Context:

The data set has information about features of silhouette extracted from the images of different cars Four "Corgie" model vehicles were used for the experiment: a double decker bus, Cheverolet van, Saab 9000 and an Opel Manta 400 cars. This particular combination of vehicles was chosen with the expectation that the bus, van and either one of the cars would be readily distinguishable, but it would be more difficult to distinguish between the cars.

Here let's apply both Hierarchial and K-Means Clustering.

Import the necessary libraries and load the dataset.

In [1]:

```
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
C:\Users\Nimisha\Anaconda3\lib\site-packages\statsmodels\tools\_testing.py:1
9: FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [2]:

```
df = pd.read_csv("vehicle.csv")
df.head()
```

Out[2]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_as
0	95	48.0	83.0	178.0	72.0	
1	91	41.0	84.0	141.0	57.0	
2	104	50.0	106.0	209.0	66.0	
3	93	41.0	82.0	159.0	63.0	
4	85	44.0	70.0	205.0	103.0	

Q1. Check for missing values in the dataset.

In [3]:
```
df.isna().sum()
```

Out[3]:
```
compactness                    0
circularity                    5
distance_circularity           4
radius_ratio                   6
pr.axis_aspect_ratio           2
max.length_aspect_ratio        0
scatter_ratio                  1
elongatedness                  1
pr.axis_rectangularity         3
max.length_rectangularity      0
scaled_variance                3
scaled_variance.1              2
scaled_radius_of_gyration      2
scaled_radius_of_gyration.1    4
skewness_about                 6
skewness_about.1               1
skewness_about.2               1
hollows_ratio                  0
class                          0
dtype: int64
```

Q2. Drop the missing values.

Note: [Use the dataset thus created after dropping missing values for the clustering algorithms.]

In [4]:
```
df = df.dropna()
```

In [5]:
```
df.isna().sum()
```

Out[5]:
```
compactness                    0
circularity                    0
distance_circularity           0
radius_ratio                   0
pr.axis_aspect_ratio           0
max.length_aspect_ratio        0
scatter_ratio                  0
elongatedness                  0
pr.axis_rectangularity         0
max.length_rectangularity      0
scaled_variance                0
scaled_variance.1              0
scaled_radius_of_gyration      0
scaled_radius_of_gyration.1    0
skewness_about                 0
skewness_about.1               0
skewness_about.2               0
hollows_ratio                  0
class                          0
dtype: int64
```

Q3. Check the shape (rows and columns), info and the basic measures of descriptive statistics from the data.

In [6]:
```
print('The number of rows of the dataframe is',df.shape[0],'.')

