The data set has information about features of silhouette extracted from the images of different c

Four "Corgie" model vehicles were used for the experiment: a double decker bus, Cheverolet van, Saab 9000 and an combination of vehicles was chosen with the expectation that the bus, van and either one of the cars would be read difficult to distinguish between the cars.

1. Read the dataset using function .dropna() - to avoid dealing with NAs as of now

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import io

from google.colab import files
uploaded = files.upload()
```

Choose Files vehicle.csv

- **vehicle.csv**(application/vnd.ms-excel) - 55762 bytes, last modified: 9/1/2019 - 100% done

```
Saving vehicle.csv to vehicle (1).csv
```

```
import numpy as np
```

```
data_raw = pd.read_csv(io.BytesIO(uploaded['vehicle.csv']))
```

```
data_raw.shape
```

```
(846, 19)
```

```
data_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 846 entries, 0 to 845
Data columns (total 19 columns):
compactness                    846 non-null int64
circularity                    841 non-null float64
distance_circularity           842 non-null float64
radius_ratio                   840 non-null float64
pr.axis_aspect_ratio           844 non-null float64
max.length_aspect_ratio        846 non-null int64
scatter_ratio                  845 non-null float64
elongatedness                  845 non-null float64
pr.axis_rectangularity         843 non-null float64
max.length_rectangularity      846 non-null int64
scaled_variance                843 non-null float64
scaled_variance.1              844 non-null float64
scaled_radius_of_gyration      844 non-null float64
scaled_radius_of_gyration.1    842 non-null float64
skewness_about                 840 non-null float64
skewness_about.1               845 non-null float64
skewness_about.2               845 non-null float64
```

```
data_raw.columns
```

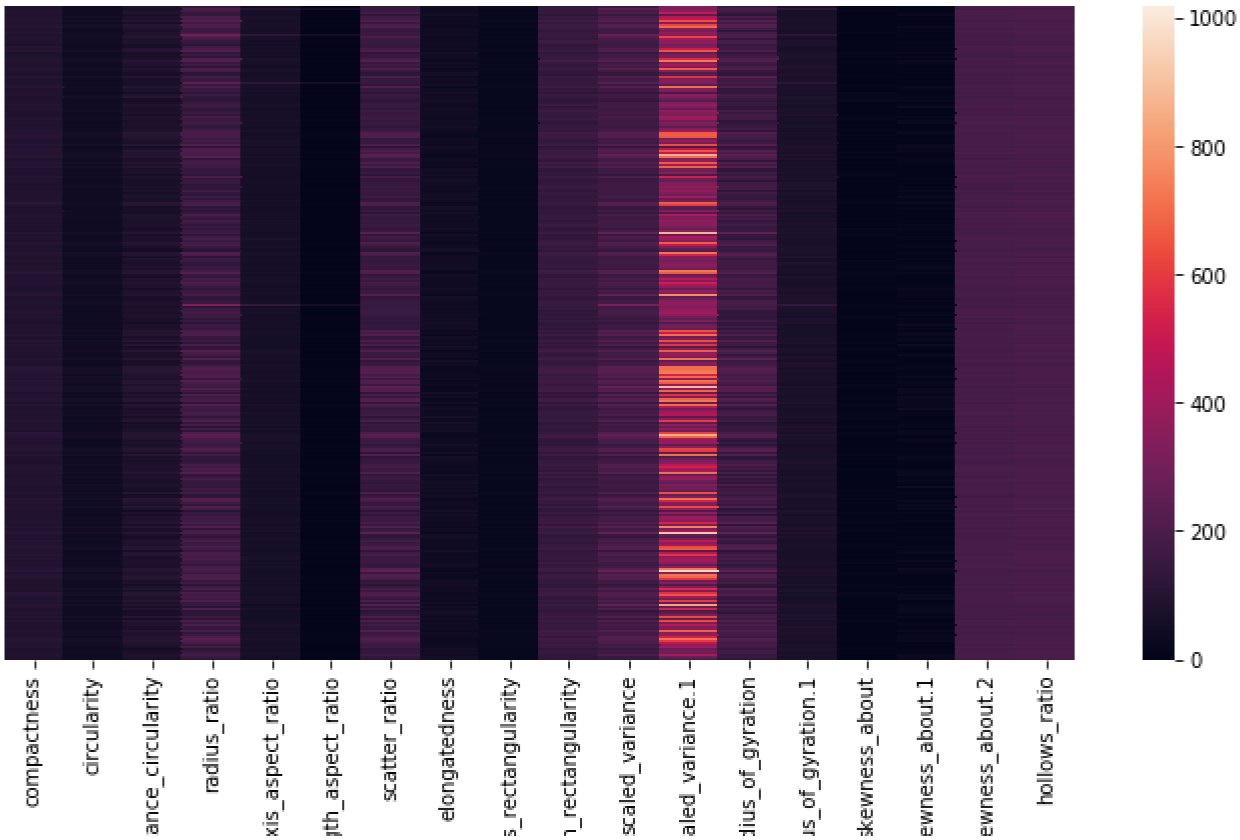
```
Index(['compactness', 'circularity', 'distance_circularity', 'radius_ratio',
       'pr.axis_aspect_ratio', 'max.length_aspect_ratio', 'scatter_ratio',
       'elongatedness', 'pr.axis_rectangularity', 'max.length_rectangularity',
       'scaled_variance', 'scaled_variance.1', 'scaled_radius_of_gyration',
       'scaled_radius_of_gyration.1', 'skewness_about', 'skewness_about.1',
       'skewness_about.2', 'hollows_ratio', 'class'],
      dtype='object')
```

```
data_raw.sample(4)
```

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio
668	94	46.0	91.0	175.0	70.0
13	89	42.0	85.0	144.0	58.0
681	96	46.0	70.0	194.0	70.0
144	95	45.0	80.0	186.0	62.0

```
## Cheking the null values for non oject columns

plt.figure(figsize=(12,6))
sns.heatmap(data_raw.select_dtypes(include=['int64','float64']),xticklabels=True,yticklabels=False,c
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7467f44fd0>
```




```
df= data_raw.copy() ## makying copy of original data
```

```
df.head()
```

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	m
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	

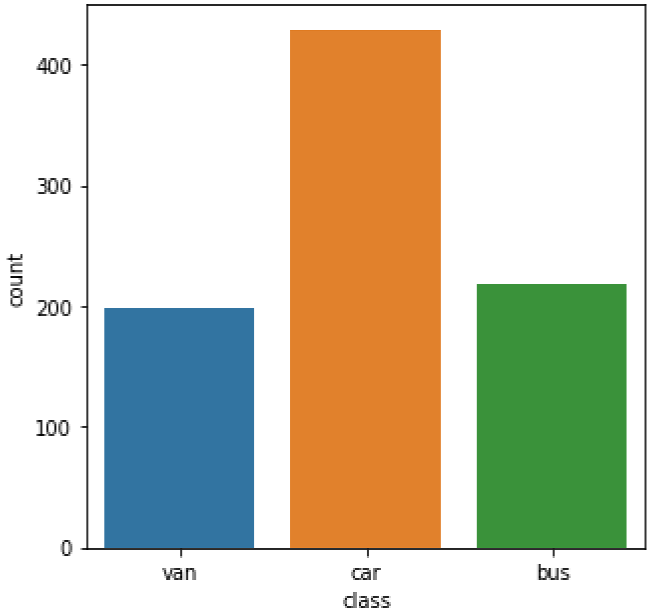
2. Print/ Plot the dependent (categorical variable) - Class column

Since the variable is categorical, you can use value_counts function

