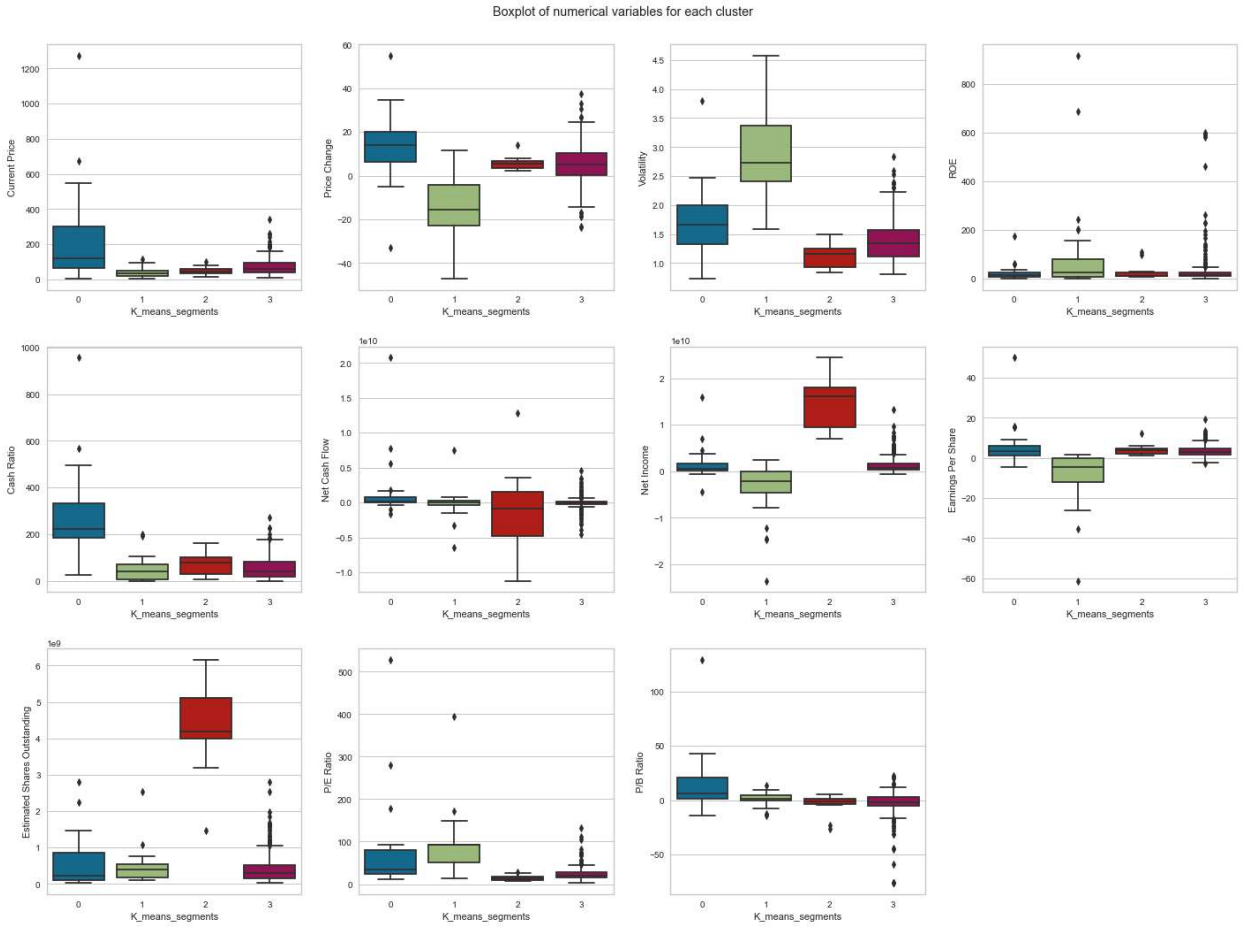
In [76]:
```
cluster_profile["count_in_each_segment"] = (
    df.groupby("K_means_segments")["Current Price"].count().values
)
```

