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 [1]:

```
# 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 dendrogr
ams
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, cophenet

# to suppress warnings
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
## Complete the code to import the data
data = pd.read_csv('stock_data.csv')
```

In [3]:

```
data.shape
```

Out[3]:

```
(340, 15)
```

In [4]:

```
# checking shape of the data
print(f"There are {data.shape[0]} rows and {data.shape[1]} columns.")
```

```
There are 340 rows and 15 columns.
```

In [5]:

```
# let's view a sample of the data
data.sample(n=10, random_state=1)
```

Out[5]:

	Ticker Symbol	Security	GICS Sector	GICS Sub Industry	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash Flow	Net Incom

	Ticker Symbol	Security	GICS Sector	GICS Sub Industry	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash Flow	Net Incom
102	DVN	Devon Energy Corp.	Energy	Oil & Gas Exploration & Production	32.000000	15.478079	2.923698	205	70	830000000	1445400000
125	FB	Facebook	Information Technology	Internet Software & Services	104.660004	16.224320	1.320606	8	958	592000000	366900000
11	AIV	Apartment Investment & Mgmt	Real Estate	REITs	40.029999	7.578608	1.163334	15	47	21818000	24871000
248	PG	Procter & Gamble	Consumer Staples	Personal Products	79.410004	10.660538	0.806056	17	129	160383000	63605600
238	OXY	Occidental Petroleum	Energy	Oil & Gas Exploration & Production	67.610001	0.865287	1.589520	32	64	-588000000	-782900000
336	YUM	Yum! Brands Inc	Consumer Discretionary	Restaurants	52.516175	-8.698917	1.478877	142	27	159000000	129300000
112	EQT	EQT Corporation	Energy	Oil & Gas Exploration & Production	52.130001	-21.253771	2.364883	2	201	523803000	8517100
147	HAL	Halliburton Co.	Energy	Oil & Gas Equipment & Services	34.040001	-5.101751	1.966062	4	189	7786000000	-67100000
89	DFS	Discover Financial Services	Financials	Consumer Finance	53.619999	3.653584	1.159897	20	99	2288000000	229700000
173	IVZ	Invesco Ltd.	Financials	Asset Management & Custody Banks	33.480000	7.067477	1.580839	12	67	412000000	96810000

In [6]:

```
# checking the column names and datatypes
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
```

In [7]:

```
# copying the data to another variable to avoid any changes to original data
df = data.copy()
```

In [8]:

```
# checking for duplicate values
```

```
df.duplicated().sum()
```

Out[8]:

```
0
```

- **Dataset has no missing or duplicate values**
- **All columns with dtype object should be dtype category in order to conserve memory**

In [9]:

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

In [10]:

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

In [11]:

```
# confirm new dataset
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
```

- **The 14 columns have three different dtypes: category(3), float64(7), int64(4)**
- **All of these dtypes are appropriate for their respective columns**

In [12]:

```
# checking for missing values in the data
df.isna().sum()
```

Out[12]:

```
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 missing values in the data.**

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.**

Let's check the statistical summary of the data.

In [13]:

```
#provide statistical summary of all categorical columns
df.describe(include='category').T
```

Out[13]:

	count	unique	top	freq
Security	340	340	3M Company	1
GICS Sector	340	11	Industrials	53
GICS Sub Industry	340	104	Oil & Gas Exploration & Production	16

Univariate analysis

In [14]:

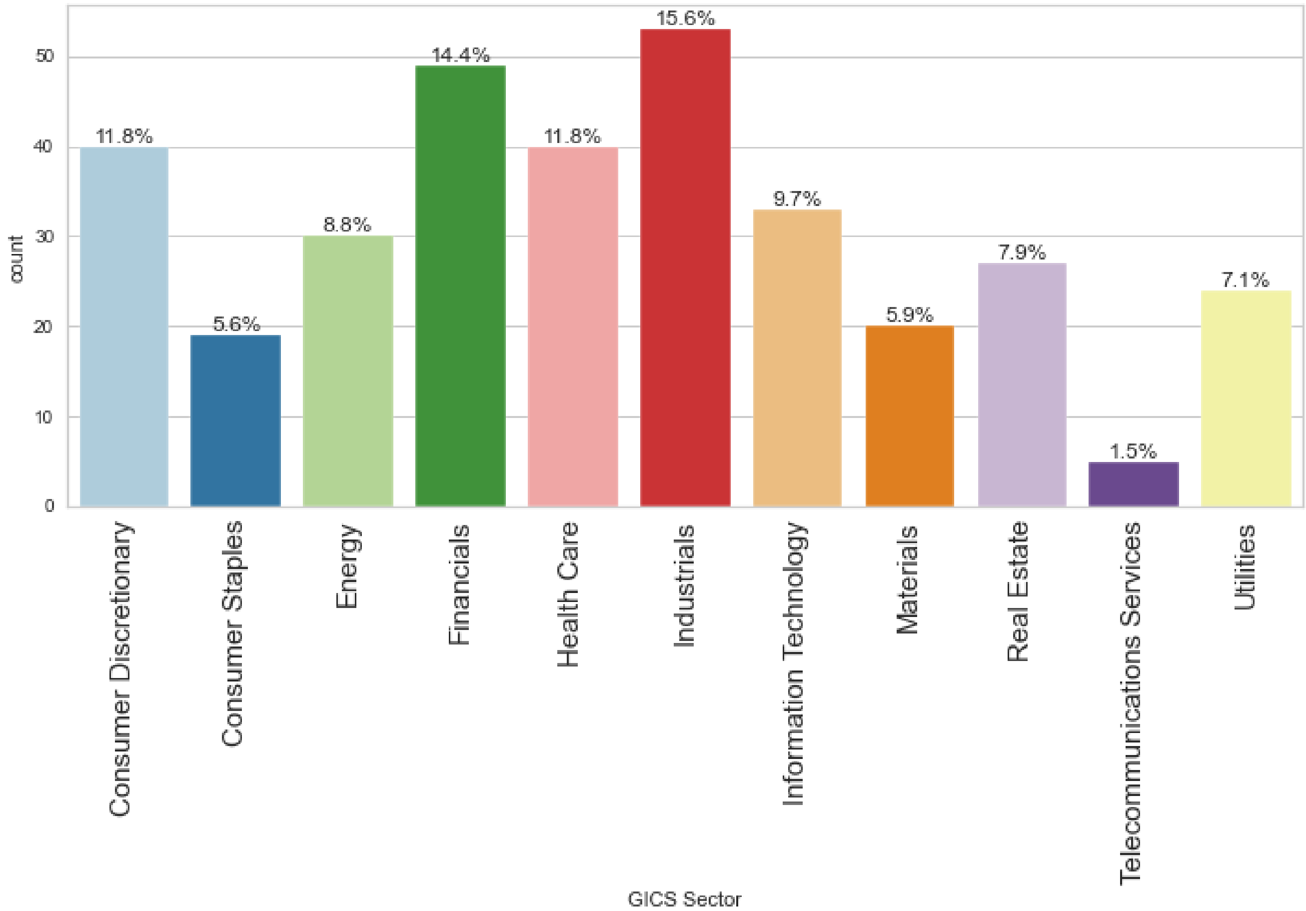
```
# function to create labeled barplots


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

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=df,
        x=feature,
        palette="Paired",
        order=df[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
```

`GICS Sector`

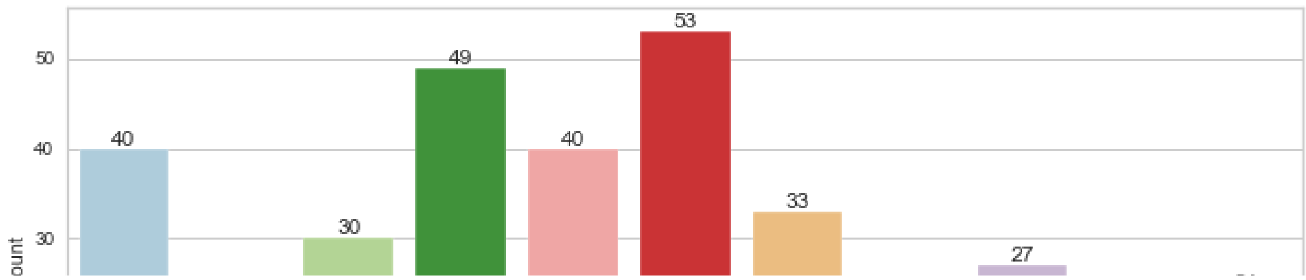
In [15]:

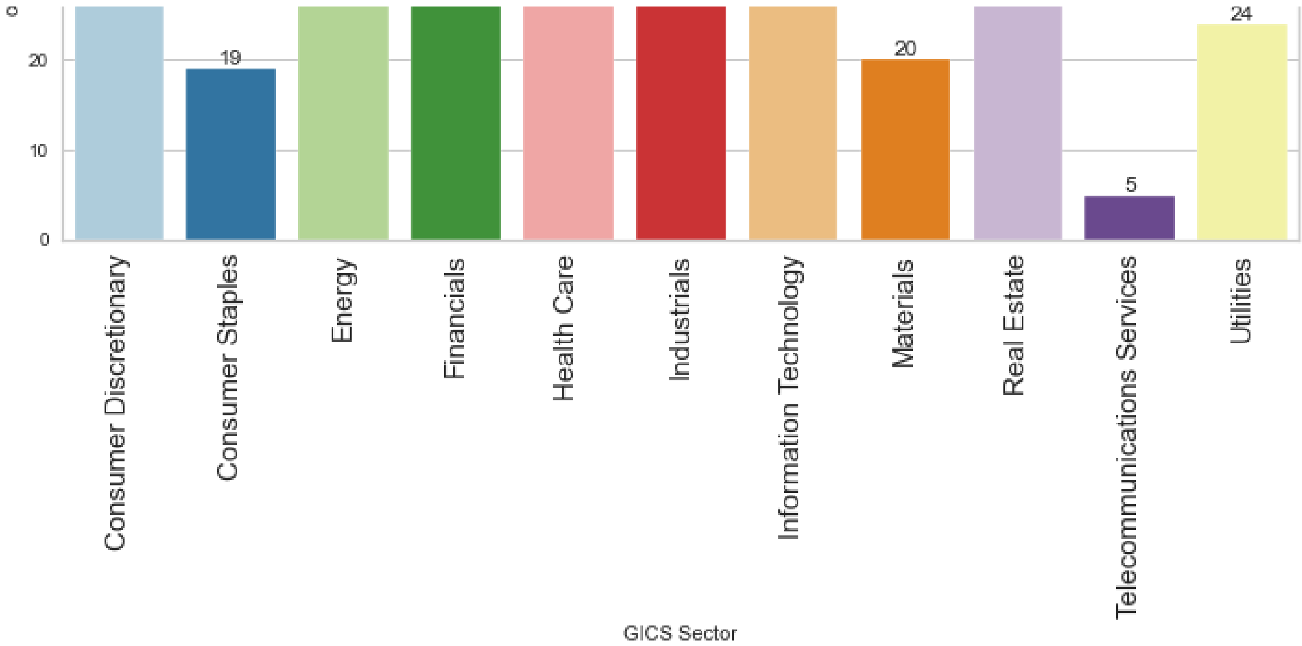
```
labeled_barplot(df, 'GICS Sector', perc=True)
```



In [16]:

```
#create labeled barplot of stocks by sector
labeled_barplot(df, 'GICS Sector')
```





In [17]:

```
#display the five sectors with the most number of stocks
df["GICS Sector"].value_counts().head(n=5)
```

Out[17]:

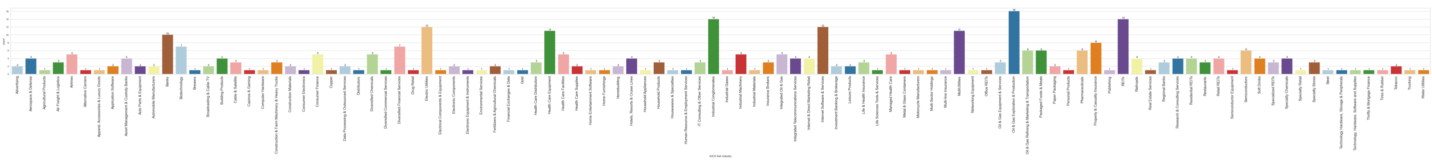
```
Industrials               53
Financials                49
Consumer Discretionary    40
Health Care               40
Information Technology    33
Name: GICS Sector, dtype: int64
```

- **The stocks are drawn from 11 different industrial sectors, with no one sector comprising more than 16% of the dataset**
- **The top 4 of the 11 sectors (industrials, financials, consumer discretionary, and health care) comprise over half of the total number of stocks**

`GICS Sub Industry`

In [18]:

```
#create labeled barplot of stocks by sub industry
labeled_barplot(df, 'GICS Sub Industry')
```



In [19]:

```
#display the five sub industries with the most number of stocks
df['GICS Sub Industry'].value_counts().head(n=5)
```

Out[19]:

```
Oil & Gas Exploration & Production    16
REITs                                 14
Industrial Conglomerates              14
Internet Software & Services          12
Electric Utilities                    12
Name: GICS Sub Industry, dtype: int64
```

- **The dataset is comprised of stocks from 104 different subindustries, with no subindustry having more than 16 stocks in the dataset**

16 stocks in the dataset

- These observations indicate that the 340 stocks held within the dataset are highly diversified across sectors and subindustries

In [20]:

```
#provide statistical summary of all numerical columns
df.describe().T
```

Out[20]:

	count	mean	std	min	25%	50%	75%	max
Current Price	340.0	8.086234e+01	9.805509e+01	4.500000e+00	3.855500e+01	5.970500e+01	9.288000e+01	1.274950e+03
Price Change	340.0	4.078194e+00	1.200634e+01	-4.712969e+01	-9.394838e-01	4.819505e+00	1.069549e+01	5.505168e+01
Volatility	340.0	1.525976e+00	5.917984e-01	7.331632e-01	1.134878e+00	1.385593e+00	1.695549e+00	4.580042e+00
ROE	340.0	3.959706e+01	9.654754e+01	1.000000e+00	9.750000e+00	1.500000e+01	2.700000e+01	9.170000e+02
Cash Ratio	340.0	7.002353e+01	9.042133e+01	0.000000e+00	1.800000e+01	4.700000e+01	9.900000e+01	9.580000e+02
Net Cash Flow	340.0	5.553762e+07	1.946365e+09	-1.120800e+10	-1.939065e+08	2.098000e+06	1.698108e+08	2.076400e+10
Net Income	340.0	1.494385e+09	3.940150e+09	-2.352800e+10	3.523012e+08	7.073360e+08	1.899000e+09	2.444200e+10
Earnings Per Share	340.0	2.776662e+00	6.587779e+00	-6.120000e+01	1.557500e+00	2.895000e+00	4.620000e+00	5.009000e+01
Estimated Shares Outstanding	340.0	5.770283e+08	8.458496e+08	2.767216e+07	1.588482e+08	3.096751e+08	5.731175e+08	6.159292e+09
P/E Ratio	340.0	3.261256e+01	4.434873e+01	2.935451e+00	1.504465e+01	2.081988e+01	3.176476e+01	5.280391e+02
P/B Ratio	340.0	-1.718249e+00	1.396691e+01	-7.611908e+01	-4.352056e+00	-1.067170e+00	3.917066e+00	1.290646e+02

- **Numerical Columns**

In [21]:

```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(df, 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=df, 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=df, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=df, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
```

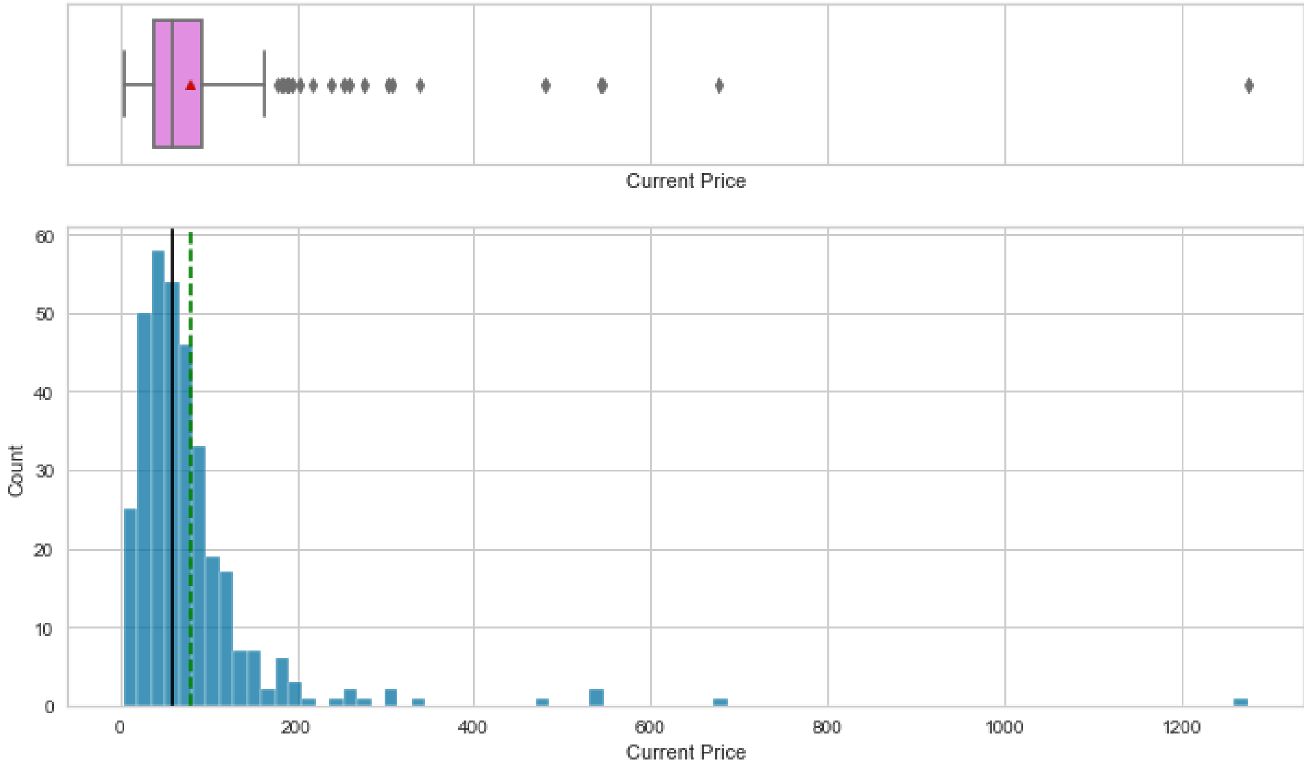
```
        df[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        df[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

Q1: What does the distribution of stock prices look like?

