Processing math: 100%

In [1]:

```
# this will help in making the Python code more structured automatically (
#%load_ext nb_black

# 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

# 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 pdist
from scipy.spatial.distance import cdist

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


# to perform hierarchical clustering, compute cophenetic correlation, and
from sklearn.cluster import AgglomerativeClustering

import warnings
warnings.filterwarnings('ignore')
!pip install yellowbrick
# to visualize the elbow curve and silhouette scores
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
from scipy.cluster.hierarchy import dendrogram, linkage, cophenet
```

Processing math: 100%

```
Requirement already satisfied: yellowbrick in c:\users\conne\anaconda3\li
b\site-packages (1.5)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in c:\users\conn
e\anaconda3\lib\site-packages (from yellowbrick) (3.7.1)
Requirement already satisfied: scipy>=1.0.0 in c:\users\conne\anaconda3\l
ib\site-packages (from yellowbrick) (1.10.1)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\users\conne\anac
onda3\lib\site-packages (from yellowbrick) (1.3.2)
Requirement already satisfied: numpy>=1.16.0 in c:\users\conne\anaconda3
\lib\site-packages (from yellowbrick) (1.24.3)
Requirement already satisfied: cycler>=0.10.0 in c:\users\conne\anaconda3
\lib\site-packages (from yellowbrick) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\conne\anacond
a3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.0.
5)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\conne\anacon
da3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.2
5.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\conne\anacon
da3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.4.
4)
Requirement already satisfied: packaging>=20.0 in c:\users\conne\anaconda
3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\conne\anaconda3
\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\conne\anacond
a3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.0.
9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\conne\ana
conda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick)
(2.8.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\conne\anaconda3
\lib\site-packages (from scikit-learn>=1.0.0->yellowbrick) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\conne\ana
conda3\lib\site-packages (from scikit-learn>=1.0.0->yellowbrick) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\conne\anaconda3\lib\s
ite-packages (from python-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yello
wbrick) (1.16.0)
```

In [2]:
```
df = pd.read_csv('/Users/conne/Downloads/stock_data.csv')
```

Processing math: 100%

In [3]:
```
df
```

Out[3]:

	Ticker Symbol	Security	GICS Sector	GICS Sub Industry	Current Price	Price Change	Volatility	R
0	AAL	American Airlines Group	Industrials	Airlines	42.349998	9.999995	1.687151	1
1	ABBV	AbbVie	Health Care	Pharmaceuticals	59.240002	8.339433	2.197887	1
2	ABT	Abbott Laboratories	Health Care	Health Care Equipment	44.910000	11.301121	1.273646	
3	ADBE	Adobe Systems Inc	Information Technology	Application Software	93.940002	13.977195	1.357679	
4	ADI	Analog Devices, Inc.	Information Technology	Semiconductors	55.320000	-1.827858	1.701169	
...	...	...	...	...	...	...	...	
335	YHOO	Yahoo Inc.	Information Technology	Internet Software & Services	33.259998	14.887727	1.845149	
336	YUM	Yum! Brands Inc	Consumer Discretionary	Restaurants	52.516175	-8.698917	1.478877	1
337	ZBH	Zimmer Biomet Holdings	Health Care	Health Care Equipment	102.589996	9.347683	1.404206	
338	ZION	Zions Bancorp	Financials	Regional Banks	27.299999	-1.158588	1.468176	
339	ZTS	Zoetis	Health Care	Pharmaceuticals	47.919998	16.678836	1.610285	

340 rows × 15 columns

In [4]:
```
df.rename(columns = {'GICS Sector':'GICS_Sector'}, inplace = True)
```

In [5]:
```
data = df.copy()
```

In [6]:
```
data.shape
```

Out[6]: (340, 15)

The dataset has The dataset has 229 rows and 15 columns

Processing math: 100%

In [7]:

```
data.head()
```

Out[7]:

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

In [8]:

```
data.tail()
```

Out[8]:

	Ticker Symbol	Security	GICS_Sector	GICS Sub Industry	Current Price	Price Change	Volatility	ROE
335	YHOO	Yahoo Inc.	Information Technology	Internet Software & Services	33.259998	14.887727	1.845149	15
336	YUM	Yum! Brands Inc	Consumer Discretionary	Restaurants	52.516175	-8.698917	1.478877	142
337	ZBH	Zimmer Biomet Holdings	Health Care	Health Care Equipment	102.589996	9.347683	1.404206	1
338	ZION	Zions Bancorp	Financials	Regional Banks	27.299999	-1.158588	1.468176	4
339	ZTS	Zoetis	Health Care	Pharmaceuticals	47.919998	16.678836	1.610285	32

Processing math: 100%

In [9]:
```
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
```

All the columns seem to have the approptiate Dtype for the data the are representing

In [10]:
```
data.duplicated().sum()
```

Out[10]:
```
0
```

I seem to have no duplicate data

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
Ticker Symbol                   0
Security                        0
GICS_Sector                     0
GICS Sub Industry               0
Current Price                   0
Price Change                    0
Volatility                      0
ROE                             0
Cash Ratio                      0
Net Cash Flow                   0
Net Income                      0
Earnings Per Share              0
Estimated Shares Outstanding    0
P/E Ratio                       0
P/B Ratio                       0
dtype: int64
```

I have no null data

Processing math: 100%

In [12]:
```
data.describe().T
```

Out[12]:

	count	mean	std	min	25%	50
Current Price	340.0	8.086234e+01	9.805509e+01	4.500000e+00	3.855500e+01	5.970500e+
Price Change	340.0	4.078194e+00	1.200634e+01	-4.712969e+01	-9.394838e-01	4.819505e+
Volatility	340.0	1.525976e+00	5.917984e-01	7.331632e-01	1.134878e+00	1.385593e+
ROE	340.0	3.959706e+01	9.654754e+01	1.000000e+00	9.750000e+00	1.500000e+
Cash Ratio	340.0	7.002353e+01	9.042133e+01	0.000000e+00	1.800000e+01	4.700000e+
Net Cash Flow	340.0	5.553762e+07	1.946365e+09	-1.120800e+10	-1.939065e+08	2.098000e+
Net Income	340.0	1.494385e+09	3.940150e+09	-2.352800e+10	3.523012e+08	7.073360e+
Earnings Per Share	340.0	2.776662e+00	6.587779e+00	-6.120000e+01	1.557500e+00	2.895000e+
Estimated Shares Outstanding	340.0	5.770283e+08	8.458496e+08	2.767216e+07	1.588482e+08	3.096751e+
P/E Ratio	340.0	3.261256e+01	4.434873e+01	2.935451e+00	1.504465e+01	2.081988e+
P/B Ratio	340.0	-1.718249e+00	1.396691e+01	-7.611908e+01	-4.352056e+00	-1.067170e+

Processing math: 100%

In [13]:

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


def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None
    """
    Boxplot and histogram combined

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

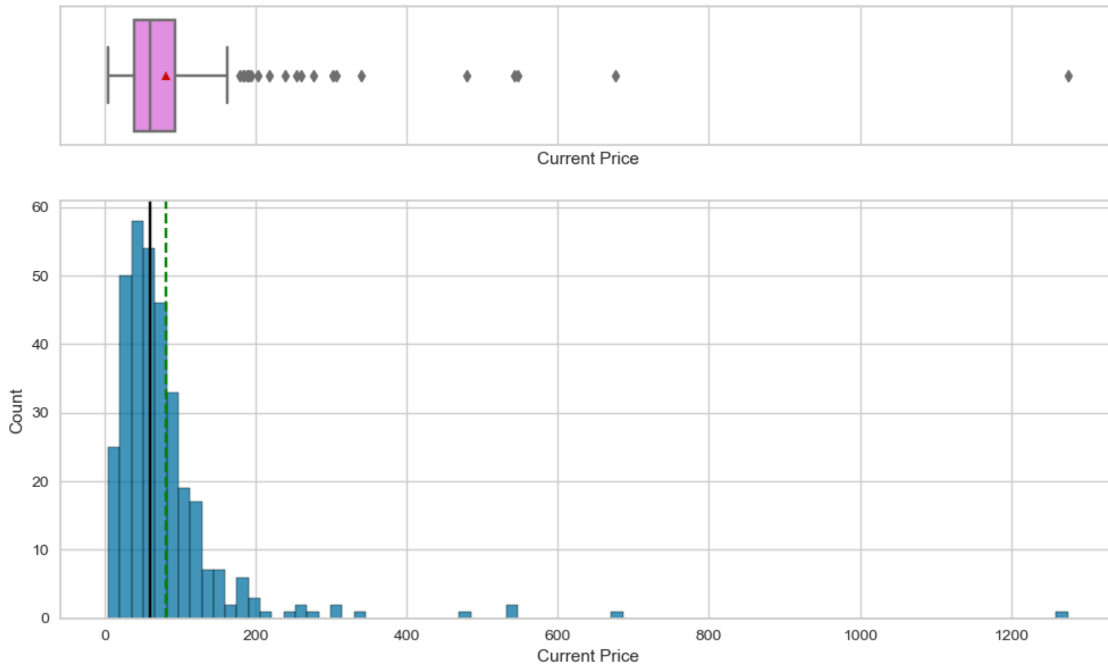
In [14]:

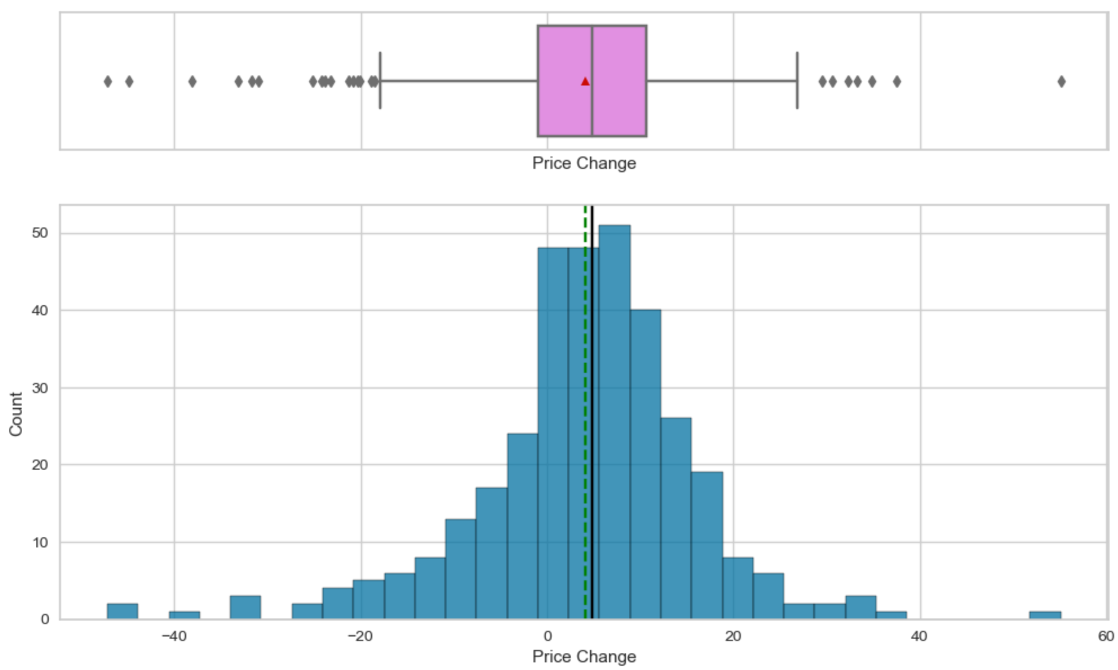
```
# selecting numerical columns
num_col = df.select_dtypes(include=np.number).columns.tolist()