In [77]:
```
# let's display cluster profiles
cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

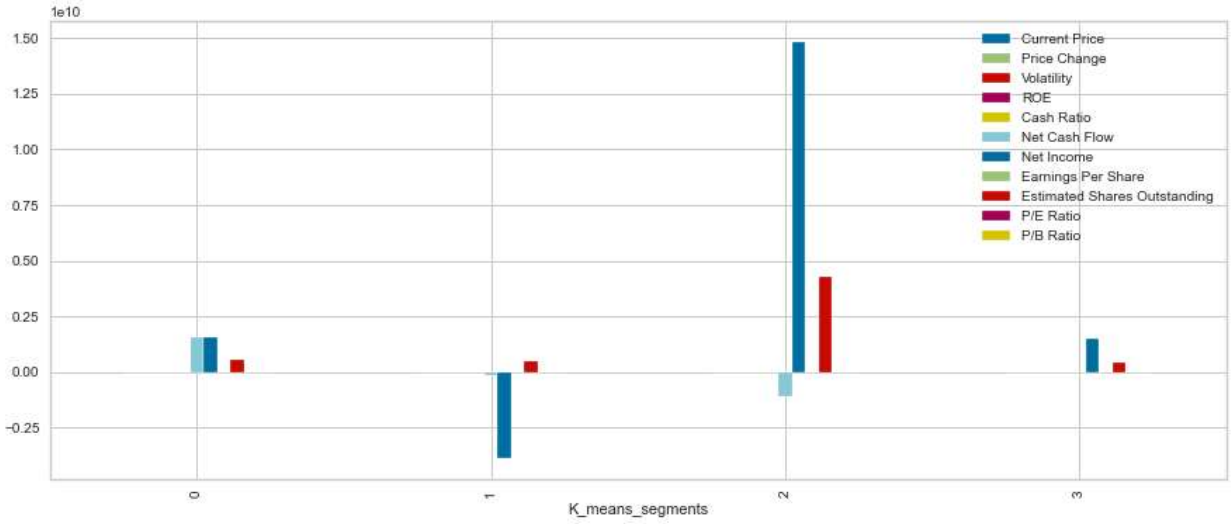
Out[77]:

K_means_segments	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash Flow	
0	234.170932	13.400685	1.729989	25.600000	277.640000	1554926560.000000	15
1	38.099260	-15.370329	2.910500	107.074074	50.037037	-159428481.481481	-38
2	50.517273	5.747586	1.130399	31.090909	75.909091	-1072272727.272727	148
3	72.399112	5.066225	1.388319	34.620939	53.000000	-14046223.826715	14

In [78]:
```
plt.figure(figsize=(20, 15))
plt.suptitle("Boxplot of numerical variables for each cluster")

for i, variable in enumerate(num_col):
    plt.subplot(3, 4, i + 1)
    sns.boxplot(data=df, x="K_means_segments", y=variable)

plt.tight_layout(pad=2.0)
```



In [79]:
```
df.groupby("K_means_segments").mean().plot.bar(figsize=(15, 6));
```



In [82]:
```
# Comparing cluster vs. GICS_Sector

pd.crosstab(df['GICS Sector'], df.K_means_segments).style.highlight_max(color = 'light
```

Out[82]:

GICS Sector	0	1	2	3
Consumer Discretionary	6	0	1	33
Consumer Staples	1	0	1	17
Energy	1	22	1	6
Financials	1	0	3	45
Health Care	9	0	2	29
Industrials	0	1	0	52
Information Technology	5	3	1	24
Materials	0	1	0	19
Real Estate	1	0	0	26
Telecommunications Services	1	0	2	2
Utilities	0	0	0	24

Insights

- **Cluster 0**:
  - Current price and Price change are very high
  - Volatility is low
  - ROE is relatively low
  - Cash Ratio and Net Cashflow are very high
  - Net Income is not very high
  - EPS is the highest amongst all clusters
  - ESO and P/E Ratio are moderately high.

- P/B ratio is highest among clusters.
- A total of 25 securities fall into this cluster.

- **Cluster 1**:
  - Current price and Price change are the lowest
  - Volatility is very high
  - ROE is very high
  - Cash Ratio and Net Cashflow are lowest
  - Net Income is very low
  - EPS is the lowest amongst all clusters
  - ESO is moderately high and P/E Ratio is the highest.
  - P/B ratio is moderate among clusters.
  - A total of 27 securities fall into this cluster.

- **Cluster 2**:
  - Current price and Price change are moderate
  - Volatility is lowest
  - ROE is moderate
  - Cash Ratio and Net Cashflow are moderate
  - Net Income is very high
  - EPS is the moderate
  - ESO is higest and P/E Ratio is lowest among clusters.
  - P/B ratio is lowest among clusters.
  - A total of 11 securities fall into this cluster.

- **Cluster 3**:
  - Current price and Price change are moderate
  - Volatility is moderate
  - ROE is moderate
  - Cash Ratio and Net Cashflow are relatively low
  - Net Income is not very high
  - EPS is the moderate upon comparison with all clusters
  - ESO and P/E Ratio are moderate.
  - P/B ratio is very low among clusters.
  - A total of 277 securities fall into this cluster.

Hierarchical Clustering

Computing Cophenetic Correlation

In [84]:
```
# list of distance metrics
distance_metrics = ["euclidean", "chebyshev", "mahalanobis", "cityblock"]

