Unsupervised Learning: Trade & Ahead

Problem Statement

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

```
# Installing the libraries with the specified version.
# uncomment and run the following line if Google Colab is being used
#!pip install scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 pandas==1.5.3 yellowbrick==1.5 -q --user
```

```
# Installing the libraries with the specified version.
# uncomment and run the following lines if Jupyter Notebook is being used
#!pip install scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 pandas==1.5.2 yellowbrick==1.5 -q --user
#!pip install --upgrade -q jinja2
```

Note: *After running the above cell, kindly restart the notebook kernel and run all cells sequentially from the start again.*

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

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

Data Overview

- Observations
- Sanity checks

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# original data
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/7 - Unsupervised Learning/Final Project/stock_data.csv')
```

```
data.head()
```

	Ticker Symbol	Security	GICS Sector	GICS Sub Industry	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
0	AAL	American Airlines Group	Industrials	Airlines	42.349998	9.999995	1.687151	135	51	-604000000	7610000000	11.39	6.681299e+08	3.718174	-8.784219
1	ABBV	AbbVie	Health Care	Pharmaceuticals	59.240002	8.339433	2.197887	130	77	51000000	5144000000	3.15	1.633016e+09	18.806350	-8.750068
2	ABT	Abbott Laboratories	Health Care	Health Care Equipment	44.910000	11.301121	1.273646	21	67	938000000	4423000000	2.94	1.504422e+09	15.275510	-0.394171
3	ADBE	Adobe Systems Inc	Information Technology	Application Software	93.940002	13.977195	1.357679	9	180	-240840000	629551000	1.26	4.996437e+08	74.555557	4.199651

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.shape
```

```
(340, 15)
```

Observations - There are 340 rows and 15 columns in the dataset

```
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
```

Observations - There are 11 numerical (4 int64 & 7 float64) and 4 objects types in the data dataset

```
data.describe()
```

	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
count	340.000000	340.000000	340.000000	340.000000	340.000000	3.400000e+02	3.400000e+02	340.000000	3.400000e+02	340.000000	340.000000
mean	80.862345	4.078194	1.525976	39.597059	70.023529	5.553762e+07	1.494385e+09	2.776662	5.770283e+08	32.612563	-1.718249
std	98.055086	12.006338	0.591798	96.547538	90.421331	1.946365e+09	3.940150e+09	6.587779	8.458496e+08	44.348731	13.966912
min	4.500000	-47.129693	0.733163	1.000000	0.000000	-1.120800e+10	-2.352800e+10	-61.200000	2.767216e+07	2.935451	-76.119077
25%	38.555000	-0.939484	1.134878	9.750000	18.000000	-1.939065e+08	3.523012e+08	1.557500	1.588482e+08	15.044653	-4.352056
50%	59.705000	4.819505	1.385593	15.000000	47.000000	2.098000e+06	7.073360e+08	2.895000	3.096751e+08	20.819876	-1.067170

```
data.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Ticker Symbol	340	340	AAL	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Security	340	340	American Airlines Group	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
GICS Sector	340	11	Industrials	53	NaN	NaN	NaN	NaN	NaN	NaN	NaN
GICS Sub Industry	340	104	Oil & Gas Exploration & Production	16	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Current Price	340.0	NaN	NaN	NaN	80.862345	98.055086	4.5	38.555	59.705	92.880001	1274.949951
Price Change	340.0	NaN	NaN	NaN	4.078194	12.006338	-47.129693	-0.939484	4.819505	10.695493	55.051683
Volatility	340.0	NaN	NaN	NaN	1.525976	0.591798	0.733163	1.134878	1.385593	1.695549	4.580042
ROE	340.0	NaN	NaN	NaN	39.597059	96.547538	1.0	9.75	15.0	27.0	917.0
Cash Ratio	340.0	NaN	NaN	NaN	70.023529	90.421331	0.0	18.0	47.0	99.0	958.0
Net Cash Flow	340.0	NaN	NaN	NaN	55537620.588235	1946365312.175789	-11208000000.0	-193906500.0	2098000.0	169810750.0	20764000000.0
Net Income	340.0	NaN	NaN	NaN	1494384602.941176	3940150279.327937	-23528000000.0	352301250.0	707336000.0	1899000000.0	24442000000.0
Earnings Per Share	340.0	NaN	NaN	NaN	2.776662	6.587779	-61.2	1.5575	2.895	4.62	50.09
Estimated Shares Outstanding	340.0	NaN	NaN	NaN	577028337.75403	845849595.417695	27672156.86	158848216.1	309675137.8	573117457.325	6159292035.0
P/E Ratio	340.0	NaN	NaN	NaN	32.612563	44.348731	2.935451	15.044653	20.819876	31.764755	528.039074

```
# ccreate a copy from the original data
df = data.copy()
```

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?

```
# Support functions

# 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



# 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=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        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


# 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,
        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
```

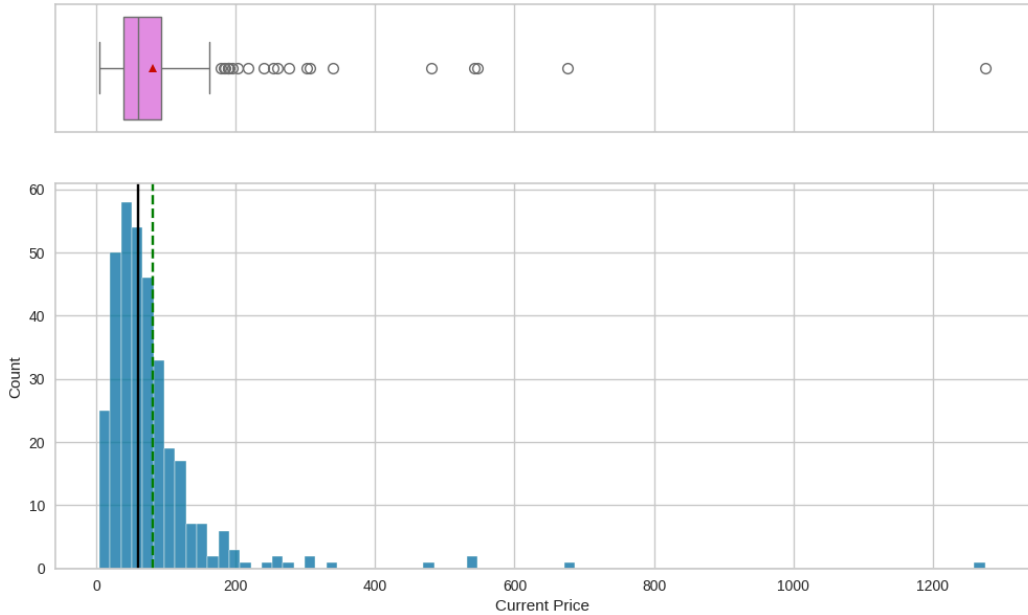
```
        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        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
```

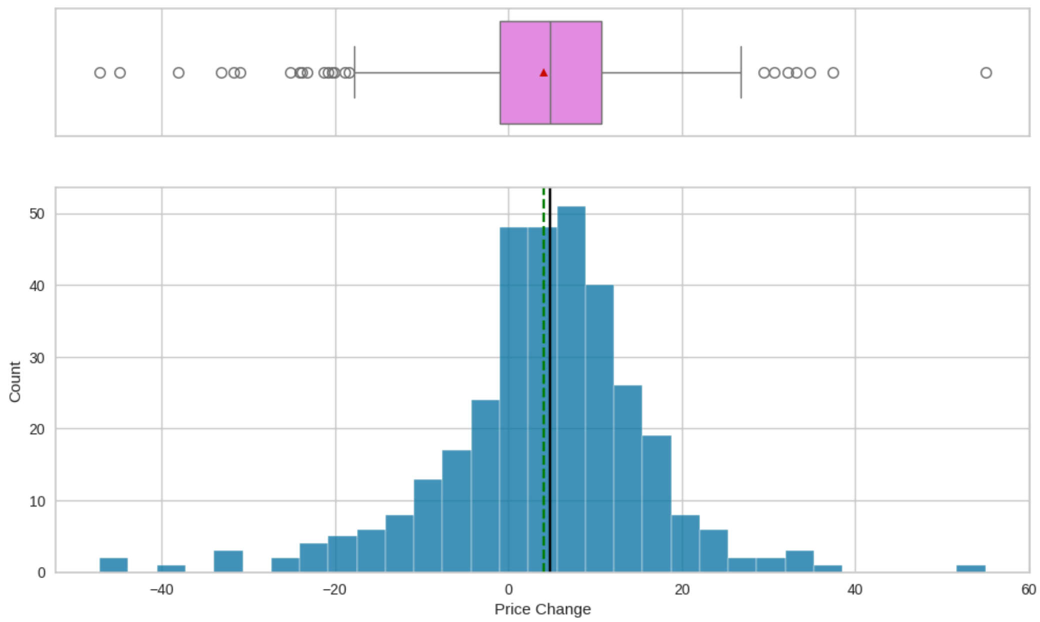
Univariate analysis

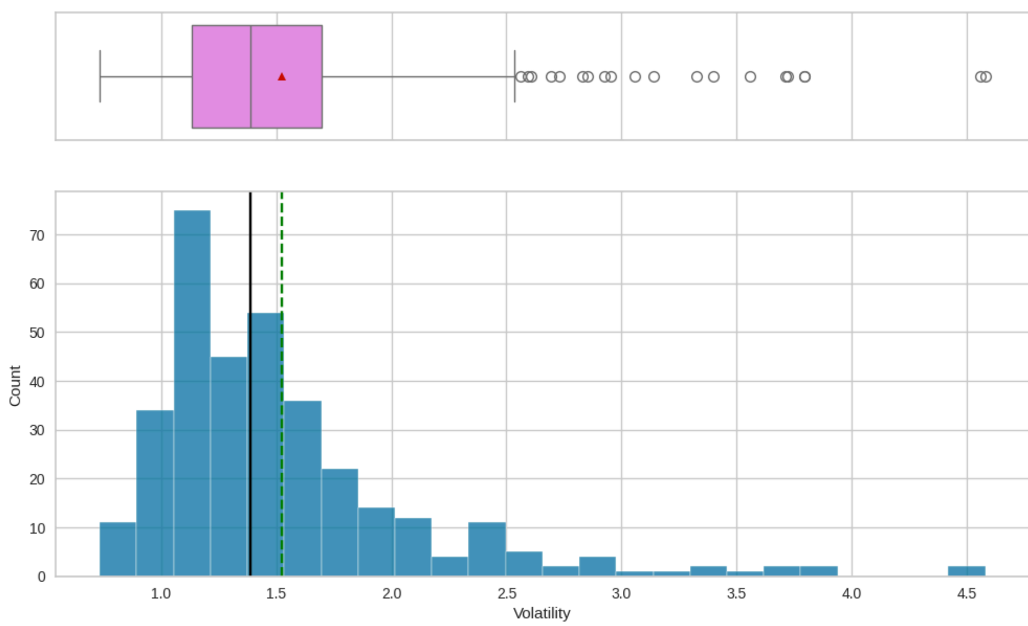
Numeric Columns

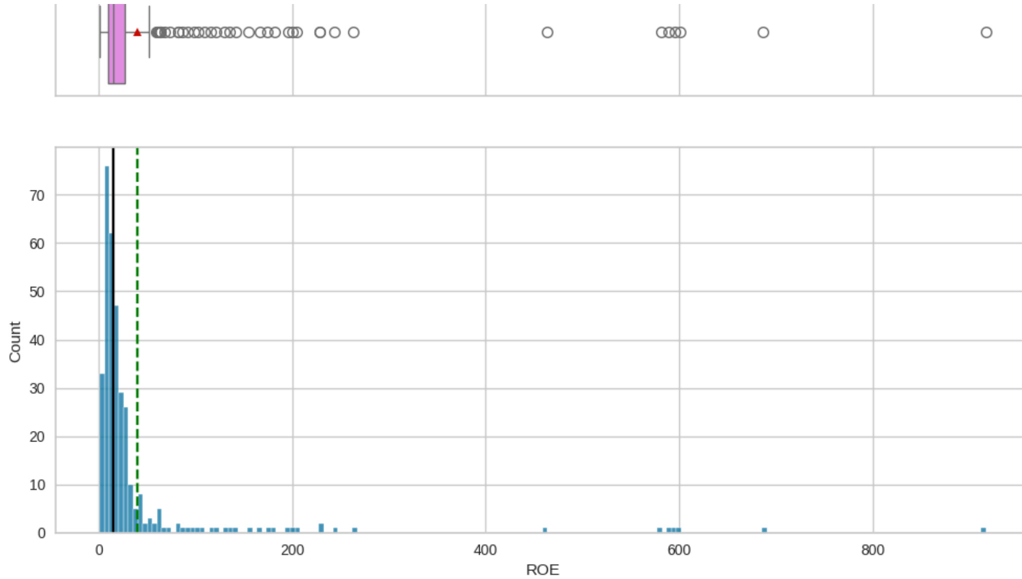
```
plt.figure(figsize=(15, 12))