```
data_raw['class'].value_counts()
```

```
plt.figure(figsize=(5,5))
sns.countplot(x=data_raw['class'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f745dd78e10>
```



Check for any missing values in the data

```
## Dropping the target

df = df.drop('class',axis=1)
```

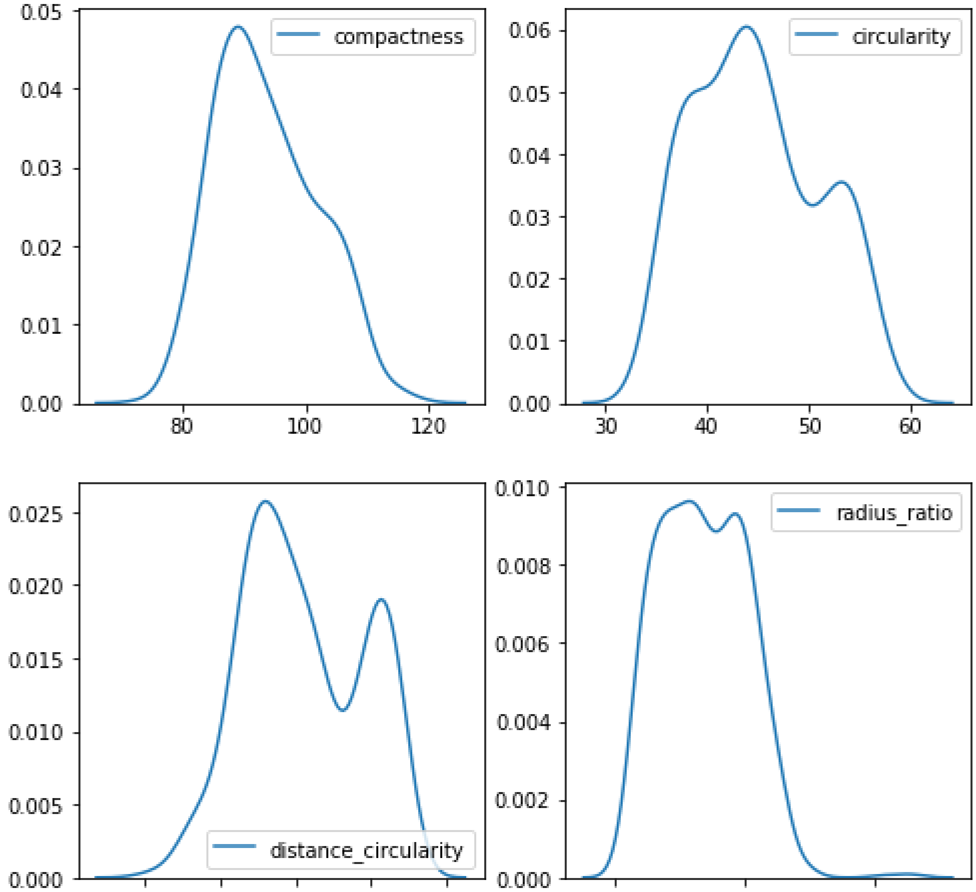
```
df.columns
```

```
Index(['compactness', 'circularity', 'distance_circularity', 'radius_ratio',
       'pr.axis_aspect_ratio', 'max.length_aspect_ratio', 'scatter_ratio',
       'elongatedness', 'pr.axis_rectangularity', 'max.length_rectangularity',
       'scaled_variance', 'scaled_variance.1', 'scaled_radius_of_gyration',
       'scaled_radius_of_gyration.1', 'skewness_about', 'skewness_about.1',
       'skewness_about.2', 'hollows_ratio'],
      dtype='object')
```

```
## Cheking the distriution of each attributes
```

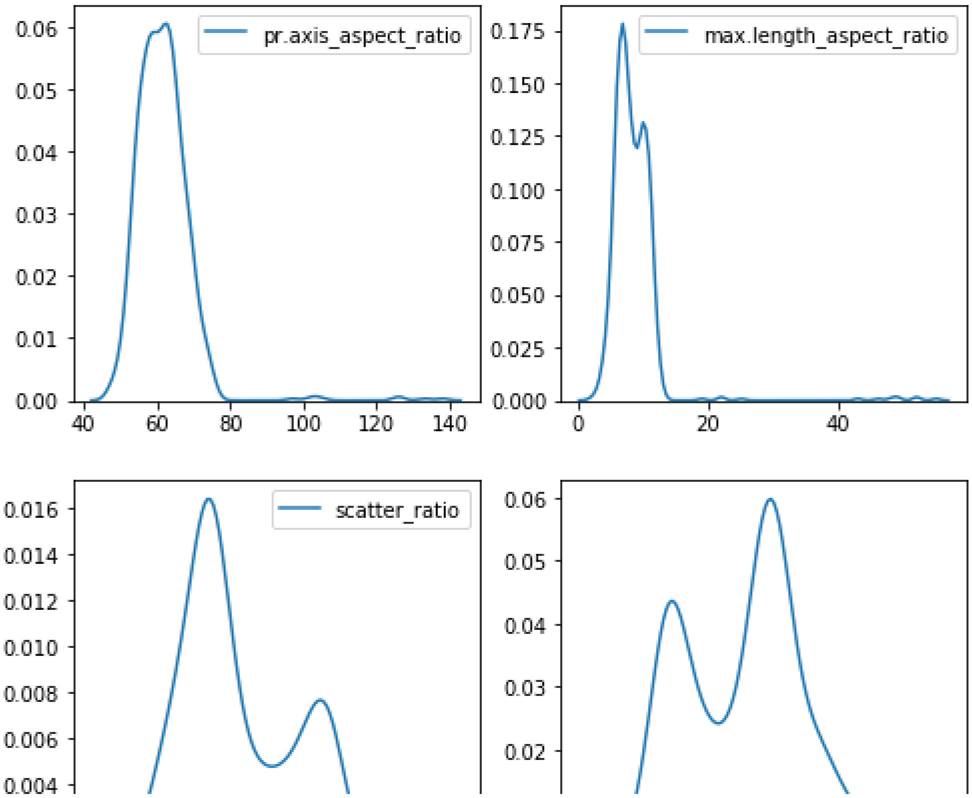
```
fig, qaxis = plt.subplots(2,2,figsize=(8,8))