print(num_col)
```

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

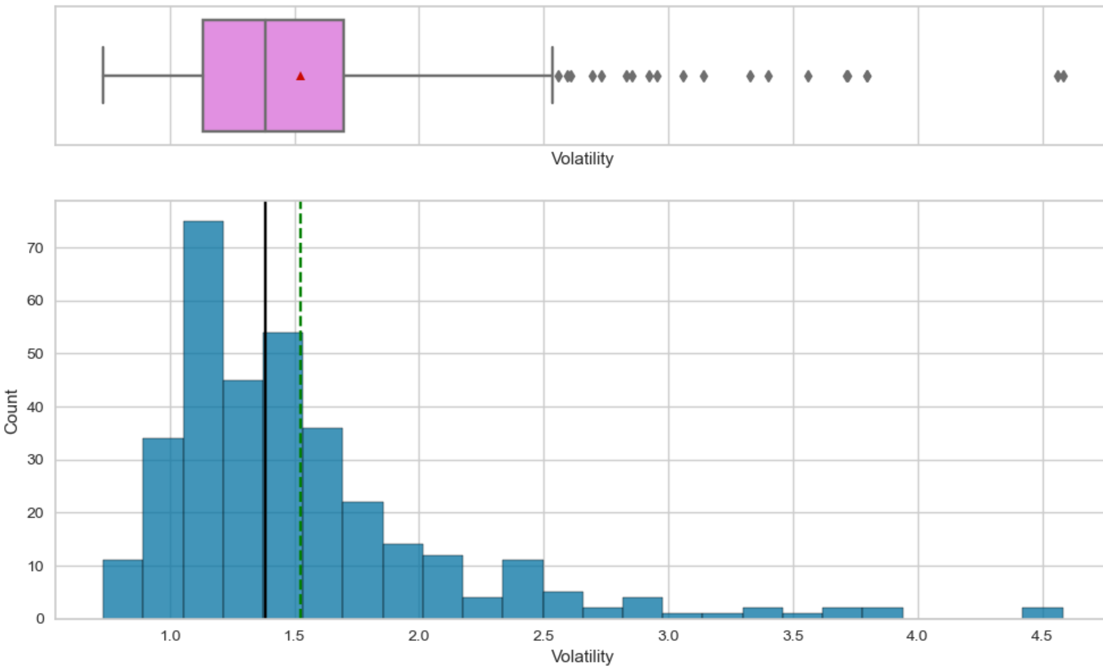
Processing math: 100%

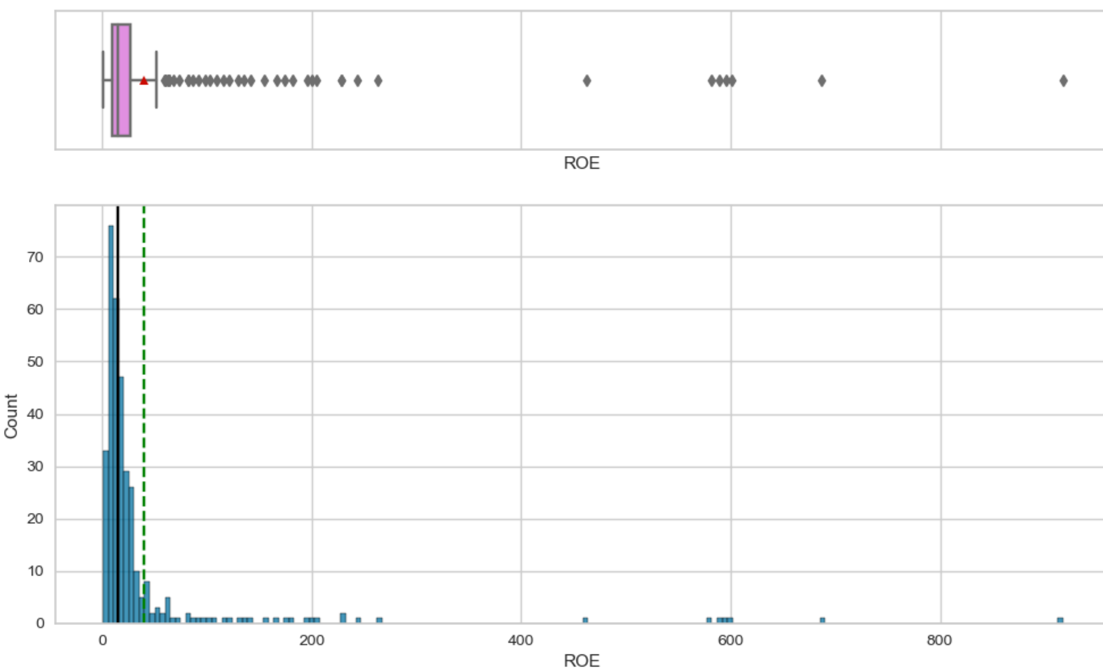
In [15]:
```
for item in num_col:
    histogram_boxplot(df, item)
```

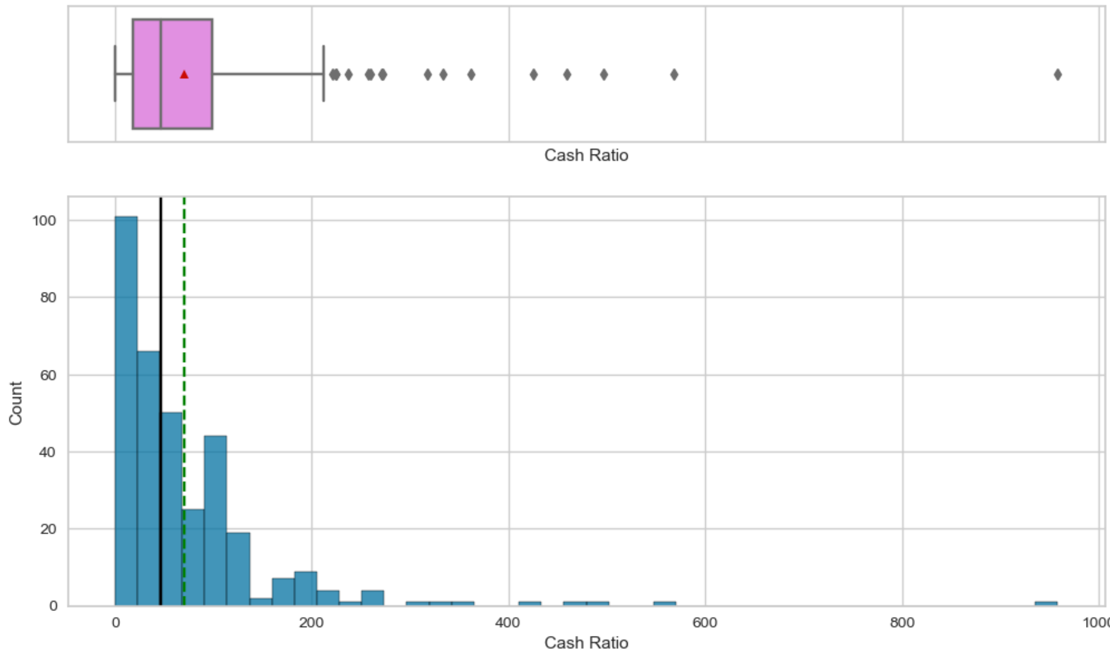


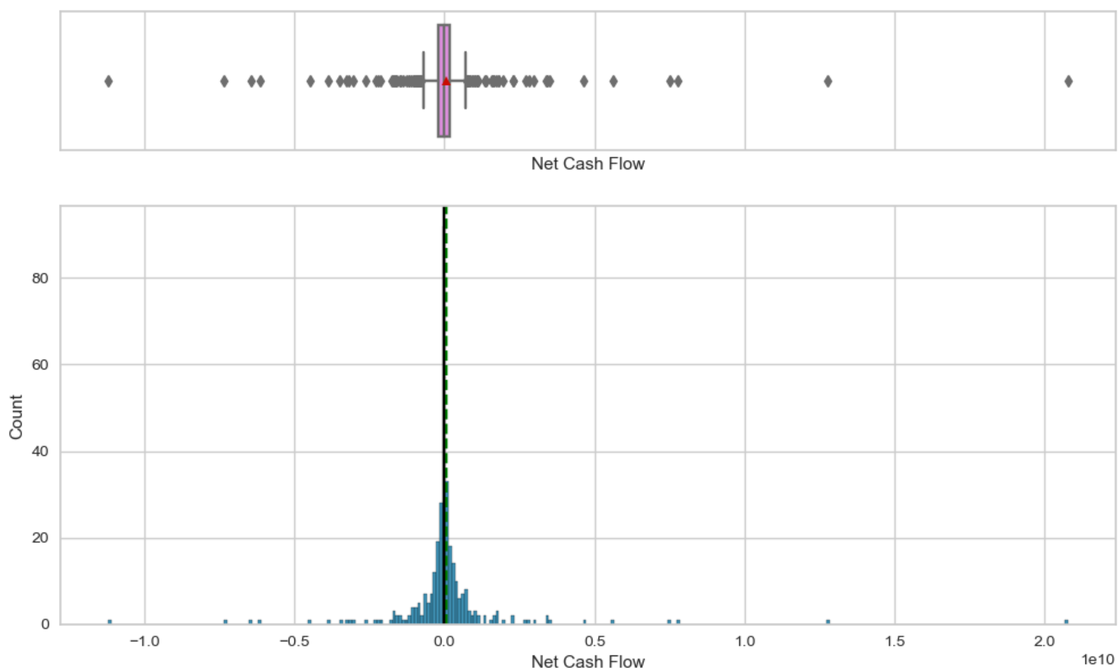


Processing math: 100%

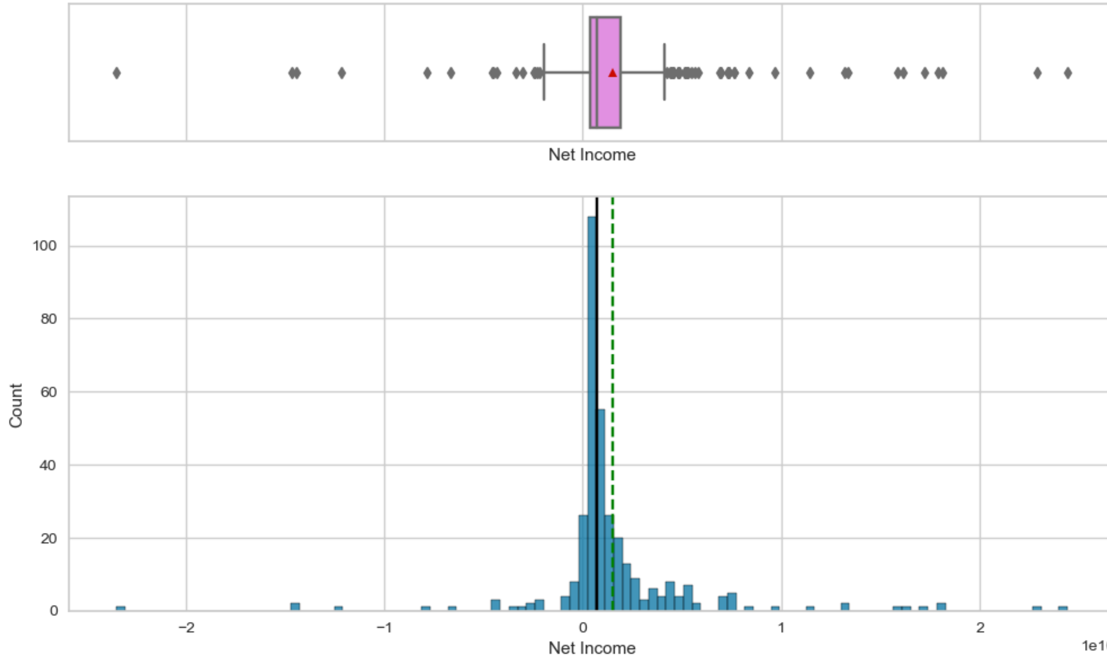

Volatility
Count
Volatility

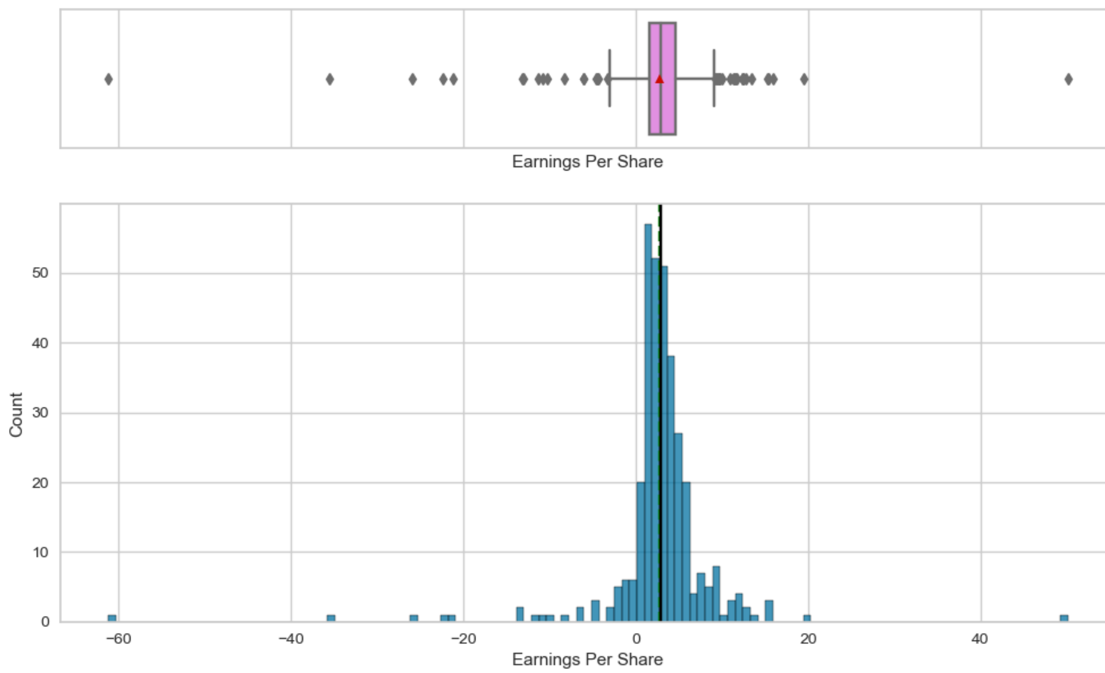