for feature in df.select_dtypes(include=np.number).columns:
    histogram_boxplot(df, feature, figsize=(12, 7), kde=False, bins=None)
```

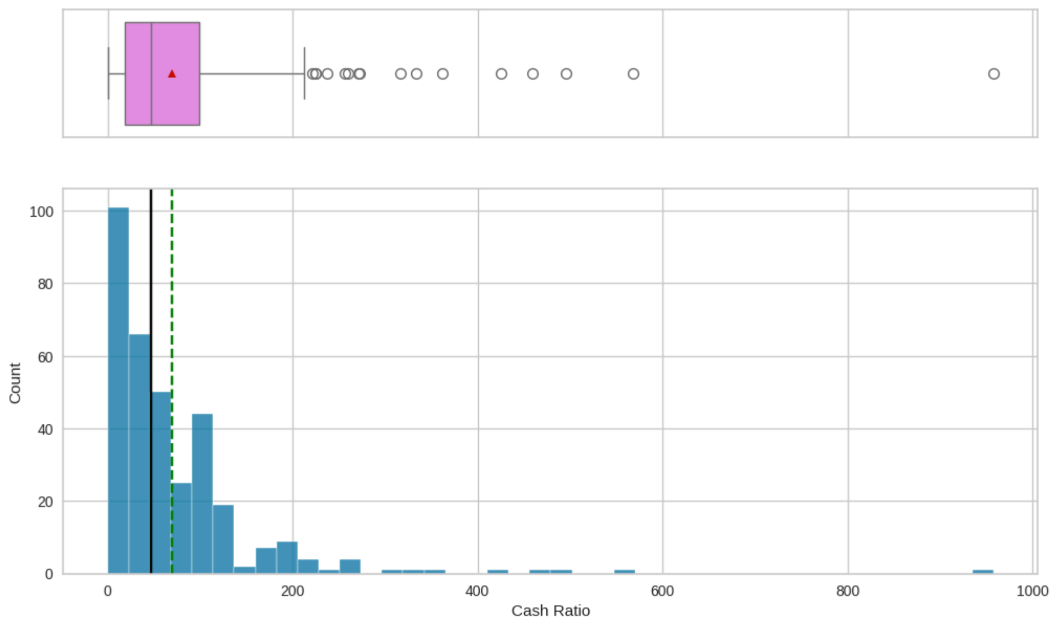
<Figure size 1500x1200 with 0 Axes>

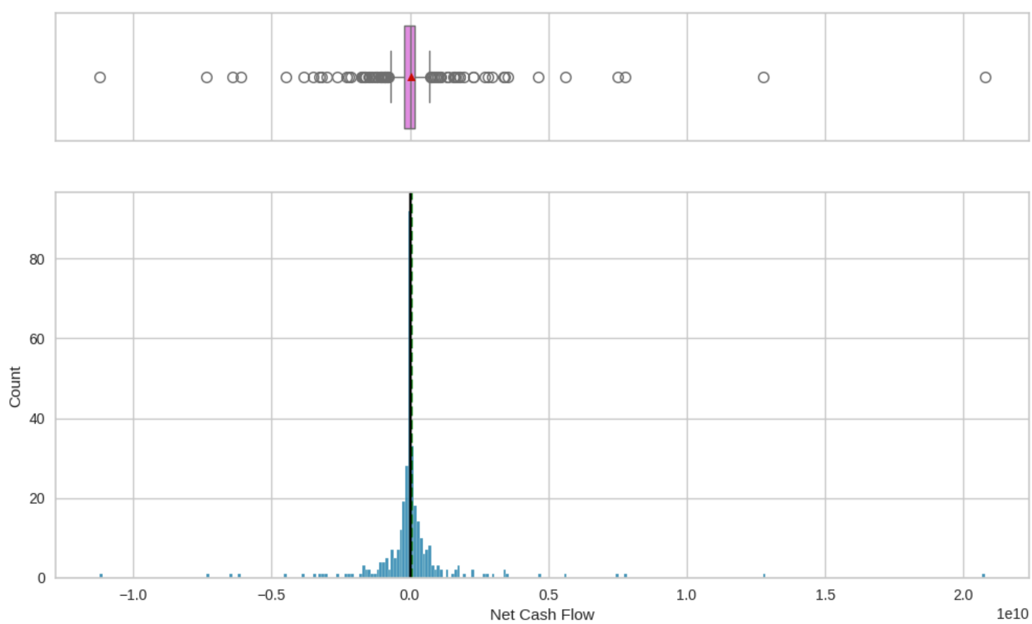


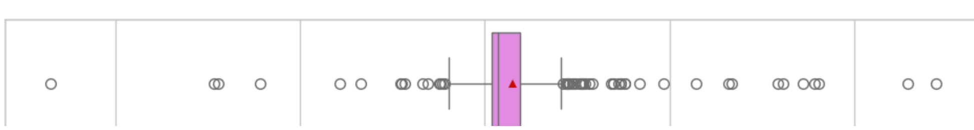


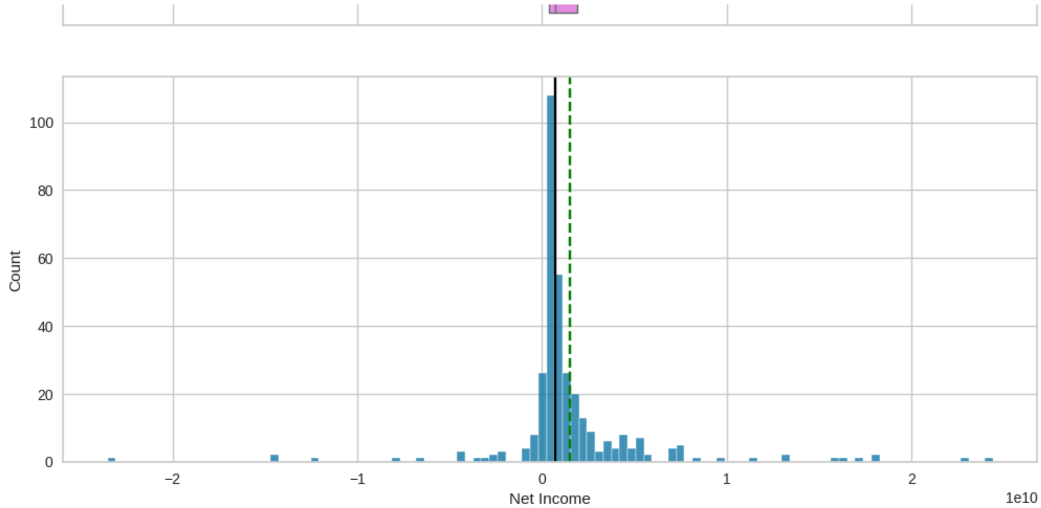


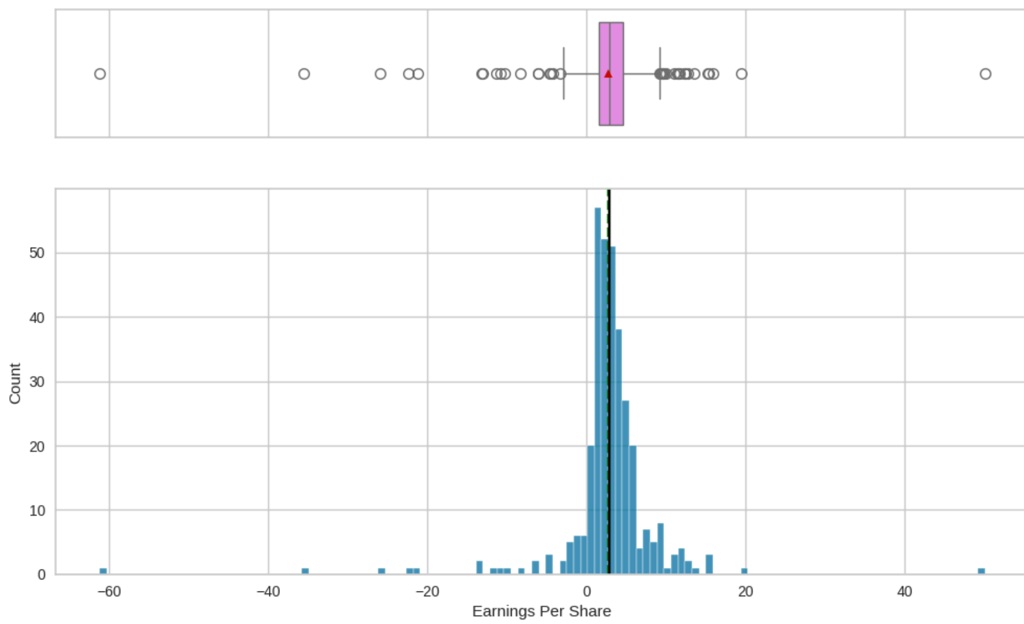

Count
ROE
0
200
400
600
800

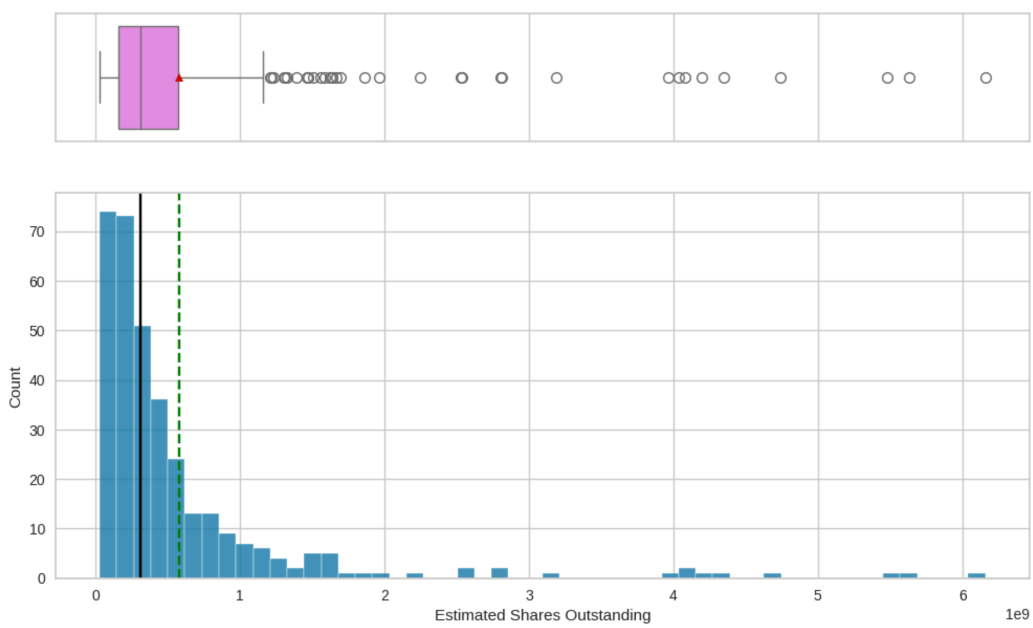

Count
Cash Ratio
0
200
400
600
800
1000

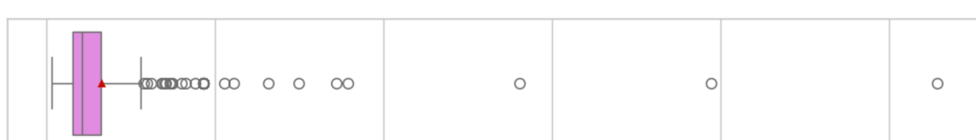

Count
Net Cash Flow
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
1e10

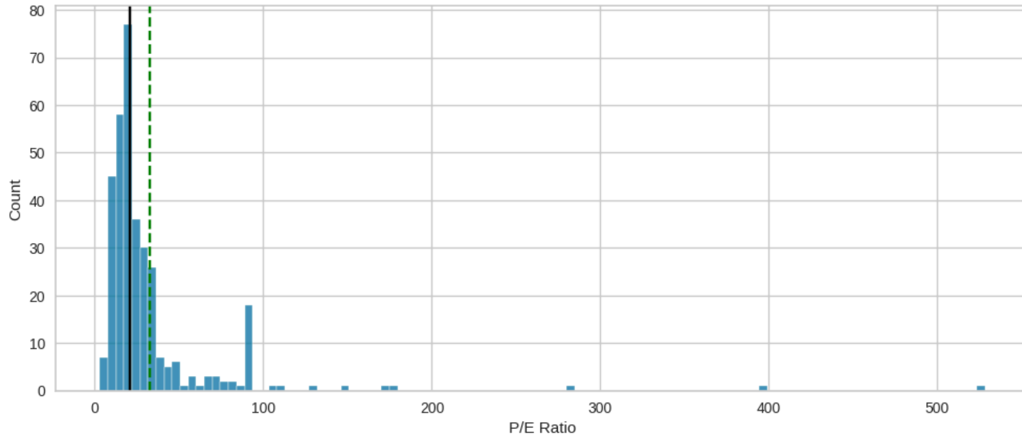


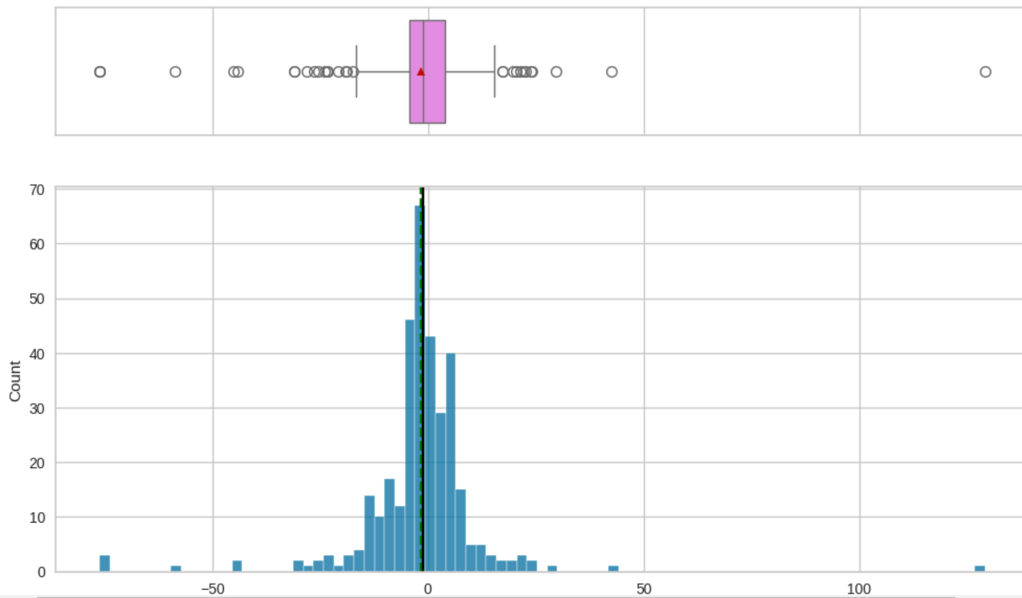

Count
Net Income
1e10


Count
Earnings Per Share


Count
Estimated Shares Outstanding
1e9

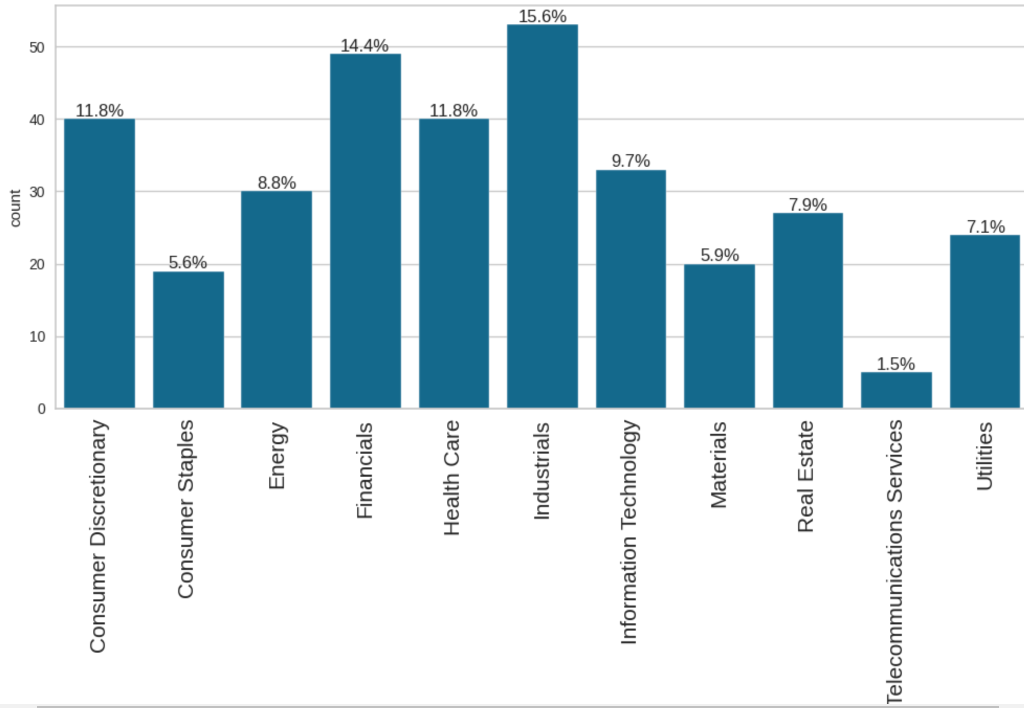



Count
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500
P/E Ratio


Count
0
10
20
30
40
50
60
70
−50
0
50
100

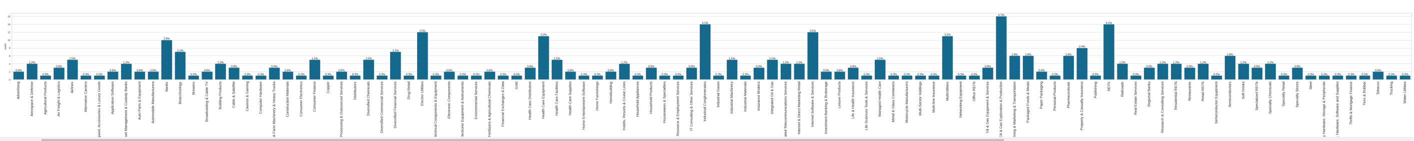
GICS Sector

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



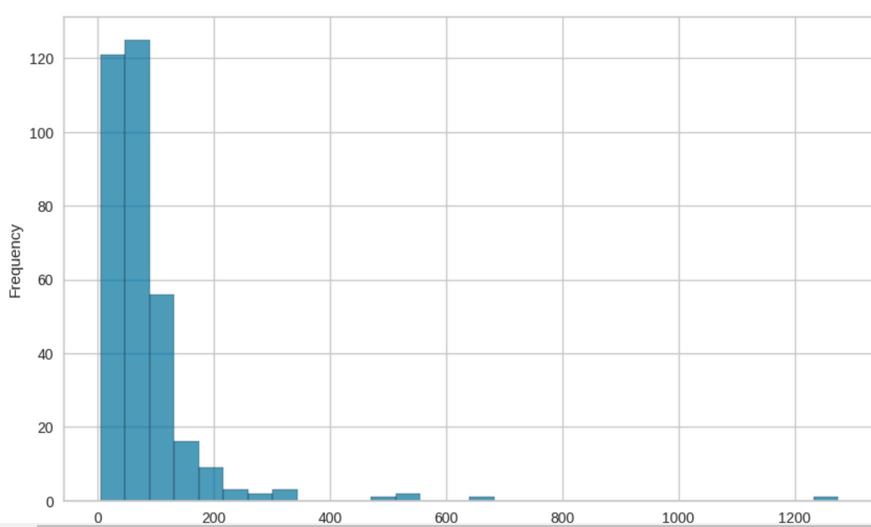
GICS Sub Industry