sns.kdeplot(df['compactness'], ax = qaxis[0,0])

sns.kdeplot(df['circularity'], ax = qaxis[0,1])

sns.kdeplot(df['distance_circularity'], ax = qaxis[1,0])

sns.kdeplot(df['radius_ratio'], ax = qaxis[1,1])
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeWarn
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeWarn
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
<matplotlib.axes._subplots.AxesSubplot at 0x7f745dc5ca58>
```



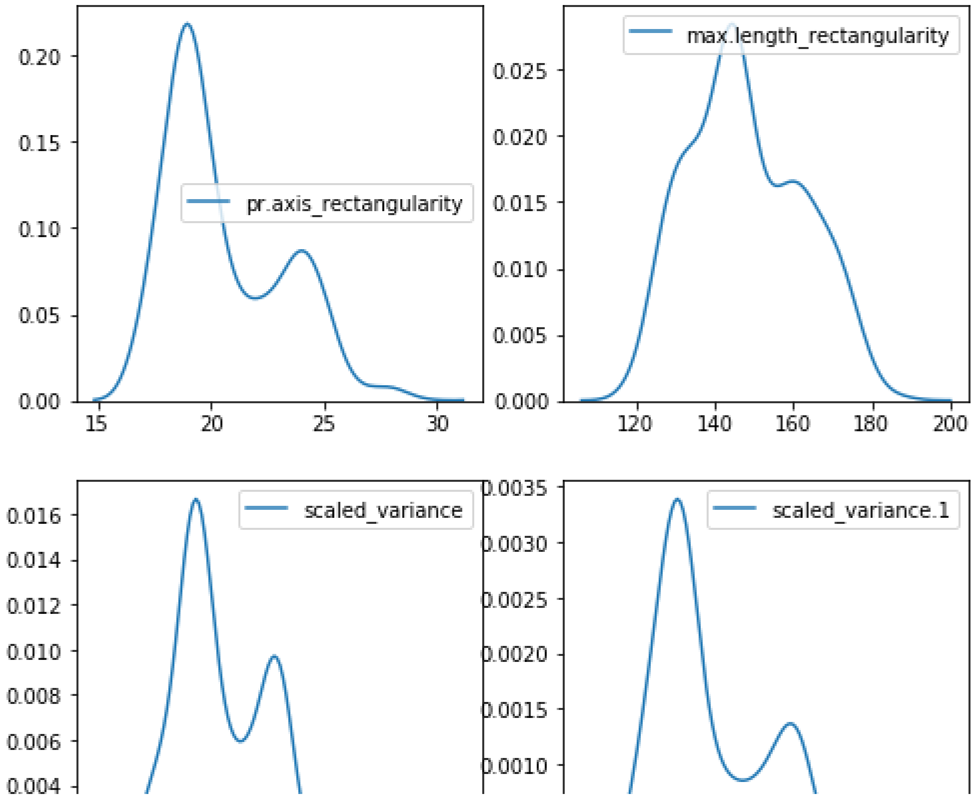

```
fig, qaxis = plt.subplots(2,2,figsize=(8,8))



sns.kdeplot(df['pr.axis_aspect_ratio'], ax = qaxis[0,0])

sns.kdeplot(df['max.length_aspect_ratio'], ax = qaxis[0,1])

sns.kdeplot(df['scatter_ratio'], ax = qaxis[1,0])

sns.kdeplot(df['elongatedness'], ax = qaxis[1,1])
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeWarn
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeWarn
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
<matplotlib.axes._subplots.AxesSubplot at 0x7f745db31978>
```



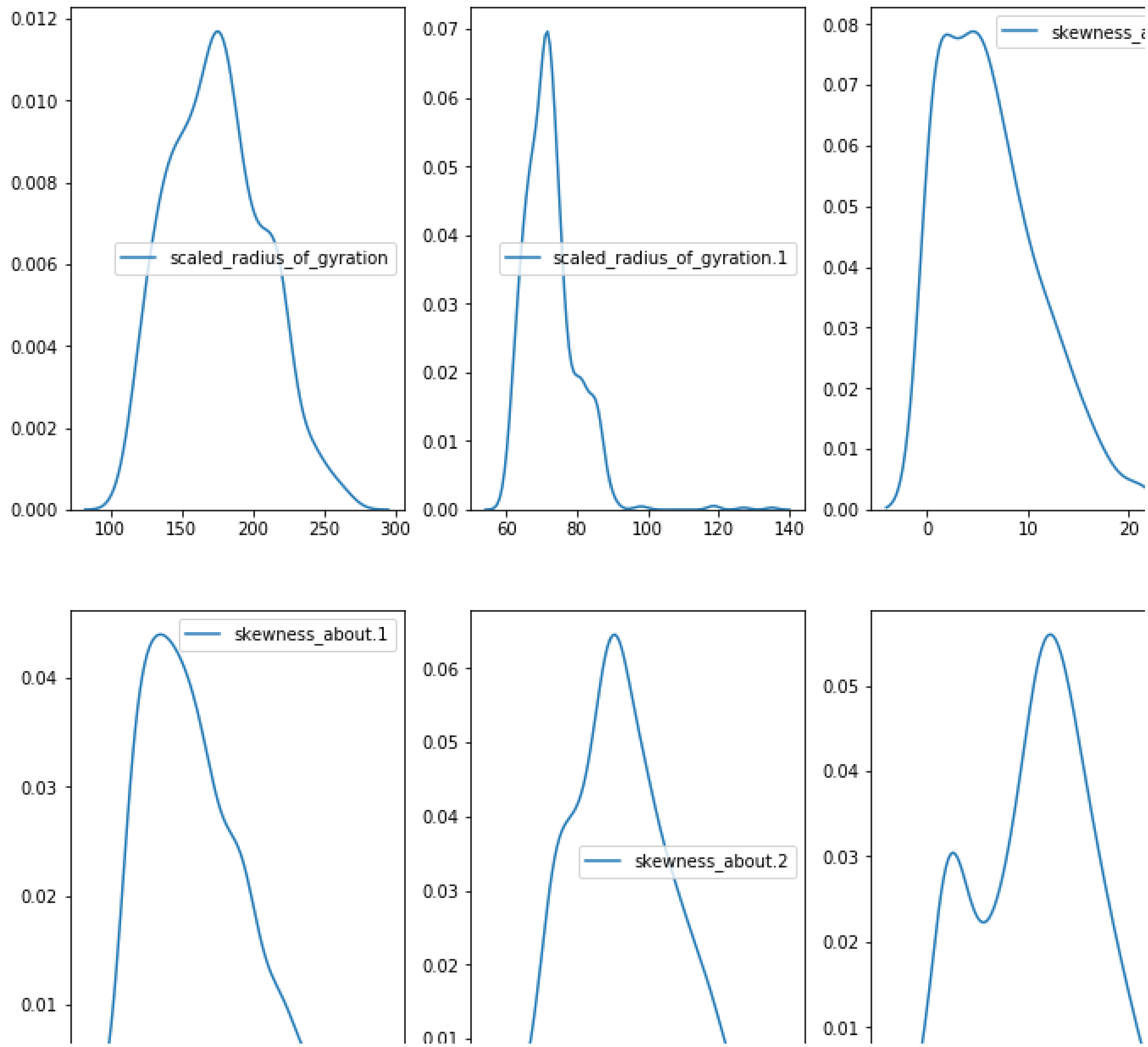

