

```
In [78]: import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
import sklearn.tree as tree
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import warnings
warnings.filterwarnings('ignore')
```

Problem statement/Business problem

In [2]: Multiclass classification of dry beans

Data gathering

In [3]: df=pd.read_excel('Dry_Bean_Dataset.xlsx')
df

Out[3]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRatio	Eccentricity	ConvexArea
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	2871
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	2917
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	2969
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	3072
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	3041
...
13606	42097	759.696	288.721612	185.944705	1.552728	0.765002	4250
13607	42101	757.499	281.576392	190.713136	1.476439	0.735702	4249
13608	42139	759.321	281.539928	191.187979	1.472582	0.734065	4256
13609	42147	763.779	283.382636	190.275731	1.489326	0.741055	4266
13610	42159	772.237	295.142741	182.204716	1.619841	0.786693	4260

13611 rows × 17 columns

In [4]: df.info

```
Out[4]: <bound method DataFrame.info of
orAxisLength  AspectRatio \
0      28395    610.291      208.178117      173.888747      1.197191
1      28734    638.018      200.524796      182.734419      1.097356
2      29380    624.110      212.826130      175.931143      1.209713
3      30008    645.884      210.557999      182.516516      1.153638
4      30140    620.134      201.847882      190.279279      1.060798
...
13606   42097    759.696      288.721612      185.944705      1.552728
13607   42101    757.499      281.576392      190.713136      1.476439
13608   42139    759.321      281.539928      191.187979      1.472582
13609   42147    763.779      283.382636      190.275731      1.489326
13610   42159    772.237      295.142741      182.204716      1.619841

          Eccentricity  ConvexArea  EquivDiameter  Extent  Solidity  roundness
\
0      0.549812      28715     190.141097      0.763923      0.988856      0.958027
1      0.411785      29172     191.272750      0.783968      0.984986      0.887034
2      0.562727      29690     193.410904      0.778113      0.989559      0.947849
3      0.498616      30724     195.467062      0.782681      0.976696      0.903936
4      0.333680      30417     195.896503      0.773098      0.990893      0.984877
...
13606   0.765002      42508     231.515799      0.714574      0.990331      0.916603
13607   0.735702      42494     231.526798      0.799943      0.990752      0.922015
13608   0.734065      42569     231.631261      0.729932      0.989899      0.918424
13609   0.741055      42667     231.653248      0.705389      0.987813      0.907906
13610   0.786693      42600     231.686223      0.788962      0.989648      0.888380

          Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFactor4 \
0      0.913358      0.007332      0.003147      0.834222      0.998724
1      0.953861      0.006979      0.003564      0.909851      0.998430
2      0.908774      0.007244      0.003048      0.825871      0.999066
3      0.928329      0.007017      0.003215      0.861794      0.994199
4      0.970516      0.006697      0.003665      0.941900      0.999166
...
13606   0.801865      0.006858      0.001749      0.642988      0.998385
13607   0.822252      0.006688      0.001886      0.676099      0.998219
13608   0.822730      0.006681      0.001888      0.676884      0.996767
13609   0.817457      0.006724      0.001852      0.668237      0.995222
13610   0.784997      0.007001      0.001640      0.616221      0.998180

          Class
0      SEKER
1      SEKER
2      SEKER
3      SEKER
4      SEKER
...
13606   DERMASON
13607   DERMASON
13608   DERMASON
13609   DERMASON
13610   DERMASON
```

```
[13611 rows x 17 columns]>
```

```
In [5]: df.shape
```

```
Out[5]: (13611, 17)
```

```
In [6]: df.describe()
```

```
Out[6]:
```

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRatio	Eccentricity
count	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000
mean	53048.284549	855.283459	320.141867	202.270714	1.583242	0.750895
std	29324.095717	214.289696	85.694186	44.970091	0.246678	0.092002
min	20420.000000	524.736000	183.601165	122.512653	1.024868	0.218951
25%	36328.000000	703.523500	253.303633	175.848170	1.432307	0.715928
50%	44652.000000	794.941000	296.883367	192.431733	1.551124	0.764441
75%	61332.000000	977.213000	376.495012	217.031741	1.707109	0.810466
max	254616.000000	1985.370000	738.860153	460.198497	2.430306	0.911423

```
In [7]: df['Area'].median()
```

```
Out[7]: 44652.0
```

```
In [8]: df['Area'].mode()
```

```
Out[8]: 0    28122
       1    29709
       2    30529
       3    33518
       4    34594
       5    34774
       6    35354
       7    35442
       8    36109
       9    38273
      10   38426
      11   38542
      12   38945
      13   40504
      14   44710
      15   47134
      16   52266
      dtype: int64
```

```
In [ ]:
```

In [9]: `df.unique()`

Out[9]:

Area	12011
Perimeter	13416
MajorAxisLength	13543
MinorAxisLength	13543
AspectRatio	13543
Eccentricity	13543
ConvexArea	12066
EquivDiameter	12011
Extent	13535
Solidity	13526
roundness	13543
Compactness	13543
ShapeFactor1	13543
ShapeFactor2	13543
ShapeFactor3	13543
ShapeFactor4	13543
Class	7

`dtype: int64`

In [10]: `df.isna().sum()`

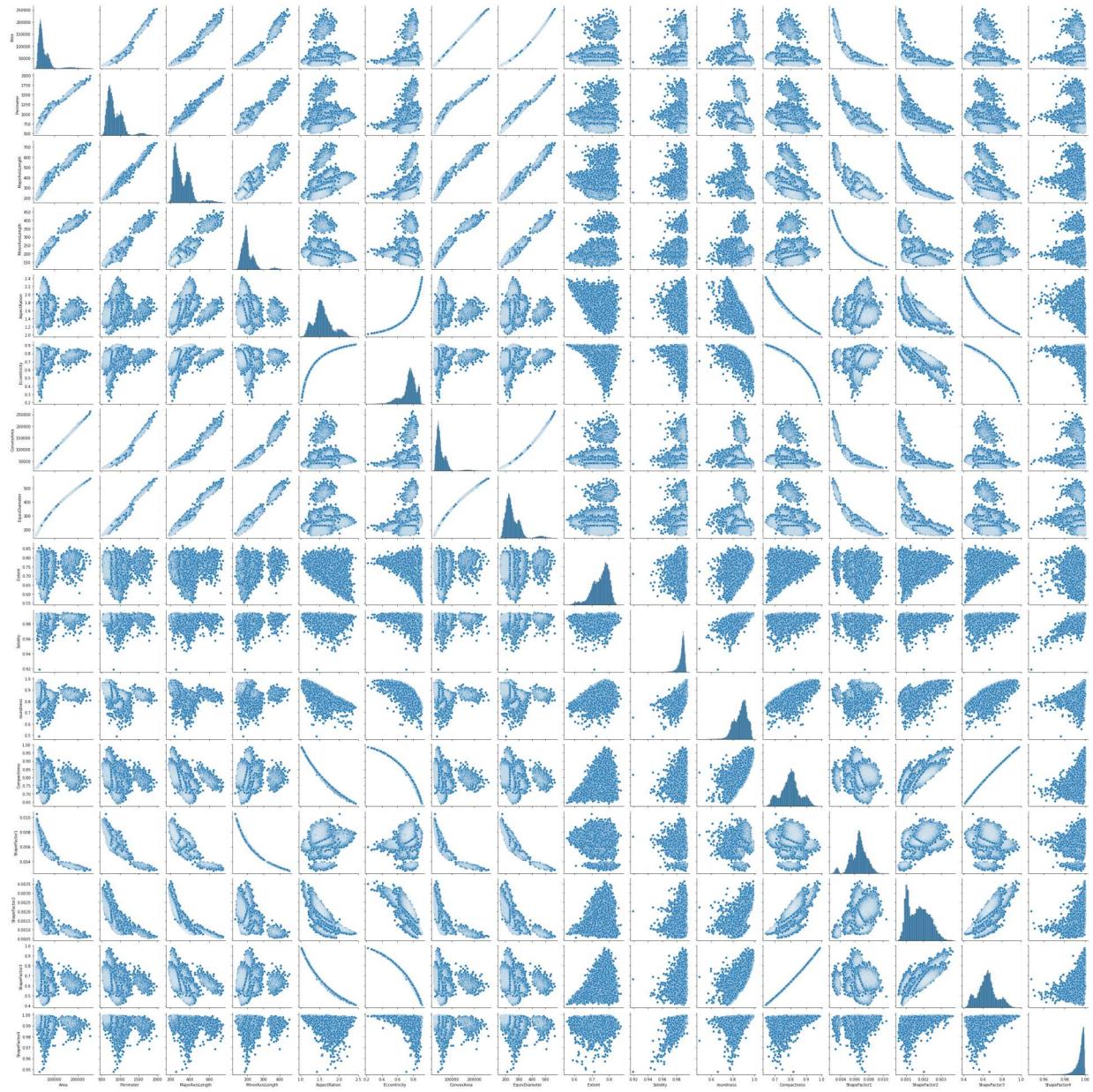
Out[10]:

Area	0
Perimeter	0
MajorAxisLength	0
MinorAxisLength	0
AspectRatio	0
Eccentricity	0
ConvexArea	0
EquivDiameter	0
Extent	0
Solidity	0
roundness	0
Compactness	0
ShapeFactor1	0
ShapeFactor2	0
ShapeFactor3	0
ShapeFactor4	0
Class	0

`dtype: int64`

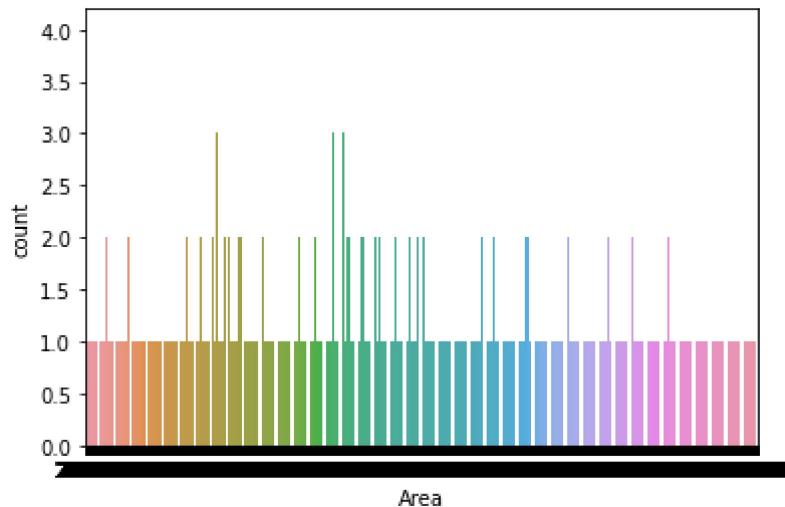
```
In [11]: sns.pairplot(df)
```

```
Out[11]: <seaborn.axisgrid.PairGrid at 0x2772f354d60>
```



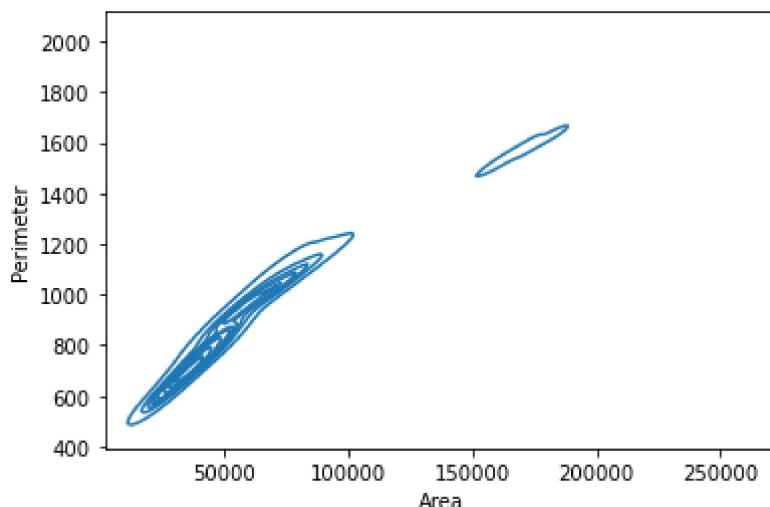
```
In [12]: sns.countplot(df['Area'])
```

```
Out[12]: <AxesSubplot:xlabel='Area', ylabel='count'>
```



```
In [13]: sns.kdeplot(df['Area'], df['Perimeter'])
```

```
Out[13]: <AxesSubplot:xlabel='Area', ylabel='Perimeter'>
```



Exploratory data analysis(EDA)

1. Area

```
In [14]: df[['Area']].isna().sum() # Area does not contain any null values or missing val
```

```
Out[14]: Area      0
          dtype: int64
```

```
In [15]: df[['Area']].value_counts()
```

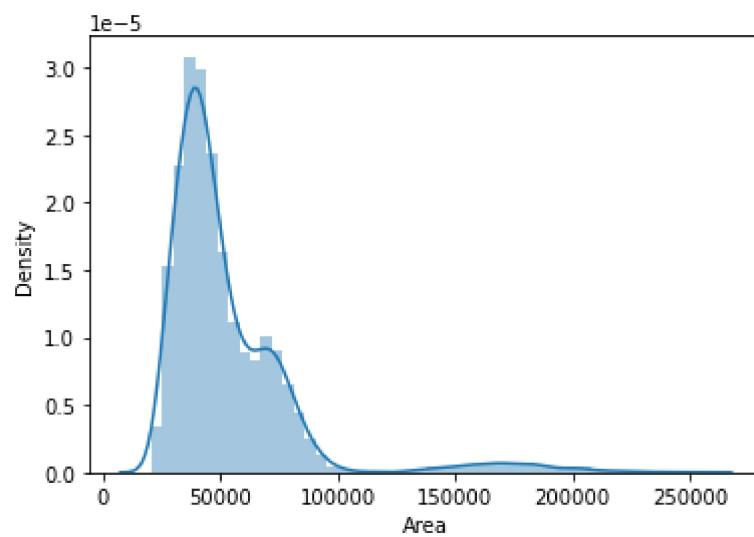
```
Out[15]: Area
34774    4
44710    4
29709    4
35442    4
38945    4
...
40127    1
40130    1
40138    1
40142    1
254616   1
Length: 12011, dtype: int64
```

```
In [16]: df[['Area']].nunique()
```

```
Out[16]: Area    12011
dtype: int64
```

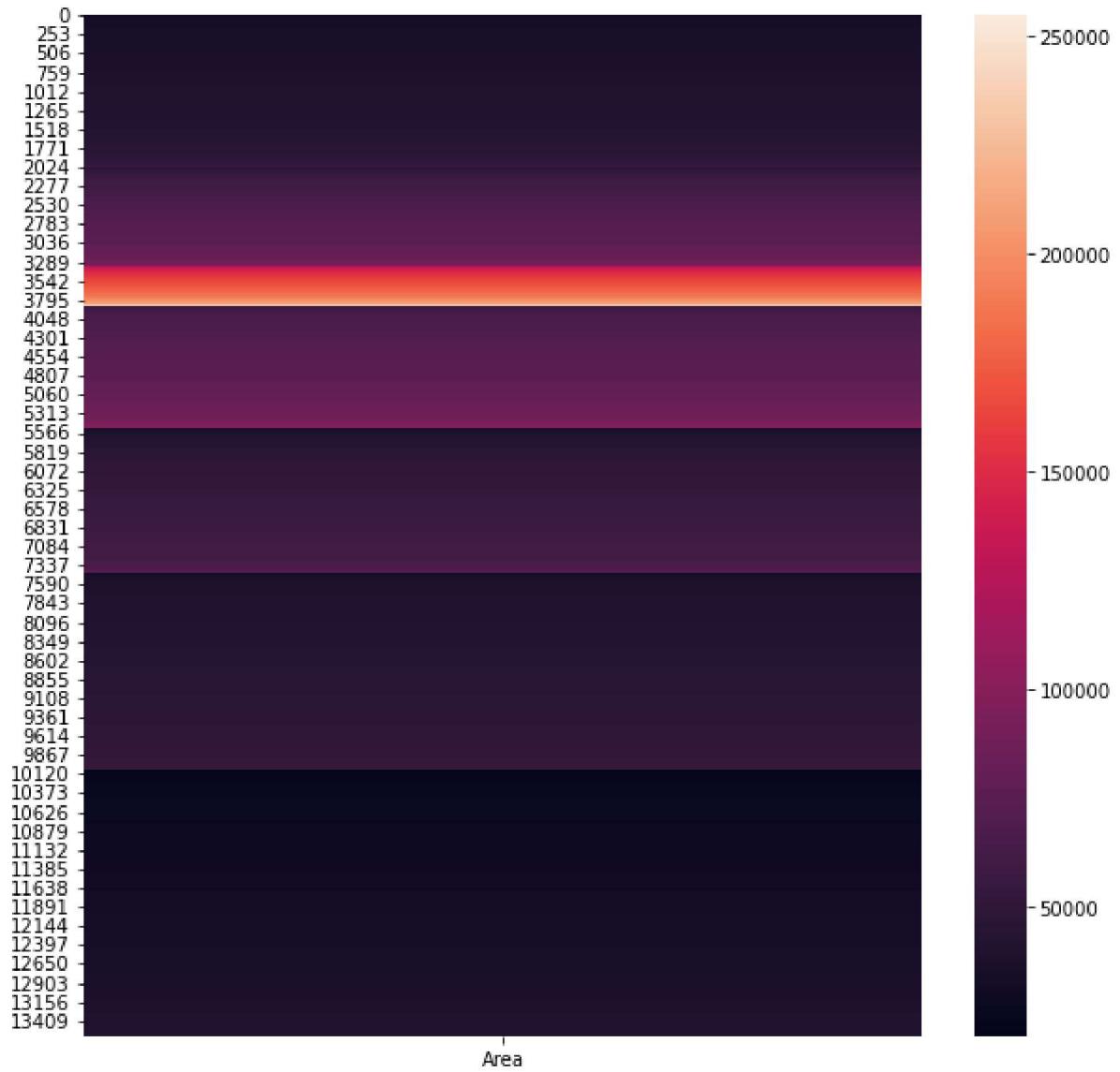
```
In [17]: sns.distplot(df['Area'])
```

```
Out[17]: <AxesSubplot:xlabel='Area', ylabel='Density'>
```



```
In [18]: plt.figure(figsize=(10, 10))
sns.heatmap(df[['Area']])
```