```
labeled_barplot(df, 'GICS Sub Industry', perc=True)
```



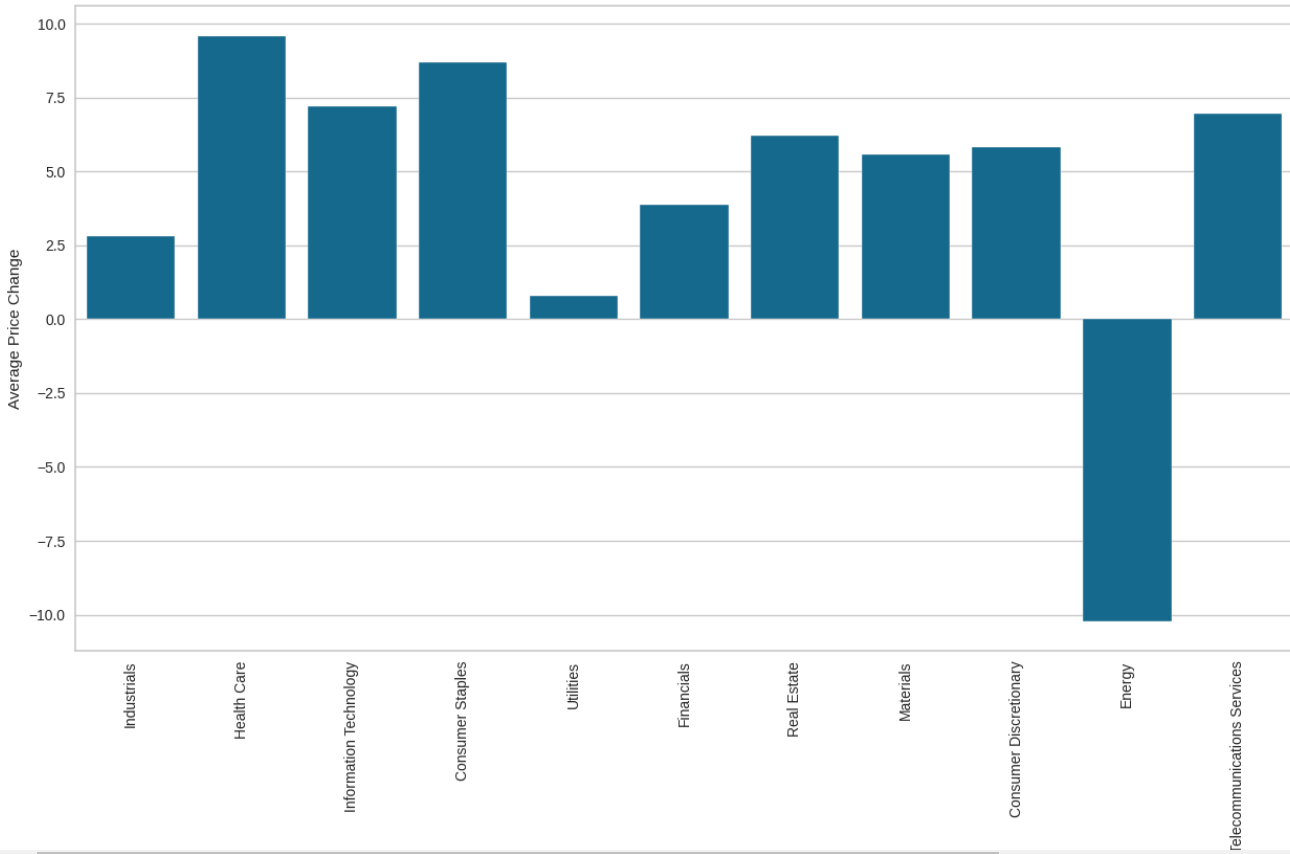
```
# 1. What does the distribution of stock prices look like?
```