```
fig, qaxis = plt.subplots(2,2,figsize=(8,8))



sns.kdeplot(df['pr.axis_rectangularity'], ax = qaxis[0,0])

sns.kdeplot(df['max.length_rectangularity'], ax = qaxis[0,1])

sns.kdeplot(df['scaled_variance'], ax = qaxis[1,0])

sns.kdeplot(df['scaled_variance.1'], ax = qaxis[1,1])
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeWarn
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeWarn
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
<matplotlib.axes._subplots.AxesSubplot at 0x7f745d9a1a90>
```




```
fig, qaxis = plt.subplots(2,3,figsize=(12,12))


sns.kdeplot(df['scaled_radius_of_gyration'], ax = qaxis[0,0])

sns.kdeplot(df['scaled_radius_of_gyration.1'], ax = qaxis[0,1])

sns.kdeplot(df['skewness_about'], ax = qaxis[0,2])

sns.kdeplot(df['skewness_about.1'], ax = qaxis[1,0])


sns.kdeplot(df['skewness_about.2'], ax = qaxis[1,1])

sns.kdeplot(df['hollows_ratio'], ax = qaxis[1,2])
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeWarn
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeWarn
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
<matplotlib.axes._subplots.AxesSubplot at 0x7f745d7e0320>
```




```
df.isnull().sum() # chking the null count
```

```
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
```

```
df = df.fillna(df.mean()) ## replacing the na values wiith mean
```

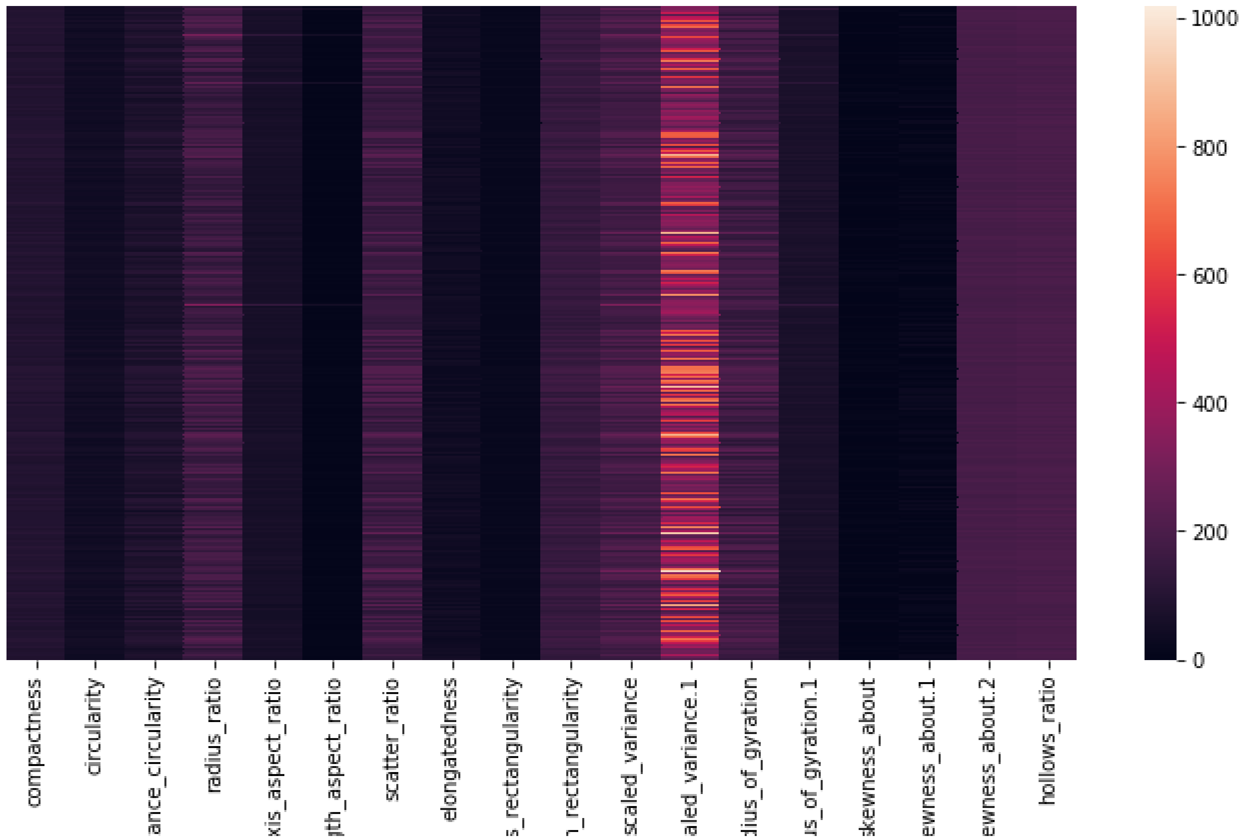
```
hollows_ratio                  0
```