print('The number of columns of the dataframe is',df.shape[1],'.')
```

```
The number of rows of the dataframe is 813 .
The number of columns of the dataframe is 19 .
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 813 entries, 0 to 845
Data columns (total 19 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   compactness                  813 non-null    int64  
 1   circularity                  813 non-null    float64
 2   distance_circularity         813 non-null    float64
 3   radius_ratio                 813 non-null    float64
 4   pr.axis_aspect_ratio         813 non-null    float64
 5   max.length_aspect_ratio      813 non-null    int64  
 6   scatter_ratio                813 non-null    float64
 7   elongatedness                813 non-null    float64
 8   pr.axis_rectangularity       813 non-null    float64
 9   max.length_rectangularity    813 non-null    int64  
 10  scaled_variance              813 non-null    float64
 11  scaled_variance.1            813 non-null    float64
 12  scaled_radius_of_gyration    813 non-null    float64
 13  scaled_radius_of_gyration.1  813 non-null    float64
 14  skewness_about               813 non-null    float64
 15  skewness_about.1             813 non-null    float64
 16  skewness_about.2             813 non-null    float64
 17  hollows_ratio                813 non-null    int64  
 18  class                        813 non-null    object 
dtypes: float64(14), int64(4), object(1)
memory usage: 127.0+ KB
```

In [8]:
```python
df.describe()
```

Out[8]:

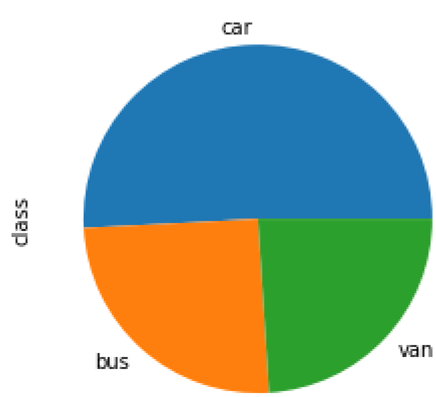
	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.lengt
count	813.000000	813.000000	813.00000	813.000000	813.000000	
mean	93.656827	44.803198	82.04305	169.098401	61.774908	
std	8.233751	6.146659	15.78307	33.615402	7.973000	
min	73.000000	33.000000	40.00000	104.000000	47.000000	
25%	87.000000	40.000000	70.00000	141.000000	57.000000	
50%	93.000000	44.000000	79.00000	167.000000	61.000000	
75%	100.000000	49.000000	98.00000	195.000000	65.000000	
max	119.000000	59.000000	112.00000	333.000000	138.000000	

Q4. Print/Plot the dependent (categorical variable) and Check for any missing values in the data

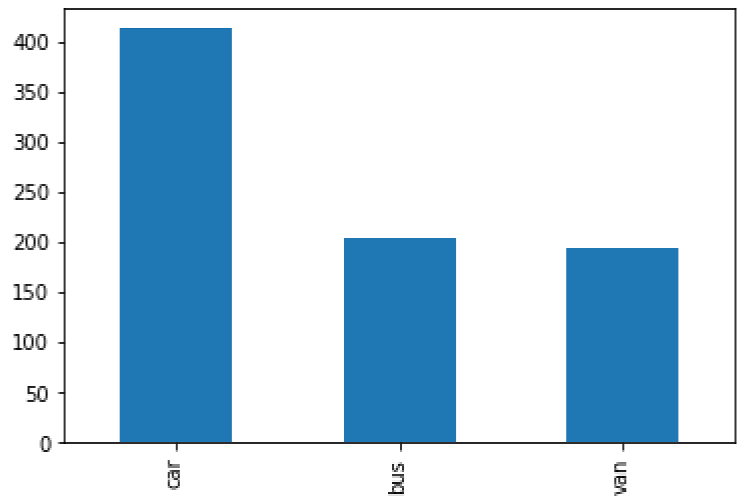
In [9]:
```python
#Since the variable is categorical, you can use value_counts function
pd.value_counts(df['class'])
```

Out[9]:
```
car    413
bus    205
van    195
Name: class, dtype: int64
```

In [10]:
```python
pd.value_counts(df["class"]).plot(kind="pie")
plt.show()
```



In [11]:
```
pd.value_counts(df["class"]).plot(kind="bar")
plt.show()
```



Q4. Standardize the data.

Drop the categorical variable before clustering the data.

In [12]:
```
DF = df.drop('class', axis=1)
```

In [13]:
```
from sklearn.preprocessing import StandardScaler
X = StandardScaler()
scaled_DF = X.fit_transform(DF)
scaled_DF
```

Out[13]:
```
array([[ 0.16323063,  0.52040788,  0.06066872, ...,  0.37128716,
        -0.3218087 ,  0.17183708],
       [-0.32287376, -0.61912319,  0.12406675, ...,  0.14710858,
         0.00340009,  0.44231829],
       [ 1.2569655 ,  0.84598818,  1.51882349, ..., -0.41333788,
        -0.1592043 ,  0.03659647],
       ...,
       [ 1.5000177 ,  1.49714879,  1.20183332, ..., -0.97378433,
        -0.3218087 ,  0.7127995 ],
       [-0.93050425, -1.43307395, -0.25632145, ...,  1.38009078,
         0.16600449, -0.09864413],
       [-1.05203035, -1.43307395, -1.01709784, ...,  0.59546574,
        -0.4844131 , -0.77484716]])
```

Now that we have scaled the data. Let us create a dataframe out of this scaled variables for clustering.

In [14]:
```
scaled_DF = pd.DataFrame(scaled_DF, index=DF.index, columns=DF.columns)
scaled_DF.head()
```

Out[14]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_asp
0	0.163231	0.520408	0.060669	0.264970	1.283254	
1	-0.322874	-0.619123	0.124067	-0.836393	-0.599253	
2	1.256966	0.845988	1.518823	1.187734	0.530251	
3	-0.079822	-0.619123	-0.002729	-0.300595	0.153750	
4	-1.052030	-0.130753	-0.763506	1.068668	5.173770	

OR

In [15]:
```
from scipy.stats import zscore
scaled_df = DF.apply(zscore)
scaled_df.head()
```

Out[15]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_asp
0	0.163231	0.520408	0.060669	0.264970	1.283254	
1	-0.322874	-0.619123	0.124067	-0.836393	-0.599253	
2	1.256966	0.845988	1.518823	1.187734	0.530251	
3	-0.079822	-0.619123	-0.002729	-0.300595	0.153750	
4	-1.052030	-0.130753	-0.763506	1.068668	5.173770	

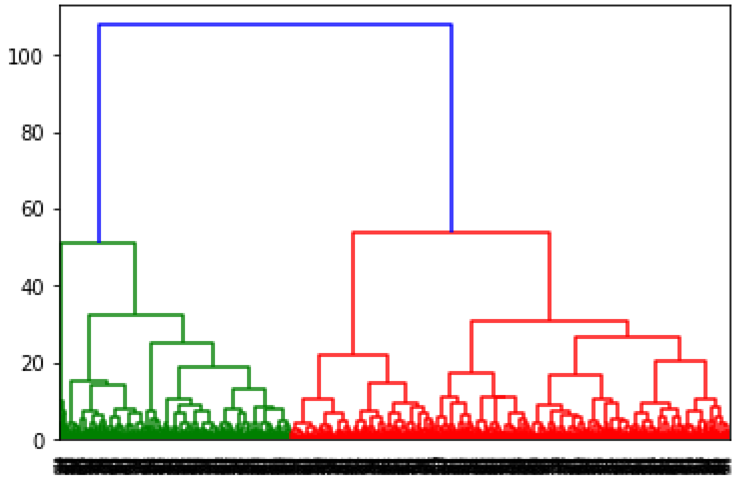
Q5. Perform Hierarchical Clustering with the Ward's linkage method and plot the dendrogram.

Note: Please do go ahead and explore other parameters under the linkage function in the Scientific Python library.

In [16]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
```