# list of linkage methods
linkage_methods = ["single", "complete", "average", "weighted"]
```

```
high_cophenet_corr = 0
high_dm_lm = [0, 0]

for dm in distance_metrics:
    for lm in linkage_methods:
        Z = linkage(subset_scaled_df, metric=dm, method=lm)
        c, coph_dists = cophenet(Z, pdist(subset_scaled_df))
        print(
            "Cophenetic correlation for {} distance and {} linkage is {}.".format(
                dm.capitalize(), lm, c
            )
        )
        if high_cophenet_corr < c:
            high_cophenet_corr = c
            high_dm_lm[0] = dm
            high_dm_lm[1] = lm
```

```
Cophenetic correlation for Euclidean distance and single linkage is 0.923227149400292
2.
Cophenetic correlation for Euclidean distance and complete linkage is 0.7873280186580
672.
Cophenetic correlation for Euclidean distance and average linkage is 0.94225406095608
14.
Cophenetic correlation for Euclidean distance and weighted linkage is 0.8693784298129
404.
Cophenetic correlation for Chebyshev distance and single linkage is 0.906253816475071
7.
Cophenetic correlation for Chebyshev distance and complete linkage is 0.5988914191112
42.
Cophenetic correlation for Chebyshev distance and average linkage is 0.93382655280304
99.
Cophenetic correlation for Chebyshev distance and weighted linkage is 0.912735589236
7.
Cophenetic correlation for Mahalanobis distance and single linkage is 0.9259195530524
59.
Cophenetic correlation for Mahalanobis distance and complete linkage is 0.79253072028
5.
Cophenetic correlation for Mahalanobis distance and average linkage is 0.924732403015
9737.
Cophenetic correlation for Mahalanobis distance and weighted linkage is 0.87083174901
80426.
Cophenetic correlation for Cityblock distance and single linkage is 0.933418636652857
4.
Cophenetic correlation for Cityblock distance and complete linkage is 0.7375328863205
818.
Cophenetic correlation for Cityblock distance and average linkage is 0.93021450485946
67.
Cophenetic correlation for Cityblock distance and weighted linkage is 0.7310455135202
81.
```

In [85]:

```
# printing the combination of distance metric and linkage method with the highest coph
print(
    "Highest cophenetic correlation is {}, which is obtained with {} distance and {} l
        high_cophenet_corr, high_dm_lm[0].capitalize(), high_dm_lm[1]
    )
)
```

```
Highest cophenetic correlation is 0.9422540609560814, which is obtained with Euclidea
n distance and average linkage.
```

Let's explore different linkage methods with Euclidean distance only.

In [86]:
```
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighted"]

high_cophenet_corr = 0
high_dm_lm = [0, 0]

for lm in linkage_methods:
    Z = linkage(subset_scaled_df, metric="euclidean", method=lm)
    c, coph_dists = cophenet(Z, pdist(subset_scaled_df))
    print("Cophenetic correlation for {} linkage is {}.".format(lm, c))
    if high_cophenet_corr < c:
        high_cophenet_corr = c
        high_dm_lm[0] = "euclidean"
        high_dm_lm[1] = lm
```