Out[18]: <AxesSubplot:>



```
In [19]: len(dir(sns))
```

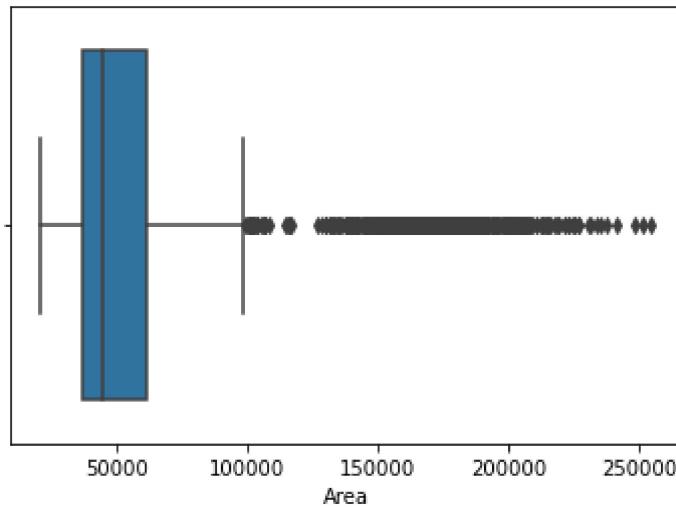
```
Out[19]: 98
```

```
In [20]: len(dir(plt))
```

```
Out[20]: 256
```

```
In [21]: sns.boxplot(df['Area']) # The area feature contains outliers
```

```
Out[21]: <AxesSubplot:xlabel='Area'>
```

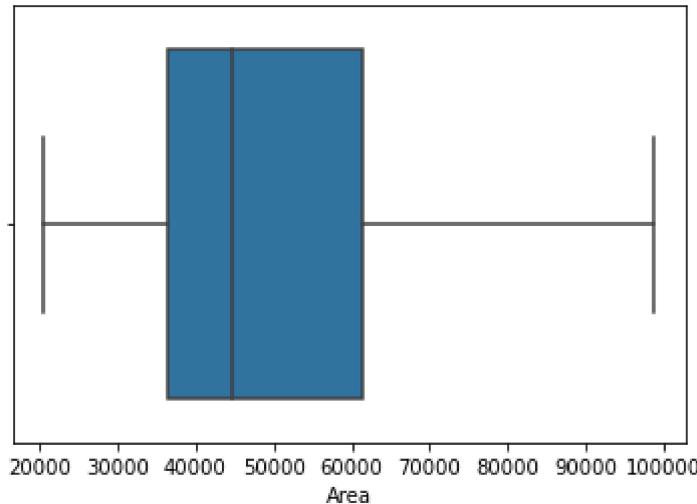


```
In [22]: # Handling outliers using iqr
q1=df['Area'].quantile(0.25)
q3=df['Area'].quantile(0.75)
iqr=q3-q1
upper_limit=q3+1.5*iqr
```

```
In [23]: df['Area']=np.where(df['Area']>upper_limit, upper_limit, df['Area'])
```

```
In [24]: sns.boxplot(df['Area'])
```

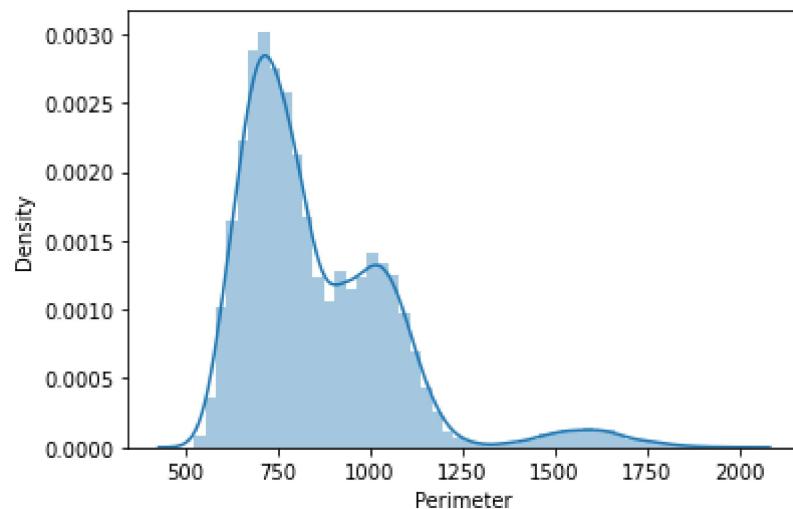
```
Out[24]: <AxesSubplot:xlabel='Area'>
```



2. Perimeter

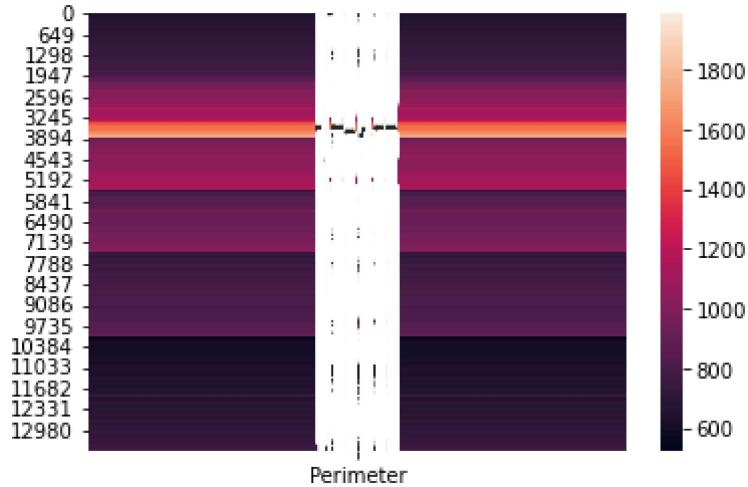
```
In [25]: sns.distplot(df['Perimeter'])
```

```
Out[25]: <AxesSubplot:xlabel='Perimeter', ylabel='Density'>
```



```
In [26]: sns.heatmap(df[['Perimeter']], annot=True)
plt.figure(figsize=(10, 50))
```

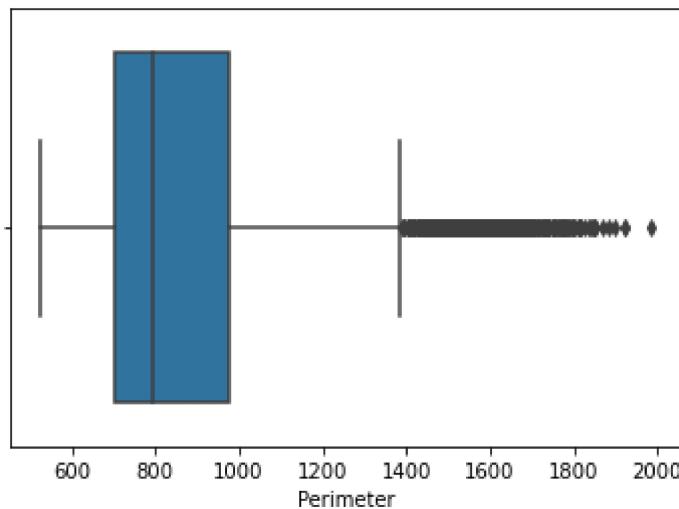
Out[26]: <Figure size 720x3600 with 0 Axes>



<Figure size 720x3600 with 0 Axes>

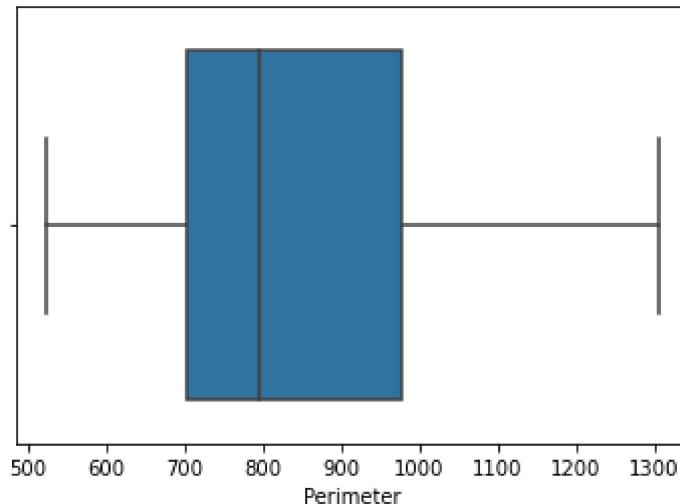
```
In [27]: sns.boxplot(df['Perimeter']) # The feature perimeter contains outliers
```