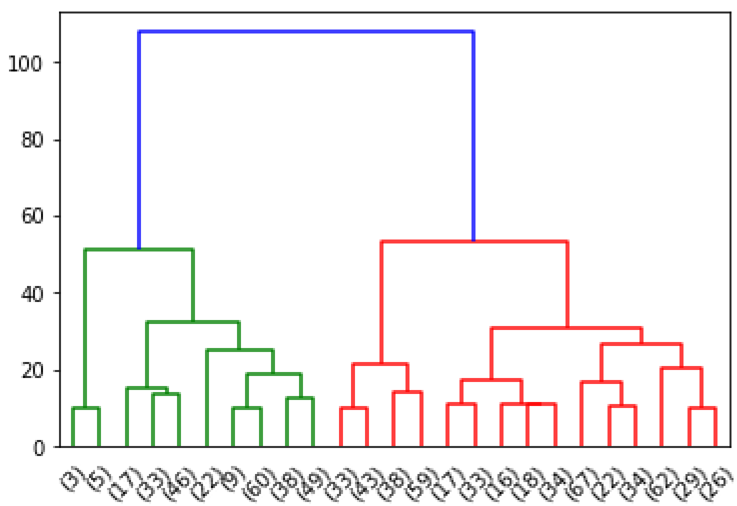
In [17]:
```
HClust = linkage(scaled_DF, method = 'ward')
```

In [18]:
```python
dend = dendrogram(HClust)
```



Q6. Plot the truncated dendrogram with the last 25 clusters.

In [19]:
```python
dend = dendrogram(HClust,
                  truncate_mode='lastp',
                  p = 25,# we are looking at the last 25 merges
                  )
```



Q7. Identify the number of clusters based on the dendrogram and add the cluster numbers to the original dataframe.