```
Cophenetic correlation for single linkage is 0.9232271494002922.
Cophenetic correlation for complete linkage is 0.7873280186580672.
Cophenetic correlation for average linkage is 0.9422540609560814.
Cophenetic correlation for centroid linkage is 0.9314012446828154.
Cophenetic correlation for ward linkage is 0.7101180299865353.
Cophenetic correlation for weighted linkage is 0.8693784298129404.
```

In [87]:
```
# printing the combination of distance metric and linkage method with the highest coph
print(
    "Highest cophenetic correlation is {}, which is obtained with {} linkage.".format(
        high_cophenet_corr, high_dm_lm[1]
    )
)
```

```
Highest cophenetic correlation is 0.9422540609560814, which is obtained with average
linkage.
```

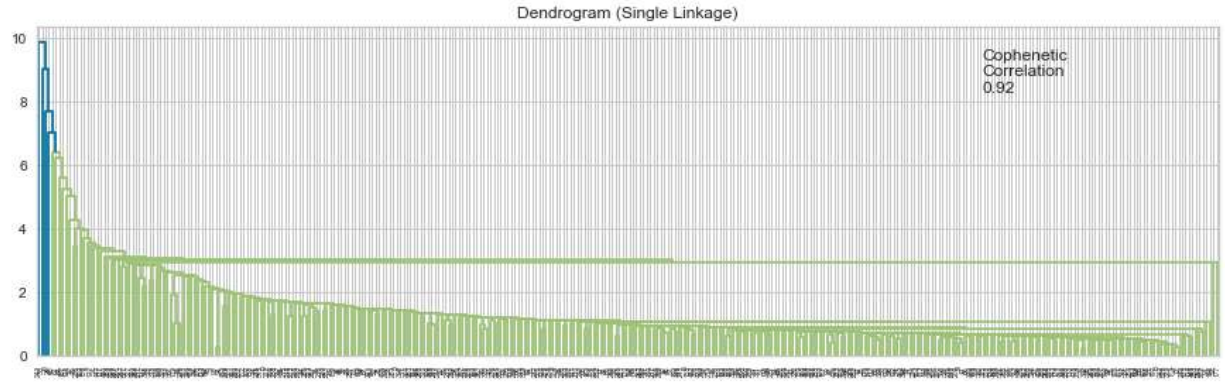
Observations

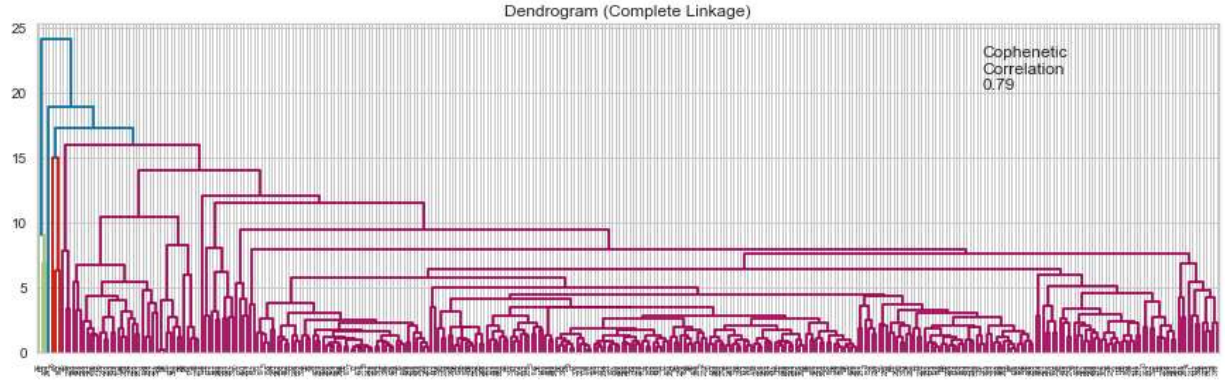
- We see that the cophenetic correlation is maximum with euclidian distance, and average linkage.

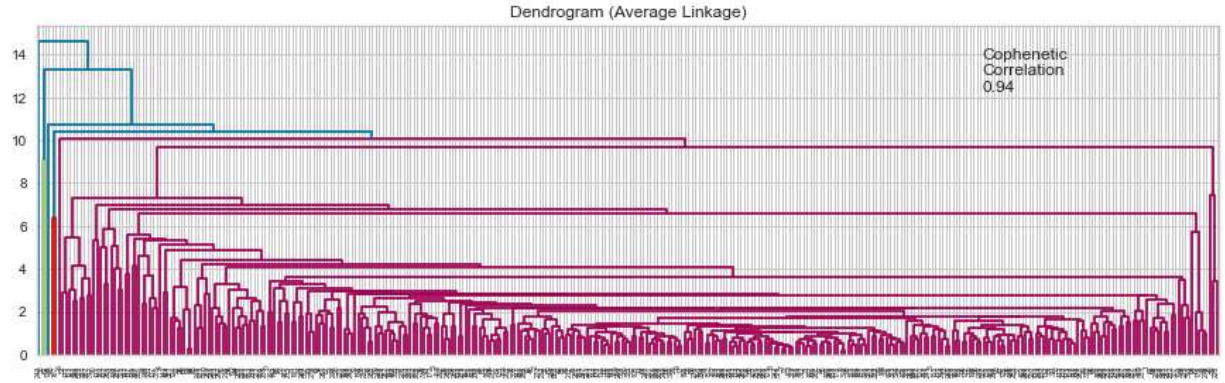
Checking Dendrograms

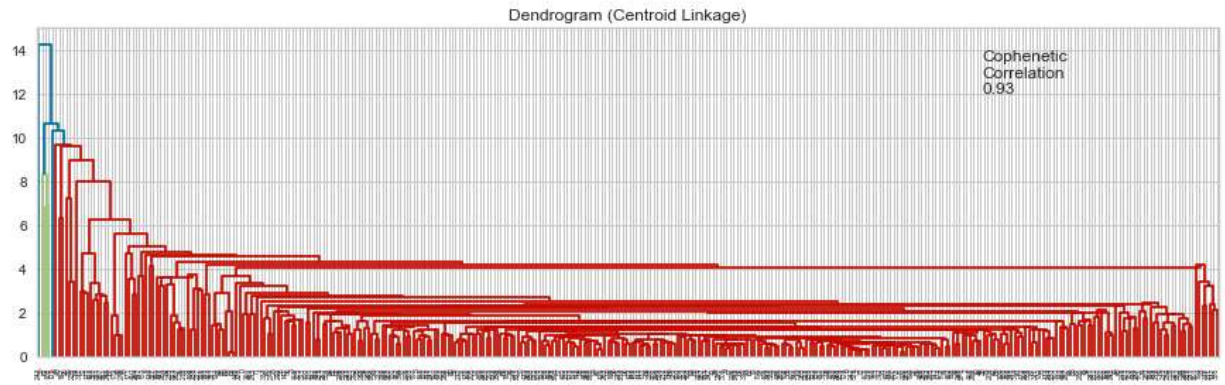
In [89]:
```
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighted"]