`Current Price`

In [22]:

```
histogram_boxplot(df, 'Current Price')
```



***The distribution is heavily right skewed, with 49 of the 340 stocks having twice the median value of all stocks**

- **As expected, no stock is listed at less of less than 0 dollars**

`Price Change`

In [23]:

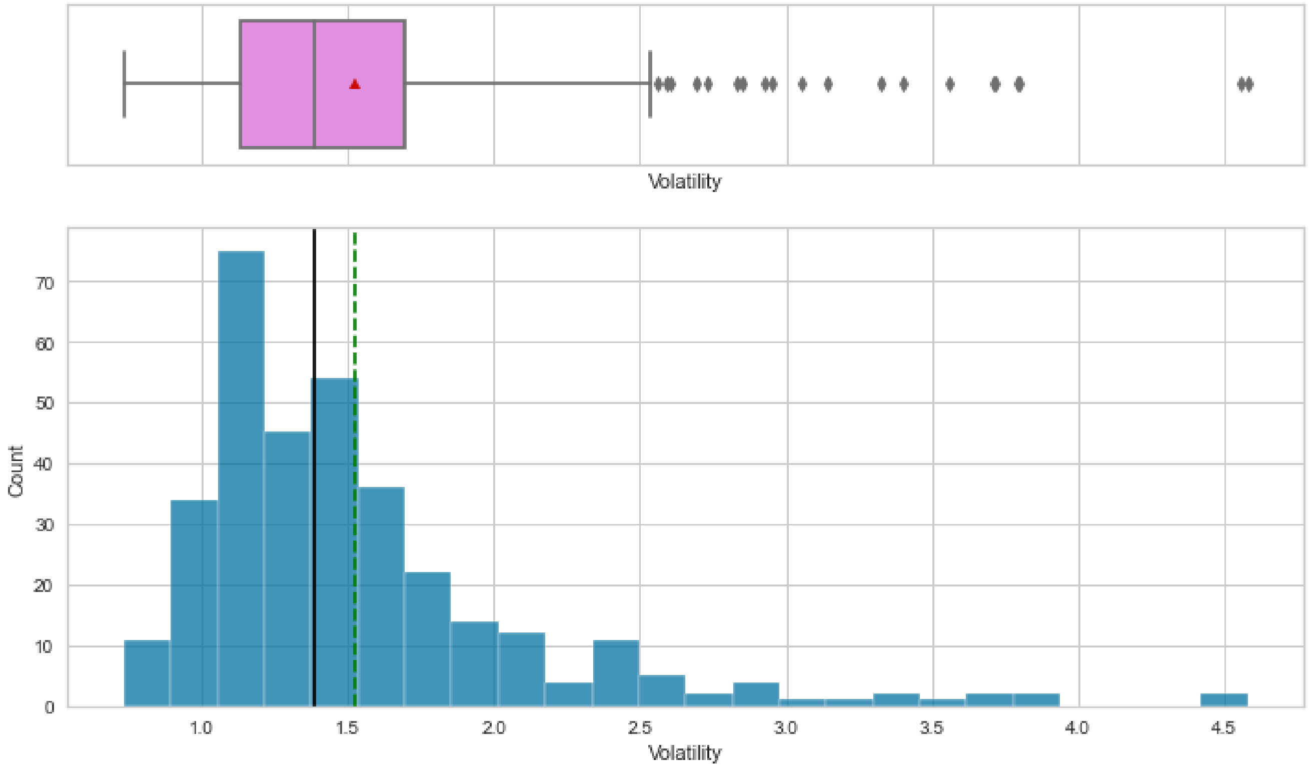
```
histogram_boxplot(df, 'Price Change')
```

- **The distribution is biased towards lower volatilities, but long tails do exist both for positive and negative price changes**
- **The most volatile stocks show as low as a 47% decrease to as high as a 55% increase over 13 weeks**

`Volatility`

In [24]:

```
histogram_boxplot(df, 'Volatility')
```

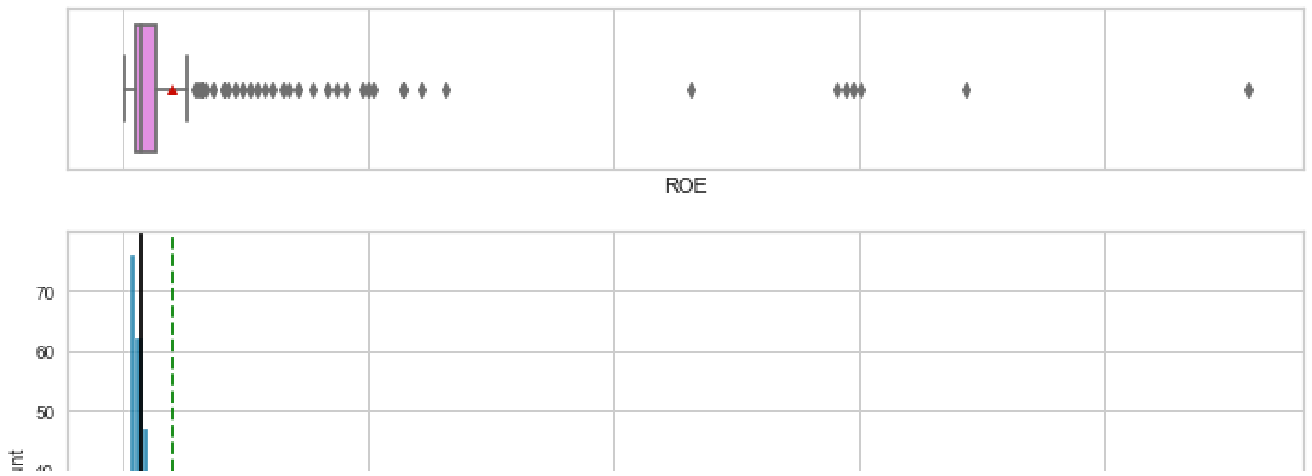


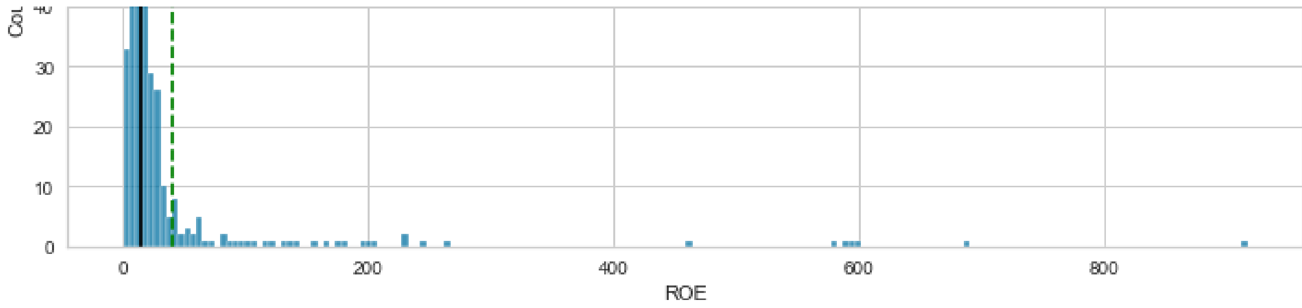
- **As expected, the distribution of standard deviations is right skewed and not normal**

`ROE`

In [25]:

```
histogram_boxplot(df, 'ROE')
```



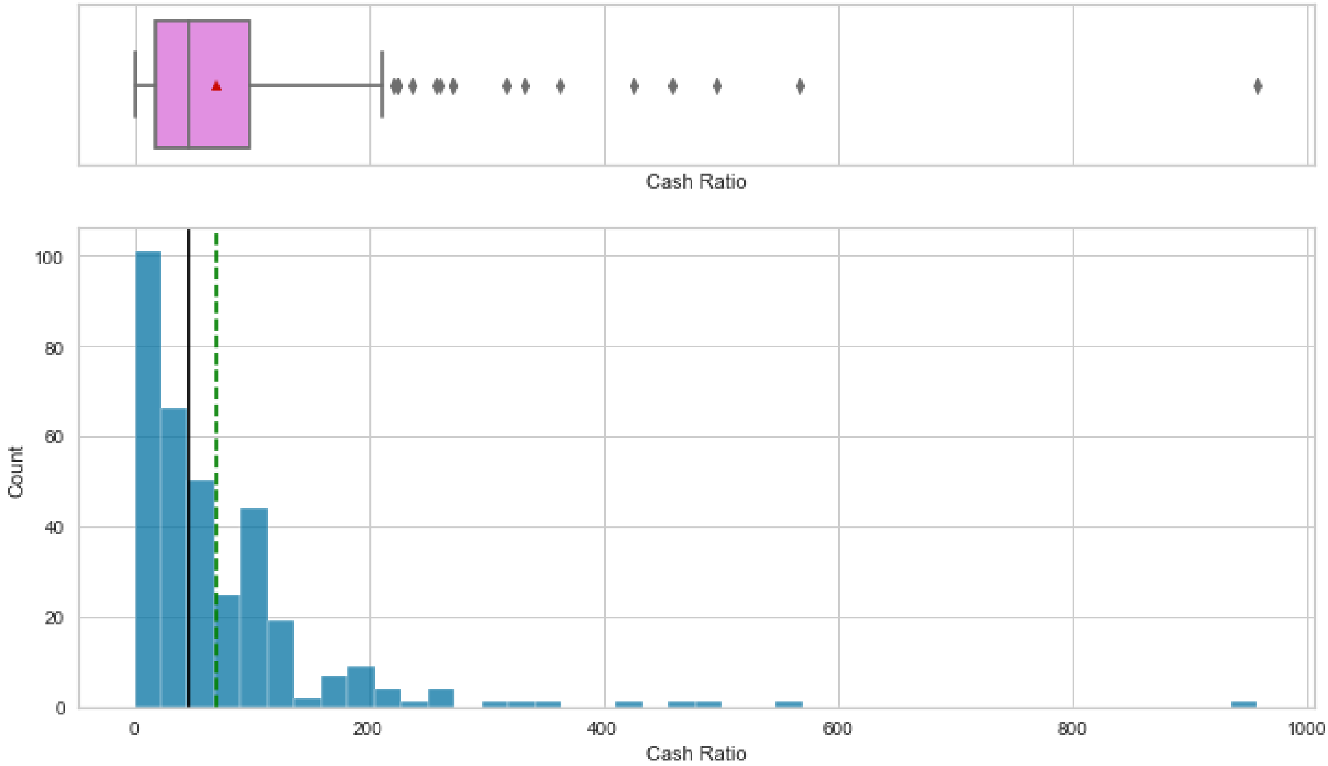


- **As expected, both distributions are heavily right skewed and no stock is listed with either metric with a value of less than 0**
- **For example, 24 stocks are listed with returns on equity of less than 5 and 25 stocks are listed with returns of over 100 percent**

`Cash Ratio`

In [26]:

```
histogram_boxplot(df, 'Cash Ratio')
```

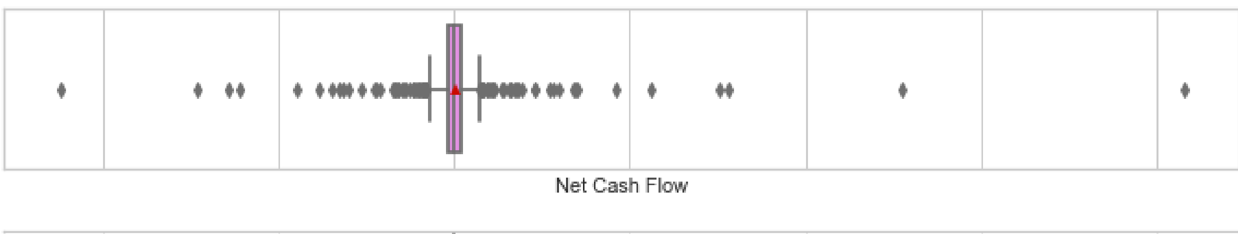


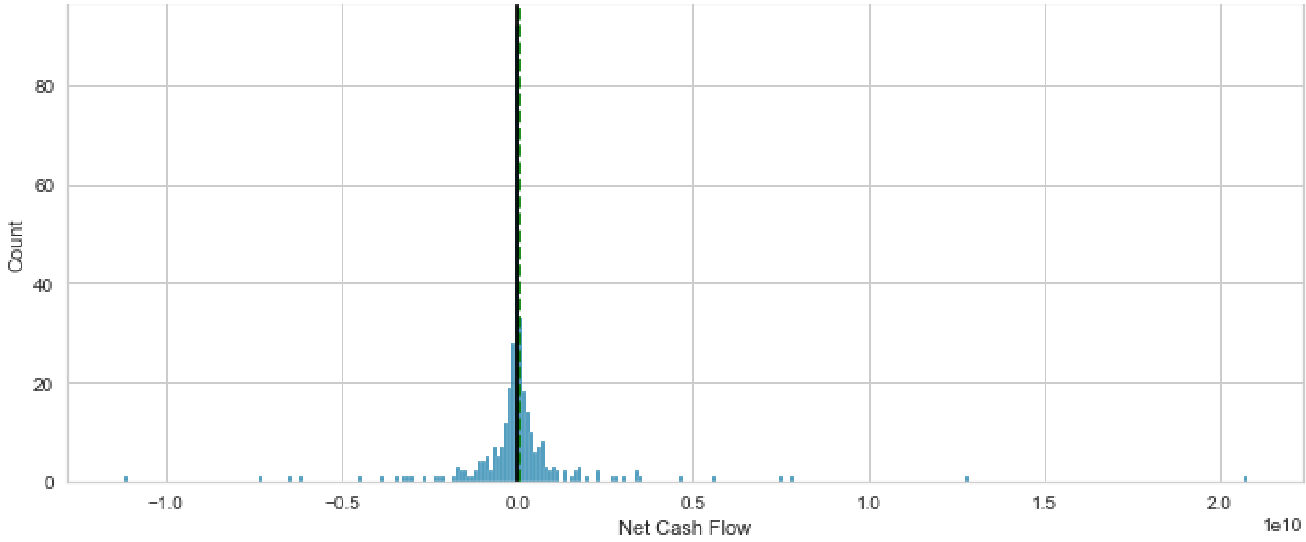
- **As expected, both distributions are heavily right skewed and no stock is listed with either metric with a value of less than 0**
- **For example, 24 stocks are listed with returns on equity of less than 5 and 25 stocks are listed with returns of over 100 percent**