ROE
Count
ROE



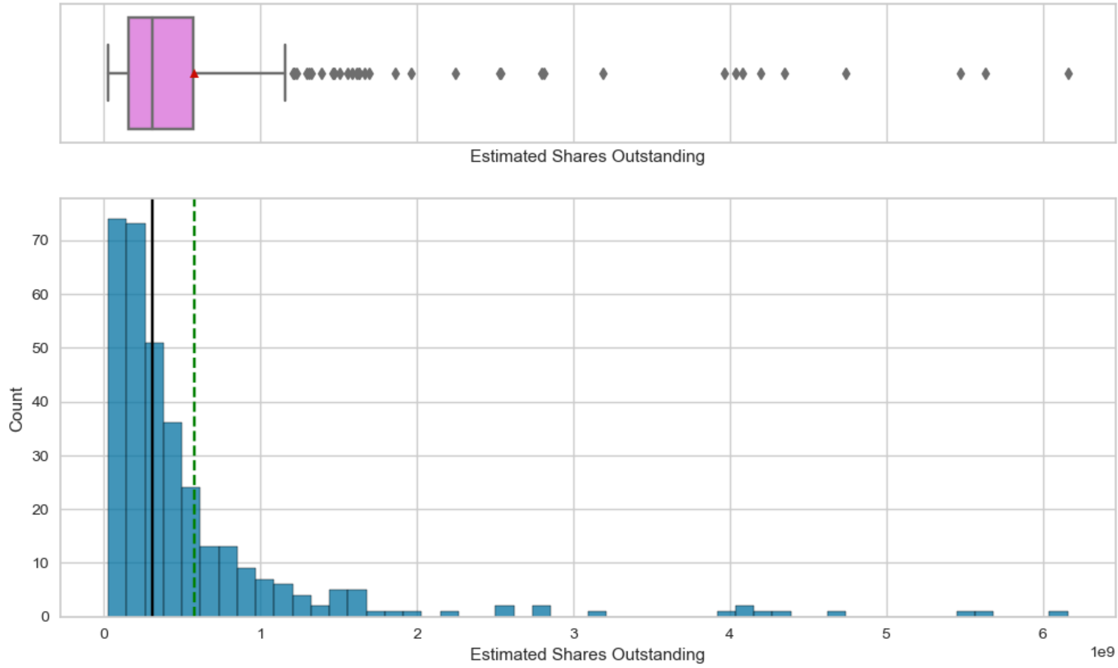


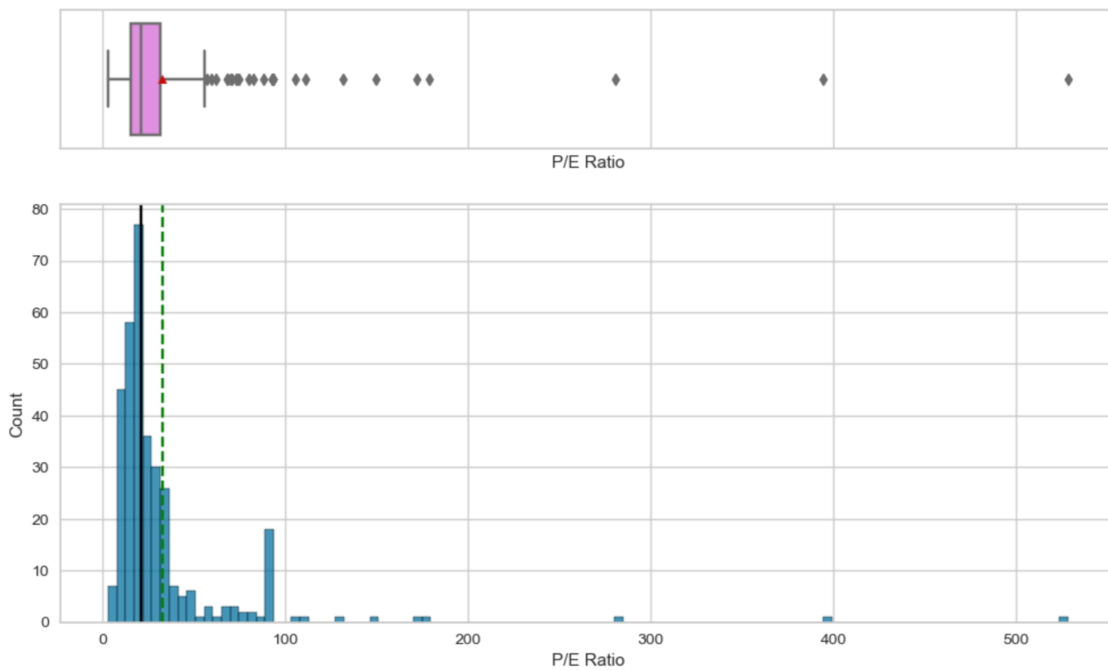
Processing math: 100%

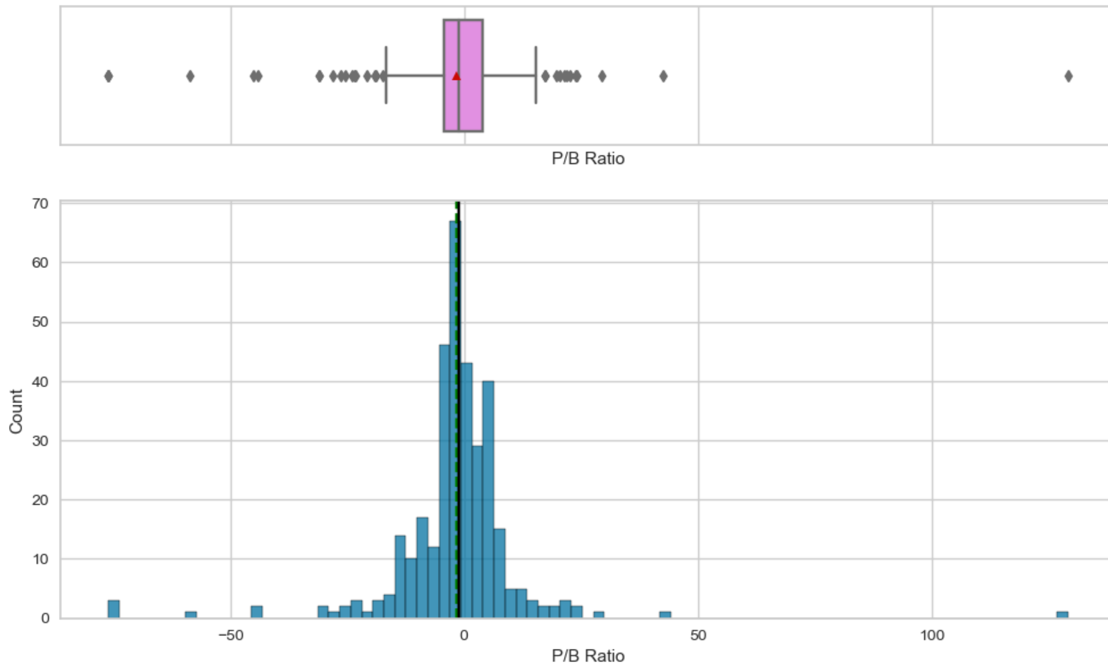




Processing math: 100%


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


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

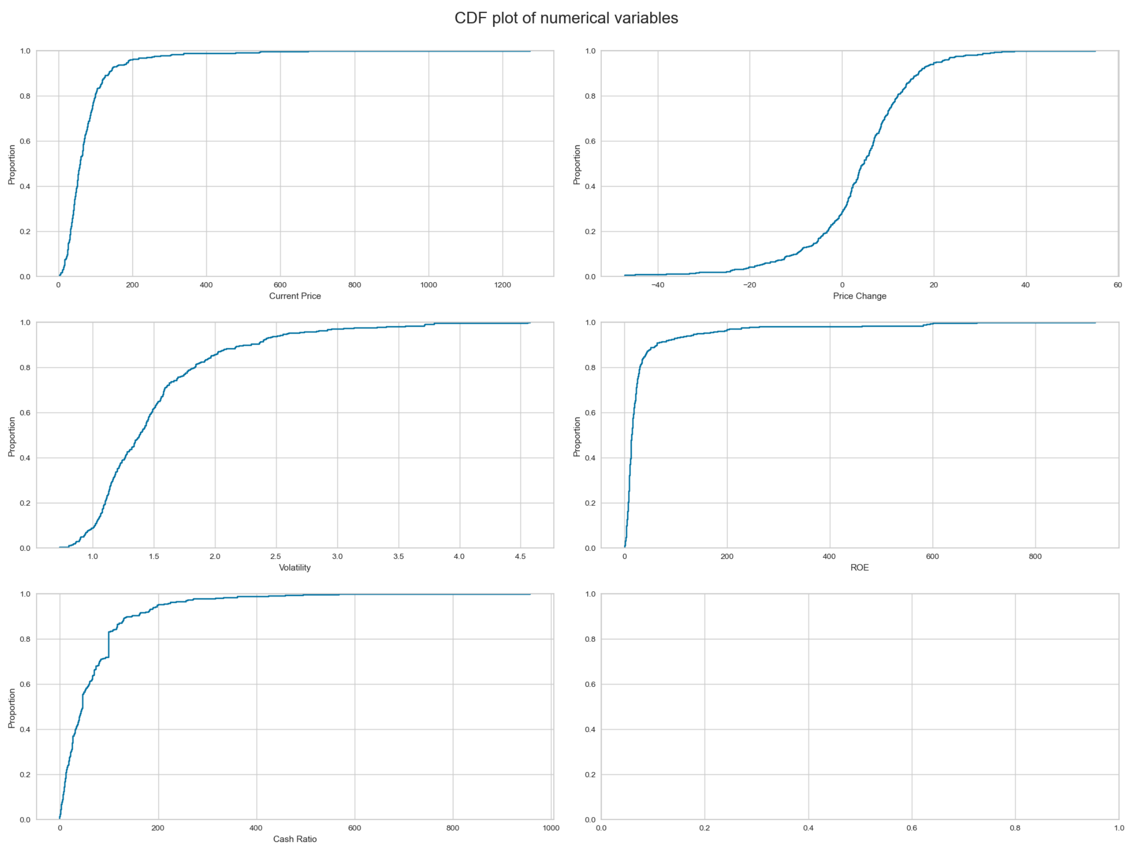


current price and price change seem to be good tools for anylizing stocks

Processing math: 100%

In [16]:

```
fig, axes = plt.subplots(3, 2, figsize=(20, 15))
fig.suptitle("CDF plot of numerical variables", fontsize=20)
counter = 0
for ii in range(3):
    sns.ecdfplot(ax=axes[ii][0], x=df[num_col[counter]])
    counter = counter + 1
    if counter != 5:
        sns.ecdfplot(ax=axes[ii][1], x=df[num_col[counter]])
        counter = counter + 1
    else:
        pass

fig.tight_layout(pad=2.0)
```



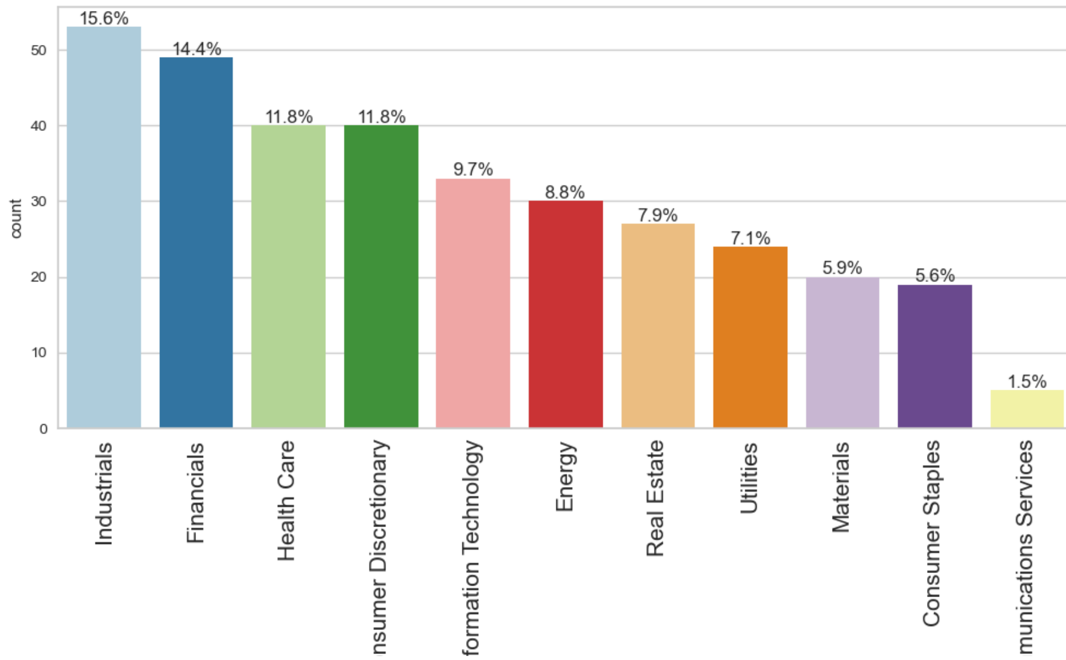
In [17]:

```
# 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 Fals
    n: displays the top n category levels (default is None, i.e., display
    """

    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,
        palette="Paired",
        order=data[feature].value_counts().index[:n],
    )

    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
```

Processing math: 100%

In [18]:

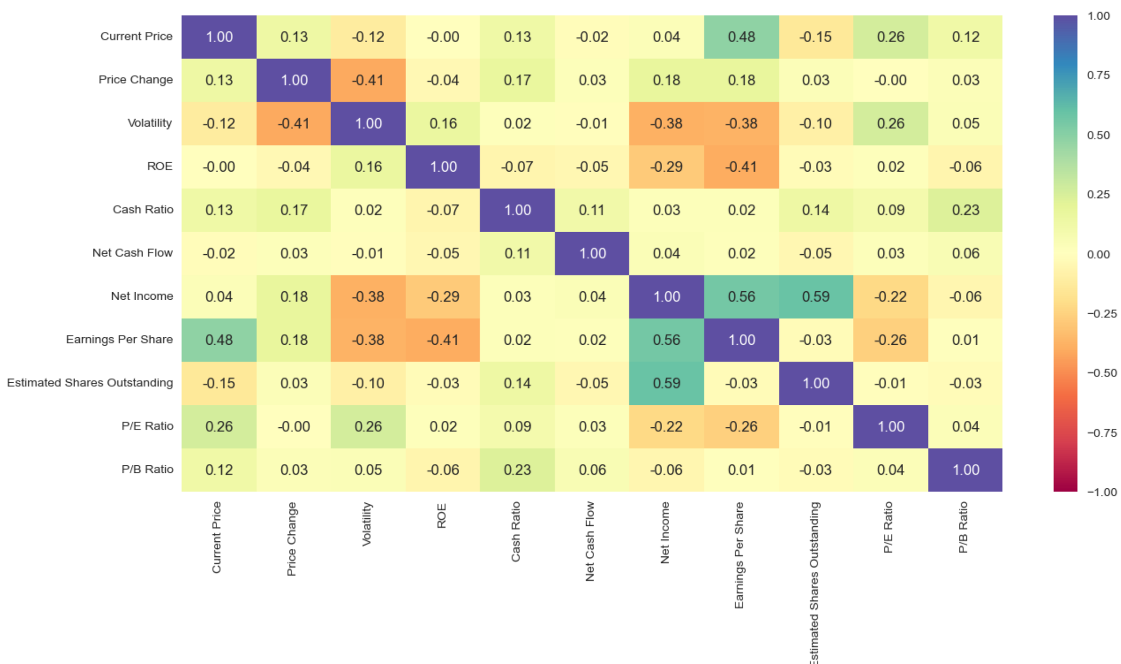
```
labeled_barplot(df, "GICS_Sector", perc=True)
```



industial financial and helth care are the top 3 largest sectors

In [19]:

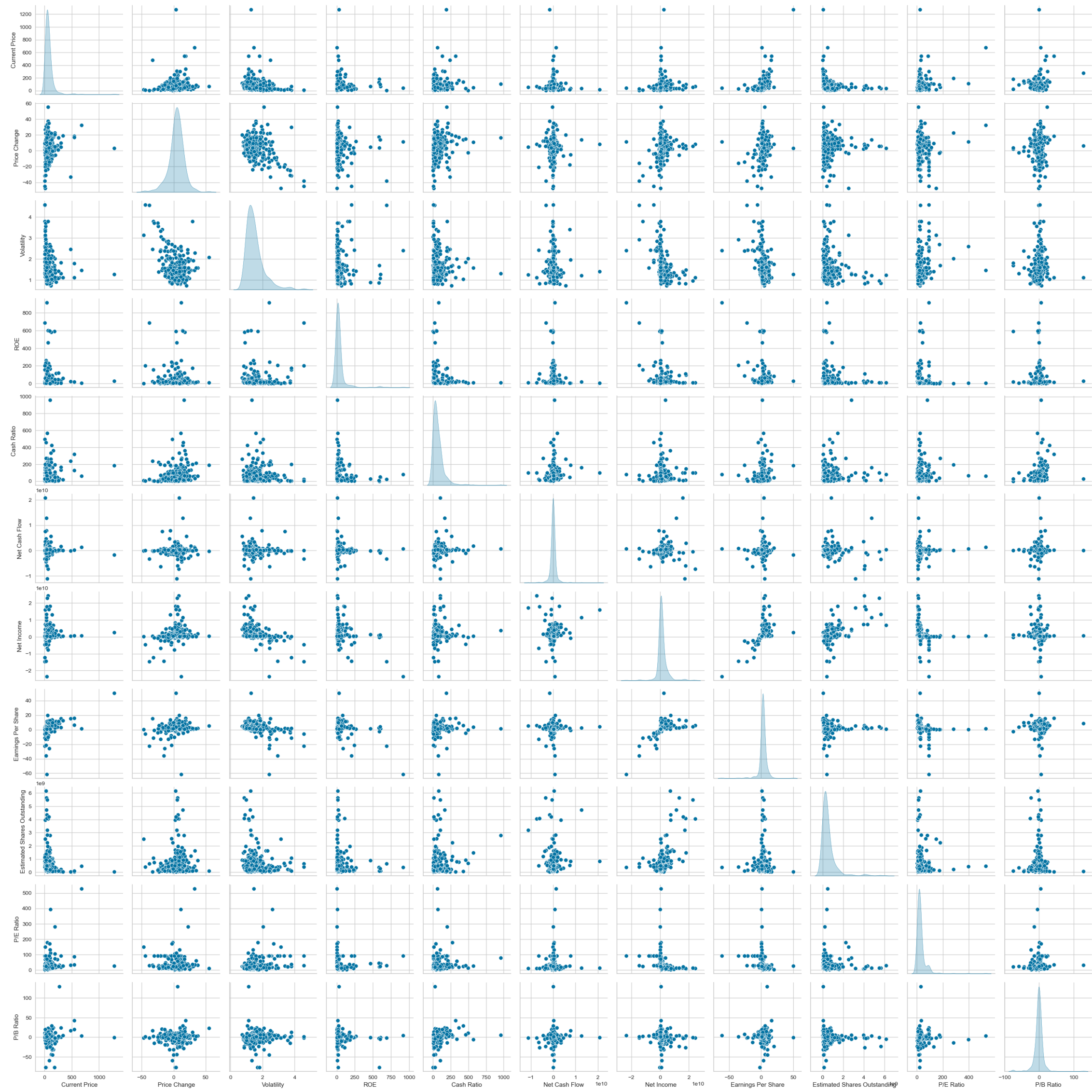
```
plt.figure(figsize=(15, 7))
sns.heatmap(df[num_col].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cm
plt.show()
```



	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

Processing math: 100%

In [20]:
```
sns.pairplot(data=df[num_col], diag_kind="kde")
plt.show()
```



In [21]:
```
# variables used for clustering
num_col
```

Out[21]:
```
['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']
```

Processing math: 100%

In [22]:
```
# scaling the dataset before clustering
scaler = StandardScaler()
subset = df[num_col].copy()
subset_scaled = scaler.fit_transform(subset)
```

In [23]:
```
# creating a dataframe of the scaled columns
subset_scaled_df = pd.DataFrame(subset_scaled, columns=subset.columns)
```

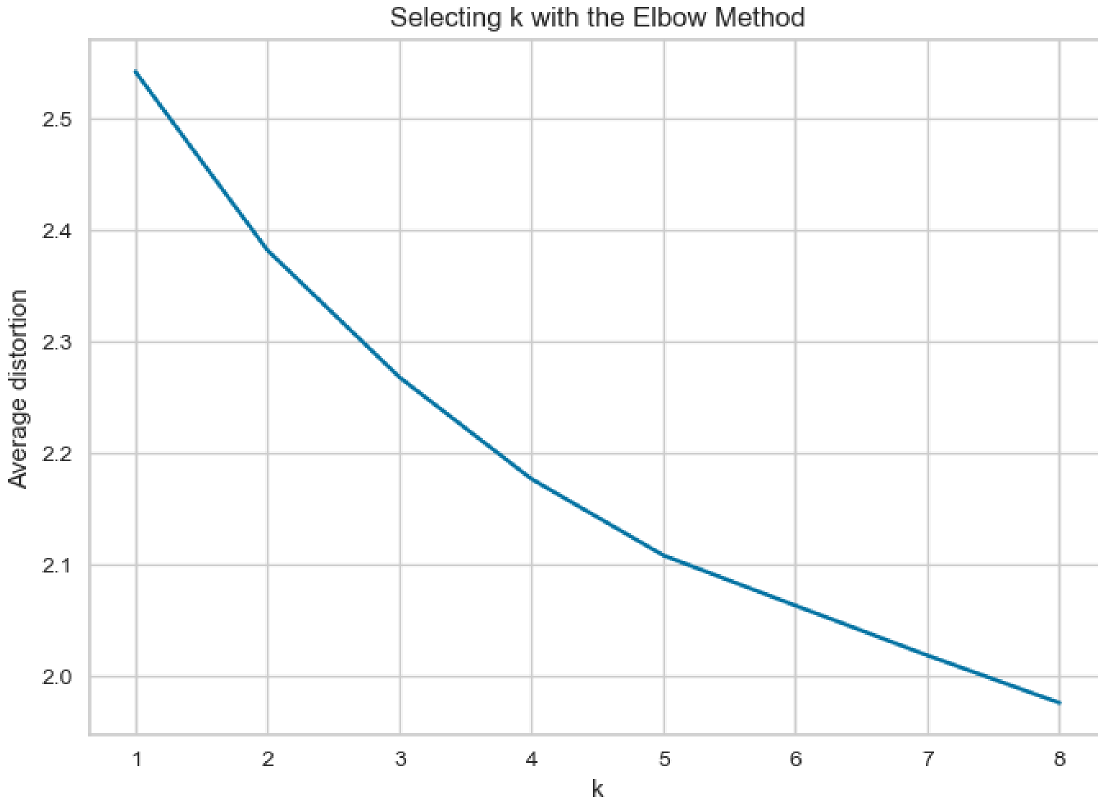
Processing math: 100%

In [24]:

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

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

    meanDistortions.append(distortion)

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

plt.plot(clusters, meanDistortions, "bx-")
plt.xlabel("k")
plt.ylabel("Average distortion")
plt.title("Selecting k with the Elbow Method")
plt.show()
```

```
Number of Clusters: 1   Average Distortion: 2.5425069919221697
Number of Clusters: 2   Average Distortion: 2.382318498894466
Number of Clusters: 3   Average Distortion: 2.2683105560042285
Number of Clusters: 4   Average Distortion: 2.177016653596875
Number of Clusters: 5   Average Distortion: 2.108395860807457
Number of Clusters: 6   Average Distortion: 2.0633627257816647
Number of Clusters: 7   Average Distortion: 2.0186475535112742
Number of Clusters: 8   Average Distortion: 1.9760595422320009
```