Out[27]: <AxesSubplot:xlabel='Perimeter'>



```
In [28]: # Handling outliers using iqr
q1=df['Perimeter'].quantile(0.25)
q3=df['Perimeter'].quantile(0.75)
iqr=q3-q1
upper_limit=q3+1.2*iqr
df['Perimeter']=np.where(df['Perimeter']>upper_limit, upper_limit, df['Perimeter'])
sns.boxplot(df['Perimeter'])
```

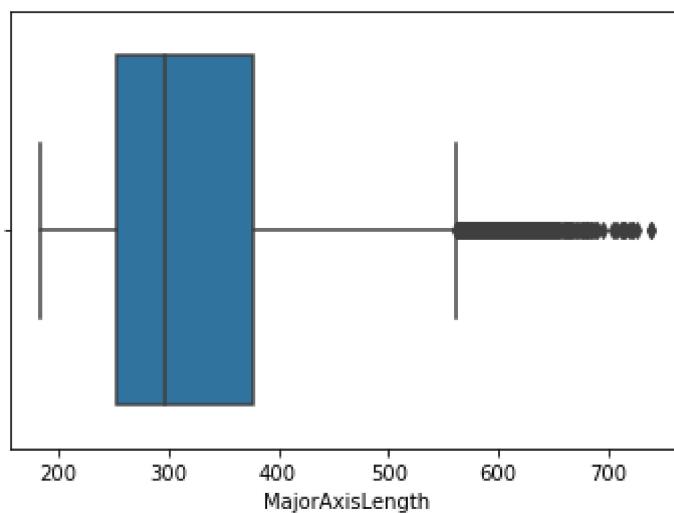
Out[28]: <AxesSubplot:xlabel='Perimeter'>



3. MajorAxisLength

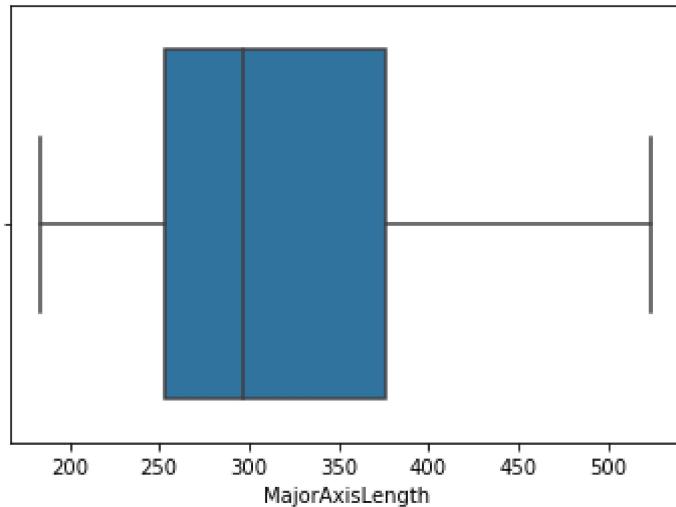
```
In [29]: sns.boxplot(df['MajorAxisLength'])
```

Out[29]: <AxesSubplot:xlabel='MajorAxisLength'>



```
In [30]: # Handling outliers using iqr  
q1=df['MajorAxisLength'].quantile(0.25)  
q3=df['MajorAxisLength'].quantile(0.75)  
iqr=q3-q1  
upper_limit=q3+1.2*iqr  
df['MajorAxisLength']=np.where(df['MajorAxisLength']>upper_limit, upper_limit, df['MajorAxisLength'])  
sns.boxplot(df['MajorAxisLength'])
```

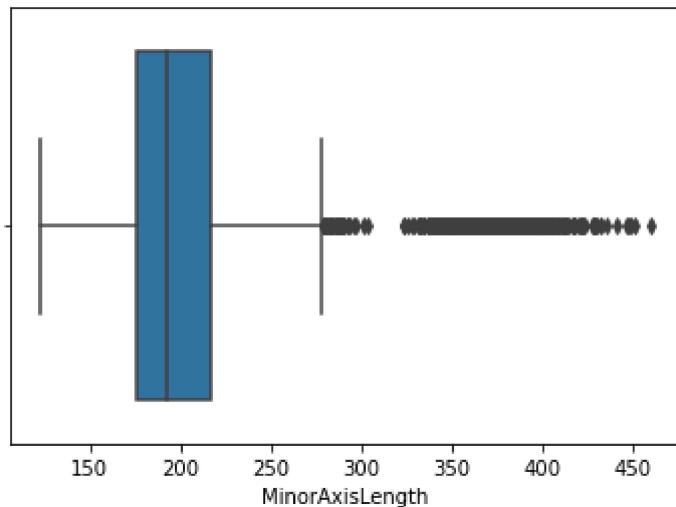
```
Out[30]: <AxesSubplot:xlabel='MajorAxisLength'>
```



4. MinorAxisLength

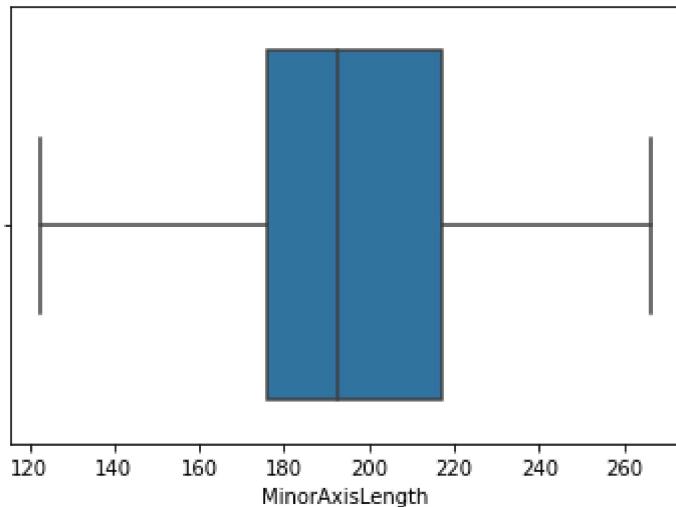
```
In [31]: sns.boxplot(df['MinorAxisLength'])
```

```
Out[31]: <AxesSubplot:xlabel='MinorAxisLength'>
```



```
In [32]: # Handling outliers using iqr  
q1=df['MinorAxisLength'].quantile(0.25)  
q3=df['MinorAxisLength'].quantile(0.75)  
iqr=q3-q1  
upper_limit=q3+1.2*iqr  
df['MinorAxisLength']=np.where(df['MinorAxisLength']>upper_limit, upper_limit, df['MinorAxisLength'])  
sns.boxplot(df['MinorAxisLength'])
```

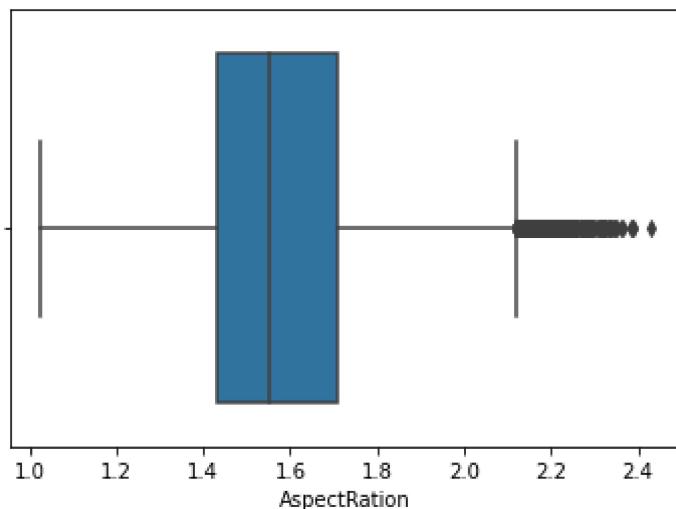
Out[32]: <AxesSubplot:xlabel='MinorAxisLength'>



5. AspectRatio

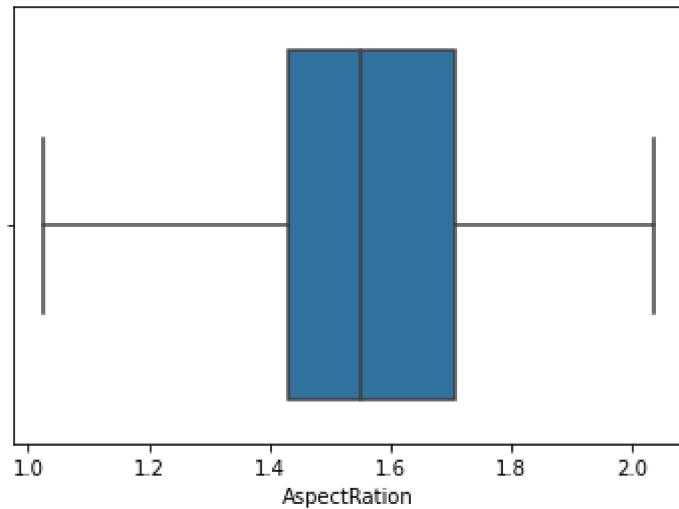
```
In [33]: sns.boxplot(df['AspectRatio'])
```