`Net Cash Flow`

In [27]:

```
histogram_boxplot(df, 'Net Cash Flow')
```

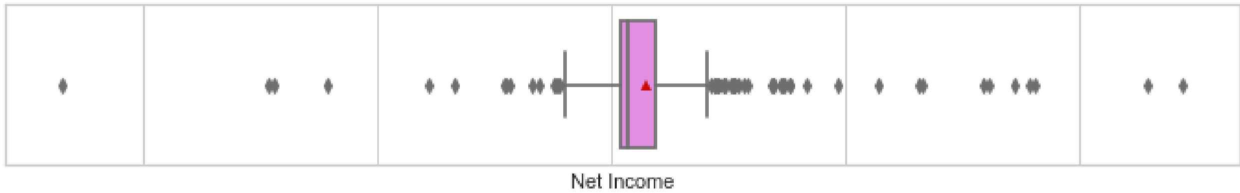


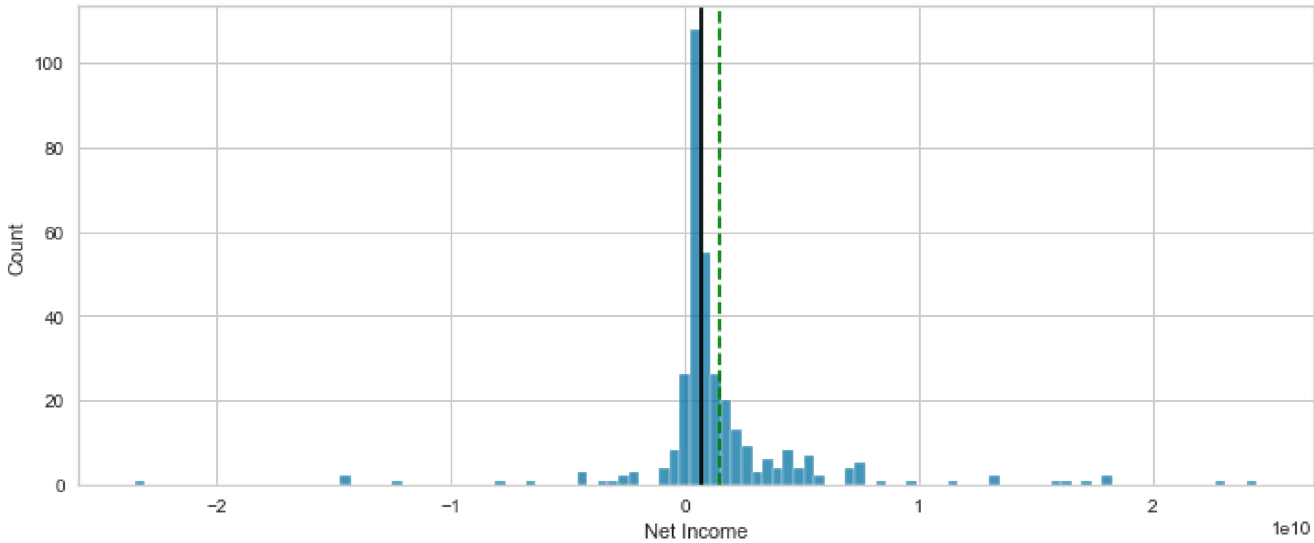


`Net Income`

In [28]:

```
histogram_boxplot(df, 'Net Income')
```



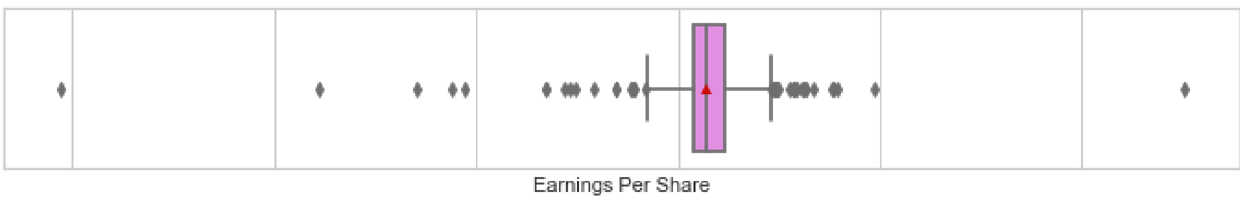


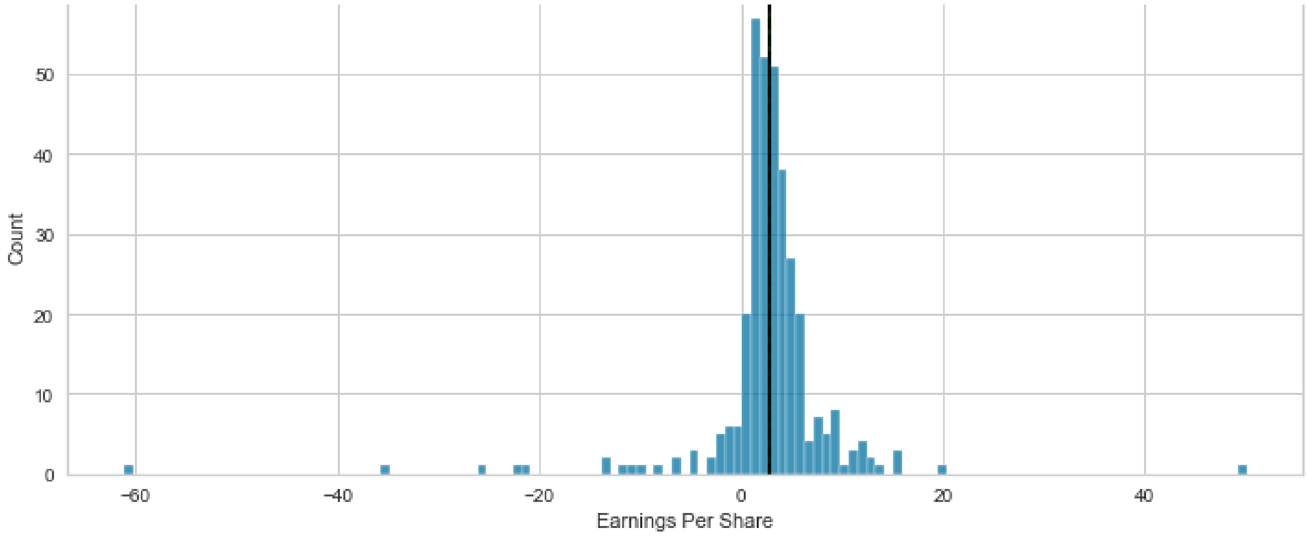
- **As expected, net income is shown to be right skewed with both long positive and negative tails I.e., most companies generate meager profits, but some are failing and some are highly successful**
- **32 companies within the dataset are showing a net income of less than 0 dollars**

`Earnings Per Share`

In [29]:

```
histogram_boxplot(df, 'Earnings Per Share')
```



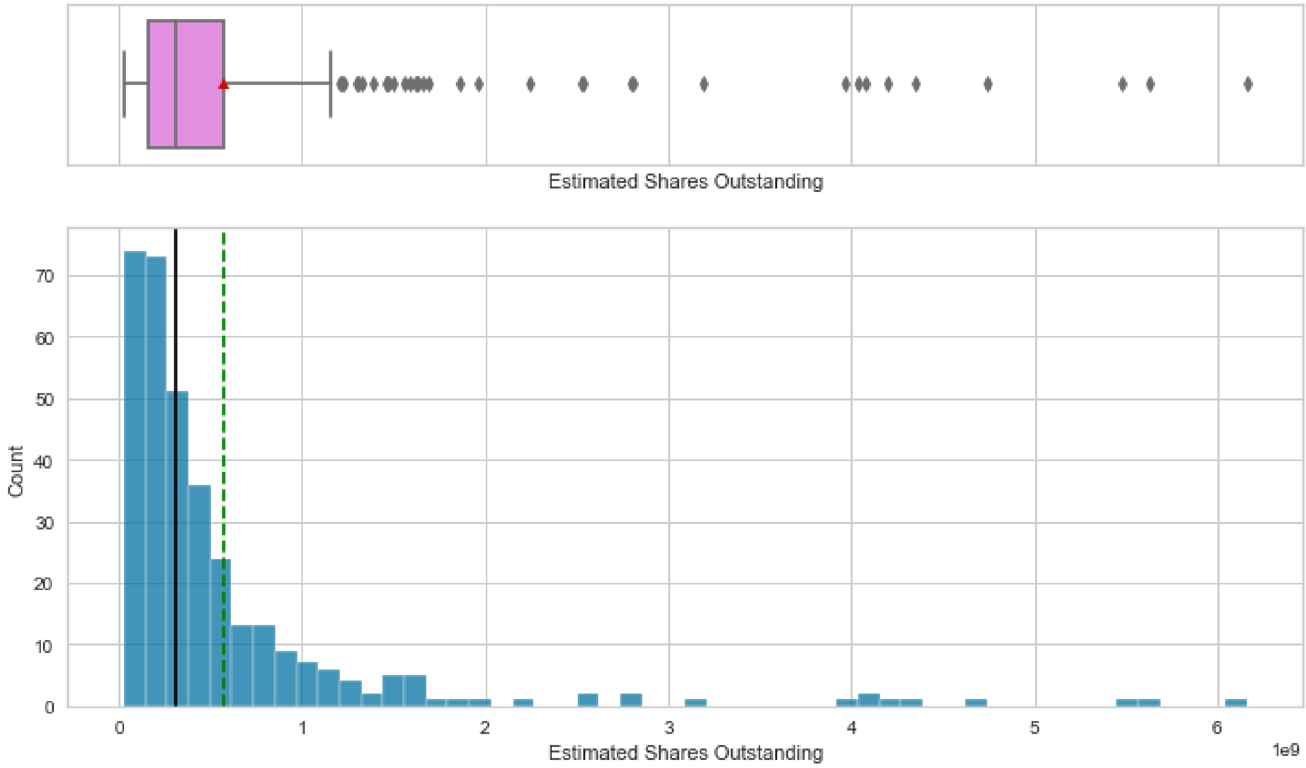


- **EPS, as a derivative of Net Income, shows a similar distribution, with most showing low positive values and a few stocks (34) showing negative values**

`Estimated Shares Outstanding`

In [30]:

```
histogram_boxplot(df, 'Estimated Shares Outstanding')
```

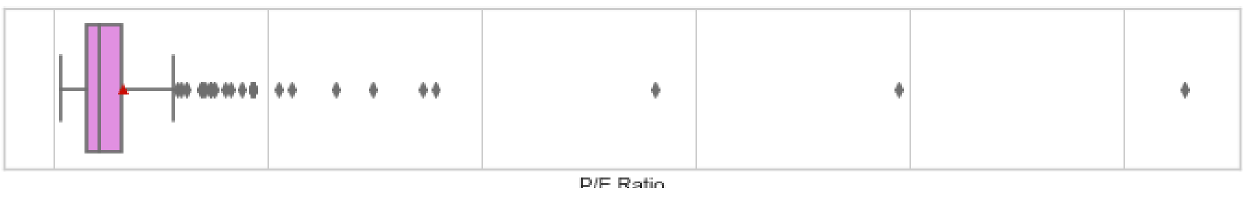


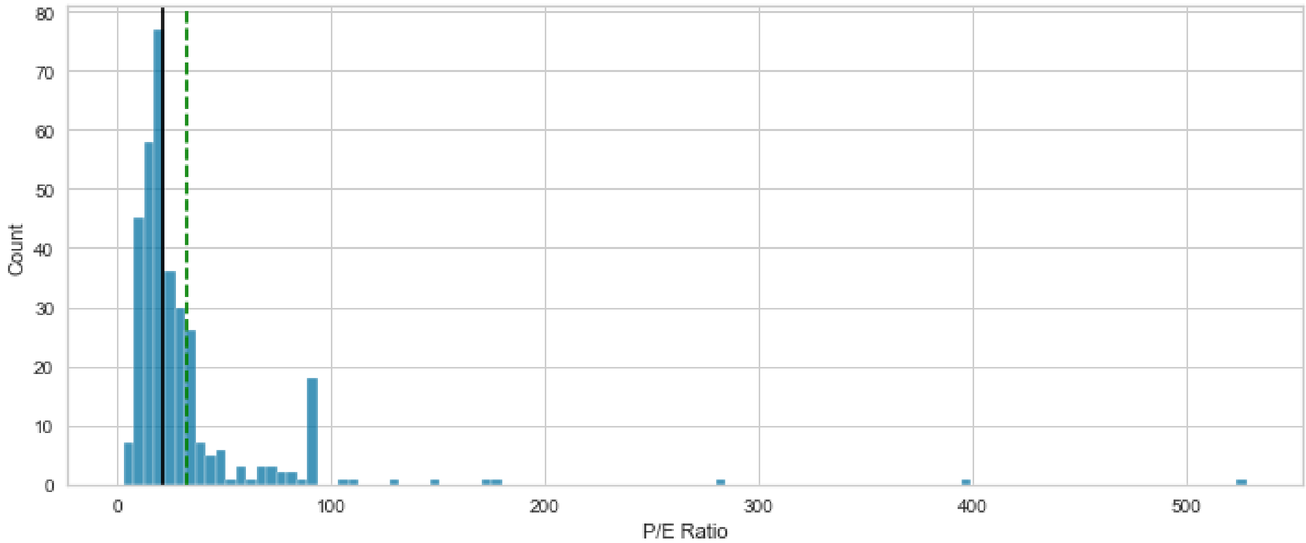
- **The distribution is highly right skewed, but no stock has a value of outstanding shares that is unrealistic**

`P/E Ratio`

In [31]:

```
histogram_boxplot(df, 'P/E Ratio')
```



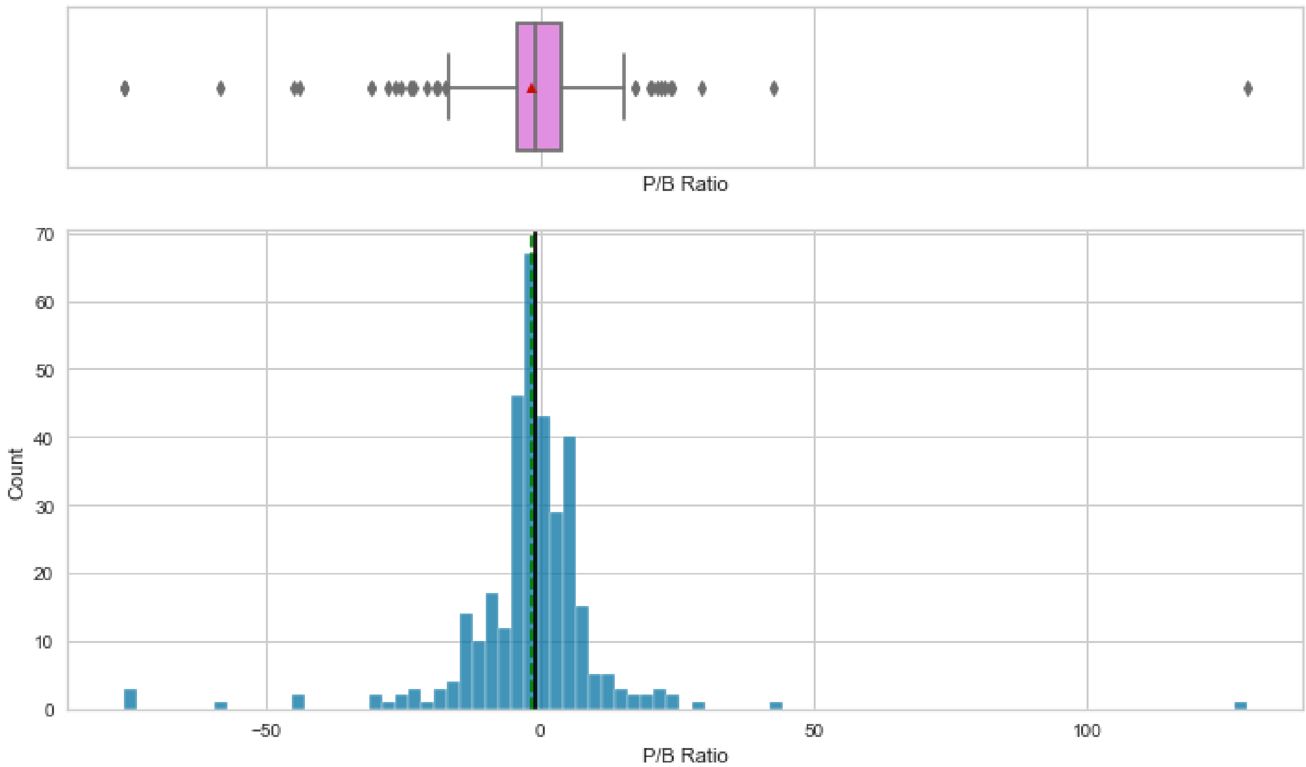


- **The distribution of P/E ratios is highly right skewed**
- **Interestingly, no stock shows a negative ratio, even though several stocks have a negative EPS and no stock stock has a price listed of less than 0**

`P/B Ratio`

In [32]:

```
histogram_boxplot(df, 'P/B Ratio')
```



- **The distribution for P/B ratios is mostly centered around 0 but with long positive and negative**
- **For example, 175 of the 340 total stocks are shown to below the 25th percentile and above the 75th percentile and**
- **Additionally, 31 of the stocks are outliers**

Conclusions

- **As expected, stocks offer uncertain returns with high upsides, mostly modest returns, and the omnipresent possibility that the value of the stock may become worthless (i.e., the company goes bankrupt)**

- **All of these variables contain a few or several outliers; however, none of these values appear to be unrealistic given the nature of stock prices and historical expectations**

Bivariate Analysis

Q2: The stocks of which economic sector have seen the maximum price increase on average?