Processing math: 100%



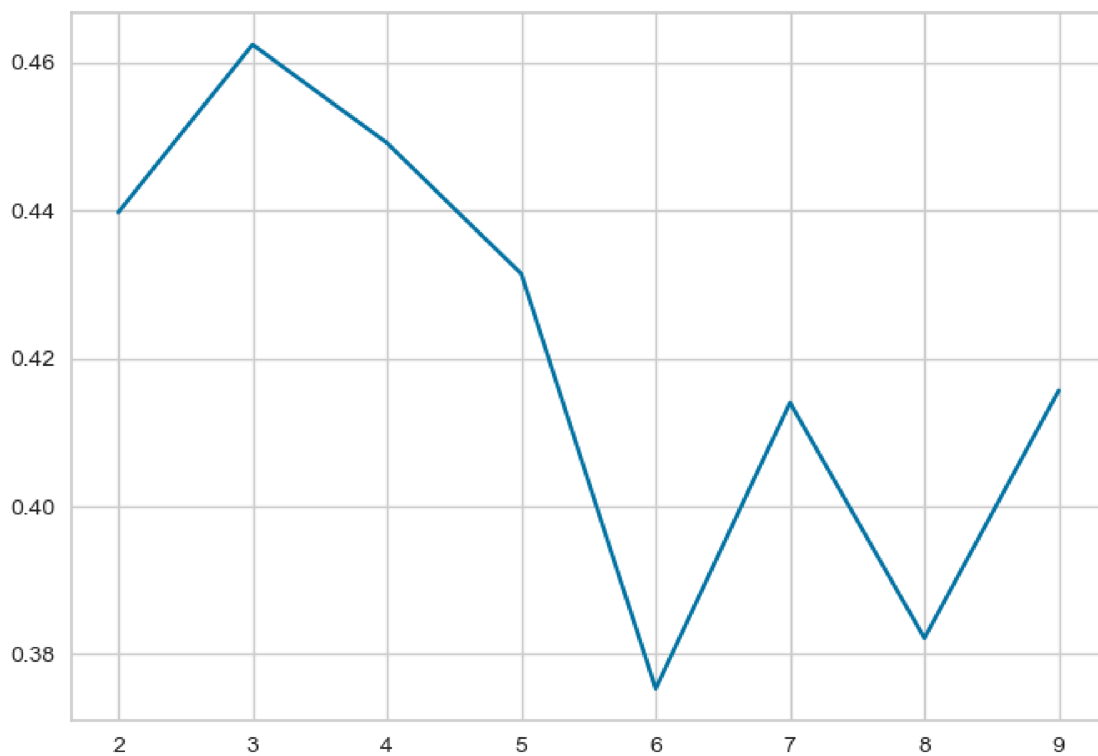
use cluster 8 because it has the lowest Average Distortion: 1.9760595422320009

Processing math: 100%

In [25]:

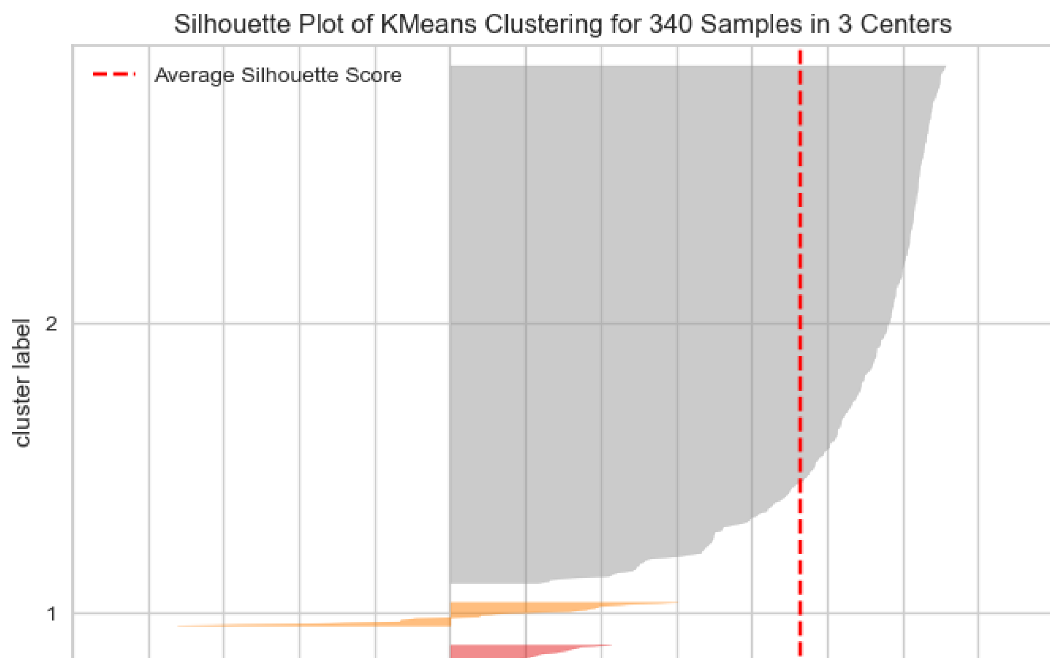
```
sil_score = []
cluster_list = list(range(2, 10))
for n_clusters in cluster_list:
    clusterer = KMeans(n_clusters=n_clusters)
    preds = clusterer.fit_predict((subset_scaled_df))
    # centers = clusterer.cluster_centers_
    score = silhouette_score(subset_scaled_df, preds)
    sil_score.append(score)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

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 0.4623841900167334)
For n_clusters = 4, the silhouette score is 0.4490996460354298)
For n_clusters = 5, the silhouette score is 0.4314106887964818)
For n_clusters = 6, the silhouette score is 0.3753563786475513)
For n_clusters = 7, the silhouette score is 0.4140059404422559)
For n_clusters = 8, the silhouette score is 0.3821962375959941)
For n_clusters = 9, the silhouette score is 0.41563727678873896)
```