```
plt.figure(figsize=(10,6))
plt.hist(df['Current Price'], bins=30, edgecolor='k', alpha=0.7)
plt.xlabel('Stock Price')
plt.ylabel('Frequency')
plt.show()
```

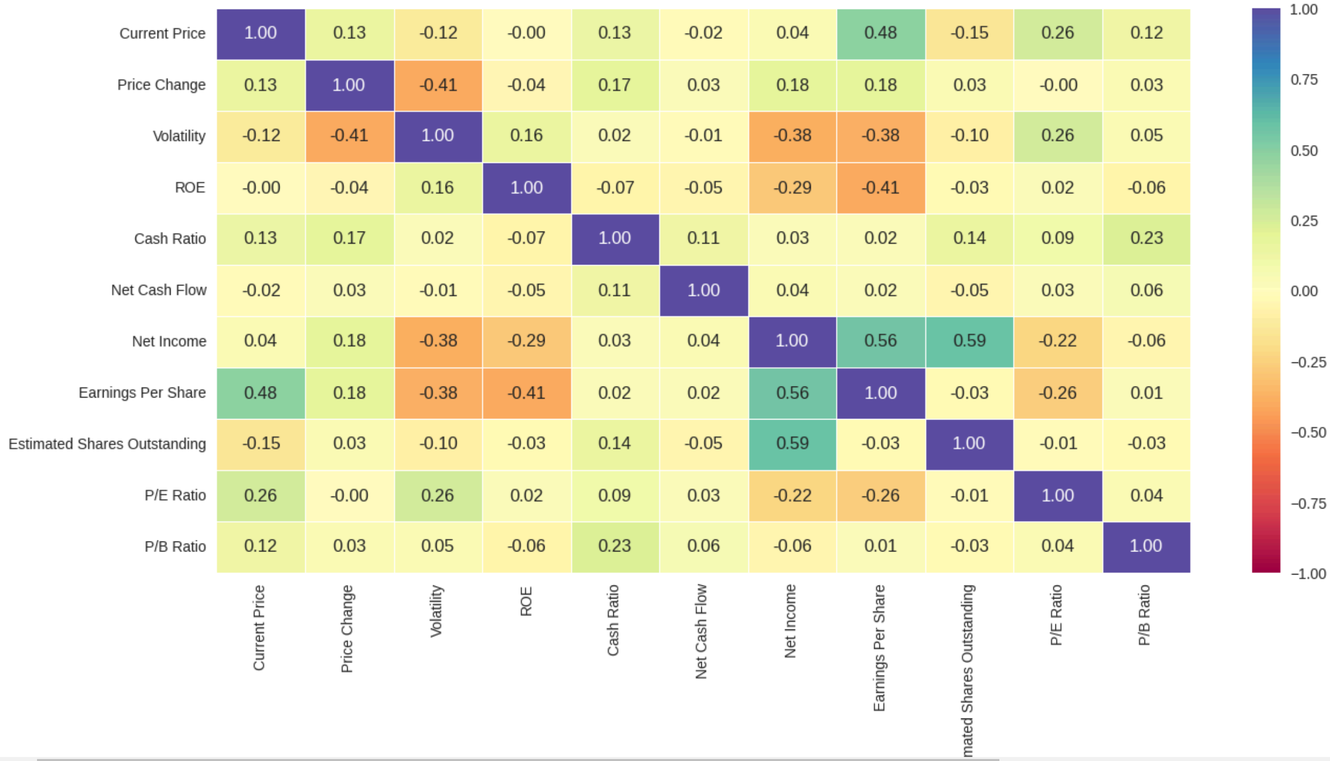


Bivariate Analysis

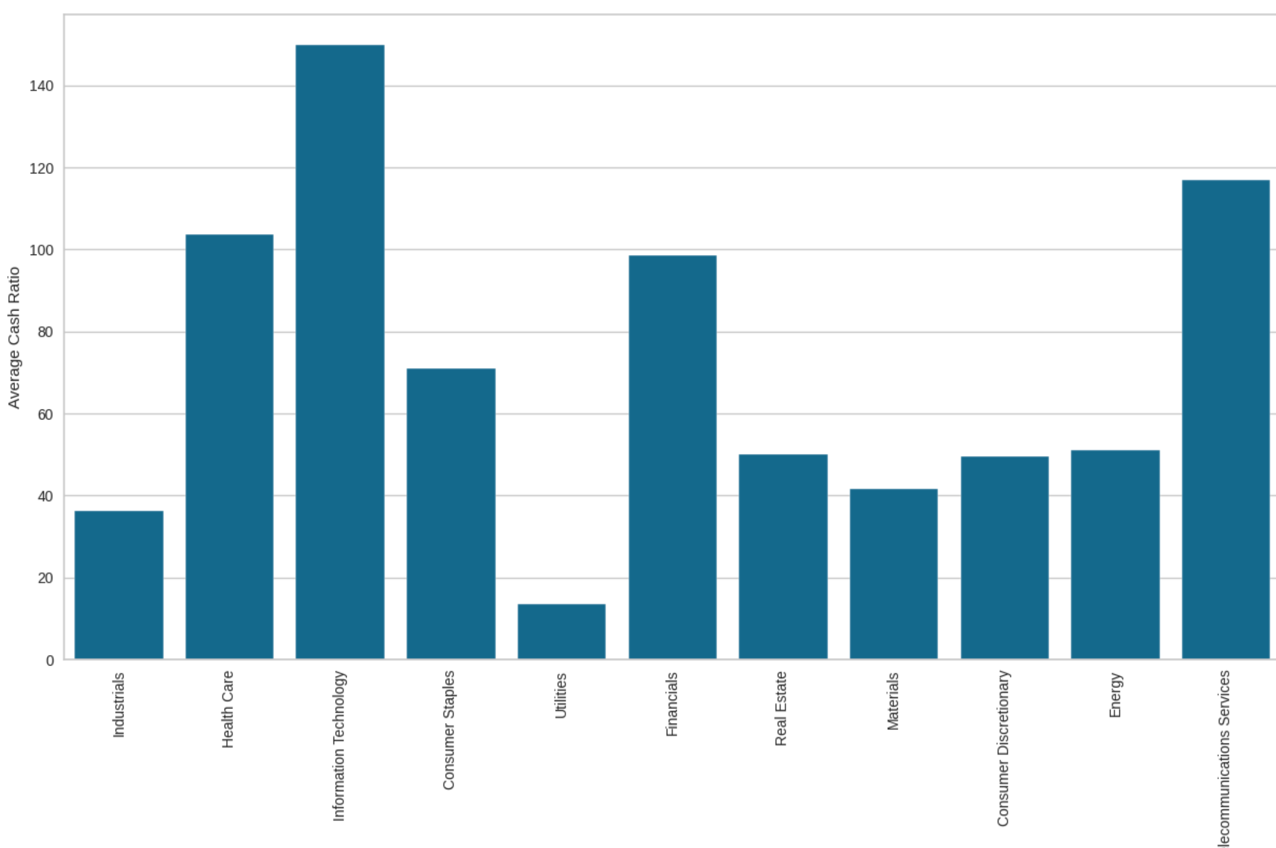
```
# 2. The stocks of which economic sector have seen the maximum price increase on average?
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='GICS Sector', y='Price Change', ci=False)
plt.xticks(rotation=90)
plt.ylabel('Average Price Change')
plt.xlabel('GICS Sector')
plt.show()
```



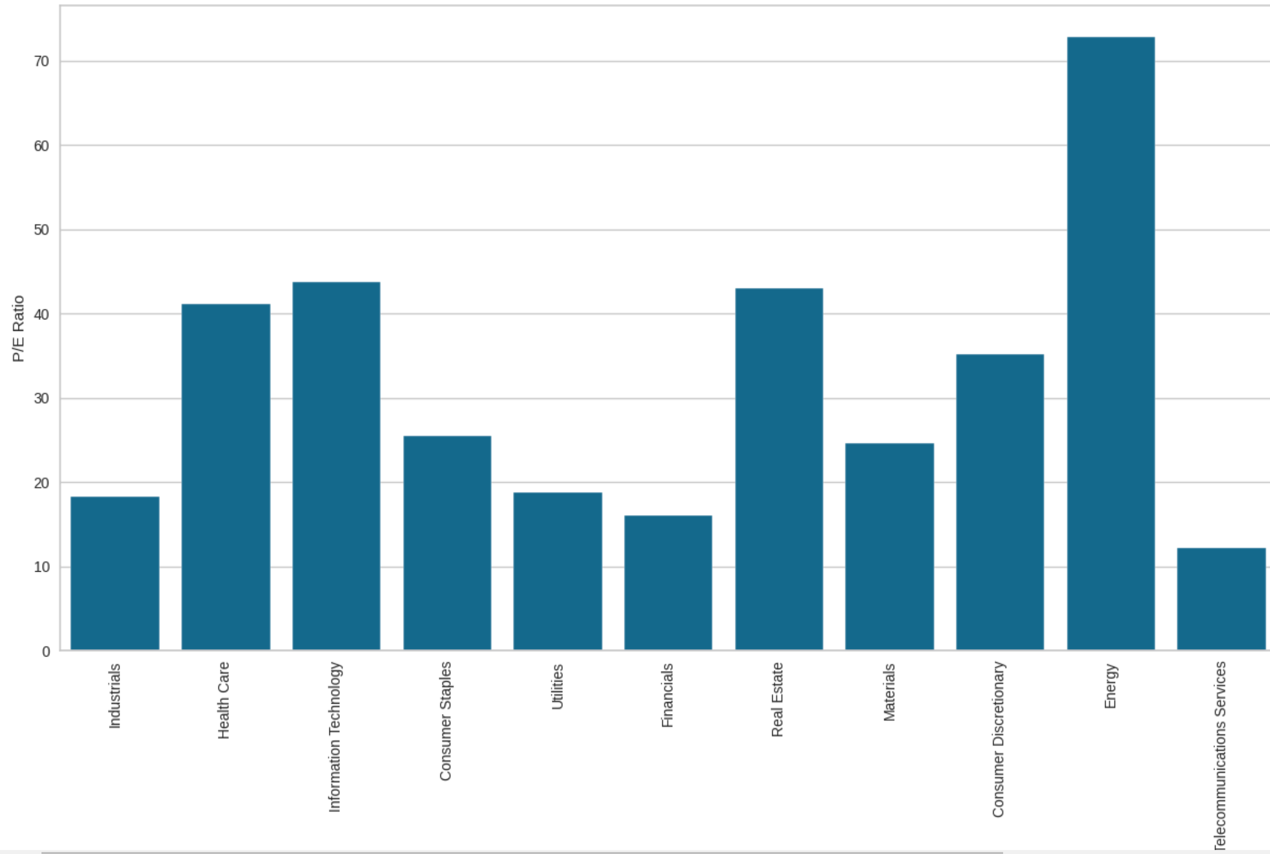