In [33]:

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

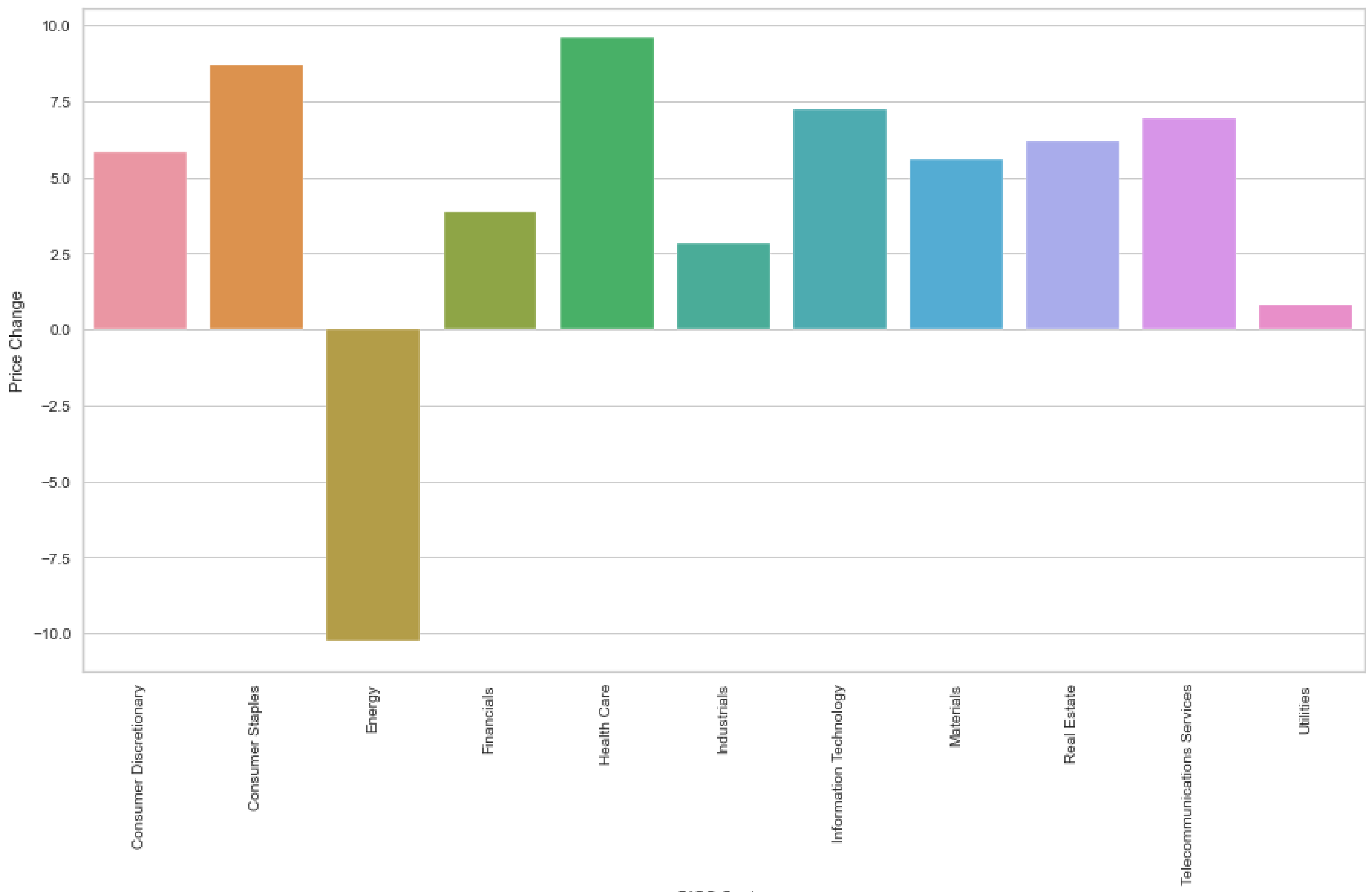
Out[33]:

```
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
```

- **Stocks within the health care sectors have shown the highest average price increase over the preeceding period**

In [34]:

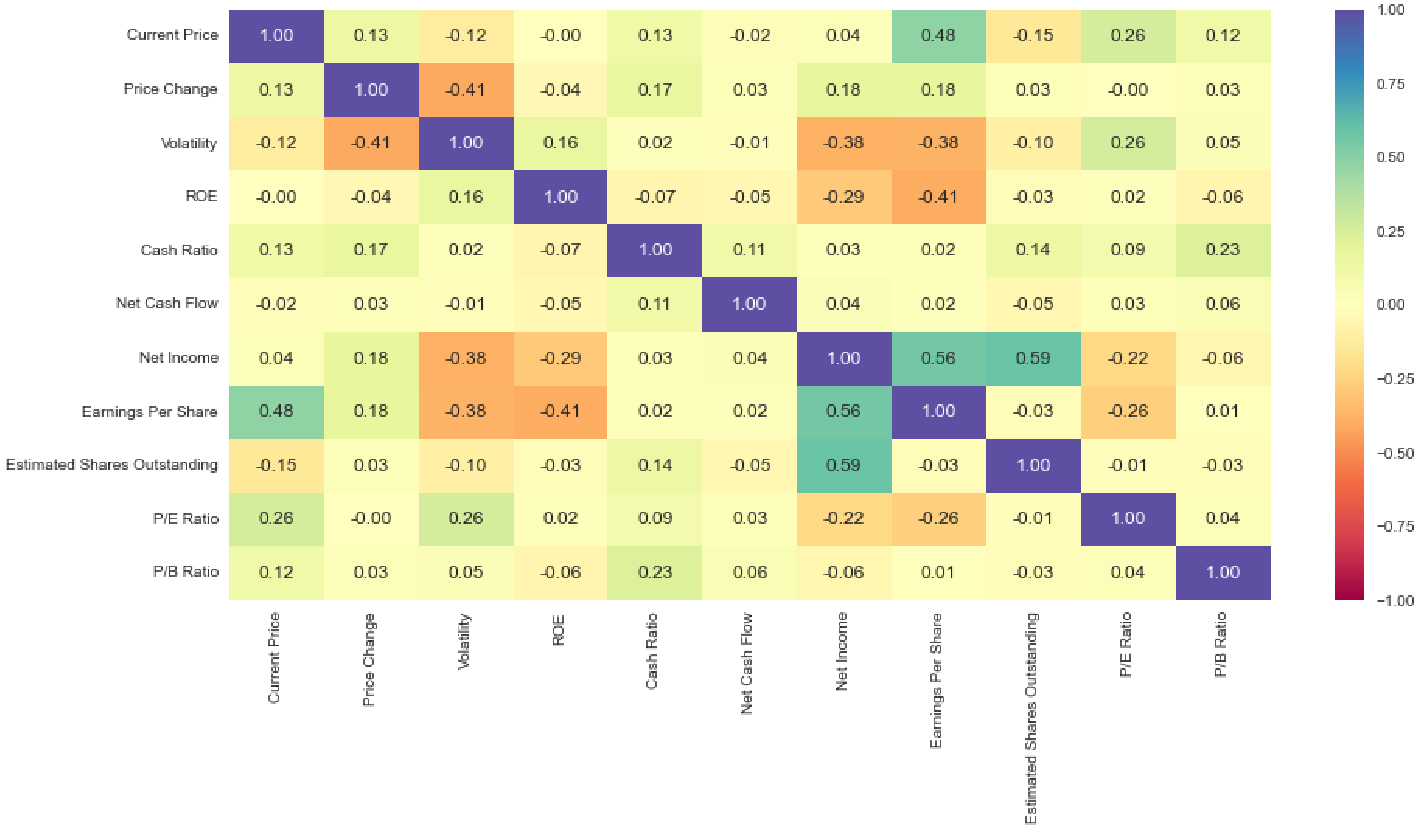
```
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='GICS Sector', y='Price Change', ci=False)   ## Complete the code to choose the right variables
plt.xticks(rotation=90)
plt.show()
```



Q3: How are the different variables correlated with each other?

In [35]:

```
# correlation check
plt.figure(figsize=(15, 7))
sns.heatmap(
    df.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



	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
Current Price	1.00	0.13	-0.12	-0.00	0.13	-0.02	0.04	0.48	-0.15	0.26	0.12
Price Change	0.13	1.00	-0.41	-0.04	0.17	0.03	0.18	0.18	0.03	-0.00	0.03
Volatility	-0.12	-0.41	1.00	0.16	0.02	-0.01	-0.38	-0.38	-0.10	0.26	0.05
ROE	-0.00	-0.04	0.16	1.00	-0.07	-0.05	-0.29	-0.41	-0.03	0.02	-0.06
Cash Ratio	0.13	0.17	0.02	-0.07	1.00	0.11	0.03	0.02	0.14	0.09	0.23
Net Cash Flow	-0.02	0.03	-0.01	-0.05	0.11	1.00	0.04	0.02	-0.05	0.03	0.06
Net Income	0.04	0.18	-0.38	-0.29	0.03	0.04	1.00	0.56	0.59	-0.22	-0.06
Earnings Per Share	0.48	0.18	-0.38	-0.41	0.02	0.02	0.56	1.00	-0.03	-0.26	0.01
Estimated Shares Outstanding	-0.15	0.03	-0.10	-0.03	0.14	-0.05	0.59	-0.03	1.00	-0.01	-0.03
P/E Ratio	0.26	-0.00	0.26	0.02	0.09	0.03	-0.22	-0.26	-0.01	1.00	0.04
P/B Ratio	0.12	0.03	0.05	-0.06	0.23	0.06	-0.06	0.01	-0.03	0.04	1.00

- **Several variables are moderately correlated (+/- .40) with one another**
  - **Volatility is negatively correlated with price change, i.e., as a stock becomes more volatile, its price is likely dropping**
  - **Net income is negatively correlayed with volatility, i.e. as a company generates higher net income its price is likely less volatile**
  - **Net income is also positively correlated with earnings per share (EPS) and estimated shares outstanding**
  - **EPS is positively correlated with current price, i.e. as a company's EPS rises, its prices is also highly likely to increase**
  - **EPS is also negatively correlated with ROE, i.e. as a company generates more equity for shareholders, an equivalent amount of net income the following periods will generate a lower return**

Q4: 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?

In [36]:

```
df.groupby('GICS Sector')['Cash Ratio'].mean().sort_values(ascending=False)
```

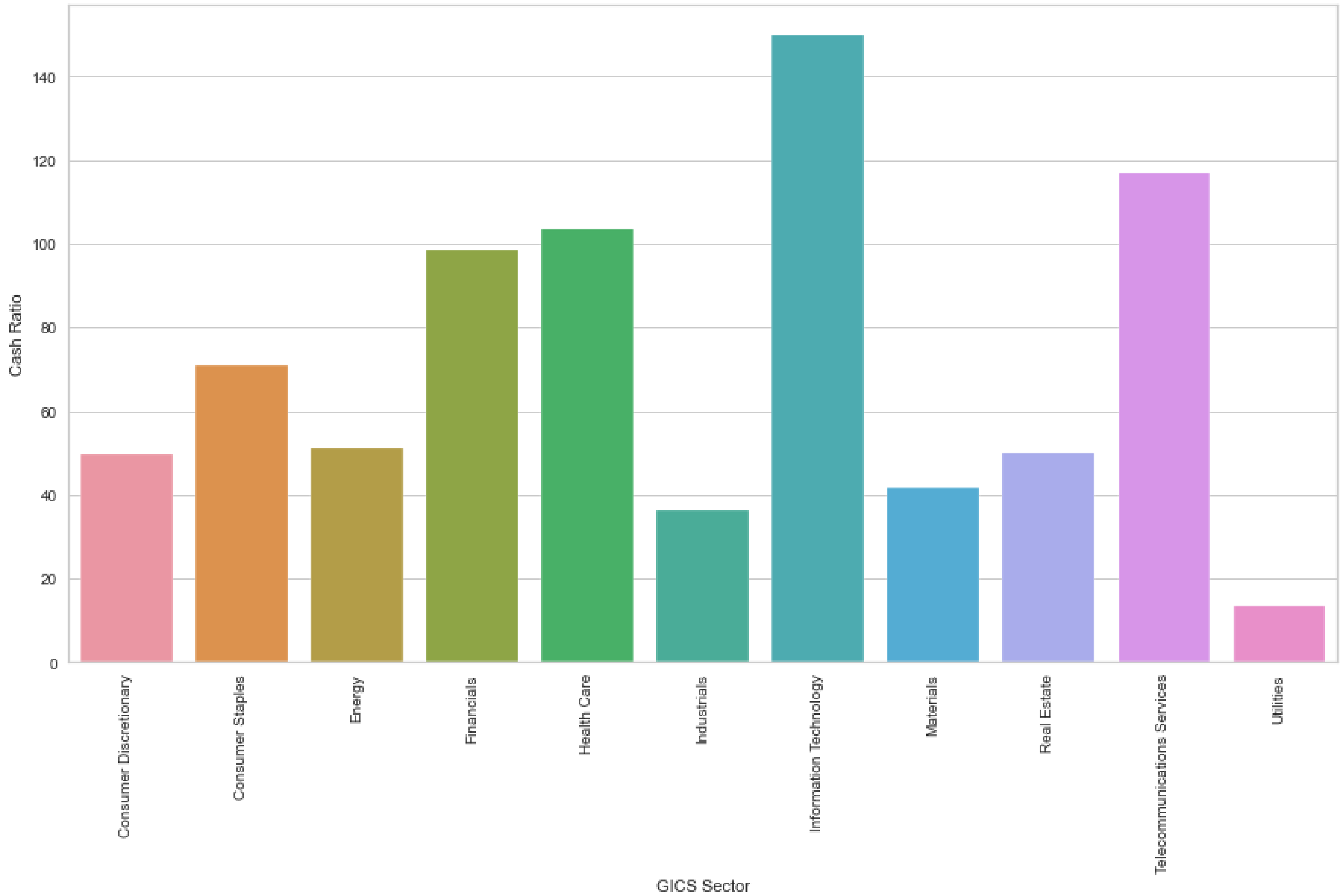
Out[36]:

```
GICS Sector
Information Technology         149.818182
Telecommunications Services    117.000000
Health Care                    103.775000
Financials                      98.591837
Consumer Staples                70.947368
Energy                          51.133333
```

```
Real Estate                  50.111111
Consumer Discretionary       49.575000
Materials                    41.700000
Industrials                  36.188679
Utilities                    13.625000
Name: Cash Ratio, dtype: float64
```

In [37]:

```
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='GICS Sector', y='Cash Ratio', ci=False)  ## Complete the code to choose the right variables
plt.xticks(rotation=90)
plt.show()
```



- **IT and Telecommunications sectors, both relatively newer and unregulated industries, are able to generate significantly higher average cash ratios than their peer sectors**
- **Utilities, a highly regulated industry, generates the lowest average cash ratios of all sectors**

Q5: 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 [38]:

```
df.groupby('GICS Sector')['P/E Ratio'].mean().sort_values(ascending=False)
```

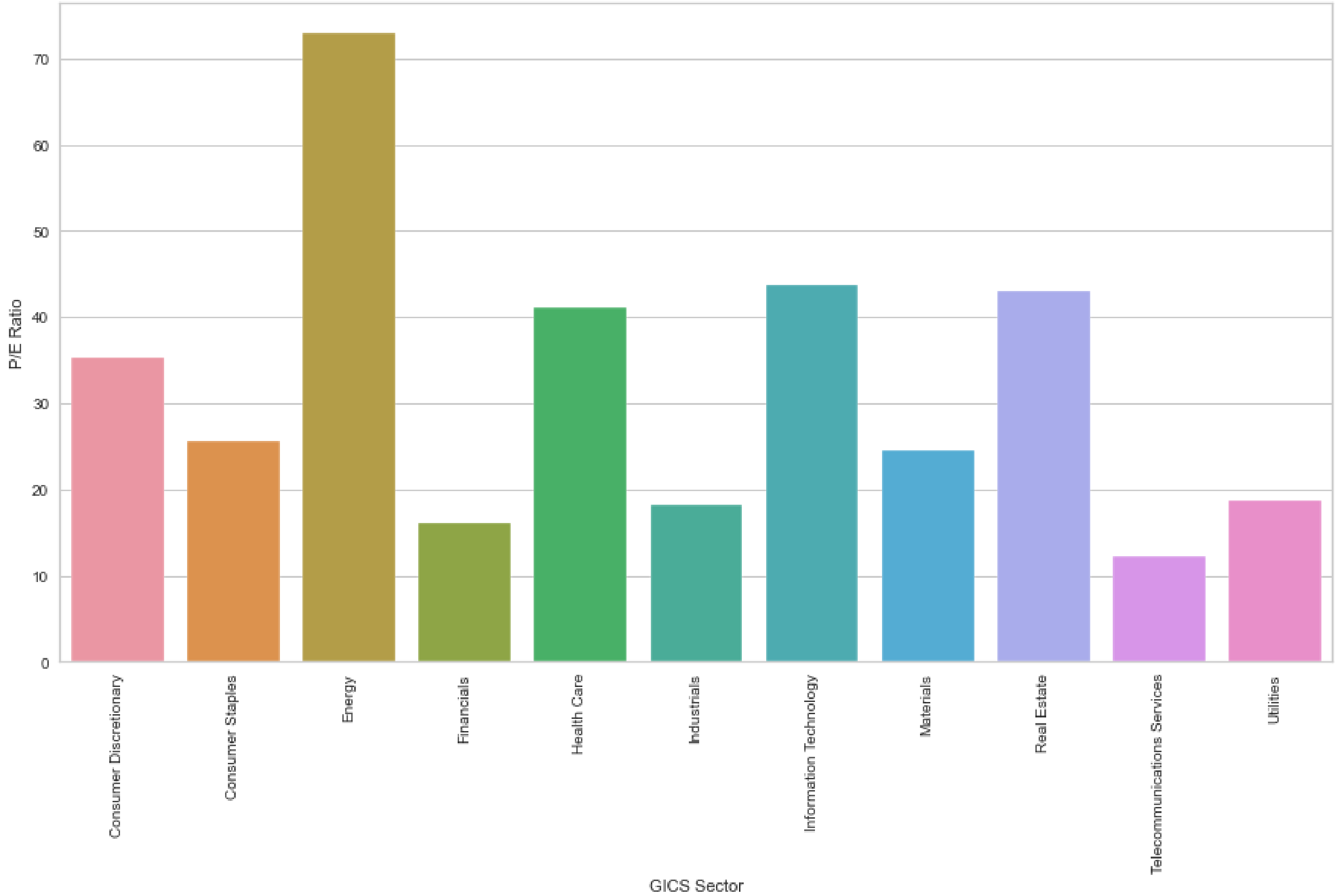
Out[38]:

```
GICS Sector
Energy                      72.897709
Information Technology      43.782546
Real Estate                 43.065585
Health Care                 41.135272
Consumer Discretionary      35.211613
Consumer Staples            25.521195
Materials                   24.585352
```

```
Utilities                         18.719412
Industrials                       18.259380
Financials                        16.023151
Telecommunications Services       12.222578
Name: P/E Ratio, dtype: float64
```

In [39]:

```
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='GICS Sector', y='P/E Ratio', ci=False)  ## Complete the code to choose the right variables
plt.xticks(rotation=90)
plt.show()
```

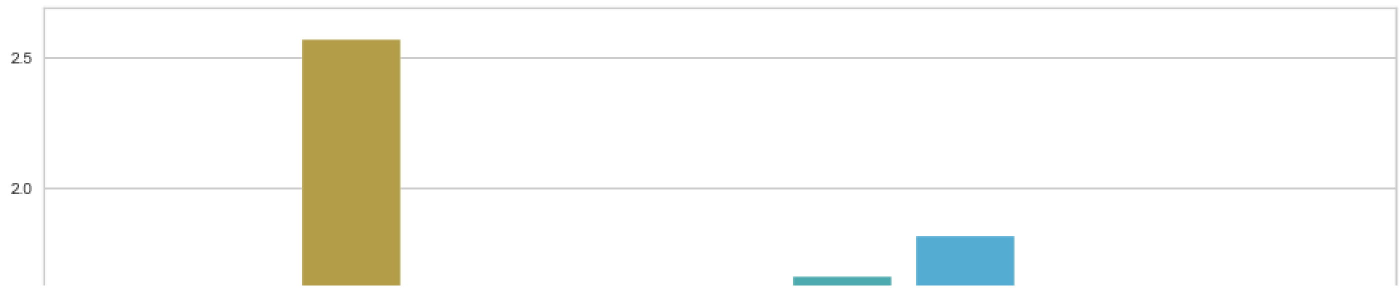


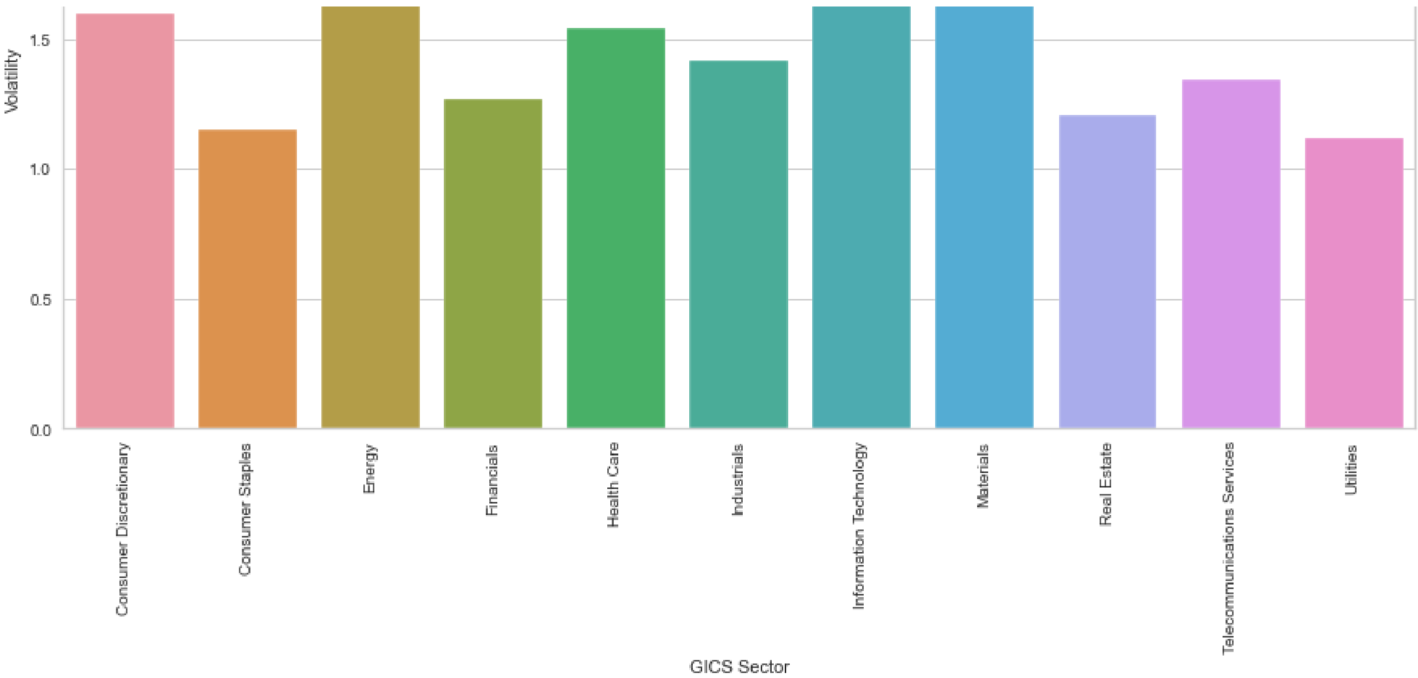
- **Energy companies have the highest average P/E ratios of all sectors by a considerable margin, with telecoms having the lowest average P/E ratios**

Volatility accounts for the fluctuation in the stock price. A stock with high volatility will witness sharper price changes, making it a riskier investment. Let's see how volatility varies, on average, across economic sectors.

In [40]:

```
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='GICS Sector', y='Volatility', ci=False)  ## Complete the code to choose the right variables
plt.xticks(rotation=90)
plt.show()
```





Data Preprocessing

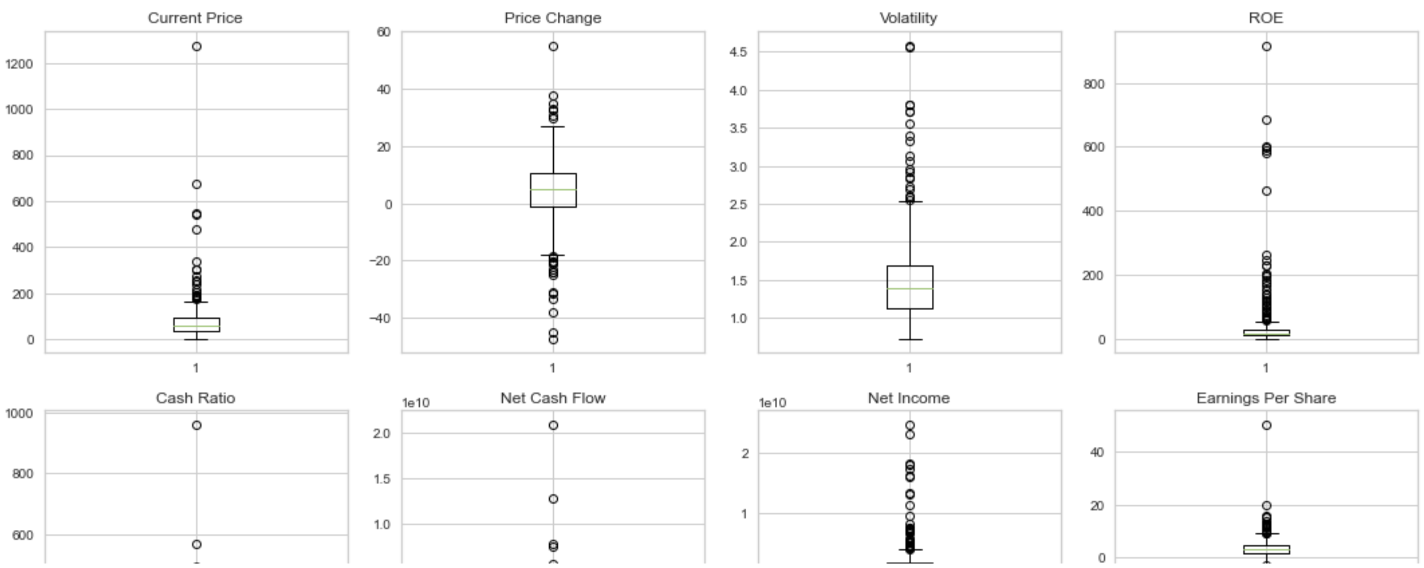
- **Duplicate value check**
- **Missing value treatment**
- **Outlier check**
- **Feature engineering (if needed)**
- **Any other preprocessing steps (if needed)**

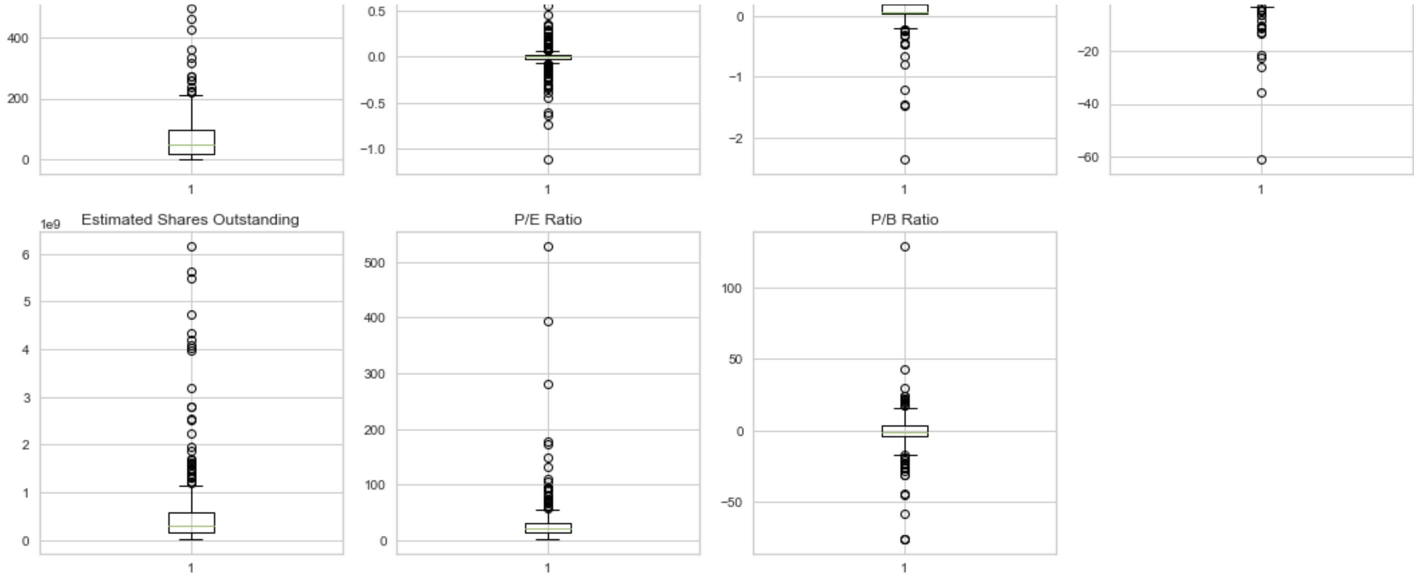
Outlier Check

- **Let's plot the boxplots of all numerical columns to check for outliers.**

In [41]:

```
plt.figure(figsize=(15, 12))

numeric_columns = df.select_dtypes(include=np.number).columns.tolist()

for i, variable in enumerate(numeric_columns):
    plt.subplot(3, 4, i + 1)
    plt.boxplot(df[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```





Scaling

- **Let's scale the data before we proceed with clustering.**

In [42]:

```
#scale the data set before clustering
scaler = StandardScaler()
subset = df[numeric_columns].copy()
subset_scaled = scaler.fit_transform(subset)
```

In [43]:

```
# creating a dataframe of the scaled data
subset_scaled_df = pd.DataFrame(subset_scaled, columns=subset.columns)
```

K-means Clustering

In [44]:

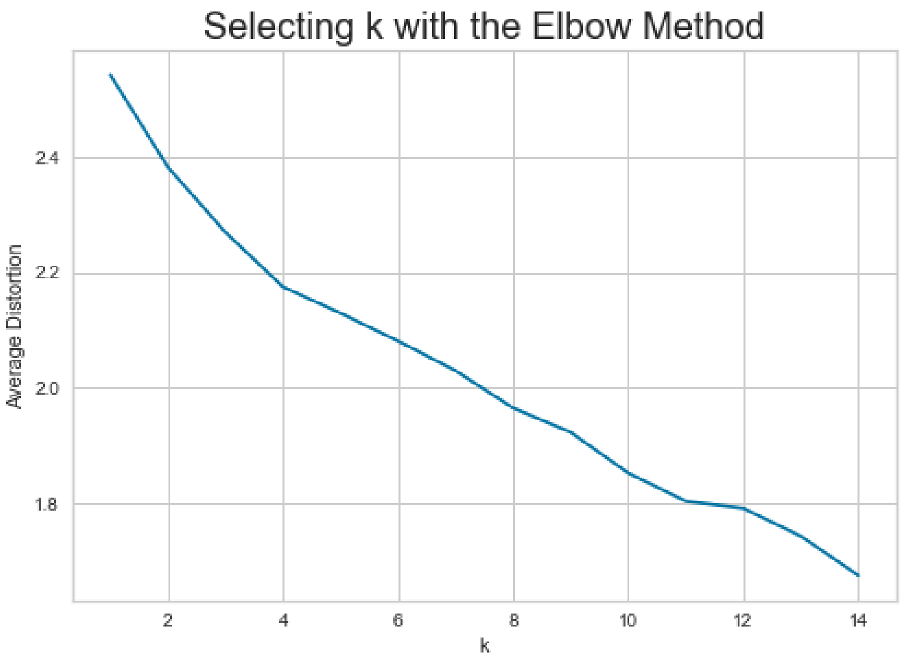
```
k_means_df = subset_scaled_df.copy()
```

In [45]:

```
clusters = range(1, 15)
meanDistortions = []

for k in clusters:
    model = KMeans(n_clusters=k, random_state=1)
    model.fit(subset_scaled_df)
    prediction = model.predict(k_means_df)
    distortion = (
        sum(np.min(cdist(k_means_df, model.cluster_centers_, "euclidean"), axis=1))
        / k_means_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)
plt.show()
```