Out[33]: <AxesSubplot:xlabel='AspectRatio'>



```
In [34]: # Handling outliers using iqr  
q1=df['AspectRation'].quantile(0.25)  
q3=df['AspectRation'].quantile(0.75)  
iqr=q3-q1  
upper_limit=q3+1.2*iqr  
df['AspectRation']=np.where(df['AspectRation']>upper_limit, upper_limit, df['AspectRation'])  
sns.boxplot(df['AspectRation'])
```

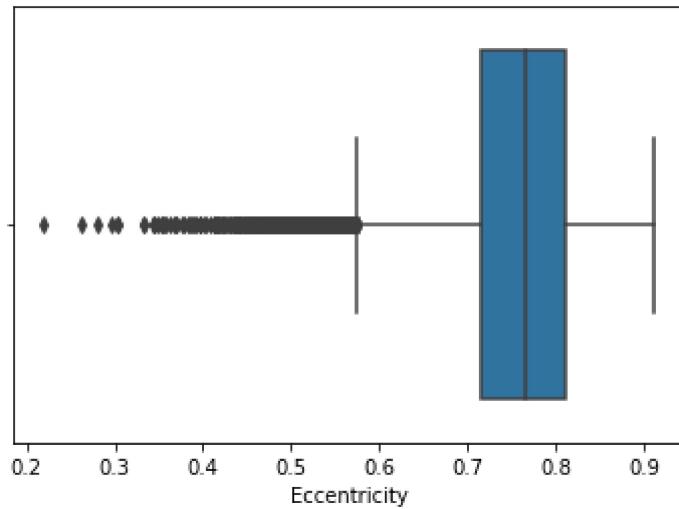
Out[34]: <AxesSubplot:xlabel='AspectRation'>



6. Eccentricity

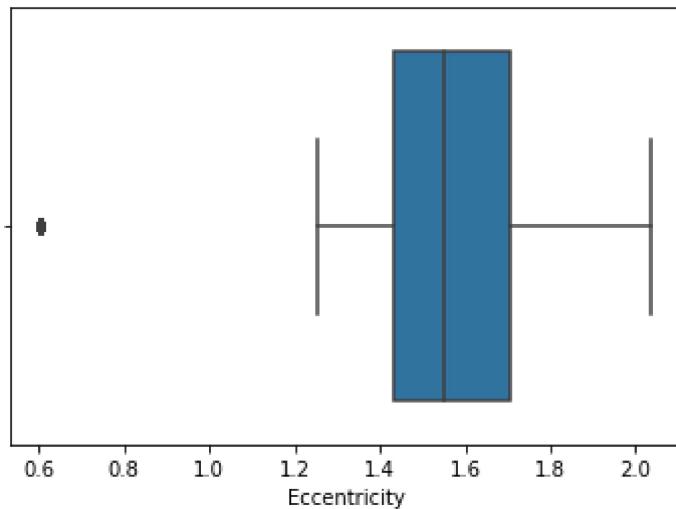
```
In [35]: sns.boxplot(df['Eccentricity'])
```

Out[35]: <AxesSubplot:xlabel='Eccentricity'>



```
In [36]: # Handling outliers using iqr  
q1=df['Eccentricity'].quantile(0.25)  
q3=df['Eccentricity'].quantile(0.75)  
iqr=q3-q1  
lower_limit=q1-1.2*iqr  
df['Eccentricity']=np.where(df['Eccentricity']<lower_limit, lower_limit, df['Aspe  
sns.boxplot(df['Eccentricity'])
```

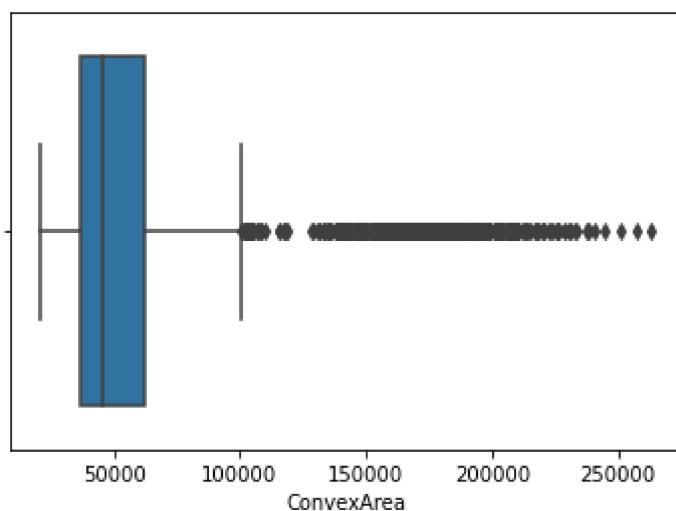
```
Out[36]: <AxesSubplot:xlabel='Eccentricity'>
```



7. ConvexArea

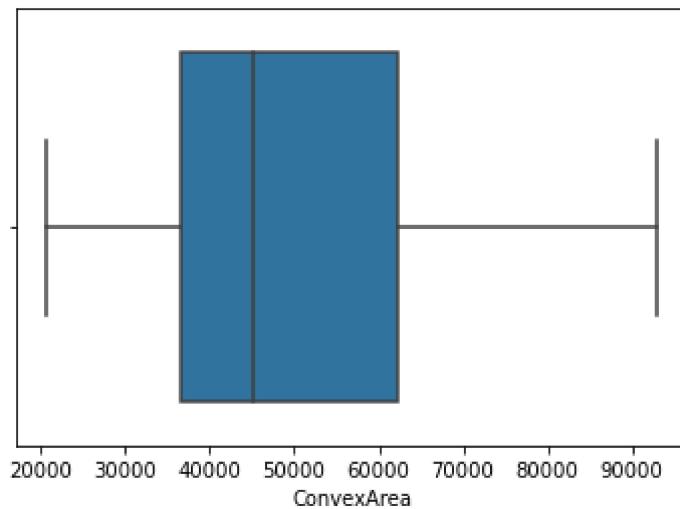
```
In [37]: sns.boxplot(df['ConvexArea'])
```

```
Out[37]: <AxesSubplot:xlabel='ConvexArea'>
```



```
In [38]: # Handling outliers using iqr  
q1=df['ConvexArea'].quantile(0.25)  
q3=df['ConvexArea'].quantile(0.75)  
iqr=q3-q1  
upper_limit=q3+1.2*iqr  
df['ConvexArea']=np.where(df['ConvexArea']>upper_limit, upper_limit, df['ConvexArea'])  
sns.boxplot(df['ConvexArea'])
```

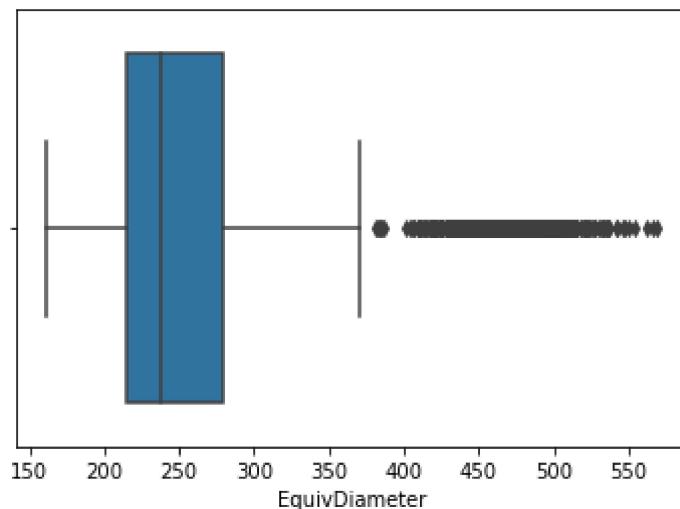
Out[38]: <AxesSubplot:xlabel='ConvexArea'>



8. EquivDiameter

```
In [39]: sns.boxplot(df['EquivDiameter'])
```

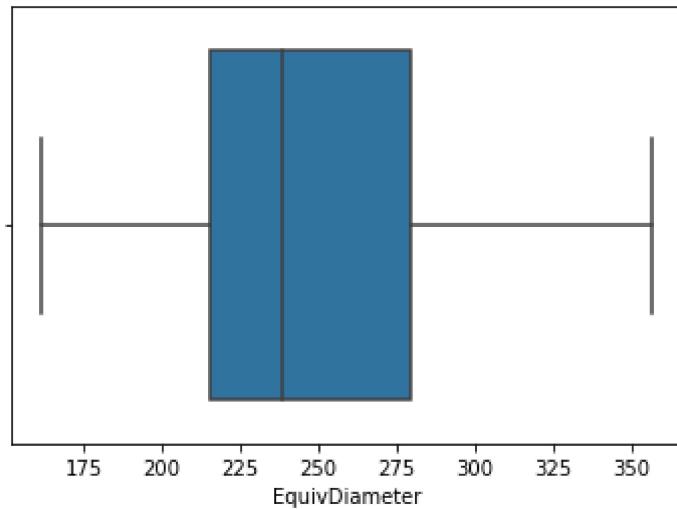
Out[39]: <AxesSubplot:xlabel='EquivDiameter'>