```
# 3. How are the different variables correlated with each other?
plt.figure(figsize=(15, 7))
sns.heatmap(df.corr(), annot=True, cmap='Spectral', linewidths=0.5, fmt='.2f', vmin=-1, vmax=1)
plt.show()
```




```
# 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?
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='GICS Sector', y='Cash Ratio', ci=False)
plt.xticks(rotation=90)
plt.ylabel('Average Cash Ratio')
plt.xlabel('GICS Sector')
plt.show()
```



```
# 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?
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='GICS Sector', y='P/E Ratio', ci=False)
plt.xticks(rotation=90)
plt.ylabel('P/E Ratio')
plt.xlabel('GICS Sector')
plt.show()
```



Data Preprocessing

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

Duplicate value check

```
print("There are",data.duplicated().sum(),"duplicated rows")
```

```
There are 0 duplicated rows
```

Missing values check

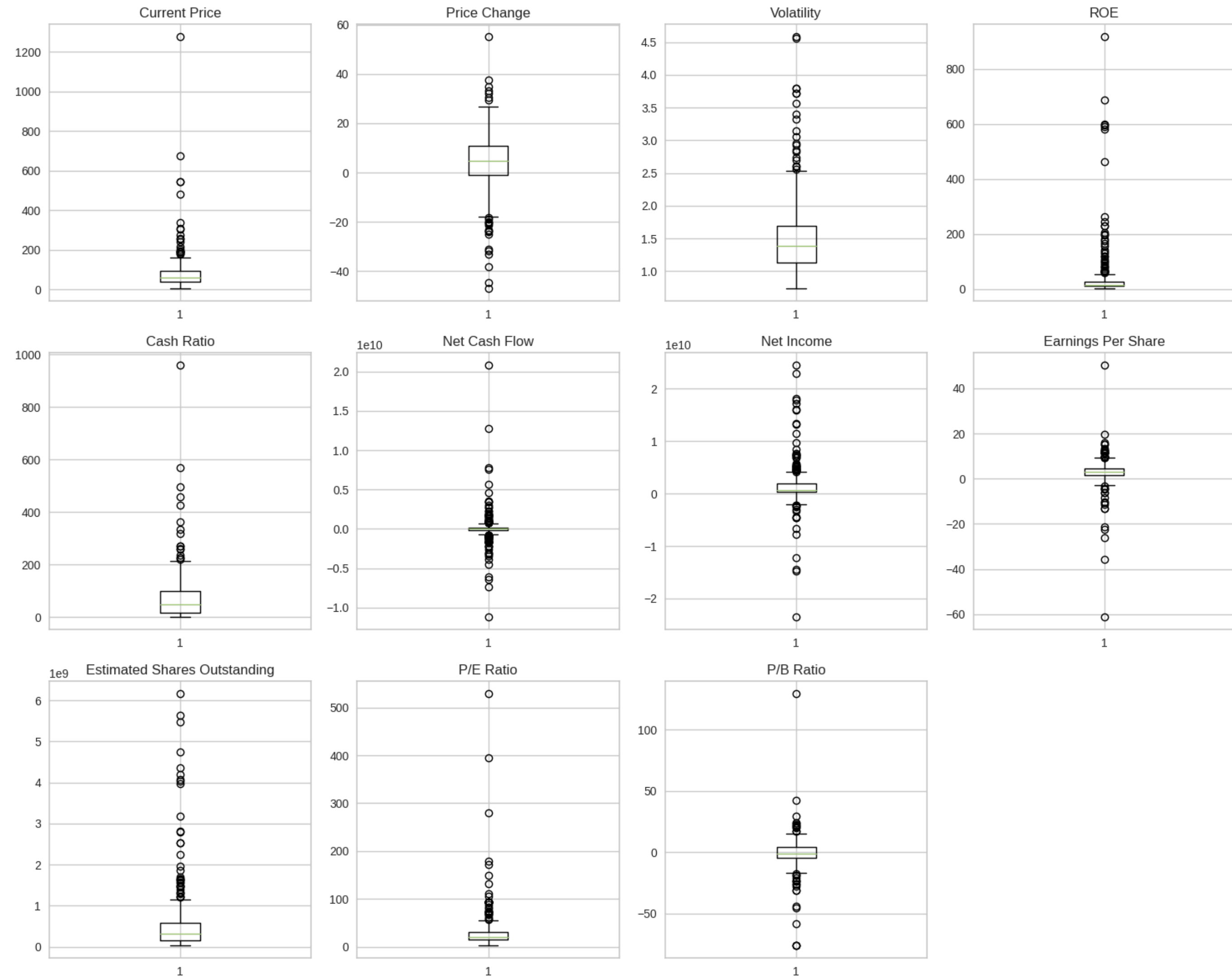
```
data.isnull().sum()
```

	0
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

Observations - There is no missing values in the data

Outlier check