In [20]:
```python
from scipy.cluster.hierarchy import fcluster
```

In [21]:

```python
#Method 1

clusters_1 = fcluster(HClust, 2, criterion='maxclust')
clusters_1
```

Out[21]:

```
array([2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1,
       2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1,
       2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1,
       2, 2, 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1,
       2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1,
       2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 2, 1,
       2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2,
       2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 2, 1, 2,
       2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1,
       2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2,
       1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2,
       1, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 2, 1,
       2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1,
       2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2,
       1, 2, 1, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 1,
       1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2,
       1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2,
       1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 2, 1,
       1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 1, 1,
       2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 1,
       1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2,
       1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1, 2,
       2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1,
       2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1,
       2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1,
       2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2,
       1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1,
       1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2,
       2, 1, 1, 2, 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 1,
       2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 1,
       2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2],
      dtype=int32)
```

In [22]:
```python
# Method 2

clusters_2 = fcluster(HClust, 60, criterion='distance')
clusters_2
```

Out[22]:
```
array([2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1,
       2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1,
       2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1,
       2, 2, 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1,
       2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1,
       2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 2, 1,
       2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2,
       2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 2, 1, 2,
       2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1,
       2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2,
       1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2,
       1, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 2, 1,
       2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1,
       2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2,
       1, 2, 1, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 1,
       1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2,
       1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2,
       1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 2, 1,
       1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 1, 1,
       2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 1,
       1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2,
       1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1, 2,
       2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1,
       2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1,
       2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1,
       2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2,
       1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1,
       1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2,
       2, 1, 1, 2, 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 1,
       2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 1,
       2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2],
      dtype=int32)
```

Now, let us go ahead and check whether the number of clusters generated by the 'maxclust' criterion is same as the number of clusters generated by the 'distance' criterion.

In [23]:
```python
np.array_equal(clusters_1,clusters_2)
```

Out[23]:
```
True
```

In [24]:
```python
DF['H_clusters'] = clusters_1
```

In [25]:
```
DF.head()
```

Out[25]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_asp
0	95	48.0	83.0	178.0	72.0	
1	91	41.0	84.0	141.0	57.0	
2	104	50.0	106.0	209.0	66.0	
3	93	41.0	82.0	159.0	63.0	
4	85	44.0	70.0	205.0	103.0	

Q8. Export the dataframe thus created with the clusters into a csv file.

In [26]:
```
df.to_csv('H_Cluster.csv')
```

Q9. Perform the K-Means clustering with 2 clusters.

In [27]:
```
from sklearn.cluster import KMeans
```

In [28]:
```
k_means2 = KMeans(n_clusters = 2,random_state=1)
k_means2.fit(scaled_DF)
k_means2.labels_
```

Out[28]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0])
```

Q10. Find out the within cluster sum of squares for 2 clusters for the K-Means algorithm.

In [29]:
```
k_means2.inertia_
```

Out[29]:
```
8623.136975986425
```

Q11. Perform the K-Means clustering with 3 clusters and find out the within cluster sum of squares.

In [30]:
```
k_means3 = KMeans(n_clusters = 3,random_state=1)
k_means3.fit(scaled_DF)
k_means3.inertia_
```

Out[30]:
```
7037.287609421165
```

Q13. Find the Within Sum of Squares (WSS) for 2 to 15 clusters.

In [31]:
```
wss =[]
```

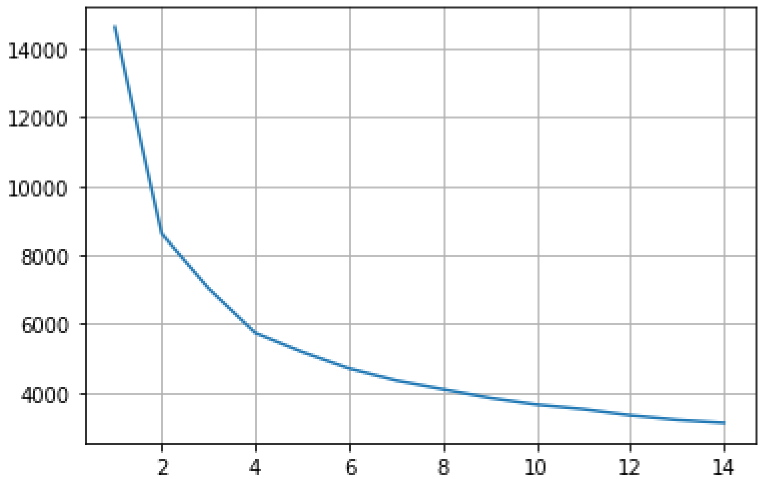
In [32]:
```
for i in range(1,15):
    KM = KMeans(n_clusters=i,random_state=1)
    KM.fit(scaled_DF)
    wss.append(KM.inertia_)
```

In [33]:
```
wss
```

Out[33]:
```
[14634.000000000007,
 8623.136975986425,
 7037.287609421163,
 5739.1692850380905,
 5186.567403420996,
 4707.768043378601,
 4355.139333861276,
 4097.9611968419185,
 3846.533696572564,
 3652.2464519244977,
 3518.5716809545643,
 3340.5704337458487,
 3208.1790781086697,
 3120.8168969715985]
```

Q14. Plot the Within Sum of Squares (WSS) plot using the values of 'inertia' computed in the last question.

In [34]:
```
plt.plot(range(1,15), wss)
plt.grid()
plt.show()
```



Q15. Find the optimum number of clusters from the WSS plot in the previous question.

Firstly, we will check with 2 clusters.

In [35]:
```python
k_means = KMeans(n_clusters = 2,random_state=1)
k_means.fit(scaled_DF)
labels = k_means.labels_
labels
```

Out[35]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0])
```