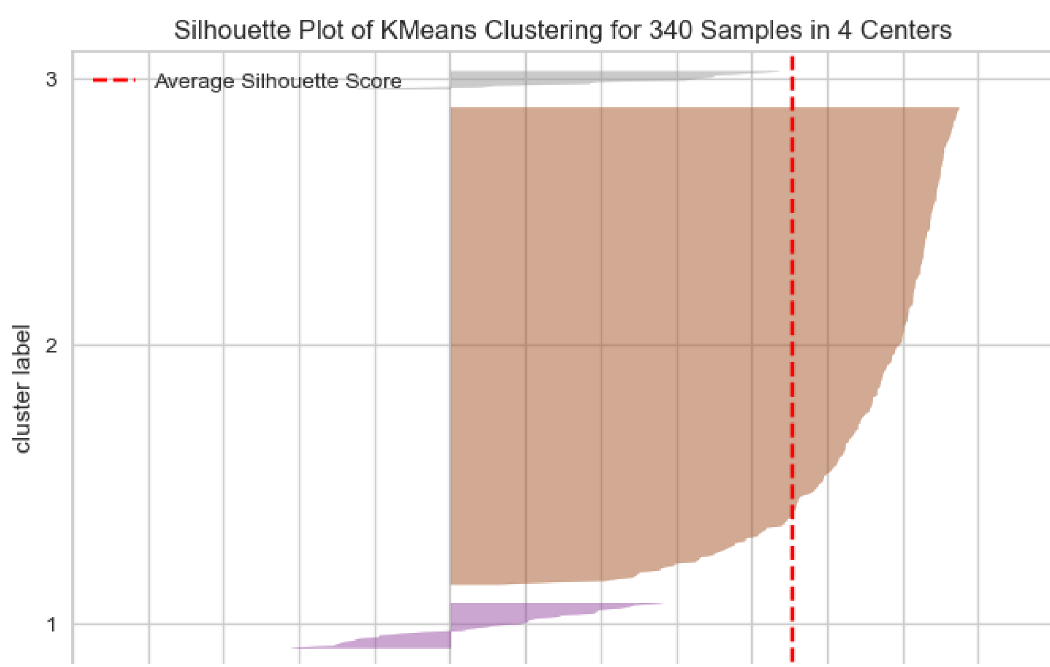


Processing math: 100%

In [26]:
```
# finding optimal no. of clusters with silhouette coefficients
visualizer = SilhouetteVisualizer(KMeans(3, random_state=1))
visualizer.fit(subset_scaled_df)
visualizer.show()
```



In [27]:
```
# finding optimal no. of clusters with silhouette coefficients
visualizer = SilhouetteVisualizer(KMeans(4, random_state=1))
visualizer.fit(subset_scaled_df)
visualizer.show()
```



Processing math: 100%

In [28]:
```python
# let's take 4 as number of clusters
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(subset_scaled_df)
```

Out[28]:
```
KMeans(n_clusters=4, random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [29]:
```python
# adding kmeans cluster labels to the original and scaled dataframes

df["K_means_segments"] = kmeans.labels_
subset_scaled_df["K_means_segments"] = kmeans.labels_
```

In [30]:
```python
cluster_profile = df.groupby("K_means_segments").mean()
```

In [31]:
```python
cluster_profile["count_in_each_segments"] = (
    df.groupby("K_means_segments")["GICS_Sector"].count().values
)
```

In [32]:
```python
# let's display cluster profiles
cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

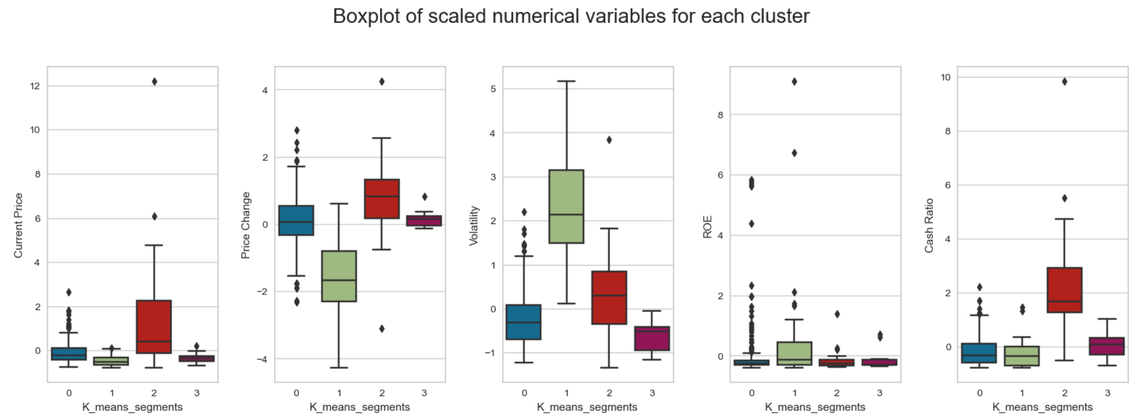
Out[32]:

K_means_segments	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash
0	72.470050	5.059104	1.388717	34.710145	52.938406	-18021028.9
1	35.165385	-16.390175	2.922214	110.961538	49.461538	-192318884.6
2	238.072932	13.508882	1.777479	25.600000	276.280000	752195440.0
3	48.103077	6.053507	1.163964	27.538462	77.230769	773230769.2

Processing math: 100%

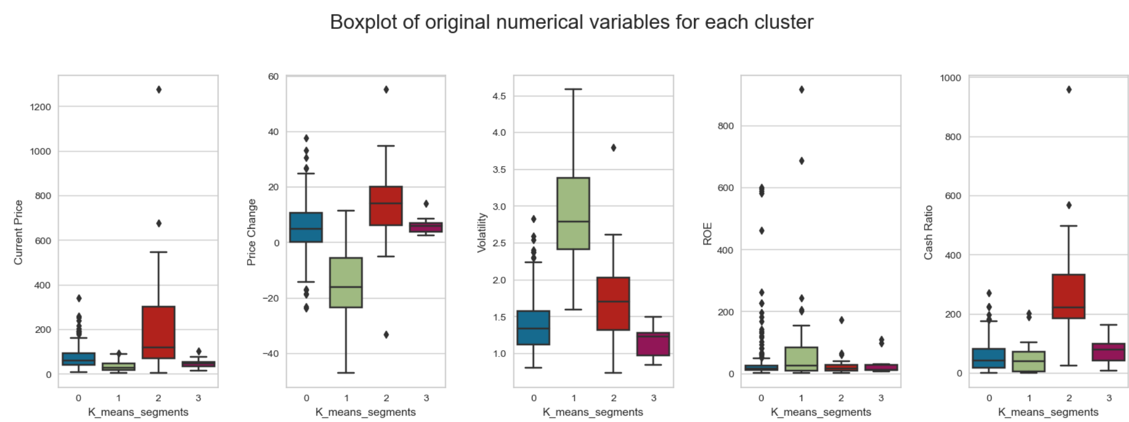
In [33]:

```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of scaled numerical variables for each cluster", fon
counter = 0
for ii in range(5):
    sns.boxplot(
        ax=axes[ii],
        y=subset_scaled_df[num_col[counter]],
        x=subset_scaled_df["K_means_segments"],
    )
    counter = counter + 1

fig.tight_layout(pad=2.0)
```



In [34]:

```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of original numerical variables for each cluster", f
counter = 0
for ii in range(5):
    sns.boxplot(ax=axes[ii], y=df[num_col[counter]], x=df["K_means_segment
    counter = counter + 1

fig.tight_layout(pad=2.0)
```



Processing math: 100%

In [35]:

```
pd.crosstab(df.K_means_segments, data.GICS_Sector).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[35]:

GICS_Sector	Consumer Discretionary	Consumer Staples	Energy	Financials	Health Care	Industrials	Inforr Techr
K_means_segments							
0	33	17	6	45	28	52	
1	0	0	22	0	0	1	
2	6	1	1	0	9	0	
3	1	1	1	4	3	0	

Processing math: 100%

In [36]:

```
# 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 {}."
                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 0.931
5698032555804.
Cophenetic correlation for Euclidean distance and complete linkage is 0.8
317589892879516.
Cophenetic correlation for Euclidean distance and average linkage is 0.94
1150484500575.
Cophenetic correlation for Euclidean distance and weighted linkage is 0.9
183038140391157.
Cophenetic correlation for Chebyshev distance and single linkage is 0.917
8912367024726.
Cophenetic correlation for Chebyshev distance and complete linkage is 0.8
027612142835183.
Cophenetic correlation for Chebyshev distance and average linkage is 0.93
3045222411398.
Cophenetic correlation for Chebyshev distance and weighted linkage is 0.9
122203090096584.
Cophenetic correlation for Mahalanobis distance and single linkage is 0.9
290783147335682.
Cophenetic correlation for Mahalanobis distance and complete linkage is
0.8151435669020113.
Cophenetic correlation for Mahalanobis distance and average linkage is 0.
9354975120525167.
Cophenetic correlation for Mahalanobis distance and weighted linkage is
0.8961307233729855.
Cophenetic correlation for Cityblock distance and single linkage is 0.924
5494341799784.
Cophenetic correlation for Cityblock distance and complete linkage is 0.6
450900595810168.
Cophenetic correlation for Cityblock distance and average linkage is 0.90
80495041593857.
Cophenetic correlation for Cityblock distance and weighted linkage is 0.6
456689300165109.
```

Processing math: 100%

In [37]:

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

```
Highest cophenetic correlation is 0.941150484500575, which is obtained wi
th Euclidean distance and average linkage.
```

The cophenetic correlation is highest for average and centroid linkage methods. We will move ahead with average linkage.

In [38]:

```
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "w

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 0.9315698032555804.
Cophenetic correlation for complete linkage is 0.8317589892879516.
Cophenetic correlation for average linkage is 0.941150484500575.
Cophenetic correlation for centroid linkage is 0.9396689710822762.
Cophenetic correlation for ward linkage is 0.7524063147770085.
Cophenetic correlation for weighted linkage is 0.9183038140391157.
```

In [39]:

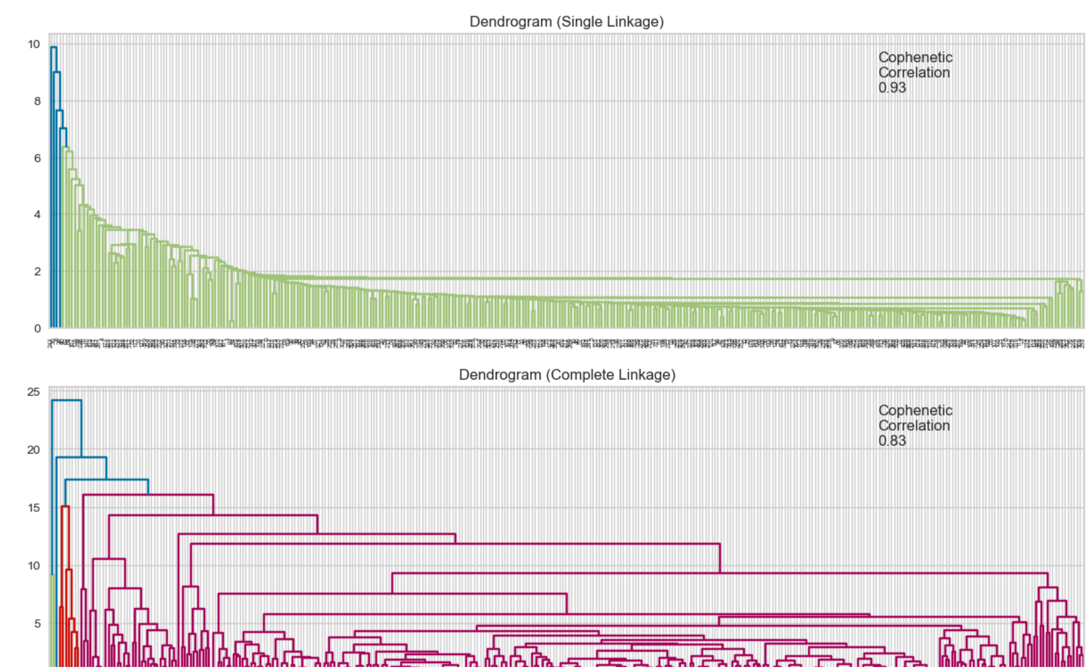
```
# printing the combination of distance metric and linkage method with the 
print(
    "Highest cophenetic correlation is {}, which is obtained with {} linka
        high_cophenet_corr, high_dm_lm[1]
    )
)
```

```
Highest cophenetic correlation is 0.941150484500575, which is obtained wi
th average linkage.
```

Processing math: 100%

In [40]:

```
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "w
# 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 c
for i, method in enumerate(linkage_methods):
    Z = linkage(subset_scaled_df, metric="euclidean", method=method)

    dendrogram(Z, ax=axs[i])
    axs[i].set_title(f"Dendrogram ({method.capitalize()} Linkage)")

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

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



Processing math: 100%

In [41]:
```
# let's create a dataframe to compare cophenetic correlations for each lin
df_cc = pd.DataFrame(compare, columns=compare_cols)
df_cc
```

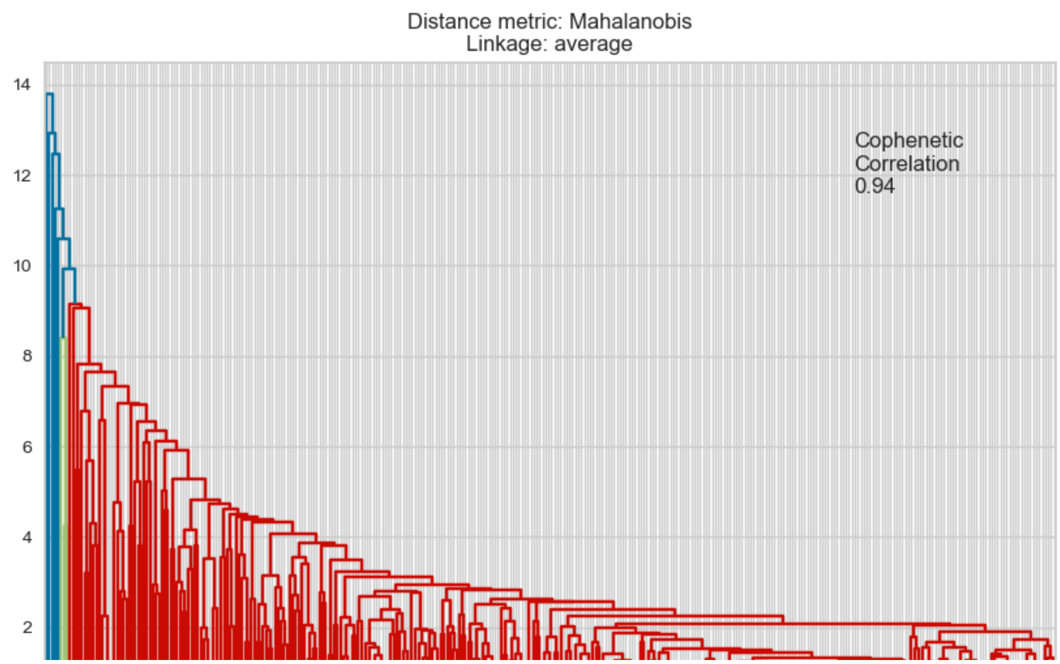
Out[41]:

	Linkage	Cophenetic Coefficient
0	single	0.931570
1	complete	0.831759
2	average	0.941150
3	centroid	0.939669
4	ward	0.752406
5	weighted	0.918304

Processing math: 100%

In [42]:

```
# list of distance metrics
distance_metrics = ["mahalanobis", "cityblock"]

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

# to create a subplot image
fig, axs = plt.subplots(
    len(distance_metrics) + len(distance_metrics), 1, figsize=(10, 30)
)

i = 0
for dm in distance_metrics:
    for lm in linkage_methods:
        Z = linkage(subset_scaled_df, metric=dm, method=lm)

        dendrogram(Z, ax=axs[i])
        axs[i].set_title("Distance metric: {}\nLinkage: {}".format(dm.capi

        coph_corr, coph_dist = cophenet(Z, pdist(subset_scaled_df))
        axs[i].annotate(
            f"Cophenetic\nCorrelation\n{coph_corr:0.2f}",
            (0.80, 0.80),
            xycoords="axes fraction",
        )
        i += 1
```



In [43]:
```
HCmodel = AgglomerativeClustering(n_clusters=3, affinity="euclidean", link
HCmodel.fit(subset_scaled_df)
```

Out[43]:
```
AgglomerativeClustering(affinity='euclidean', n_clusters=3)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [44]:
```
# adding hierarchical cluster labels to the original and scaled dataframes

subset_scaled_df["HC_Clusters"] = HCmodel.labels_
df["HC_Clusters"] = HCmodel.labels_
```

In [45]:
```
cluster_profile = df.groupby("HC_Clusters").mean()
```

In [46]:
```
cluster_profile["count_in_each_segments"] = (
    df.groupby("HC_Clusters")["GICS_Sector"].count().values
)
```

In [47]:
```
# let's display cluster profiles
cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

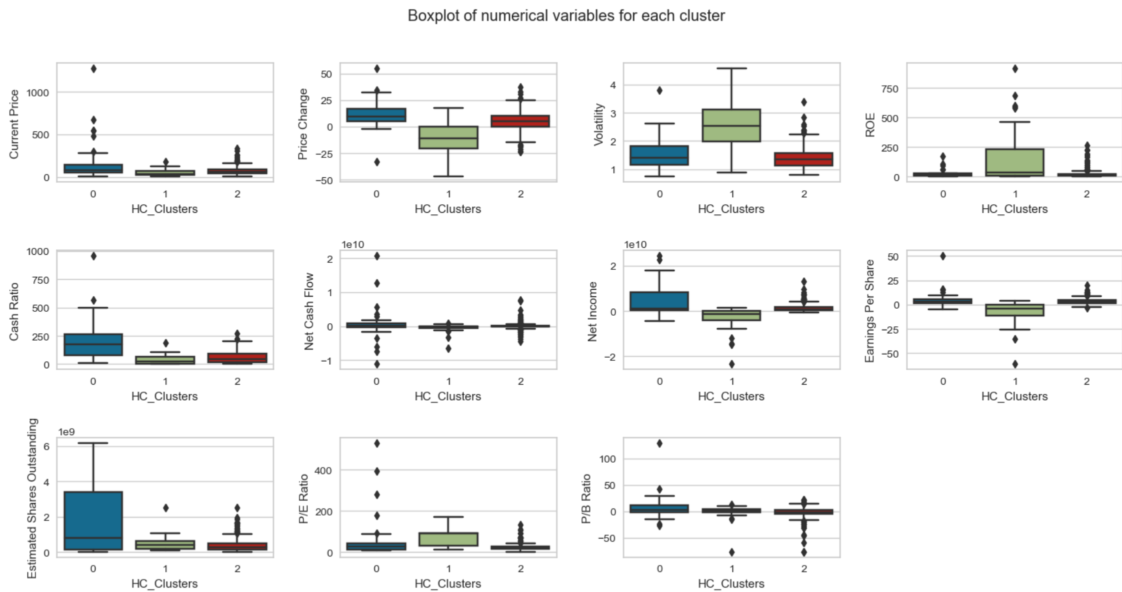
Out[47]:

HC_Clusters	Current Price	Price Change	Volatility	ROE	Cash Ratio	Net Cash Flow
0	173.243702	11.572262	1.549346	26.555556	210.805556	581166277.777778
1	47.081001	-11.442082	2.602381	190.333333	39.300000	-481429800.000000
2	72.423335	4.792872	1.405051	24.806569	54.890511	45268974.452555

Processing math: 100%

In [48]:

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

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

plt.tight_layout(pad=2.0)
```



Business recommendations for hierarchical clustering use cluster 0 Cluster 0 stocks are good places to provide stock advice based on cluster profiling done above. for K-mean clustering use cluster 2 Cluster 2 stocks are good places to provide stock advice based on cluster profiling done above. It also has the best data in the reliveant columns.

Type *Markdown* and LaTeX: α^2

Processing math: 100%