```
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

```
scaler = StandardScaler()
subset = df[['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']]
subset_scaled = scaler.fit_transform(subset)
```

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

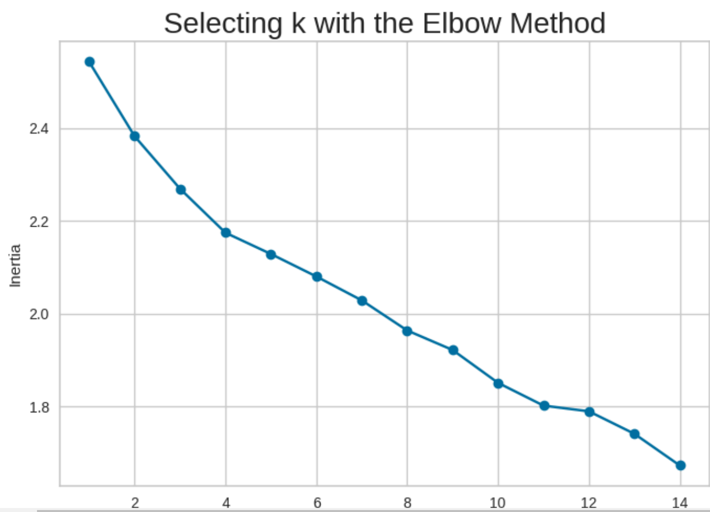
K-means Clustering

Using the Elbow Method

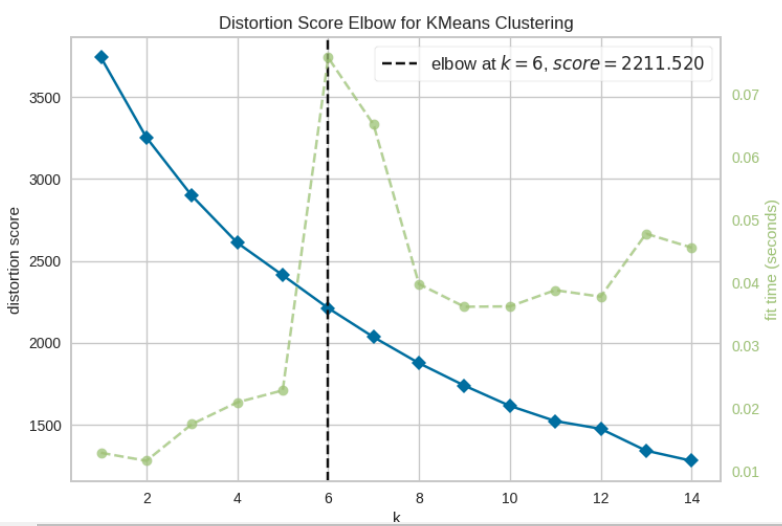
```
k_means_df = subset_scaled_df.copy()
```

```
# Elbow Method
clusters = range(1, 15)
inertia = []

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]
    )

    inertia.append(distortion)
```

```
plt.plot(clusters, inertia, marker='o')
plt.xlabel("Number of clusters (k)")
plt.ylabel("Inertia")
plt.title("Selecting k with the Elbow Method", fontsize=20)
plt.show()
```

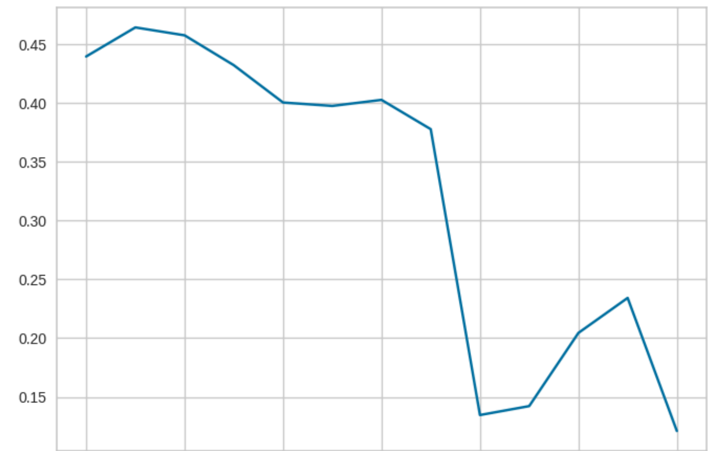



```
model = KMeans(random_state=1)
visualizer = KElbowVisualizer(model, k=(1, 15), timings=True)
visualizer.fit(k_means_df)
visualizer.show()
```

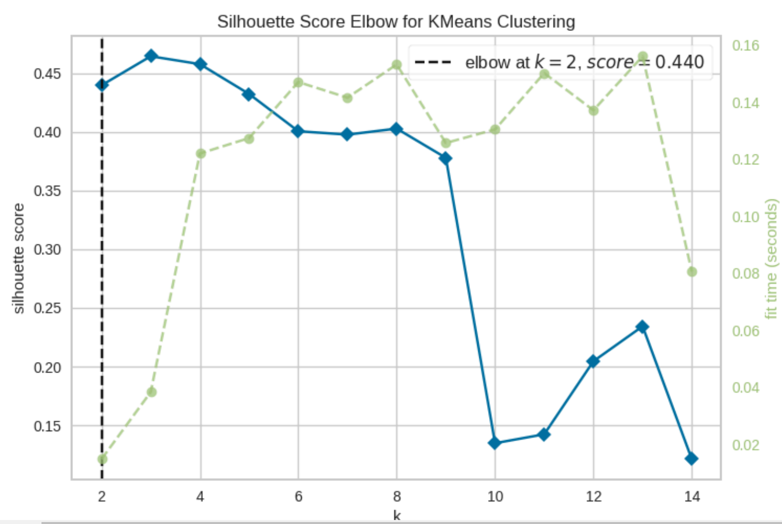


Silhouette Score

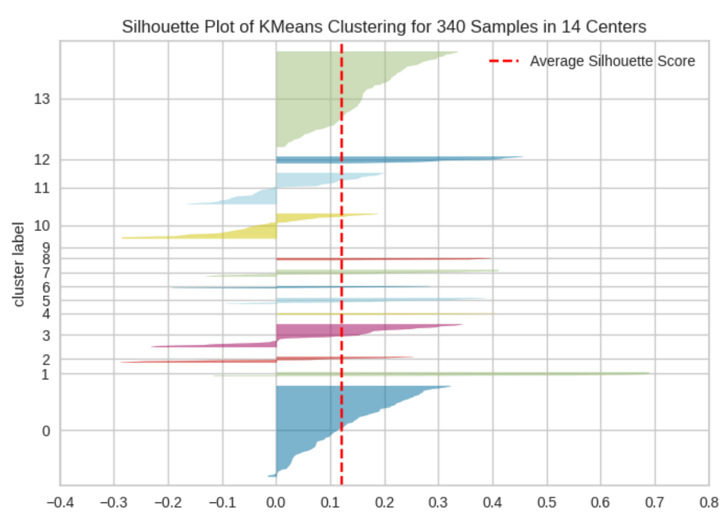
```
silhouette_scores = []
cluster_list = range(2, 15)
for n_clusters in cluster_list:
    clusterer = KMeans(n_clusters=n_clusters, random_state=1)
    preds = clusterer.fit_predict((subset_scaled_df))
    score = silhouette_score(k_means_df, preds)
    silhouette_scores.append(score)

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




```
model = KMeans(random_state=1)
visualizer = KElbowVisualizer(model, k=(2, 15), metric="silhouette", timings=True)
visualizer.fit(k_means_df)
visualizer.show()
```




```
kmeans = KMeans(n_clusters=n_clusters, random_state=1)
visualizer = SilhouetteVisualizer(kmeans, colors='yellowbrick')
visualizer.fit(k_means_df)
visualizer.show()
```



Final model