# lists to save results of cophenetic correlation calculation
compare_cols = ["Linkage", "Cophenetic Coefficient"]
compare = []

# to create a subplot image
fig, axs = plt.subplots(len(linkage_methods), 1, figsize=(15, 30))

# We will enumerate through the list of linkage methods above
# For each linkage method, we will plot the dendrogram and calculate the cophenetic co
for i, method in enumerate(linkage_methods):
    Z = linkage(subset_scaled_df, metric="euclidean", method=method)

    dendrogram(Z, ax=axs[i])
```

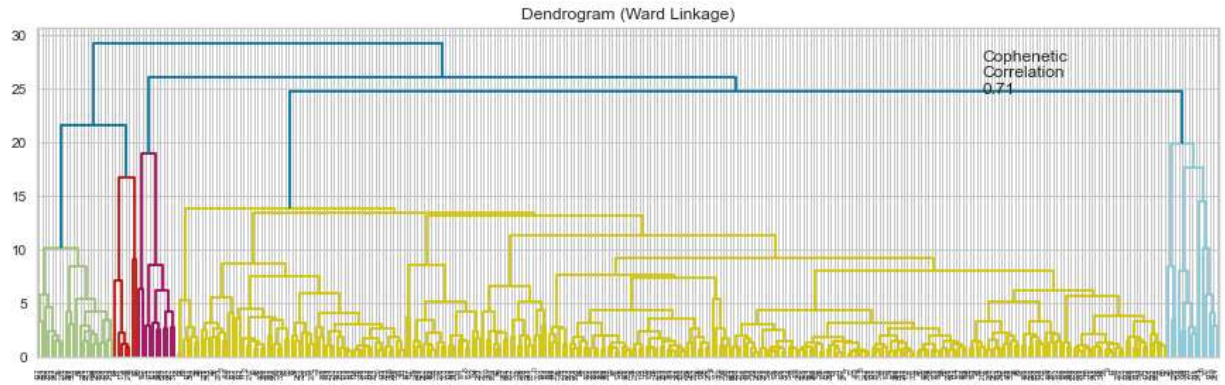
```
    axs[i].set_title(f"Dendrogram ({method.capitalize()} Linkage)")

    coph_corr, coph_dist = cophenet(Z, pdist(subset_scaled_df))
    axs[i].annotate(
        f"Cophenetic\nCorrelation\n{coph_corr:0.2f}",
        (0.80, 0.80),
        xycoords="axes fraction",
    )