Now, let us check with 4 clusters.

In [36]:
```
k_means4 = KMeans(n_clusters = 4,random_state=1)
k_means4.fit(scaled_DF)
labels_4 = k_means4.labels_
labels_4
```

Out[36]:
```
array([0, 0, 1, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 2, 2, 0, 0, 1,
       0, 2, 1, 1, 2, 0, 0, 0, 1, 0, 2, 3, 1, 2, 1, 2, 2, 0, 1, 2, 2, 2,
       2, 0, 2, 0, 1, 0, 1, 0, 0, 2, 1, 2, 1, 2, 2, 2, 0, 2, 1, 0, 1, 1,
       0, 2, 0, 1, 0, 2, 2, 1, 0, 2, 0, 1, 0, 2, 0, 2, 1, 0, 1, 0, 2, 1,
       2, 2, 1, 2, 3, 0, 0, 2, 1, 1, 2, 2, 1, 0, 0, 2, 2, 2, 0, 1, 1, 0,
       2, 2, 0, 2, 2, 2, 2, 2, 0, 1, 1, 0, 0, 2, 1, 3, 2, 0, 2, 0, 0, 1,
       2, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 2, 0, 2, 1, 0, 0, 1, 1,
       0, 1, 2, 2, 1, 1, 0, 1, 0, 0, 0, 0, 2, 1, 2, 0, 2, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 2, 1, 2, 2, 2, 0, 0, 1, 1, 0, 0, 0, 2, 1, 0, 0, 0,
       1, 2, 2, 1, 2, 0, 1, 2, 2, 2, 0, 1, 0, 1, 2, 2, 2, 2, 1, 0, 2, 0,
       1, 2, 0, 0, 2, 1, 2, 2, 0, 0, 1, 2, 1, 2, 0, 0, 1, 0, 0, 1, 1, 2,
       0, 0, 0, 1, 2, 0, 0, 2, 2, 0, 0, 1, 0, 2, 2, 1, 0, 0, 2, 2, 1, 2,
       0, 1, 2, 0, 3, 0, 0, 1, 0, 1, 2, 0, 0, 1, 0, 0, 0, 2, 0, 1, 1, 1,
       1, 2, 0, 1, 2, 2, 2, 0, 2, 1, 1, 2, 1, 0, 2, 1, 2, 0, 0, 1, 1, 2,
       1, 1, 2, 1, 0, 0, 0, 2, 2, 1, 1, 1, 0, 0, 0, 1, 2, 0, 2, 1, 0, 0,
       1, 0, 1, 1, 1, 0, 2, 2, 1, 2, 2, 2, 0, 0, 0, 0, 2, 1, 1, 2, 2, 1,
       2, 1, 2, 1, 0, 2, 0, 2, 3, 1, 2, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2, 1, 0, 0, 2, 2, 1, 0, 2, 0, 0,
       1, 0, 1, 0, 1, 1, 2, 2, 1, 0, 2, 2, 0, 1, 1, 2, 0, 1, 1, 2, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 2, 2, 0, 1, 0, 0, 1, 0, 2, 1, 2, 2, 1, 0, 2,
       1, 1, 1, 2, 1, 1, 2, 0, 2, 1, 1, 0, 0, 2, 2, 1, 0, 2, 1, 1, 0, 2,
       1, 1, 0, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 0, 0, 1, 2, 0, 1, 0, 2, 2,
       0, 1, 2, 0, 0, 2, 3, 1, 0, 1, 1, 0, 2, 0, 1, 1, 2, 2, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 2, 2, 2, 0, 0, 1, 2, 2, 0, 2, 1, 0, 1, 2, 2, 1, 1,
       0, 1, 0, 0, 0, 1, 0, 2, 0, 1, 0, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 1,
       1, 1, 0, 1, 2, 0, 1, 2, 2, 2, 0, 2, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 2, 1, 2, 2, 0, 2, 0, 0, 2, 2, 1, 1, 2, 0, 1, 0, 1, 0,
       0, 1, 0, 2, 1, 2, 1, 2, 2, 0, 2, 0, 1, 1, 2, 1, 0, 0, 2, 0, 2, 1,
       0, 1, 2, 0, 0, 0, 2, 2, 2, 0, 1, 0, 1, 2, 0, 0, 0, 0, 1, 0, 2, 1,
       0, 1, 0, 0, 1, 2, 1, 2, 0, 2, 0, 2, 1, 0, 2, 0, 1, 2, 1, 0, 2, 1,
       2, 0, 2, 0, 0, 2, 0, 1, 1, 0, 0, 1, 1, 3, 0, 2, 0, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 2, 1, 0, 2, 1, 1, 1, 0, 1, 2, 2, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 2, 0, 2, 0, 1, 0, 2, 0, 0, 0, 2, 1, 2, 2,
       2, 1, 1, 2, 1, 1, 2, 0, 0, 1, 0, 2, 1, 1, 2, 0, 0, 1, 1, 1, 2, 1,
       0, 1, 1, 2, 2, 1, 2, 1, 0, 2, 0, 1, 1, 0, 2, 0, 1, 1, 0, 0, 2, 0,
       0, 1, 2, 0, 1, 2, 2, 1, 2, 0, 2, 2, 2, 0, 1, 1, 0, 2, 1, 0, 1, 1,
       2, 0, 1, 2, 2, 0, 0, 1, 2, 2, 1, 2, 0, 0, 0, 0, 0, 0, 1, 0, 2])
```