In [40]: # Handling outliers using iqr

```
q1=df['EquivDiameter'].quantile(0.25)
q3=df['EquivDiameter'].quantile(0.75)
iqr=q3-q1
upper_limit=q3+1.2*iqr
df['EquivDiameter']=np.where(df['EquivDiameter']>upper_limit, upper_limit, df['EquivDiameter'])
sns.boxplot(df['EquivDiameter'])
```

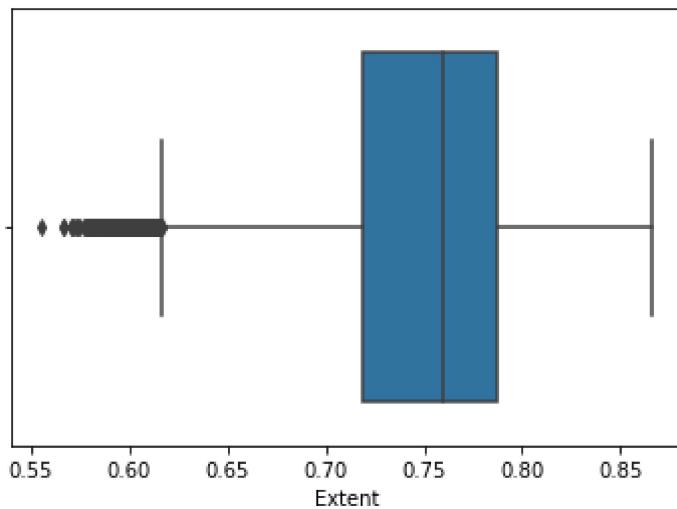
Out[40]: <AxesSubplot:xlabel='EquivDiameter'>



9. Extent

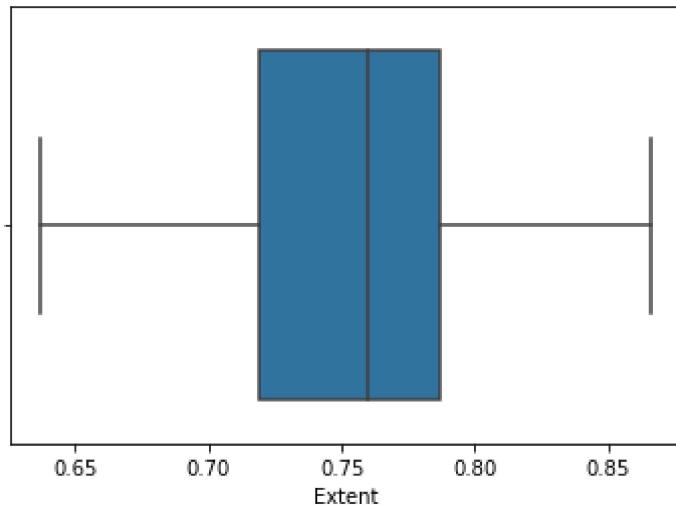
In [41]: sns.boxplot(df['Extent'])

Out[41]: <AxesSubplot:xlabel='Extent'>



```
In [42]: # Handling outliers using iqr
q1=df['Extent'].quantile(0.25)
q3=df['Extent'].quantile(0.75)
iqr=q3-q1
lower_limit=q1-1.2*iqr
df['Extent']=np.where(df['Extent']<lower_limit, lower_limit, df['Extent'])
sns.boxplot(df['Extent'])
```

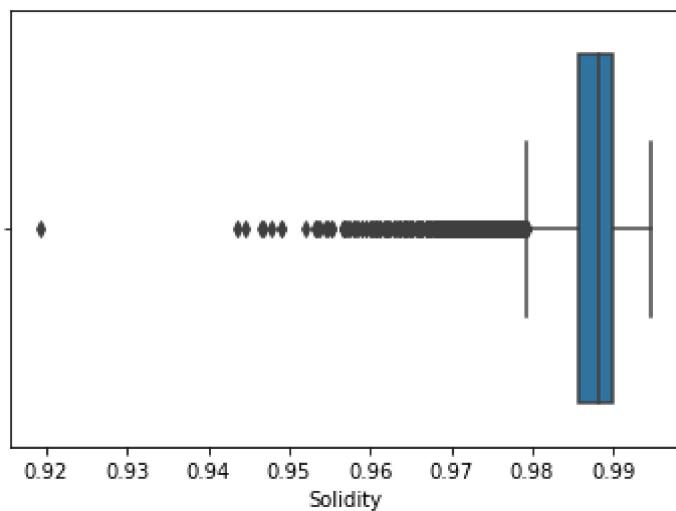
Out[42]: <AxesSubplot:xlabel='Extent'>



10. Solidity

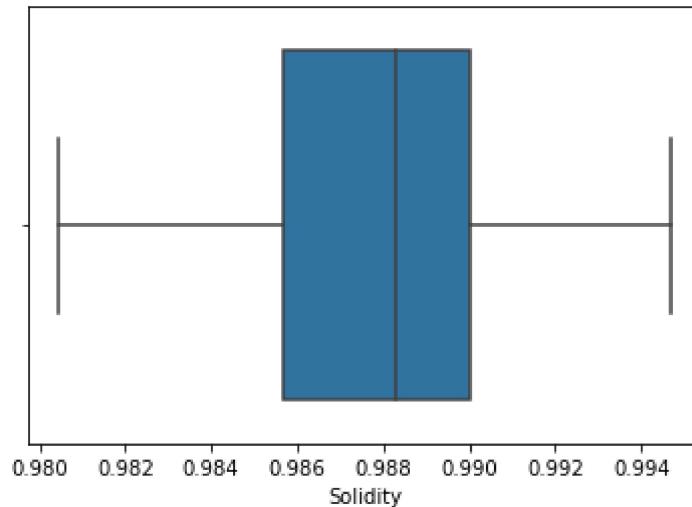
```
In [43]: sns.boxplot(df['Solidity'])
```

Out[43]: <AxesSubplot:xlabel='Solidity'>



```
In [44]: # Handling outliers using iqr
q1=df['Solidity'].quantile(0.25)
q3=df['Solidity'].quantile(0.75)
iqr=q3-q1
lower_limit=q1-1.2*iqr
df['Solidity']=np.where(df['Solidity']<lower_limit, lower_limit, df['Solidity'])
sns.boxplot(df['Solidity'])
```

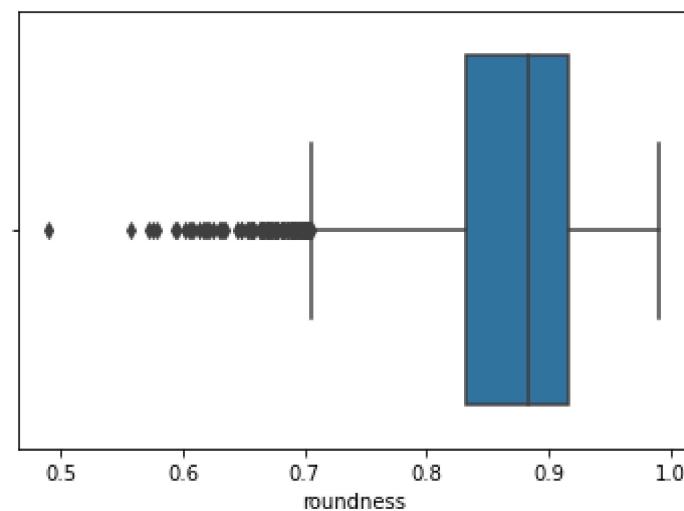
```
Out[44]: <AxesSubplot:xlabel='Solidity'>
```