```
## again checking the values null
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 846 entries, 0 to 845
Data columns (total 18 columns):
compactness                    846 non-null int64
circularity                    846 non-null float64
distance_circularity           846 non-null float64
radius_ratio                   846 non-null float64
pr.axis_aspect_ratio           846 non-null float64
max.length_aspect_ratio        846 non-null int64
scatter_ratio                  846 non-null float64
elongatedness                  846 non-null float64
pr.axis_rectangularity         846 non-null float64
max.length_rectangularity      846 non-null int64
scaled_variance                846 non-null float64
scaled_variance.1              846 non-null float64
scaled_radius_of_gyration      846 non-null float64
scaled_radius_of_gyration.1    846 non-null float64
skewness_about                 846 non-null float64
skewness_about.1               846 non-null float64
skewness_about.2               846 non-null float64
hollows_ratio                  846 non-null int64
dtypes: float64(14), int64(4)
memory usage: 119.0 KB
```

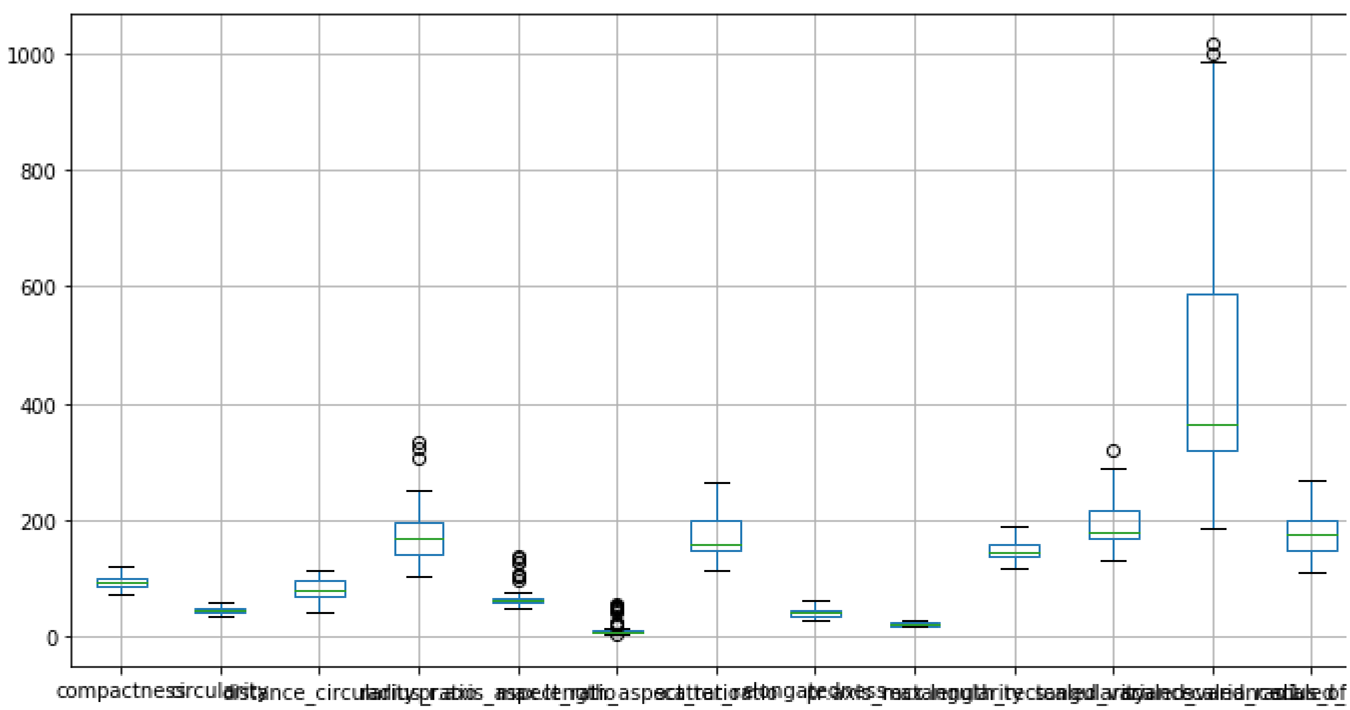
```
## Cheking the null values for non oject columns

plt.figure(figsize=(12,6))
sns.heatmap(df,xticklabels=True,yticklabels=False,cbar='inferno')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f745db11da0>
```




```
## boxplot
plt.figure(figsize=(16,6))
df.boxplot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f745dd3d080>
```



3. Standardize the data

```
from scipy.stats import zscore
```

Since the dimensions of the data are not really known to us, it would be wise to standardize the data using z scores methods. You can use zscore function to do this

K - Means Clustering

4. Plotting Elbow/ Scree Plot

```
# Numerical libraries
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.cluster import KMeans
```

```
from sklearn import metrics
```

```
cluster_range = range( 2, 10)   # expect 3 to four clusters from the pair panel visual inspection he
cluster_errors = []
cluster_sil_scores = []
for num_clusters in cluster_range:
  clusters = KMeans( num_clusters, n_init = 5)
```

```
  clusters.fit(df_z)
  labels = clusters.labels_                     # capture the cluster lables
  centroids = clusters.cluster_centers_         # capture the centroids
  cluster_errors.append( clusters.inertia_ )    # capture the intertia
  cluster_sil_scores.append(metrics.silhouette_score(df_z, labels, metric='euclidean'))
```

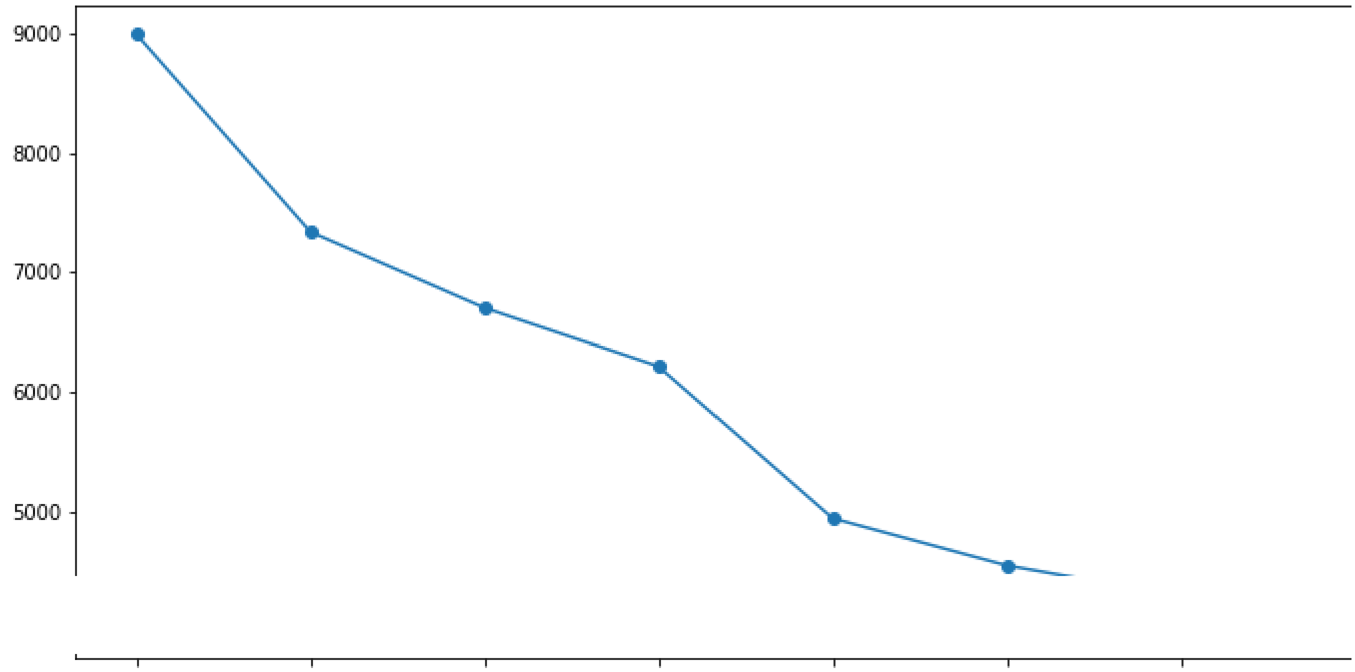
```
# combine the cluster_range and cluster_errors into a dataframe by combining them
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors": cluster_errors , "Avg
clusters_df[0:15]
```

	num_clusters	cluster_errors	Avg Sil Score
0	2	8990.751211	0.388462
1	3	7338.006447	0.291290
2	4	6704.150244	0.245967
3	5	6215.117977	0.222094
4	6	4944.556096	0.209766
5	7	4550.406799	0.220192
6	8	4295.459241	0.217656
7	9	4026.975073	0.204496

Use Matplotlib to plot the scree plot - Note: Scree plot plots distortion vs the no of clusters