    compare.append([method, coph_corr])
```

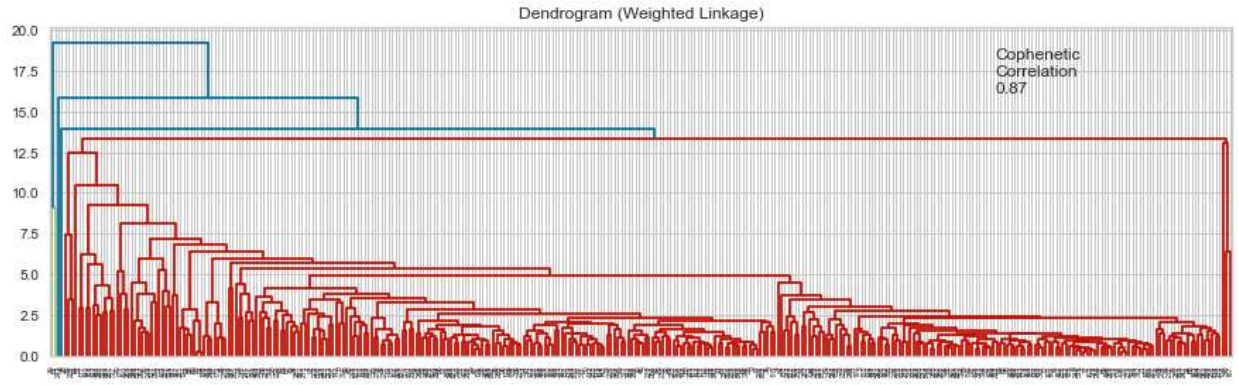

Dendrogram (Single Linkage)
Cophenetic
Correlation
0.92


Dendrogram (Complete Linkage)
Cophenetic
Correlation
0.79


Dendrogram (Average Linkage)
Cophenetic
Correlation
0.94


Dendrogram (Centroid Linkage)
Cophenetic
Correlation
0.93


Dendrogram (Ward Linkage)
Cophenetic
Correlation
0.71



In [90]:
```
# let's create a dataframe to compare cophenetic correlations for each linkage method
df_cc = pd.DataFrame(compare, columns=compare_cols)
df_cc
```

Out[90]:

	Linkage	Cophenetic Coefficient
0	single	0.923227
1	complete	0.787328
2	average	0.942254
3	centroid	0.931401
4	ward	0.710118
5	weighted	0.869378

Creating Final Model

Let's create 4 clusters.

- Ward linkage clusters appear to be distinctly separate as compared to average linkage

In [128…
```
HCmodel = AgglomerativeClustering(n_clusters=4, affinity="euclidean", linkage="ward")
HCmodel.fit(subset_scaled_df)
```

Out[128]:
```
AgglomerativeClustering(n_clusters=4)