11. roundness

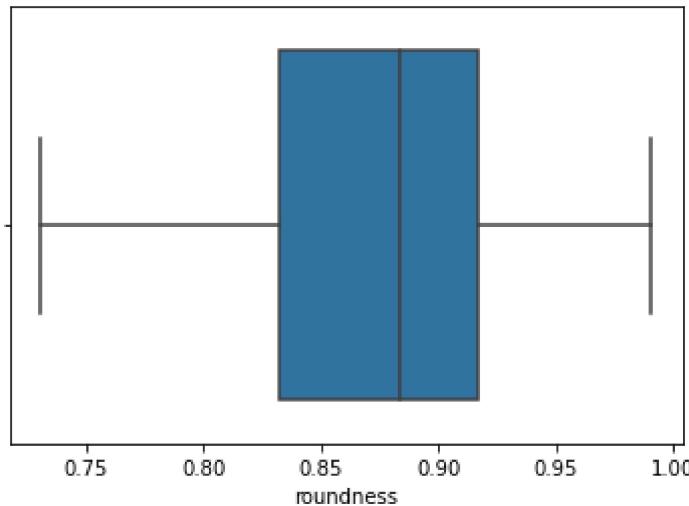
```
In [45]: sns.boxplot(df['roundness'])
```

```
Out[45]: <AxesSubplot:xlabel='roundness'>
```



```
In [46]: # Handling outliers using iqr
q1=df['roundness'].quantile(0.25)
q3=df['roundness'].quantile(0.75)
iqr=q3-q1
lower_limit=q1-1.2*iqr
df['roundness']=np.where(df['roundness']<lower_limit, lower_limit, df['roundness'])
sns.boxplot(df['roundness'])
```

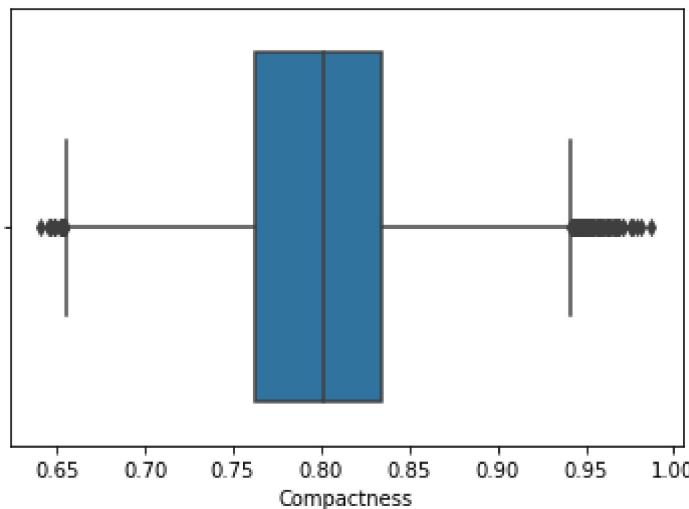
Out[46]: <AxesSubplot:xlabel='roundness'>



12. Compactness

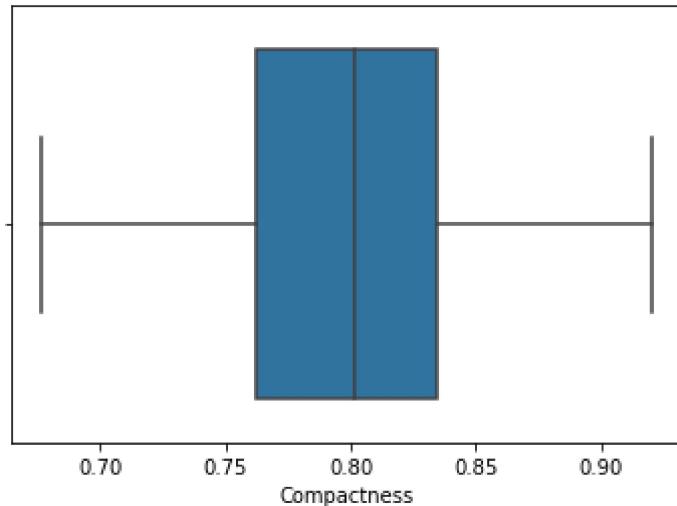
```
In [47]: sns.boxplot(df['Compactness'])
```

Out[47]: <AxesSubplot:xlabel='Compactness'>



```
In [48]: # Handling outliers using iqr
q1=df['Compactness'].quantile(0.25)
q3=df['Compactness'].quantile(0.75)
iqr=q3-q1
upper_limit=q3+1.2*iqr
lower_limit=q1-1.2*iqr
df['Compactness']=np.where(df['Compactness']>upper_limit, upper_limit, df['Compactness'])
df['Compactness']=np.where(df['Compactness']<lower_limit, lower_limit, df['Compactness'])
sns.boxplot(df['Compactness'])
```

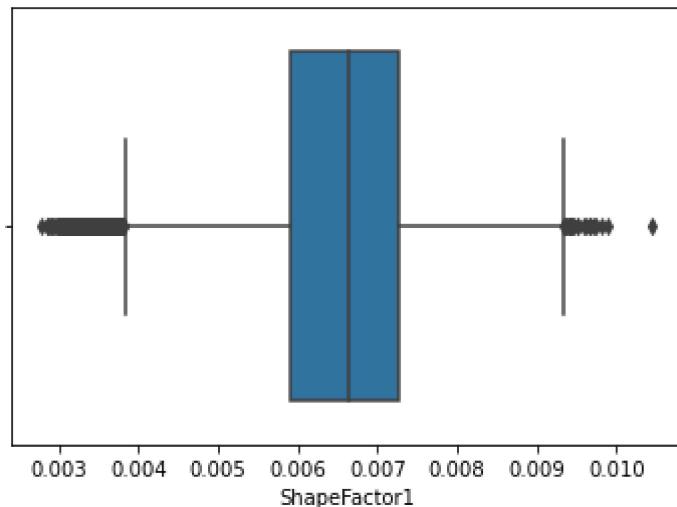
```
Out[48]: <AxesSubplot:xlabel='Compactness'>
```



13. ShapeFactor1

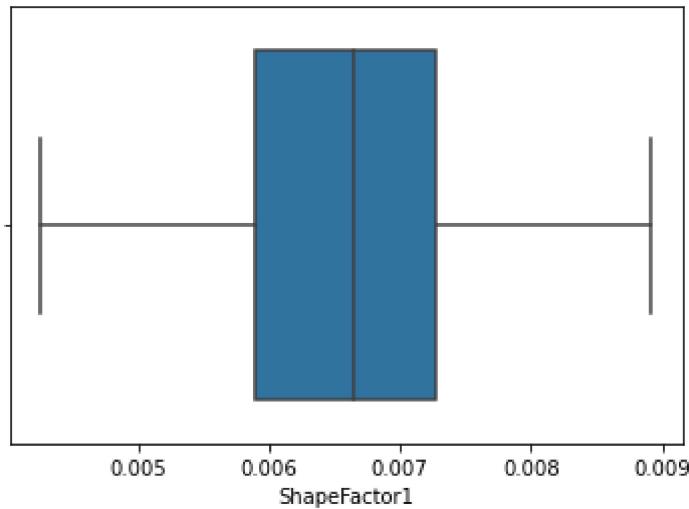
```
In [49]: sns.boxplot(df['ShapeFactor1'])
```

```
Out[49]: <AxesSubplot:xlabel='ShapeFactor1'>
```



```
In [50]: # Handling outliers using iqr
q1=df['ShapeFactor1'].quantile(0.25)
q3=df['ShapeFactor1'].quantile(0.75)
iqr=q3-q1
upper_limit=q3+1.2*iqr
lower_limit=q1-1.2*iqr
df['ShapeFactor1']=np.where(df['ShapeFactor1']>upper_limit, upper_limit, df['ShapeFactor1'])
df['ShapeFactor1']=np.where(df['ShapeFactor1']<lower_limit, lower_limit, df['ShapeFactor1'])
sns.boxplot(df['ShapeFactor1'])
```

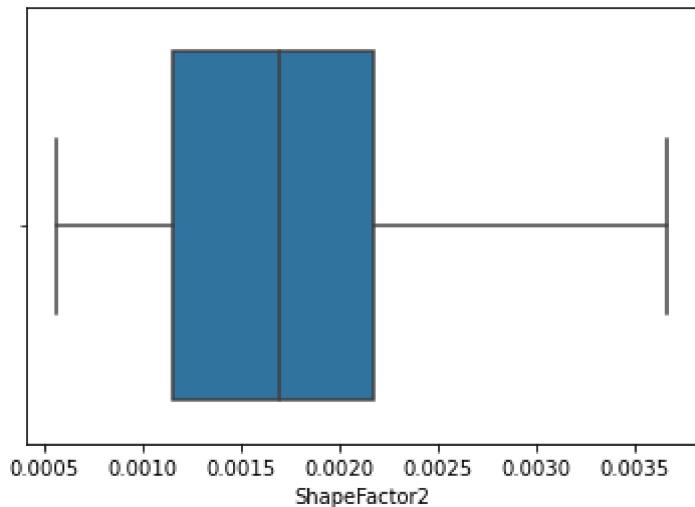
Out[50]: <AxesSubplot:xlabel='ShapeFactor1'>