```
# Elbow plot

plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
```

```
[<matplotlib.lines.Line2D at 0x7f745d45d978>]
```



Find out the optimal value of K

```
# Elbow plot

# optimum value for k is  = 3
```

Using optimal value of K - Cluster the data.

Note: Since the data has more than 2 dimension we cannot visualize the data. As an alternative, we can observe the distributed across different dimensions

```
cluster = KMeans( n_clusters = 3, random_state = 1 )
cluster.fit(df_z)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=1, tol=0.0001, verbose=0)
```

You can use kmeans.cluster_centers_ function to pull the centroid information from the instance

5. Store the centroids in a dataframe with column names from the original dataset given

```
# Get the centroids.... using function cluster_centers_
```

```
centroids = cluster.cluster_centers_
centroids
```

```
array([[ 1.13537065e+00,  1.16647789e+00,  1.19091375e+00,
         1.00571242e+00,  1.87546774e-01,  3.07529633e-01,
         1.27440296e+00, -1.19585143e+00,  1.27348442e+00,
         1.09485243e+00,  1.21730138e+00,  1.28001321e+00,
         1.08268523e+00, -2.09311115e-02,  1.66150782e-01,
         2.47377842e-01,  1.08524087e-03,  1.69066938e-01,
         3.32103321e-02],
       [-2.33494033e-01, -5.71862506e-01, -3.06054559e-01,
        -1.77764335e-02,  2.30200857e-01, -8.99271612e-02,
        -4.66244648e-01,  3.30858305e-01, -4.88530396e-01,
        -5.36102706e-01, -4.12673596e-01, -4.66625570e-01,
        -6.04057792e-01, -6.18605848e-01, -5.94407949e-02,
         1.90398963e-02,  8.08735491e-01,  7.04200047e-01,
         2.00923077e+00],
       [-9.27199540e-01, -5.21040780e-01, -8.93079584e-01,
        -1.06708290e+00, -5.02561817e-01, -2.16456813e-01,
        -7.75334771e-01,  8.66187148e-01, -7.45367601e-01,
        -4.89886514e-01, -7.83079023e-01, -7.80921078e-01,
        -3.88355666e-01,  8.26876928e-01, -1.02834414e-01,
        -2.92909446e-01, -1.05253254e+00, -1.09872862e+00,
         1.01600000e+00]])
```

Hint: Use pd.Dataframe function

```
# Let us put the raw centroid values into a dataframe under respective columns

centroid_df = pd.DataFrame(centroids, columns = list(df_z) )
centroid_df
```

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	m
0	1.135371	1.166478	1.190914	1.005712	0.187547	
1	-0.233494	-0.571863	-0.306055	-0.017776	0.230201	
2	-0.927200	-0.521041	-0.893080	-1.067083	-0.502562	

```
df['predict']= pd.DataFrame(cluster.labels_)
```

```
df['Actual']= data_raw['class']
```

Use kmeans.labels_ function to print out the labels of the classes

```
cluster.labels_
```

```
array([1, 1, 0, 1, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 1, 0, 0, 2, 2,
       1, 1, 0, 1, 2, 0, 0, 2, 1, 1, 1, 0, 1, 1, 2, 0, 0, 2, 0, 2, 2, 1,
       0, 2, 2, 2, 2, 1, 2, 1, 0, 1, 0, 1, 1, 2, 0, 2, 0, 2, 2, 2, 1, 2,
       2, 0, 1, 0, 0, 0, 1, 2, 1, 0, 1, 2, 0, 2, 2, 0, 1, 2, 1, 0, 1, 2,
       1, 2, 0, 1, 0, 1, 2, 0, 2, 2, 0, 2, 2, 1, 1, 2, 0, 0, 0, 2, 2, 0,
       1, 1, 2, 2, 2, 1, 0, 0, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 0, 0, 1,
       1, 2, 0, 0, 2, 1, 2, 1, 1, 2, 0, 2, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 2, 1, 1, 2, 0, 1, 1, 0, 0, 1, 0, 2, 2, 0, 0, 1, 0, 1, 1,
       1, 1, 1, 2, 0, 2, 1, 2, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 2, 0, 2,
       2, 2, 1, 1, 0, 0, 1, 1, 1, 2, 2, 0, 1, 1, 1, 0, 2, 1, 2, 0, 2, 1,
       0, 2, 0, 2, 2, 1, 0, 1, 0, 2, 2, 2, 2, 0, 1, 2, 1, 2, 0, 2, 1, 1,
       2, 0, 2, 2, 1, 1, 0, 2, 2, 0, 2, 1, 1, 0, 1, 1, 0, 0, 2, 1, 1, 1,
       0, 2, 2, 1, 1, 2, 2, 1, 1, 1, 0, 1, 2, 2, 0, 1, 1, 2, 2, 0, 2, 1,
       1, 2, 0, 2, 1, 1, 1, 1, 0, 1, 0, 2, 1, 1, 0, 1, 1, 1, 2, 1, 0, 0,
       0, 0, 0, 2, 1, 0, 2, 2, 2, 1, 2, 0, 0, 0, 2, 0, 1, 2, 0, 2, 1, 1,
       1, 0, 0, 2, 0, 0, 2, 0, 1, 1, 1, 2, 2, 0, 0, 0, 0, 1, 1, 1, 0, 2,
       1, 2, 0, 1, 1, 0, 1, 0, 0, 0, 1, 2, 2, 0, 2, 2, 2, 1, 1, 1, 1, 1,
       2, 0, 0, 2, 2, 0, 2, 0, 2, 0, 1, 2, 1, 2, 0, 0, 2, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 2, 2, 1, 1, 1, 2, 2, 1, 2, 0, 1, 1,
       2, 1, 2, 0, 1, 2, 1, 1, 0, 1, 0, 1, 0, 0, 2, 2, 0, 1, 2, 2, 1, 0,
       0, 2, 1, 0, 0, 2, 0, 0, 0, 1, 1, 1, 1, 1, 0, 2, 2, 1, 0, 1, 1, 0,
       1, 2, 0, 2, 2, 0, 0, 1, 2, 0, 0, 0, 2, 0, 0, 2, 1, 2, 0, 0, 1, 1,
       2, 2, 0, 1, 2, 0, 0, 1, 2, 0, 0, 1, 0, 2, 2, 0, 0, 0, 2, 2, 0, 0,
       0, 1, 1, 0, 2, 1, 0, 1, 2, 2, 1, 0, 2, 1, 1, 2, 1, 1, 0, 1, 0, 0,
       1, 2, 1, 0, 0, 2, 2, 1, 0, 1, 0, 0, 1, 1, 1, 1, 2, 2, 2, 1, 1, 0,
       2, 2, 1, 2, 0, 1, 0, 2, 2, 0, 0, 1, 0, 1, 1, 1, 0, 1, 2, 1, 0, 1,
       1, 2, 0, 0, 0, 0, 1, 2, 2, 2, 0, 0, 0, 1, 0, 2, 1, 0, 2, 2, 2, 1,
       2, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 2, 0, 2, 2, 1, 2, 1,
       1, 2, 2, 0, 0, 2, 1, 0, 1, 0, 1, 1, 0, 1, 2, 0, 2, 0, 2, 2, 1, 2,
       1, 0, 0, 2, 0, 1, 1, 2, 1, 2, 0, 1, 0, 2, 1, 1, 1, 2, 2, 2, 1, 0,
       1, 0, 2, 1, 1, 1, 1, 0, 1, 2, 0, 1, 0, 1, 1, 0, 2, 0, 2, 1, 1, 1,
       2, 0, 1, 2, 1, 0, 2, 0, 1, 1, 0, 2, 1, 2, 1, 1, 2, 1, 0, 0, 1, 1,
       0, 0, 1, 1, 2, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 2,
       0, 1, 2, 0, 0, 0, 1, 0, 2, 2, 0, 0, 0, 1, 0, 1, 1, 0, 1, 2, 1, 2,
       1, 0, 1, 2, 1, 1, 1, 2, 0, 2, 2, 2, 0, 0, 2, 0, 0, 2, 1, 1, 0, 1,
       2, 0, 0, 2, 1, 1, 0, 0, 0, 2, 0, 1, 0, 0, 2, 2, 0, 2, 0, 1, 2, 1,
       0, 0, 1, 2, 1, 0, 0, 1, 1, 2, 1, 1, 0, 2, 1, 0, 2, 2, 0, 2, 1, 2,
```