```

In [129…
```
# adding hierarchical cluster labels to the original and scaled dataframes
#df.drop("K_means_segments", axis=1, inplace=True)
subset_scaled_df["HC_Clusters"] = HCmodel.labels_
df["HC_Clusters"] = HCmodel.labels_
```

Cluster Profiling

In [130…
```
cluster_profile = df.groupby("HC_Clusters").mean()
```

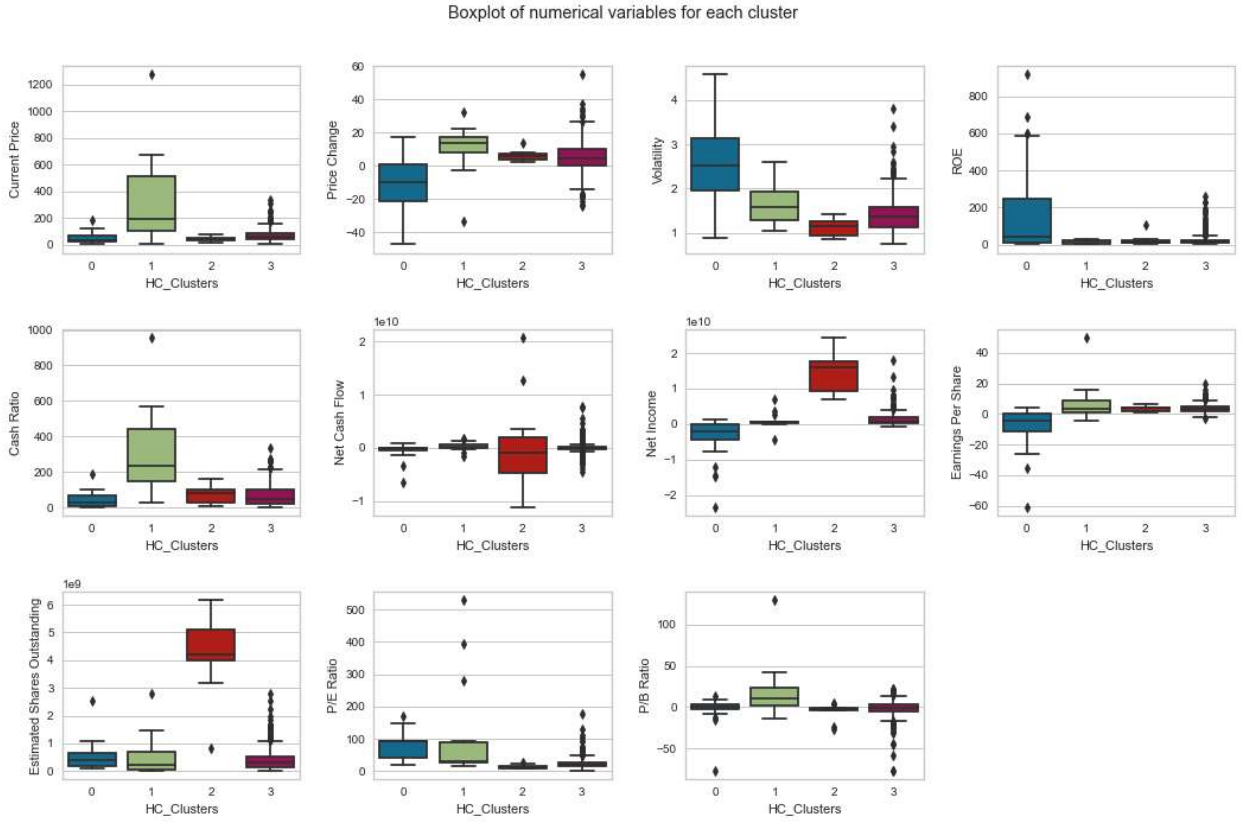
In [131…
```
    cluster_profile["count_in_each_segments"] = (
        df.groupby("HC_Clusters")["Current Price"].count().values
    )
```

In [133…
```
# let's display cluster profiles
cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

Out[133]:

HC_Clusters	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash Flow	Net
0	48.006208	-11.263107	2.590247	196.551724	40.275862	-495901724.137931	-359724465.
1	326.198218	10.563242	1.642560	14.400000	309.466667	288850666.666667	86449853.
2	42.848182	6.270446	1.123547	22.727273	71.454545	558636363.636364	1463127272
3	72.760400	5.213307	1.427078	25.603509	60.392982	79951512.280702	153859432.