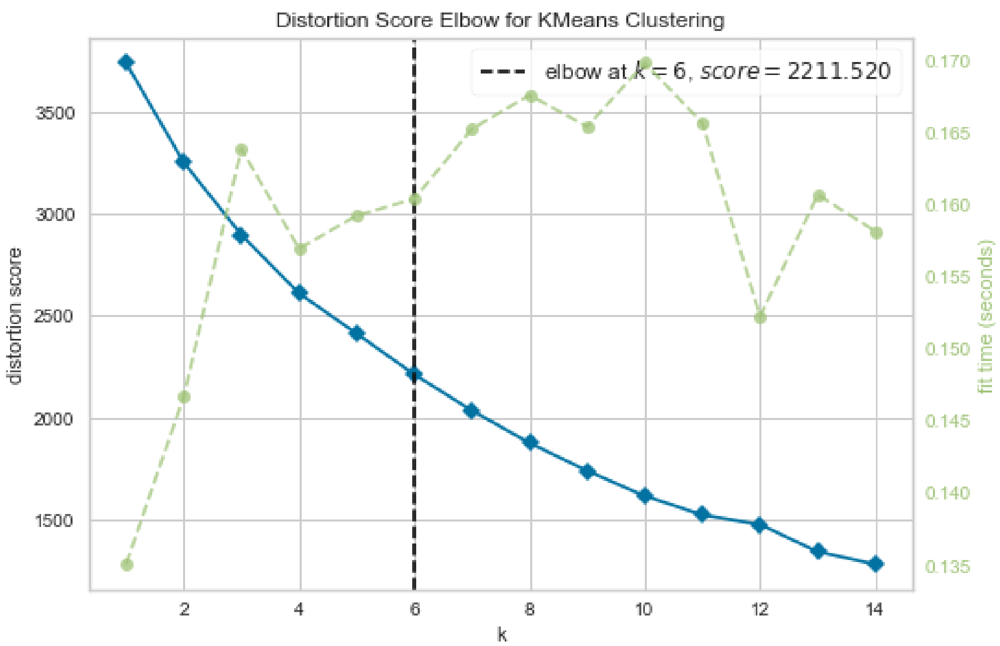
```
Number of Clusters: 1   Average Distortion: 2.5425069919221697
Number of Clusters: 2   Average Distortion: 2.382318498894466
```

```
Number of Clusters: 3  Average Distortion: 2.2692367155390745
Number of Clusters: 4  Average Distortion: 2.1745559827866363
Number of Clusters: 5  Average Distortion: 2.128799332840716
Number of Clusters: 6  Average Distortion: 2.080400099226289
Number of Clusters: 7  Average Distortion: 2.0289794220177395
Number of Clusters: 8  Average Distortion: 1.964144163389972
Number of Clusters: 9  Average Distortion: 1.9221492045198068
Number of Clusters: 10  Average Distortion: 1.8513913649973124
Number of Clusters: 11  Average Distortion: 1.8024134734578485
Number of Clusters: 12  Average Distortion: 1.7900931879652673
Number of Clusters: 13  Average Distortion: 1.7417609203336912
Number of Clusters: 14  Average Distortion: 1.673559857259703
```



In [46]:

```
model = KMeans(random_state=1)
visualizer = KElbowVisualizer(model, k=(1, 15), timings=True)
visualizer.fit(k_means_df)  # fit the data to the visualizer
visualizer.show()  # finalize and render figure
```



Out[46]:

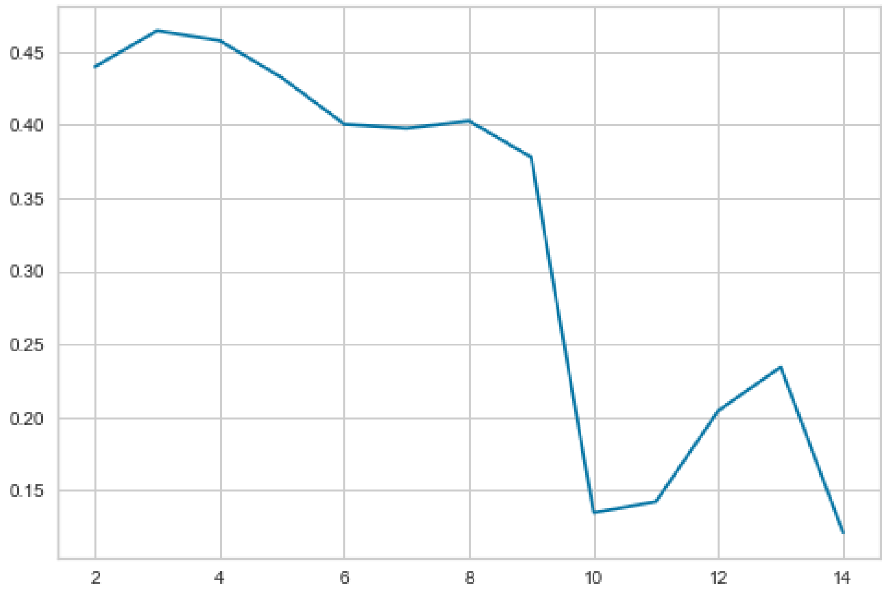
```
<AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>
```

Let's check the silhouette scores.

In [47]:

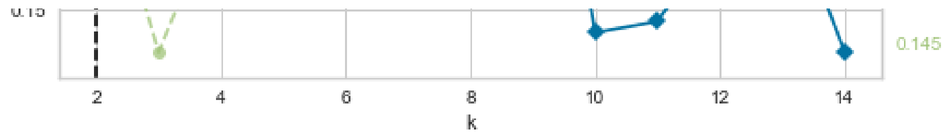
```
sil_score = []
cluster_list = range(2, 15)
for n_clusters in cluster_list:
    clusterer = KMeans(n_clusters=n_clusters, random_state=1)
    preds = clusterer.fit_predict((subset_scaled_df))
    score = silhouette_score(k_means_df, preds)
    sil_score.append(score)
    print("For n_clusters = {}, the silhouette score is {})".format(n_clusters, score))

plt.plot(cluster_list, sil_score)
plt.show()
```

```
For n_clusters = 2, the silhouette score is 0.43969639509980457)
For n_clusters = 3, the silhouette score is 0.4644405674779404)
For n_clusters = 4, the silhouette score is 0.4577225970476733)
For n_clusters = 5, the silhouette score is 0.43228336443659804)
For n_clusters = 6, the silhouette score is 0.4005422737213617)
For n_clusters = 7, the silhouette score is 0.3976335364987305)
For n_clusters = 8, the silhouette score is 0.40278401969450467)
For n_clusters = 9, the silhouette score is 0.3778585981433699)
For n_clusters = 10, the silhouette score is 0.13458938329968687)
For n_clusters = 11, the silhouette score is 0.1421832155528444)
For n_clusters = 12, the silhouette score is 0.2044669621527429)
For n_clusters = 13, the silhouette score is 0.23424874810104204)
For n_clusters = 14, the silhouette score is 0.12102526472829901)
```



In [50]:

```
model = KMeans(random_state=1)
visualizer = KElbowVisualizer(model, k=(2, 15), metric="silhouette", timings=True)
visualizer.fit(k_means_df)  # fit the data to the visualizer
visualizer.show()  # finalize and render figure
```

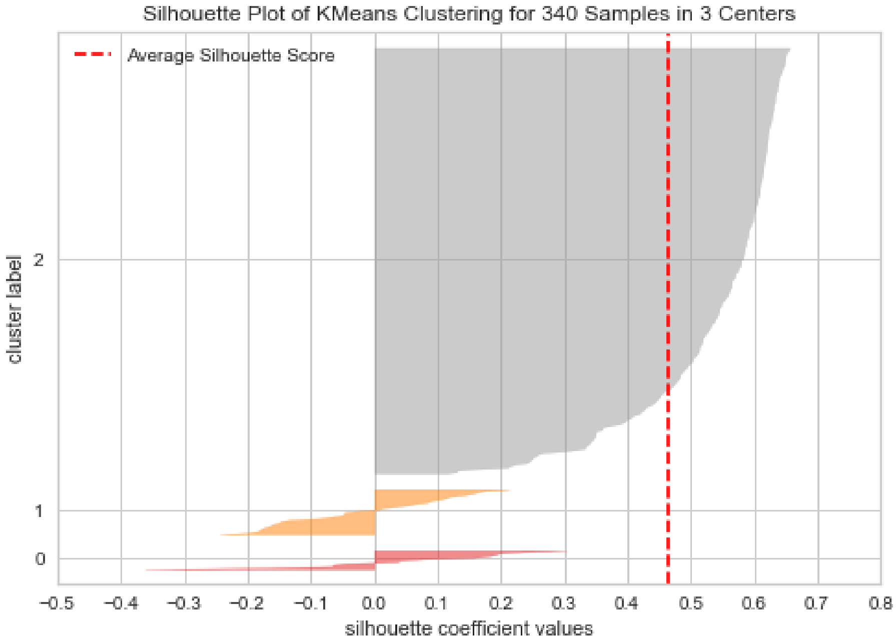


Out[50]:

```
<AxesSubplot:title={'center':'Silhouette Score Elbow for KMeans Clustering'}, xlabel='k',
ylabel='silhouette score'>
```

In [49]:

```
# finding optimal no. of clusters with silhouette coefficients
visualizer = SilhouetteVisualizer(KMeans(3, random_state=1))  ## Complete the code to vi
sualize the silhouette scores for certain number of clusters
visualizer.fit(k_means_df)
visualizer.show()
```



Out[49]:

```
<AxesSubplot:title={'center':'Silhouette Plot of KMeans Clustering for 340 Samples in 3 C
enters'}, xlabel='silhouette coefficient values', ylabel='cluster label'>
```

- **Between the Elbow and Silhouette plots, the number of clusters with the best performance appears to be 3**

In [52]:

```
# final K-means model
kmeans = KMeans(n_clusters=3, random_state=1)
kmeans.fit(k_means_df)
```

Out[52]:

```
▾ KMeans
KMeans(n_clusters=3, random_state=1)
```

In [53]:

```
# creating a copy of the original data
df1 = df.copy()

# adding kmeans cluster labels to the original and scaled dataframes
k_means_df["KM_segments"] = kmeans.labels_
df1["KM_segments"] = kmeans.labels_
```

Cluster Profiles

In [54]:

```
km_cluster_profile = df1.groupby("KM_segments").mean()   ## Complete the code to groupby t
he cluster labels
```

In [55]:

```
km_cluster_profile["count_in_each_segment"] = (
    df1.groupby("KM_segments")["Security"].count().values   ## Complete the code to group
by the cluster labels
)
```

In [56]:

```
km_cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

Out[56]:

KM_segments	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash Flow	Net Income	Earnings Per Share	Est
0	52.142857	6.779993	1.175153	26.142857	140.142857	760285714.285714	13368785714.285715	3.769286	3838
1	64.183438	-10.557046	2.797776	96.531250	70.718750	159171125.000000	-3250005968.750000	-7.886875	526
2	84.045331	5.542488	1.404255	34.040816	66.608844	10698350.340136	1445333183.673469	3.890051	427

In [57]:

```
## Complete the code to print the companies in each cluster
for cl in df1["KM_segments"].unique():
    print("In cluster {}, the following companies are present:".format(cl))
    print(df1[df1["KM_segments"] == cl]["Security"].unique())
    print()
```

```
In cluster 2, the following companies are present:
['American Airlines Group', 'AbbVie', 'Abbott Laboratories', 'Adobe Systems Inc', 'Archer
-Daniels-Midland Co', ..., 'Yahoo Inc.', 'Yum! Brands Inc', 'Zimmer Biomet Holdings', 'Zi
ons Bancorp', 'Zoetis']
Length: 294
Categories (340, object): ['3M Company', 'AFLAC Inc', 'AMETEK Inc', 'AT&T Inc', ..., 'Zim
mer Biomet Holdings', 'Zions Bancorp', 'Zoetis', 'eBay Inc.']

In cluster 1, the following companies are present:
['Analog Devices, Inc.', 'Alexion Pharmaceuticals', 'Amazon.com Inc', 'Apache Corporation
', 'Anadarko Petroleum Corp', ..., 'Southwestern Energy', 'Teradata Corp.', 'Williams Cos
.', 'Wynn Resorts Ltd', 'Cimarex Energy']
Length: 32
Categories (340, object): ['3M Company', 'AFLAC Inc', 'AMETEK Inc', 'AT&T Inc', ..., 'Zim
mer Biomet Holdings', 'Zions Bancorp', 'Zoetis', 'eBay Inc.']

In cluster 0, the following companies are present:
['Bank of America Corp', 'Citigroup Inc.', 'Ford Motor', 'Facebook', 'Gilead Sciences', .
.., 'Pfizer Inc.', 'AT&T Inc', 'Verizon Communications', 'Wells Fargo', 'Exxon Mobil Corp
.']
Length: 14
Categories (340, object): ['3M Company', 'AFLAC Inc', 'AMETEK Inc', 'AT&T Inc', ..., 'Zim
mer Biomet Holdings', 'Zions Bancorp', 'Zoetis', 'eBay Inc.']
```

In [58]:

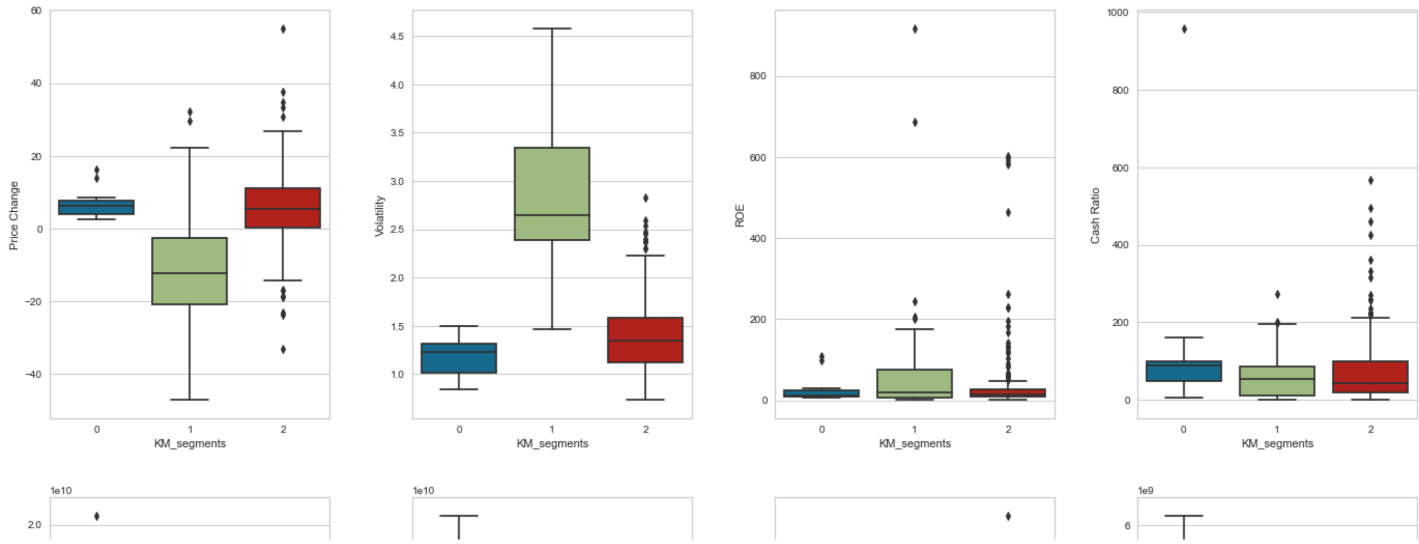
```
df1.groupby(["KM_segments", "GICS Sector"])['Security'].count()
```

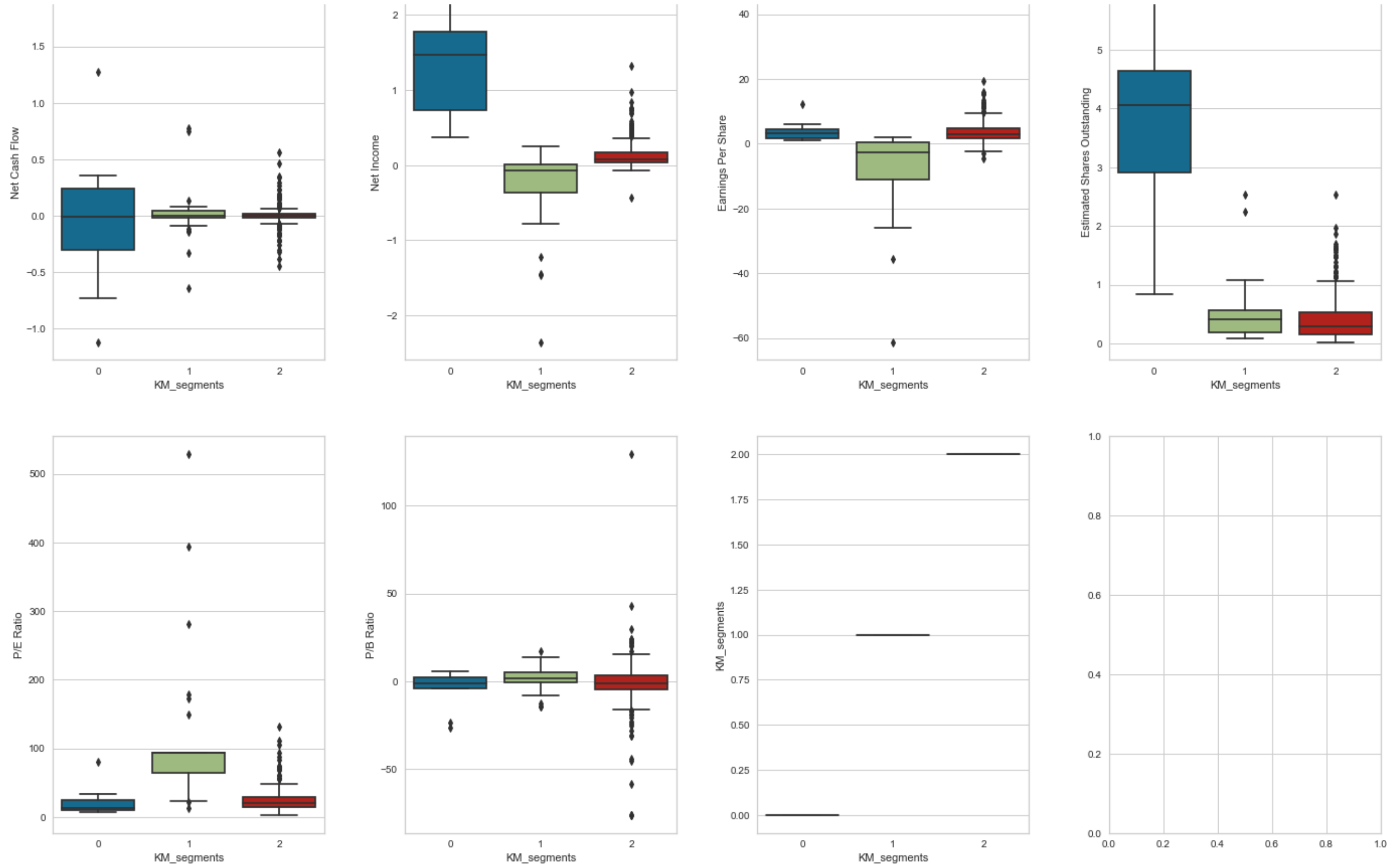
Out[58]:

```
KM_segments  GICS Sector
0            Consumer Discretionary          1
             Consumer Staples                1
             Energy                          1
             Financials                      4
             Health Care                     3
             Industrials                     0
             Information Technology          2
             Materials                       0
             Real Estate                     0
             Telecommunications Services     2
             Utilities                       0
1            Consumer Discretionary          2
             Consumer Staples                0
             Energy                         23
             Financials                      0
             Health Care                     1
             Industrials                     1
             Information Technology          4
             Materials                       1
             Real Estate                     0
             Telecommunications Services     0
             Utilities                       0
2            Consumer Discretionary         37
             Consumer Staples               18
             Energy                          6
             Financials                     45
             Health Care                    36
             Industrials                    52
             Information Technology         27
             Materials                      19
             Real Estate                    27
             Telecommunications Services     3
             Utilities                      24
Name: Security, dtype: int64
```

In [59]:

```python
fig, axes = plt.subplots(3, 4, figsize=(20, 20))
counter = 0

for ii in range(3):
    for jj in range(4):
        if counter < 11:
            sns.boxplot(
                ax=axes[ii][jj],
                data=df1,
                y=df1.columns[4+counter],
                x="KM_segments",
            )
            counter = counter + 1

fig.tight_layout(pad=3.0)
```





KMeans Clusters

Cluster 0 - Large Market Capitalization / Dow Jones Industrial Average

- **14 stocks, comprised mostly of stocks within the Financials, Health Care, Information Technology (IT), Telecommunications services and Consumer Discretionary sectors**
- **Companies within this cluster have:**
  - **Low volatility**
  - **Most of the companies with the highest outflows of cash**
  - **The highest net incomes**
  - **The highest number of shares outstanding**

Cluster 1

- **32 stocks, comprised mostly of stocks within the Energy, IT, Materials, Health caare, Industrials and Consumer Discretionary sectors**
- **Companies within this cluster have:**
  - **Highest volatility**
  - **Low earnings per share**
  - **Most of the P/B ratio**

Cluster 2 - S&P 500 / Diversification

- **284 stocks (~84% of all stocks in the dataset) drawn from all sectors present in the dataset**
- **Companies within this cluster have:**
  - **Low P/E ratios**
  - **Most of the outliers on negative P/B ratios**

Hierarchical Clustering

In [60]:

```
hc_df = subset_scaled_df.copy()
```

In [61]:

```
# list of distance metrics
distance_metrics = ["euclidean", "chebyshev", "mahalanobis", "cityblock"] ## Complete th
e code to add distance metrics

# list of linkage methods
linkage_methods = ["single", "complete", "average", "weighted"] ## Complete the code to a
dd linkages

high_cophenet_corr = 0
high_dm_lm = [0, 0]

for dm in distance_metrics:
    for lm in linkage_methods:
        Z = linkage(hc_df, metric=dm, method=lm)
        c, coph_dists = cophenet(Z, pdist(hc_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

# printing the combination of distance metric and linkage method with the highest cophene
tic correlation
print('*'*100)
print(
    "Highest cophenetic correlation is {}, which is obtained with {} distance and {} link
age.".format(
        high_cophenet_corr, high_dm_lm[0].capitalize(), high_dm_lm[1]
    )
)
```

```
Cophenetic correlation for Euclidean distance and single linkage is 0.9232271494002922.
Cophenetic correlation for Euclidean distance and complete linkage is 0.7873280186580672.
Cophenetic correlation for Euclidean distance and average linkage is 0.9422540609560814.
Cophenetic correlation for Euclidean distance and weighted linkage is 0.8693784298129404.
Cophenetic correlation for Chebyshev distance and single linkage is 0.9062538164750717.
Cophenetic correlation for Chebyshev distance and complete linkage is 0.598891419111242.
Cophenetic correlation for Chebyshev distance and average linkage is 0.9338265528030499.
Cophenetic correlation for Chebyshev distance and weighted linkage is 0.9127355892367.
Cophenetic correlation for Mahalanobis distance and single linkage is 0.925919553052459.
Cophenetic correlation for Mahalanobis distance and complete linkage is 0.792530720285.
Cophenetic correlation for Mahalanobis distance and average linkage is 0.9247324030159737
.
Cophenetic correlation for Mahalanobis distance and weighted linkage is 0.870831749018042
6.
Cophenetic correlation for Cityblock distance and single linkage is 0.9334186366528574.
Cophenetic correlation for Cityblock distance and complete linkage is 0.7375328863205818.
Cophenetic correlation for Cityblock distance and average linkage is 0.9302145048594667.
Cophenetic correlation for Cityblock distance and weighted linkage is 0.731045513520281.
*********************************************************************************************
***********
Highest cophenetic correlation is 0.9422540609560814, which is obtained with Euclidean di
stance and average linkage.
```

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

In [62]:

```
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighted"] ## C
omplete the code to add linkages

high_cophenet_corr = 0
```

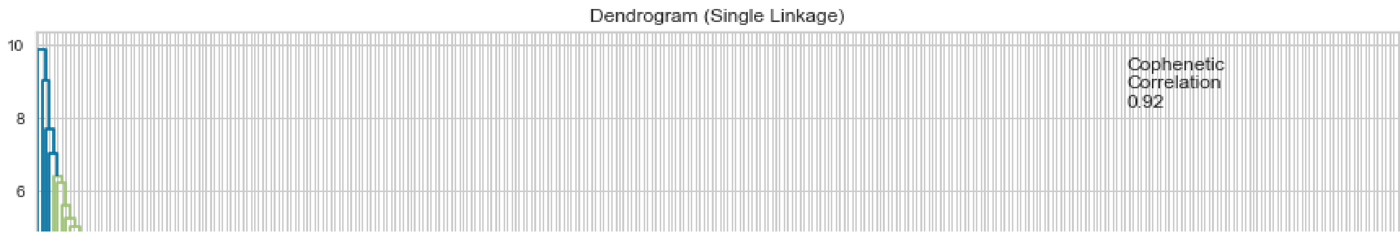
```
high_dm_lm = [0, 0]

for lm in linkage_methods:
    Z = linkage(hc_df, metric="euclidean", method=lm)
    c, coph_dists = cophenet(Z, pdist(hc_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

# printing the combination of distance metric and linkage method with the highest cophene
tic correlation
print('*'*100)
print(
    "Highest cophenetic correlation is {}, which is obtained with {} linkage.".format(
        high_cophenet_corr, high_dm_lm[1]
    )
)
```

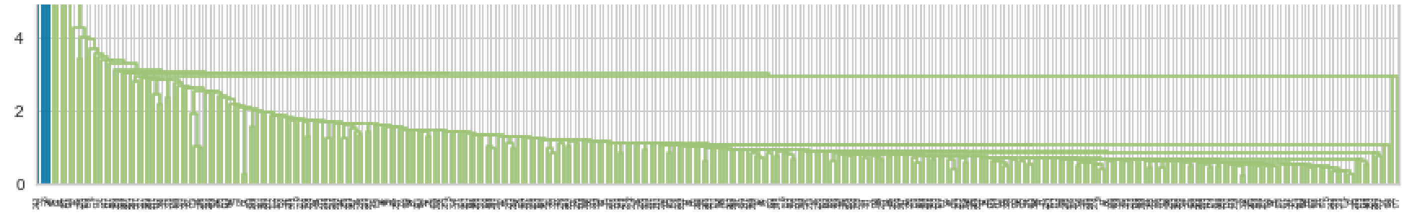
```
Cophenetic correlation for single linkage is 0.9232271494002922.
Cophenetic correlation for complete linkage is 0.7873280186580672.
Cophenetic correlation for average linkage is 0.9422540609560814.
Cophenetic correlation for centroid linkage is 0.9314012446828154.
Cophenetic correlation for ward linkage is 0.7101180299865353.
Cophenetic correlation for weighted linkage is 0.8693784298129404.
*****************************************************************************************
***********
Highest cophenetic correlation is 0.9422540609560814, which is obtained with average link
age.
```

Let's view the dendrograms for the different linkage methods with Euclidean distance.

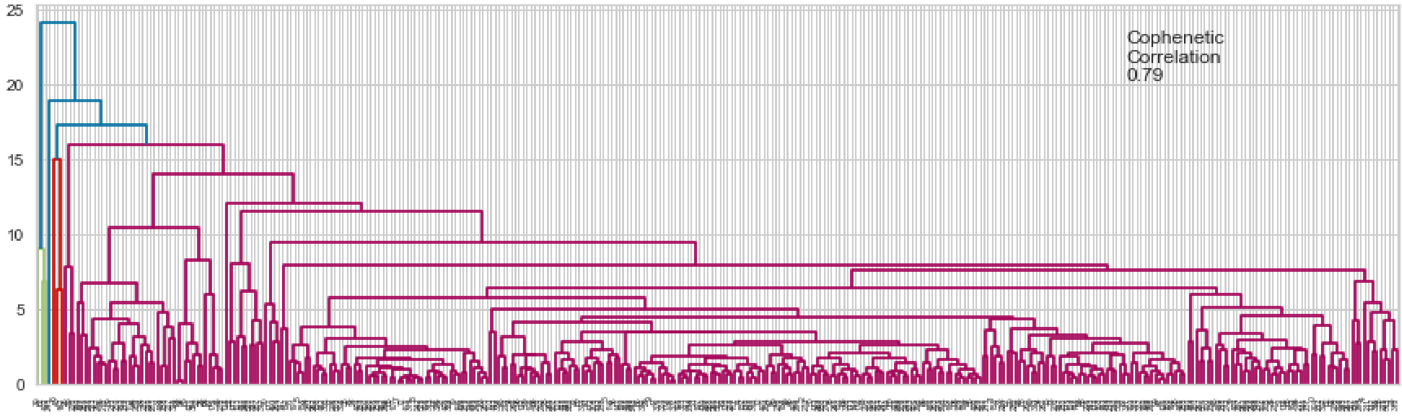
In [63]:

```
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighted"] ## C
omplete the code to add linkages

# 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 corre
lation
for i, method in enumerate(linkage_methods):
    Z = linkage(hc_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(hc_df))
    axs[i].annotate(
        f"Cophenetic\nCorrelation\n{coph_corr:0.2f}",
        (0.80, 0.80),
        xycoords="axes fraction",
    )

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

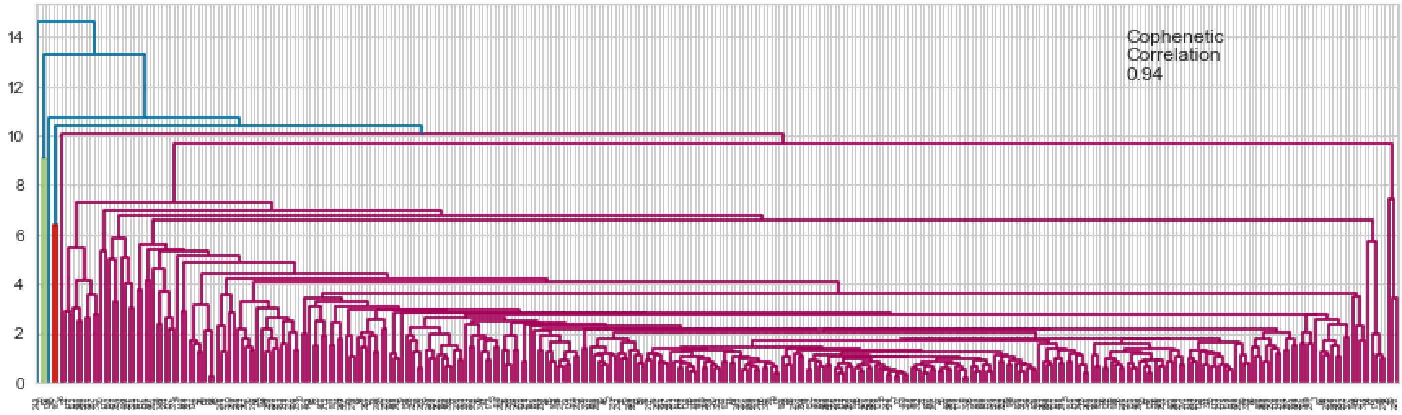




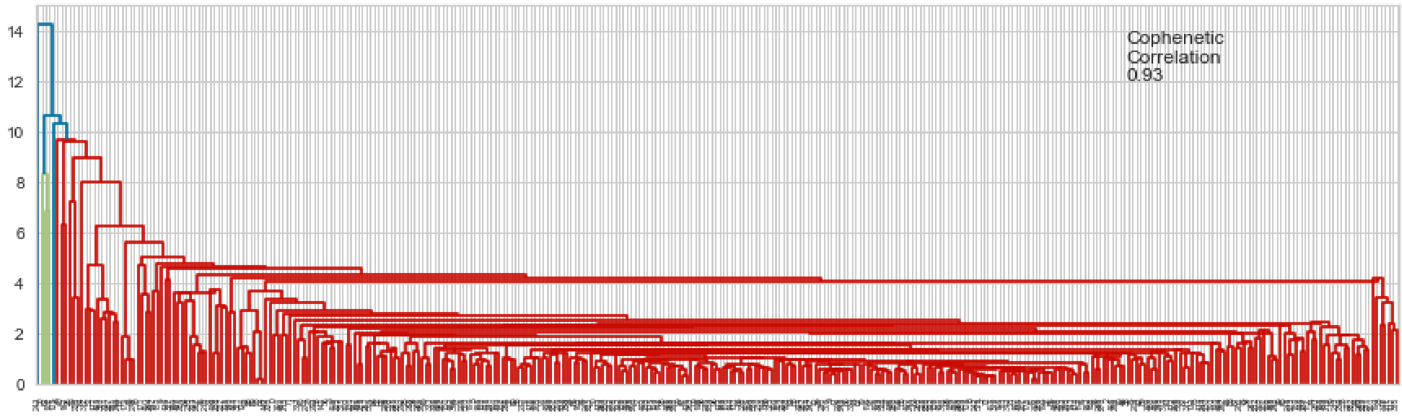
Dendrogram (Complete Linkage)