Q16. Check the average silhouette score and silhouette width of the cluster(s) thus created.

In [37]:
```
DF_Kmeans = DF.drop('H_clusters',axis=1)
```

In [38]:
```
from sklearn.metrics import silhouette_samples, silhouette_score
```

Let us check the silhouette score and silhouette width for 2 clusters.

In [39]:
```
silhouette_score(scaled_DF,labels)
```

Out[39]: 0.38978847975148845

In [40]:
```
silhouette_samples(scaled_DF,labels).min()
```

Out[40]: 0.00036697237344667964

Let us check the silhouette score and silhouette width for 4 clusters.

In [41]:
```
silhouette_score(scaled_DF,labels_4)
```

Out[41]: 0.3044797739071198

In [42]:
```
silhouette_samples(scaled_DF,labels_4).min()
```

Out[42]: -0.03412954997487397

Q17. Add the cluster labels to the dataset which has the cluster labels of Hierarchical Clustering.

Here, we will be going with 2 clusters from the K-Means Clustering as well. This is based on the Silhouette Score and Silhouette width.

In [43]:
```
DF['Kmeans_clusters'] = labels
```

In [44]:
```
DF.head()
```

Out[44]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_as
0	95	48.0	83.0	178.0	72.0	
1	91	41.0	84.0	141.0	57.0	
2	104	50.0	106.0	209.0	66.0	
3	93	41.0	82.0	159.0	63.0	
4	85	44.0	70.0	205.0	103.0	

Q18. Export the new dataframe with both the cluster labels of Hierarchical Clustering and K-Means clustering into a csv. Do not include the 'class' variable in this particular dataframe.

In [45]:
```
DF.to_csv('Cluster.csv')
```

We can go ahead and try to read into the significance of the clusters. We can try other methods of scaling and check whether the answers are coming out to be different or not. We can also try to use different linkage methods and check. Since Clustering is an Unsupervised Learning Technique, we can dig deep and spend some time on imputing the missing values rather than dropping them. We can try to perform measures of Exploratory Data Analysis to understand the data better and then perform the Clustering algorithm.