In [134…
```
plt.figure(figsize=(15, 10))
plt.suptitle("Boxplot of numerical variables for each cluster")

for i, variable in enumerate(num_col):
    plt.subplot(3, 4, i + 1)
    sns.boxplot(data=df, x="HC_Clusters", y=variable)

plt.tight_layout(pad=2.0)
```



In [135…
```
# Comparing cluster vs. GICS_Sector
```

```
pd.crosstab(df['GICS Sector'], df.HC_Clusters).style.highlight_max(color = 'lightgreen
```

Out[135]:

GICS Sector	0	1	2	3
Consumer Discretionary	1	3	1	35
Consumer Staples	2	1	1	15
Energy	22	0	1	7
Financials	1	0	4	44
Health Care	0	5	1	34
Industrials	1	0	0	52
Information Technology	1	4	1	27
Materials	1	0	0	19
Real Estate	0	1	0	26
Telecommunications Services	0	1	2	2
Utilities	0	0	0	24

- **Cluster 0**:
  - Current price is moderate and Price change is low
  - Volatility is very high
  - ROE is the highest
  - Cash Ratio and Net Cashflow are the lowest
  - Net Income is the lowest
  - EPS is the lowest amongst all clusters
  - ESO and P/E Ratio are moderately high.
  - P/B ratio is moderately low.
  - A total of 29 securities fall into this cluster.
- **Cluster 1**:
  - Current price is highest and Price change is too
  - Volatility is moderate
  - ROE is low
  - Cash Ratio and Net Cashflow are among the highest
  - Net Income is relatively moderate
  - EPS is the highest amongst all clusters
  - ESO is lowest P/E Ratio is moderately high.
  - P/B ratio is high.
  - A total of 15 security fall into this cluster.
- **Cluster 2**:
  - Current price and Price change are lowest
  - Volatility is moderate

- ROE is low
- Cash Ratio and Net Cashflow are among lowest
- Net Income is highest
- EPS is moderate
- ESO is highest and P/E Ratio is lowest.
- P/B ratio is moderate.
- A total of 11 securities fall into this cluster.

- **Cluster 3**:
  - Current price is moderate and Price change is lower
  - Volatility is very low
  - ROE is low
  - Cash Ratio and Net Cashflow are moderate
  - Net Income is moderate
  - EPS is moderate upon comparison with all clusters
  - ESO and P/E Ratio are modereate.
  - P/B ratio is moderate among clusters.
  - A total of 285 security fall into this cluster.

K-means vs Hierarchical Clustering

- Which clustering technique took less time for execution?
  - *Both the KMeans model and the Agglomerative Clustering model fit the dataset within a few seconds.*
- Which clustering technique gave you more distinct clusters, or are they the same?
  - *Both techniques gave around the same number of clusters. 4 was identified to be an optimal number*
- How many observations are there in the similar clusters of both algorithms?
  - *Both clustering techniques gave about the same number of clusters. KMeans has 25,7,11 and 277 securities in each cluster as opposed to 29, 15, 11 and 285*
- How many clusters are obtained as the appropriate number of clusters from both algorithms?
  - *Both algorithms yield similar clusters based on the outliers within the 11 variables*

Actionable Insights and Recommendations

- Trade&Ahead should first understand the requirement, financial goal and risk tolerance of its customers.

- Trade&Ahead should share all the options with its customers,i.e. Large Cap, Mid Cap, Small Cap and Flexi Cap, and walk them through about the Pros and Cons of each investment by observing the trend of past few years that best suits its customers.
- Trade&Ahead should also share the Large Cap - Dividend making stocks with its ultra rich investors who are not bothered about liquidity and can keep capital invested.
- Trade&Ahead should also suggest ETFs, Mutual Funds etc. that invest in a wide variety of stocks lowering the dependency on a single stock giving an overall gain later.
- These clusterings should be a starting point in targeting the investments and not the end.
- A thorough and frequent analysis should be done based on the market performance instead of just relying on the clusters.