```
# setup a variable for number of clusters
num_clusters = 5


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

KMeans

```
# 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 Profiling

```
km_cluster_profile = df1.groupby("KM_segments").mean()
```

```
km_cluster_profile["count_in_each_segment"] = (
    df1.groupby("KM_segments")["Security"].count().values
)
```

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

KM_segments	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	count_in_each_
0	65.174668	-11.542247	2.690220	37.300000	65.366667	195008366.666667	-1677736033.333333	-4.401667	544473664.718000	113.488924	1.424161	
1	72.738269	5.179897	1.380738	34.825455	53.138182	-10147287.272727	1488641570.909091	3.636164	437961614.918582	23.680917	-3.395254	
2	233.251108	13.682869	1.719008	29.333333	296.523810	1398716380.952381	1835686380.952381	7.126190	508721791.962857	37.805996	16.758218	
3	50.517273	5.747586	1.130399	31.090909	75.909091	-1072272727.272727	14833090909.090910	4.154545	4298826628.727273	14.803577	-4.552119	

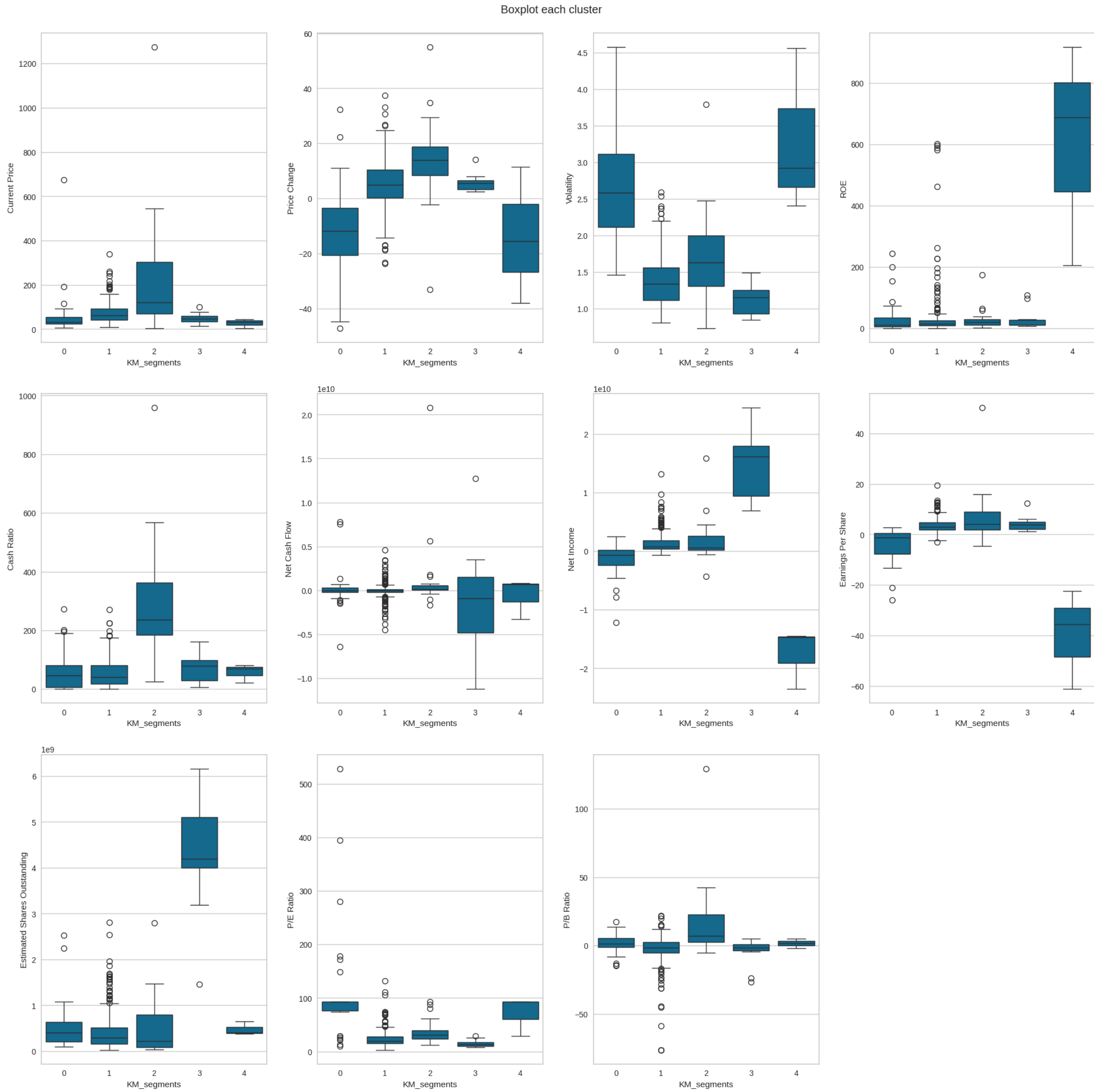
```
for cl in df1["KM_segments"].unique():
    print("In cluster {}, the following companies are present:".format(cl))
    print(df1[df1["KM_segments"] == cl]["Security"].unique())
    print()
```

```
['Apache Corporation' 'Chesapeake Energy' 'Devon Energy Corp.']

In cluster 3, the following companies are present:
['Citigroup Inc.' 'Ford Motor' 'Gilead Sciences' 'Intel Corp.'
 'JPMorgan Chase & Co.' 'Coca Cola Company' 'Pfizer Inc.' 'AT&T Inc'
 'Verizon Communications' 'Wells Fargo' 'Exxon Mobil Corp.']
```

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

KM_segments	GICS Sector	Security
0	Consumer Discretionary	1
	Energy	21
	Health Care	1
	Industrials	1
	Information Technology	4
	Materials	2
1	Consumer Discretionary	33
	Consumer Staples	17
	Energy	5
	Financials	45
	Health Care	29
	Industrials	52
	Information Technology	24
	Materials	18
	Real Estate	26
	Telecommunications Services	2
	Utilities	24
2	Consumer Discretionary	5
	Consumer Staples	1
	Financials	1
	Health Care	8
	Information Technology	4
	Real Estate	1
	Telecommunications Services	1
3	Consumer Discretionary	1
	Consumer Staples	1
	Energy	1
	Financials	3
	Health Care	2
	Information Technology	1
	Telecommunications Services	2
4	Energy	3

```
plt.figure(figsize=(20, 20))
plt.suptitle("Boxplot each cluster")

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

for i, variable in enumerate(num_col):
    plt.subplot(3, 4, i + 1)
    sns.boxplot(data=df1, x="KM_segments", y=variable)

plt.tight_layout(pad=2.0)
```



Hierarchical Clustering

Computing Cophenetic Correlation

```
hc_df = subset_scaled_df.copy()


# 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 0.9232271494002922.
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 0.7925307202850002.
Cophenetic correlation for Mahalanobis distance and average linkage is 0.9247324030159736.
Cophenetic correlation for Mahalanobis distance and weighted linkage is 0.8708317490180428.
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.
```

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

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

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

```
# 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
```

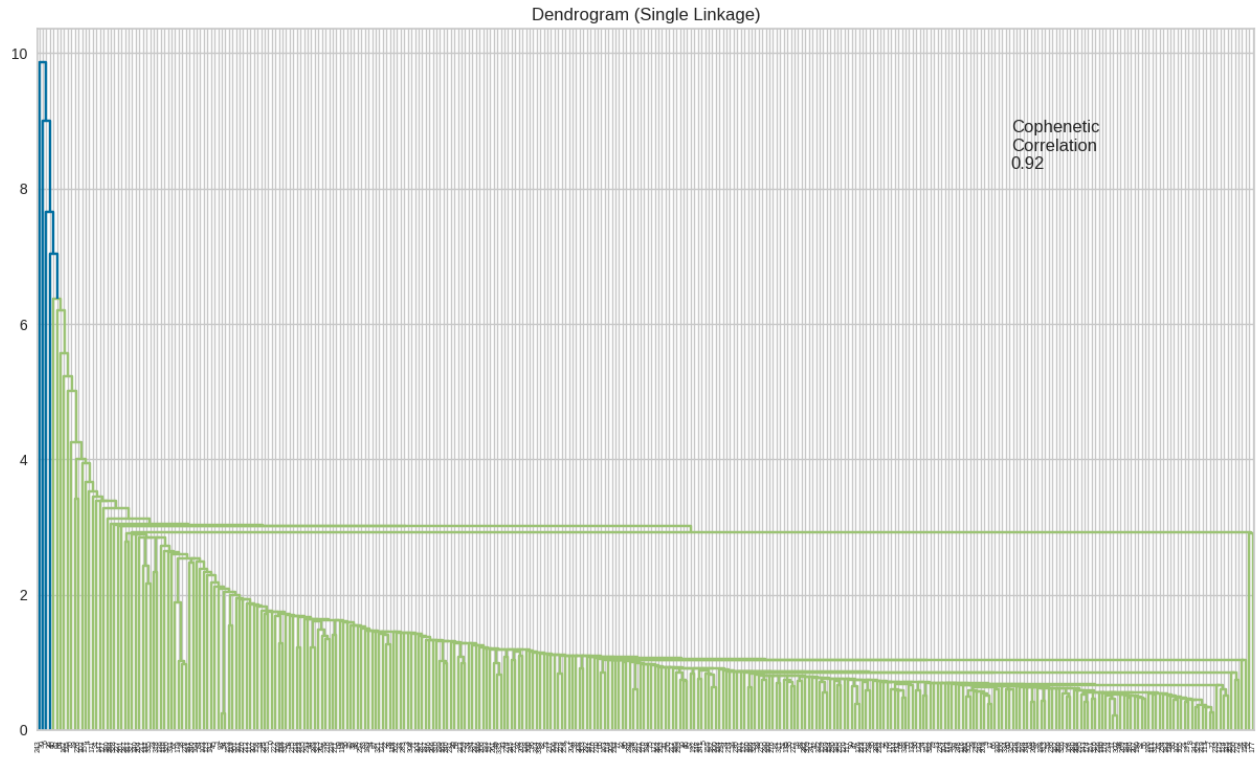
```
Cophenetic correlation for single linkage is 0.9232271494002922.
Cophenetic correlation for complete linkage is 0.7873280186580672.
Cophenetic correlation for average linkage is 0.9422540609560814.
Cophenetic correlation for centroid linkage is 0.9314012446828154.
Cophenetic correlation for ward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

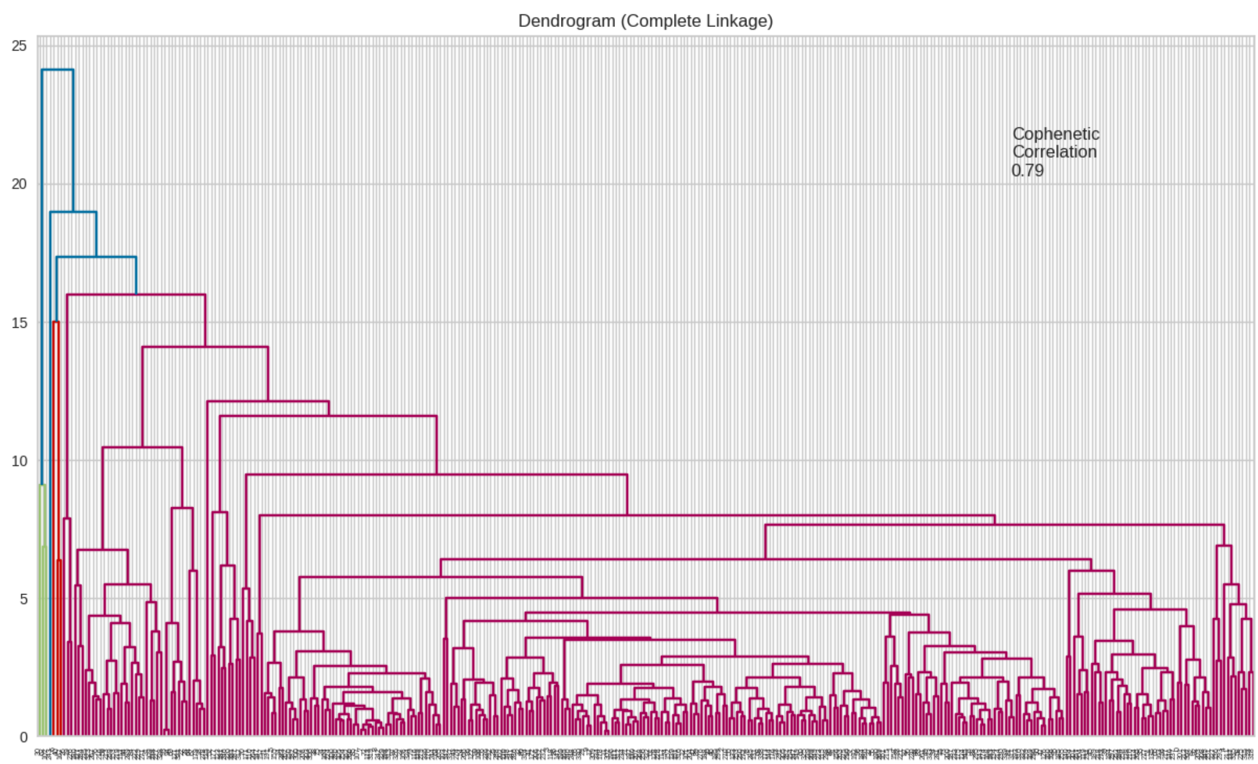
```
# printing the combination of distance metric and linkage method with the highest cophenetic correlation
print(
    "Highest cophenetic correlation is {}, which is obtained with {} linkage.".format(
        high_cophenet_corr, high_dm_lm[1]
    )
)
```