```
df['Actual'].value_counts()
```

```
car    429
bus    218
van    199
Name: Actual, dtype: int64
```

```
df['predict'].value_counts()
```

```
1    325
0    271
2    250
Name: predict, dtype: int64
```

Hierarchical Clustering

6. Variable creation

For Hierarchical clustering, we will create datasets using multivariate normal distribution to visually observe how the

```
# Getting the values and plotting it
plt.figure(figsize=(5,5))
f1 = data['V1'].values

f2 = data['V2'].values

X = np.array(list(zip(f1, f2)))

plt.scatter(f1, f2, c='black', s=7)
```

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
a = np.random.multivariate_normal([10, 0], [[3, 1], [1, 4]], size=[100,])
b = np.random.multivariate_normal([0, 20], [[3, 1], [1, 4]], size=[50,])
c = np.random.multivariate_normal([10, 20], [[3, 1], [1, 4]], size=[100,])
```

7. Combine all three arrays a,b,c into a dataframe

```
X = np.concatenate((a,b,c),axis=0)
```

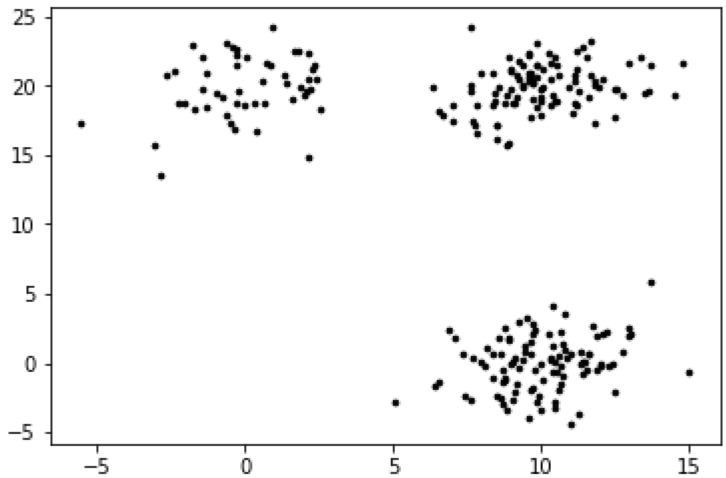
```
data = pd.DataFrame(X)
data.head()
```

	0	1
0	11.403874	0.076128
1	8.169289	1.074573
2	8.598782	1.859506
3	10.816401	3.580541
4	10.424222	-0.593139

8. Use scatter matrix to print all the 3 distributions

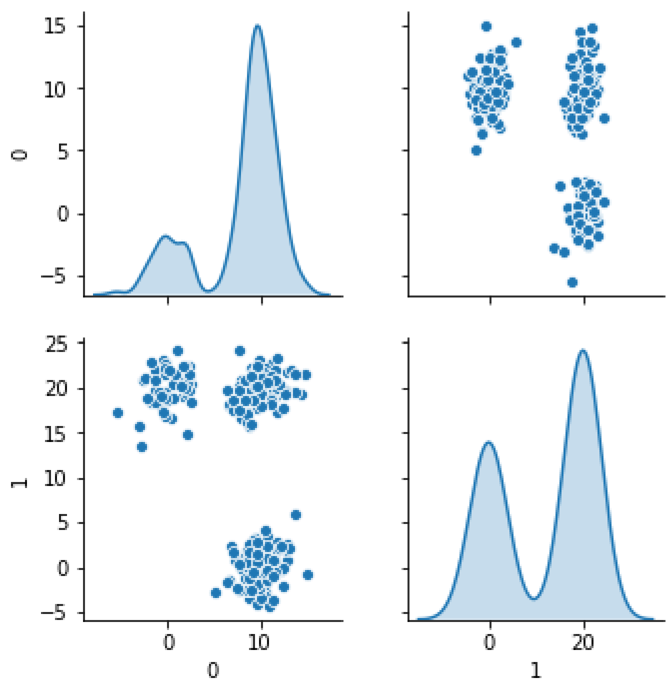
```
f1 = data.iloc[:,0].values

f2 = data.iloc[:,1].values

# X = np.array(list(zip(f1, f2)))

plt.scatter(f1, f2, c='black', s=7)
```

```
<matplotlib.collections.PathCollection at 0x7f745bd47a58>
```



```
sns.pairplot(data, diag_kind ='kde')
```

```
<seaborn.axisgrid.PairGrid at 0x7f745bdbdf60>
```



9. Find out the linkage matrix

```
from sklearn.cluster import AgglomerativeClustering


model = AgglomerativeClustering(n_clusters=6, affinity='euclidean',  linkage='ward')
```

```
model.fit(data)
```