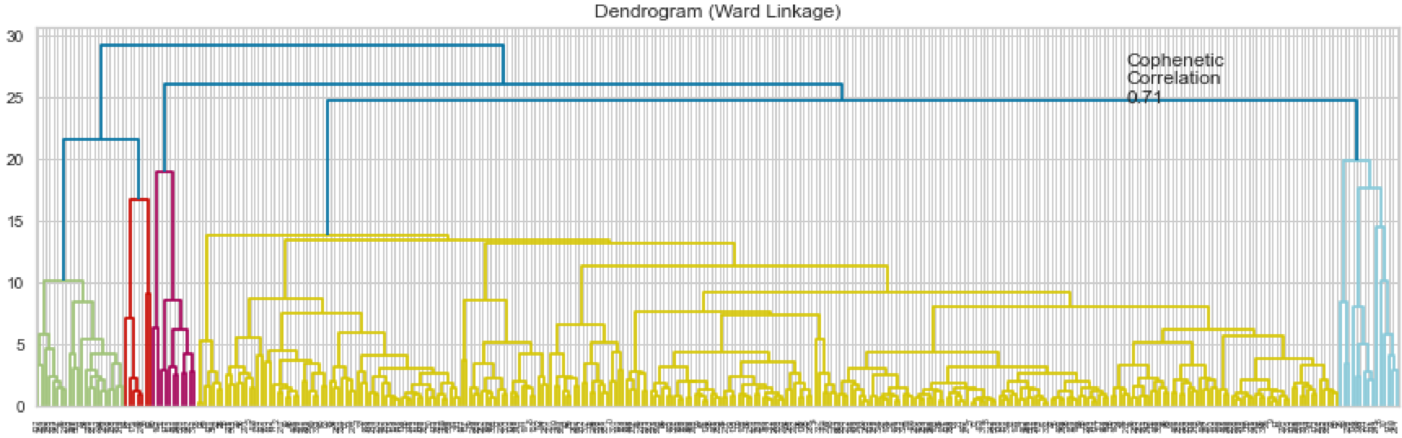
Dendrogram (Average Linkage)



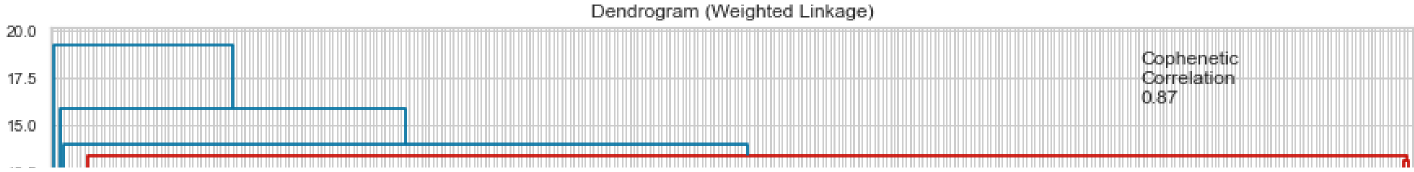
Dendrogram (Centroid Linkage)

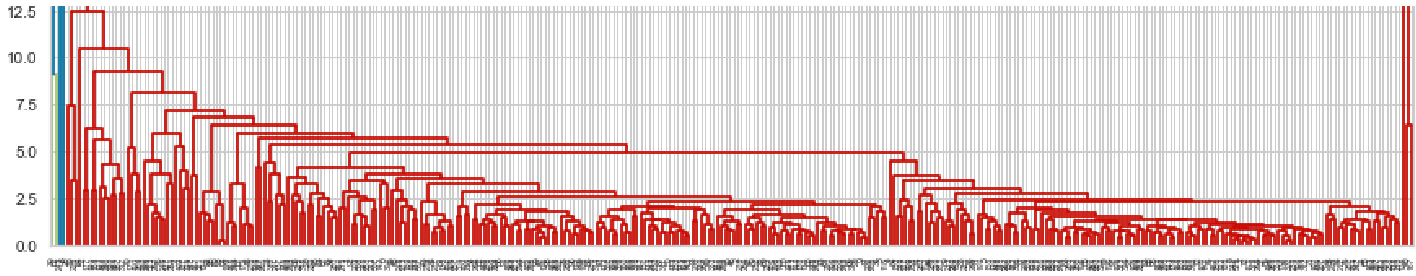


Dendrogram (Ward Linkage)



Dendrogram (Weighted Linkage)





- **The cophenetic correlation is highest for average and centroid linkage methods, but the dendrogram for average appears to provide better clusters**
- **5 appears to be the appropriate number of clusters for the average linkage method**

In [64]:

```
# create and print a dataframe to compare cophenetic correlations for different linkage m
ethods
df_cc = pd.DataFrame(compare, columns=compare_cols)
df_cc = df_cc.sort_values(by="Cophenetic Coefficient")
df_cc
```

Out[64]:

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

In [72]:

```
HCmodel = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='average')
## Complete the code to define the hierarchical clustering model
HCmodel.fit(hc_df)
```

Out[72]:

```
▾ AgglomerativeClustering
AgglomerativeClustering(linkage='average', n_clusters=3)
```

In [73]:

```
# creating a copy of the original data
df2 = df.copy()

# adding hierarchical cluster labels to the original and scaled dataframes
hc_df["HC_segments"] = HCmodel.labels_
df2["HC_segments"] = HCmodel.labels_
```

Cluster Profiling

In [74]:

```
hc_cluster_profile = df2.groupby("HC_segments").mean()  ## Complete the code to groupby t
he cluster labels
```

In [75]:

```
hc_cluster_profile["count_in_each_segment"] = (
    df2.groupby("HC_segments")["Security"].count().values  ## Complete the code to group
```

```
by the cluster labels
)
```

In [76]:

```
hc_cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

Out[76]:

HC_segments	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash Flow	Net Income	Earnings Per Share
0	77.653642	4.184271	1.515129	35.103858	69.798220	68662246.290801	1613508620.178041	2.900905
1	1274.949951	3.190527	1.268340	29.000000	184.000000	-1671386000.000000	2551360000.000000	50.090000
2	24.485001	-13.351992	3.482611	802.000000	51.000000	-1292500000.000000	-19106500000.000000	-41.815000

In [77]:

```
## Complete the code to print the companies in each cluster
for cl in df2["HC_segments"].unique():
    print("In cluster {}, the following companies are present:".format(cl))
    print(df2[df2["HC_segments"] == cl]["Security"].unique())
    print()
```

```
In cluster 0, the following companies are present:
['American Airlines Group', 'AbbVie', 'Abbott Laboratories', 'Adobe Systems Inc', 'Analog
Devices, Inc.', ..., 'Yahoo Inc.', 'Yum! Brands Inc', 'Zimmer Biomet Holdings', 'Zions Ba
ncorp', 'Zoetis']
Length: 337
Categories (340, object): ['3M Company', 'AFLAC Inc', 'AMETEK Inc', 'AT&T Inc', ..., 'Zim
mer Biomet Holdings', 'Zions Bancorp', 'Zoetis', 'eBay Inc.']

In cluster 2, the following companies are present:
['Apache Corporation', 'Chesapeake Energy']
Categories (340, object): ['3M Company', 'AFLAC Inc', 'AMETEK Inc', 'AT&T Inc', ..., 'Zim
mer Biomet Holdings', 'Zions Bancorp', 'Zoetis', 'eBay Inc.']

In cluster 1, the following companies are present:
['Priceline.com Inc']
Categories (340, object): ['3M Company', 'AFLAC Inc', 'AMETEK Inc', 'AT&T Inc', ..., 'Zim
mer Biomet Holdings', 'Zions Bancorp', 'Zoetis', 'eBay Inc.']
```

In [78]:

```
df2.groupby(["HC_segments", "GICS Sector"])['Security'].count()
```

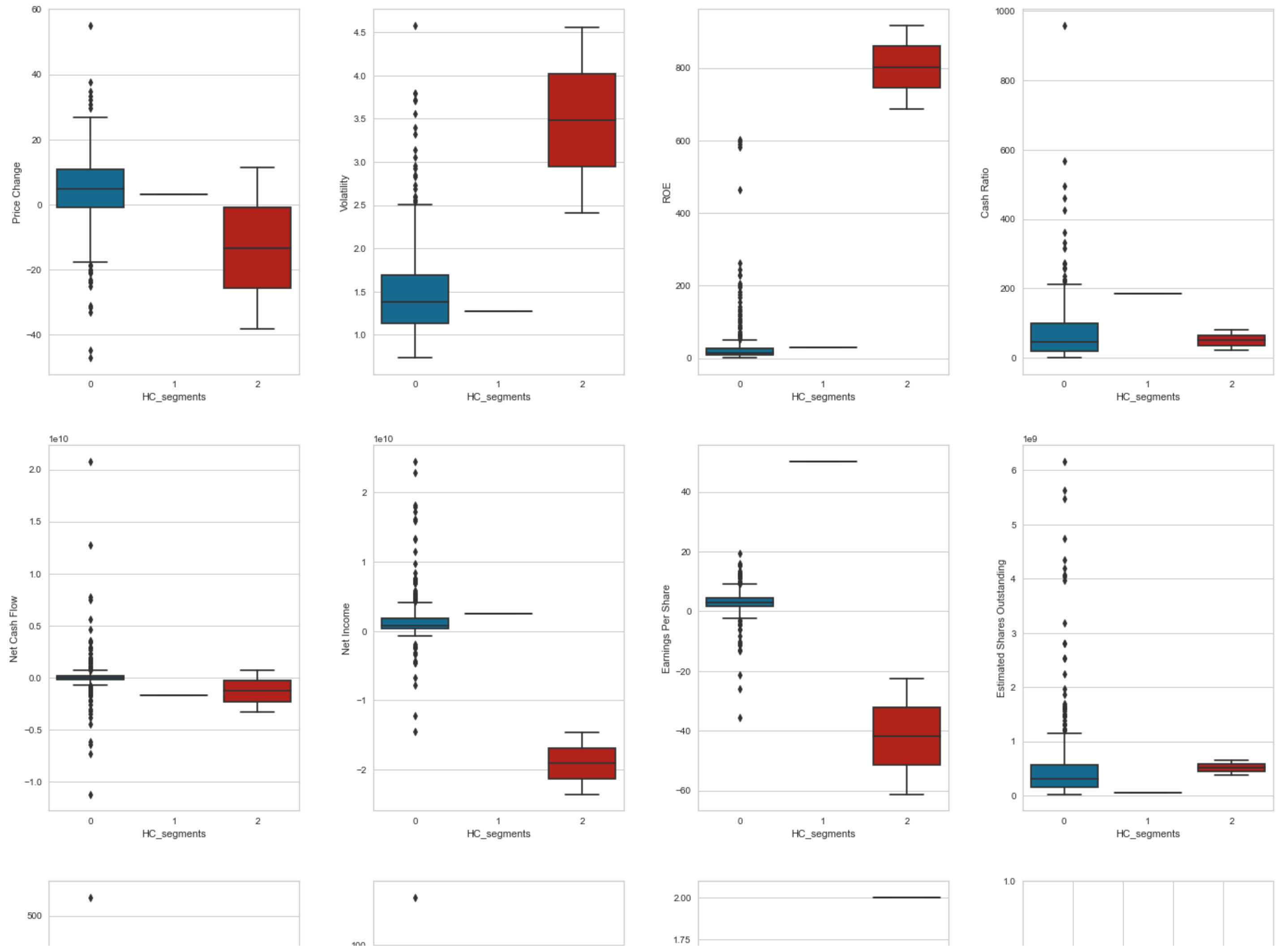
Out[78]:

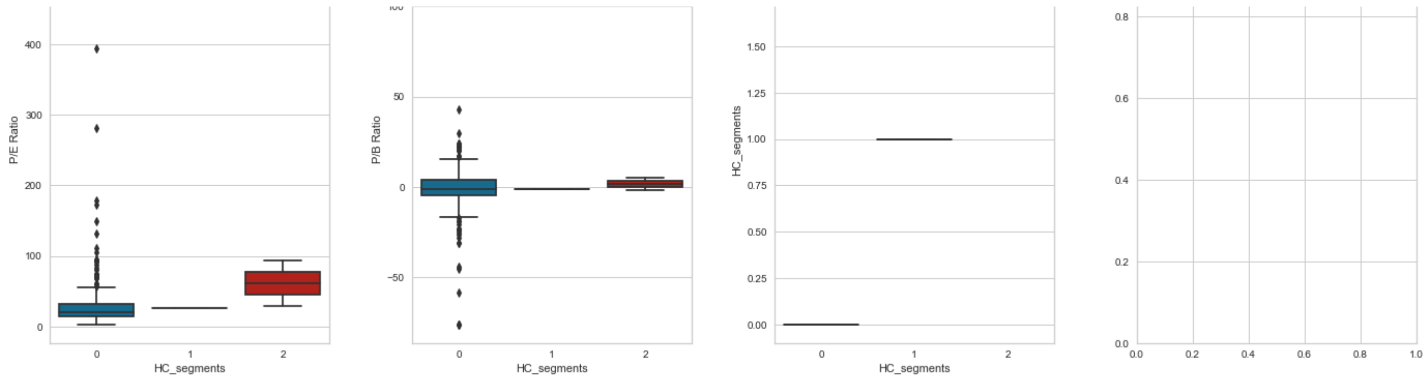
```
HC_segments  GICS Sector
0            Consumer Discretionary         39
             Consumer Staples               19
             Energy                         28
             Financials                     49
             Health Care                    40
             Industrials                    53
             Information Technology         33
             Materials                      20
             Real Estate                    27
             Telecommunications Services     5
             Utilities                      24
1            Consumer Discretionary          1
             Consumer Staples                0
             Energy                          0
             Financials                      0
             Health Care                     0
```

```
               Health Care                         0
               Industrials                         0
               Information Technology              0
               Materials                           0
               Real Estate                         0
               Telecommunications Services         0
               Utilities                           0
2              Consumer Discretionary              0
               Consumer Staples                    0
               Energy                              2
               Financials                          0
               Health Care                         0
               Industrials                         0
               Information Technology              0
               Materials                           0
               Real Estate                         0
               Telecommunications Services         0
               Utilities                           0
Name: Security, dtype: int64
```

In [79]:

```python
fig, axes = plt.subplots(3, 4, figsize=(20, 20))
counter = 0

for ii in range(3):
    for jj in range(4):
        if counter < 11:
            sns.boxplot(
                ax=axes[ii][jj],
                data=df2,
                y=df2.columns[4+counter],
                x="HC_segments",
            )
            counter = counter + 1

fig.tight_layout(pad=3.0)
```





Hierarchical Clusters

Cluster 0

- 337 stocks, (~99% of all stocks in the dataset) drawn from all sectors present in the dataset
  - Companies within this cluster have:
  - Most of stocks with the high price change
  - Significant outliers in Estimated shares outstanding
  - Low volartility

Cluster 1

- 1 stock, comprised pf Consumer Discretionary
- Companies within this cluster have:
  - Mostly negative net cashflow

Cluster 2

- only 2 stocks, comprised mostly of stocks within the energy sector
  - Companies within this cluster have:
    - High Volatatility
    - Negative Price change and earnings per share

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 ~0.1s

Which clustering technique gave you more distinct clusters, or are they the same? How many observations are there in the similar clusters of both algorithms?

- Both algorithms give similar clusters, with a single cluster of a majority of the stocks and the remaining four clusters containing 7-29 stocks

How many clusters are obtained as the appropriate number of clusters from both algorithms?

- For both algorithms, 3 clusters provided distinct clusters with sufficient observations in each to reasonably differentiate which "type" of stock is representative of the cluster

Differences or similarities in the cluster profiles from both the clustering techniques

- Both algorithms yielded similar clusters based on the outliers within the 11 variables

Actionable Insights and Recommendations

- Trade&Ahead should first identify the financial goals, risk tolerance, and investment behaviors of their clilents, then recommend a cluster as a potential portfolio of stocks which will fit these needs

- However, many of these clusters, based on the characteristics of the stocks within them, are essentially substitutes for standard indexes, such as the Dow Jones Industrial Average and the S&P 500, which could more easily achieve these goals
- Alternatively, Trade&Ahead could use these clusters as an starting point for further financial statement analysis, particularly which individual stocks do not fit the "profile" of the cluster
  - Assuming selecting individual stocks is a component of a client's investment strategy, Trade&Ahead may then be able to identify stocks which should outperform its peers (i.e., price will rise = buy recommendation) or likely fall behind its peers (i.e., price will fall = sell recommendation)