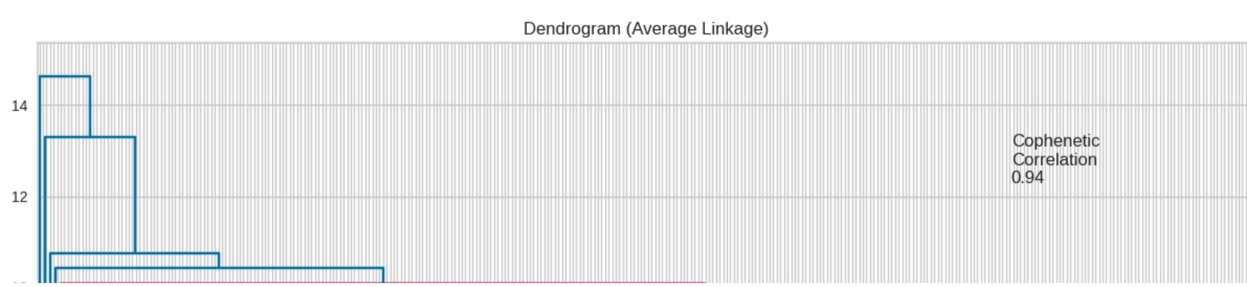
```
Highest cophenetic correlation is 0.9422540609560814, which is obtained with average linkage.
```

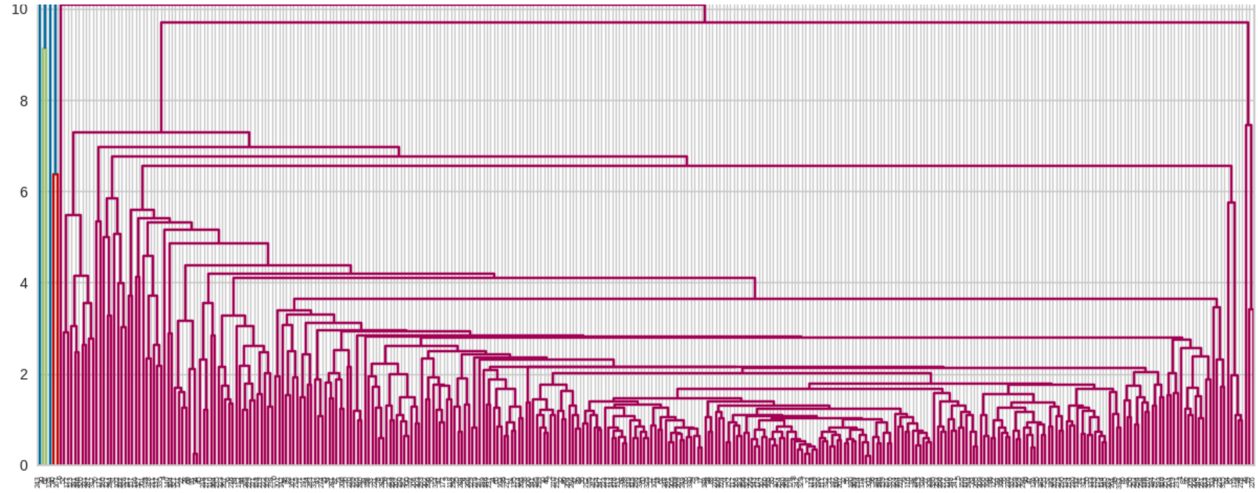
Let's see the dendrograms for the different linkage methods.

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

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

fig, axs = plt.subplots(len(linkage_methods), 1, figsize=(15, 40))

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])
```

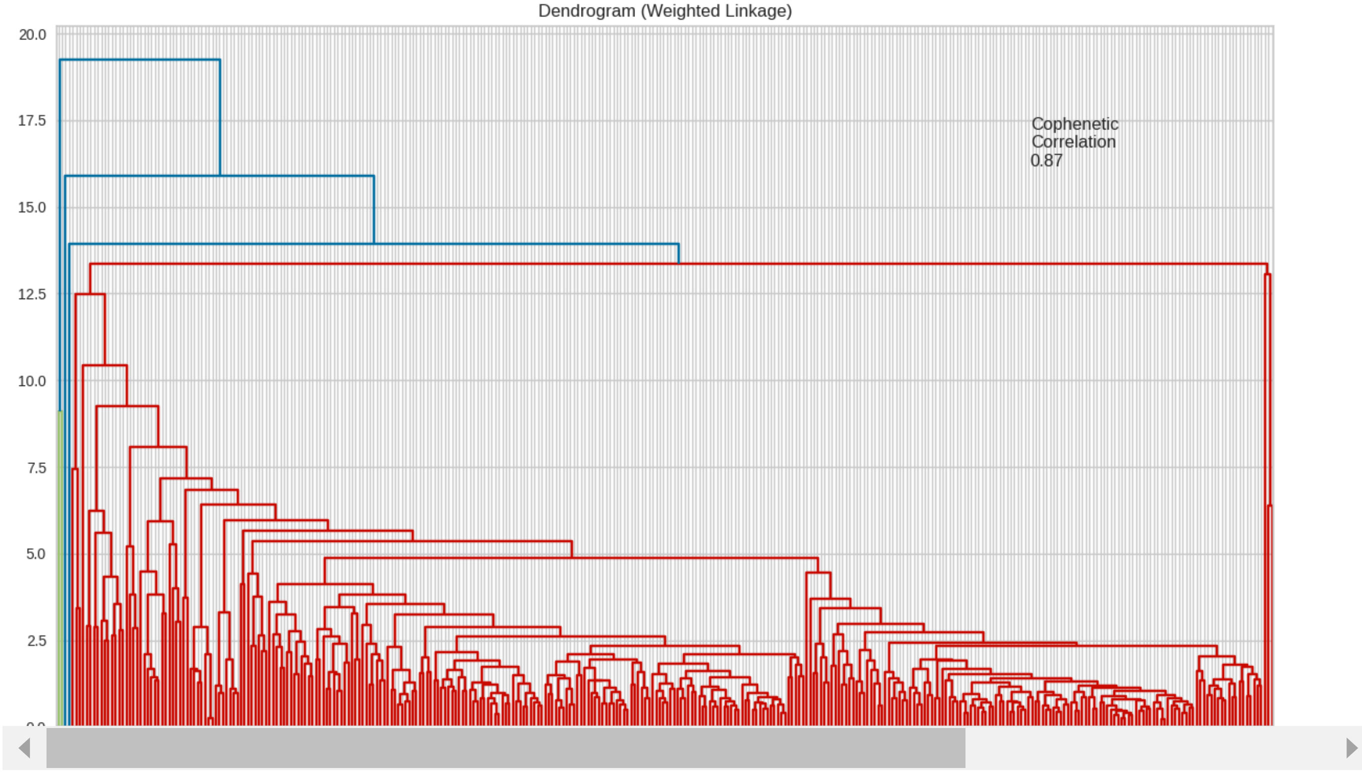

Dendrogram (Single Linkage)
Cophenetic
Correlation
0.92
0
2
4
6
8
10


Dendrogram (Complete Linkage)
Cophenetic
Correlation
0.79
0
5
10
15
20
25


Dendrogram (Average Linkage)
Cophenetic
Correlation
0.94
12
14



Dendrogram (Weighted Linkage)



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

	Linkage	Cophenetic Coefficient
1	complete	0.787328
3	weighted	0.869378
0	single	0.923227

Next steps: Generate code with df_cc | View recommended plots | New interactive sheet

Creating Model using sklearn

```
HCmodel = AgglomerativeClustering(n_clusters=num_clusters, affinity="euclidean", linkage="average")
HCmodel.fit(subset_scaled_df)
```

```
AgglomerativeClustering
```

```
# 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

```
hc_cluster_profile = df2.groupby("HC_segments").mean()
```

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

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

HC_segments	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	count_in_each_
0	77.884243	4.105986	1.516865	35.320359	66.775449	-32825817.365269	1535255703.592814	2.903308	559027333.145509	32.437511	-1.781988	
1	25.640000	11.237908	1.322355	12.500000	130.500000	16755500000.000000	13654000000.000000	3.295000	2791829362.100000	13.649696	1.508484	
2	24.485001	-13.351992	3.482611	802.000000	51.000000	-1292500000.000000	-19106500000.000000	-41.815000	519573983.250000	60.748608	1.565141	
3	104.660004	16.224320	1.320606	8.000000	958.000000	592000000.000000	3669000000.000000	1.310000	2800763359.000000	79.893133	5.884467	

```
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()
```

```
 'United Continental Holdings' 'UDR Inc' 'Universal Health Services, Inc.'
 'United Health Group Inc.' 'Unum Group' 'Union Pacific'
 'United Parcel Service' 'United Technologies' 'Varian Medical Systems'
 'Valero Energy' 'Vulcan Materials' 'Vornado Realty Trust'
 'Verisk Analytics' 'Verisign Inc.' 'Vertex Pharmaceuticals Inc'
 'Ventas Inc' 'Verizon Communications' 'Waters Corporation'
 'Wec Energy Group Inc' 'Wells Fargo' 'Whirlpool Corp.'
 'Waste Management Inc.' 'Williams Cos.' 'Western Union Co'
 'Weyerhaeuser Corp.' 'Wyndham Worldwide' 'Wynn Resorts Ltd'
```