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='ward', memory=None, n_clusters=6,
                        pooling_func='deprecated')
```

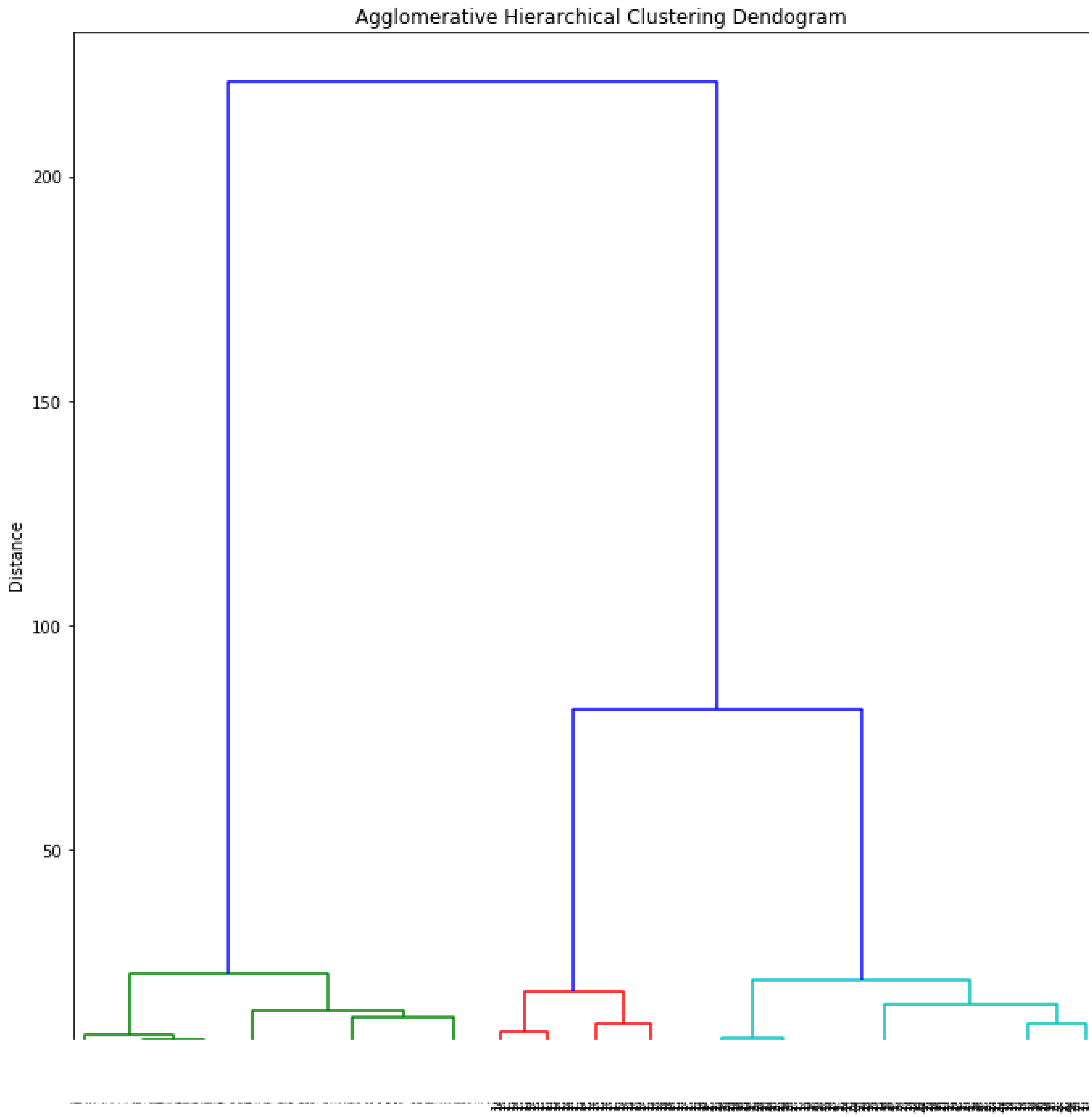
```
from scipy.cluster.hierarchy import cophenet, dendrogram, linkage
from scipy.spatial.distance import pdist  #Pairwise distribution between data points


# cophenet index is a measure of the correlation between the distance of points in feature space and
# closer it is to 1, the better is the clustering

Z = linkage(data, 'ward')
c, coph_dists = cophenet(Z , pdist(data))

c
```

```
0.9585251916947939
```

Use ward as linkage metric and distance as Eucledian

10. Plot the dendrogram for the consolidated dataframe

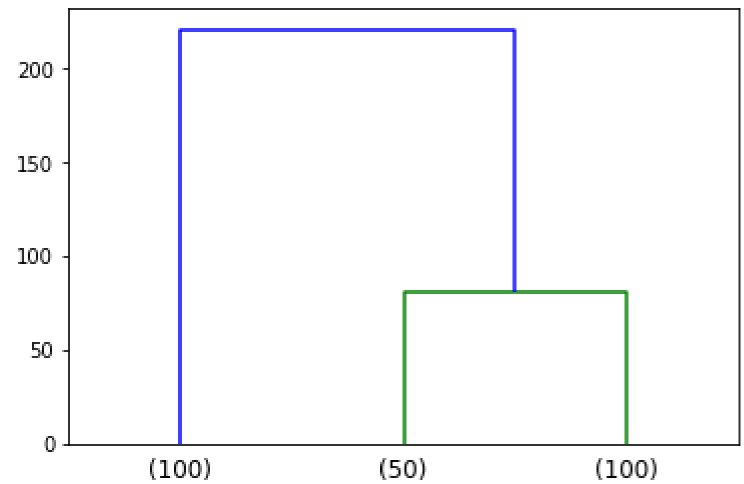
```
plt.figure(figsize=(10, 10))
plt.title('Agglomerative Hierarchical Clustering Dendogram')
plt.xlabel('sample index')
plt.ylabel('Distance')
dendrogram(Z, leaf_rotation=90.,color_threshold = 30, leaf_font_size=8. )
plt.tight_layout()
```



11. Recreate the dendrogram for last 3 merged clusters

1. List item
2. List item

```
dendrogram(
    Z,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=3,  # show only the last p merged clusters
)
plt.show()
```



Hint: Use truncate_mode='lastp' attribute in dendrogram function to arrive at dendrogram

12. From the truncated dendrogram, find out the optimal distance between clusters which u want

```
# the optimal distance between clusters is 75
```

13. Using this distance measure and fcluster function to cluster the data into 3 different groups

```
from scipy.cluster.hierarchy import fcluster

Z_plt = fcluster(Z, 75, criterion='distance')
```

Use matplotlib to visually observe the clusters in 2D space

```
f1 = data.iloc[:,0].values
f2 = data.iloc[:,1].values
# X = np.array(list(zip(f1, f2)))
```

```
plt.scatter(f1, f2, c=Z_plt, s=7)
```

```
<matplotlib.collections.PathCollection at 0x7f745b6e30b8>
```