14. ShapeFactor2

```
In [51]: sns.boxplot(df['ShapeFactor2'])
```

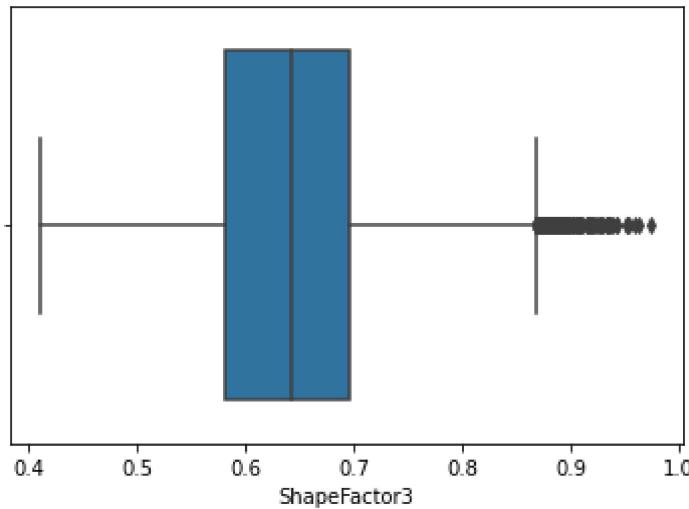
Out[51]: <AxesSubplot:xlabel='ShapeFactor2'>



15. ShapeFactor3

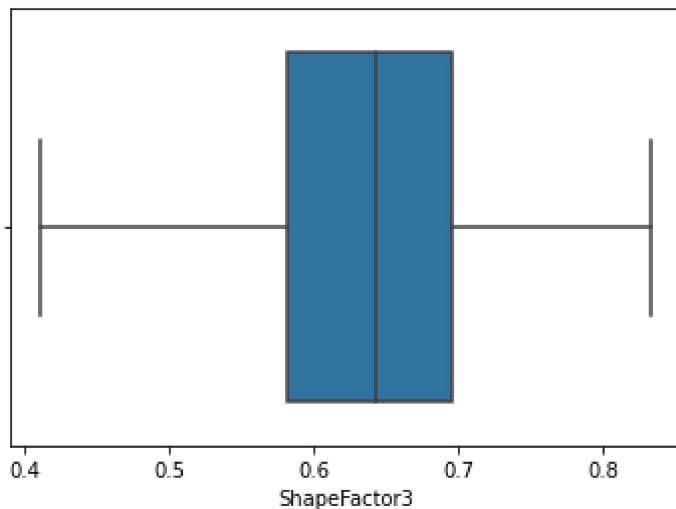
```
In [52]: sns.boxplot(df['ShapeFactor3'])
```

```
Out[52]: <AxesSubplot:xlabel='ShapeFactor3'>
```



```
In [53]: # Handling outliers using iqr
q1=df['ShapeFactor3'].quantile(0.25)
q3=df['ShapeFactor3'].quantile(0.75)
iqr=q3-q1
upper_limit=q3+1.2*iqr
df['ShapeFactor3']=np.where(df['ShapeFactor3']>upper_limit, upper_limit, df['ShapeFactor3'])
sns.boxplot(df['ShapeFactor3'])
```

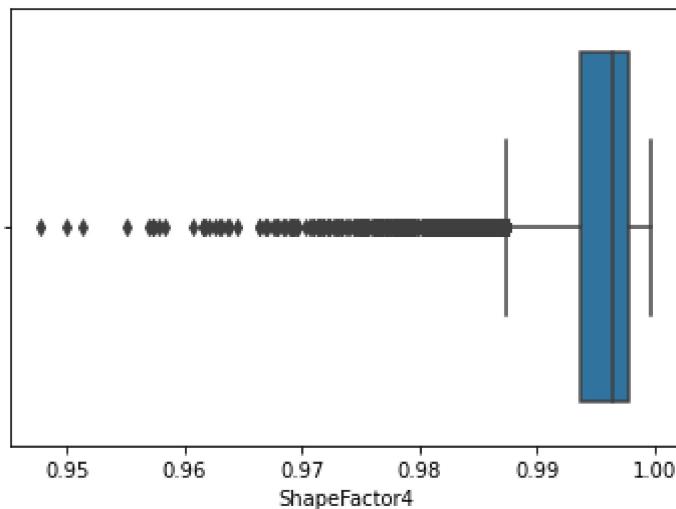
```
Out[53]: <AxesSubplot:xlabel='ShapeFactor3'>
```



16. ShapeFactor4

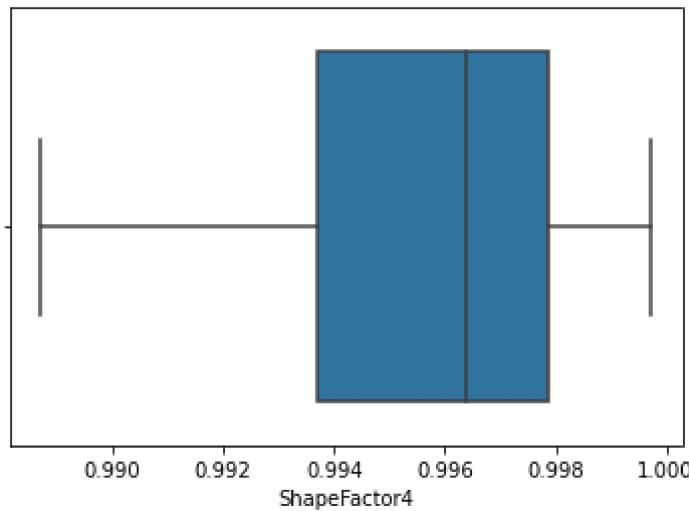
```
In [54]: sns.boxplot(df['ShapeFactor4'])
```

```
Out[54]: <AxesSubplot:xlabel='ShapeFactor4'>
```



```
In [55]: # Handling outliers using iqr
q1=df['ShapeFactor4'].quantile(0.25)
q3=df['ShapeFactor4'].quantile(0.75)
iqr=q3-q1
lower_limit=q1-1.2*iqr
df['ShapeFactor4']=np.where(df['ShapeFactor4']<lower_limit, lower_limit, df['ShapeFactor4'])
sns.boxplot(df['ShapeFactor4'])
```

```
Out[55]: <AxesSubplot:xlabel='ShapeFactor4'>
```



17. Class

In [56]: `label_encoder=LabelEncoder()
label_encoder`

Out[56]: `LabelEncoder()`

In [57]: `df['Class']=label_encoder.fit_transform(df['Class'])`

In [58]: `df`

Out[58]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRatio	Eccentricity	ConvexA
0	28395.0	610.291	208.178117	173.888747	1.197191	0.602482	2871
1	28734.0	638.018	200.524796	182.734419	1.097356	0.602482	2917
2	29380.0	624.110	212.826130	175.931143	1.209713	0.602482	2969
3	30008.0	645.884	210.557999	182.516516	1.153638	0.602482	3072
4	30140.0	620.134	201.847882	190.279279	1.060798	0.602482	3041
...
13606	42097.0	759.696	288.721612	185.944705	1.552728	1.552728	4250
13607	42101.0	757.499	281.576392	190.713136	1.476439	1.476439	4249
13608	42139.0	759.321	281.539928	191.187979	1.472582	1.472582	4256
13609	42147.0	763.779	283.382636	190.275731	1.489326	1.489326	4266
13610	42159.0	772.237	295.142741	182.204716	1.619841	1.619841	4260

13611 rows × 17 columns

Feature selection

In [59]: `x=df.drop(['Class'], axis=1)
y=df['Class']`

In [60]: `x.head()`

Out[60]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRatio	Eccentricity	ConvexArea
0	28395.0	610.291	208.178117	173.888747	1.197191	0.602482	28715.0
1	28734.0	638.018	200.524796	182.734419	1.097356	0.602482	29172.0
2	29380.0	624.110	212.826130	175.931143	1.209713	0.602482	29690.0
3	30008.0	645.884	210.557999	182.516516	1.153638	0.602482	30724.0
4	30140.0	620.134	201.847882	190.279279	1.060798	0.602482	30417.0

```
In [61]: y.head()
```

```
Out[61]: 0    5  
1    5  
2    5  
3    5  
4    5  
Name: Class, dtype: int32
```

Model training

```
In [62]: x_train, x_test, y_train, y_test= train_test_split(x, y, random_state=5, test_size=0.2)
```

```
In [63]: x_train.shape
```

```
Out[63]: (10888, 16)
```

```
In [64]: x_test.shape
```

```
Out[64]: (2723, 16)
```

Model building

1. Logistic Model

```
In [65]: lgt_model=LogisticRegression()
```

```
In [66]: lgt_model.fit(x_train, y_train)
```

```
Out[66]: LogisticRegression()
```

In [67]: # Testing Accuracy

```
y_pred=lgt_model.predict(x_test)
accu_score=accuracy_score(y_test, y_pred)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_test, y_pred)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_test, y_pred)
print('Classification report: \n', clf_report)
print('*'*30)
```

Accuracy score: 0.753947851634227

Confusion matrix:

```
[[133  0  74  0  48  0  10]
 [ 0 104  0  0  0  0  0]
 [ 79  3 230  0  12  1  1]
 [ 0  0  0 625  0  44  40]
 [ 45  0  1 22 287  0  31]
 [ 10  0  0 77  0 272  47]
 [ 15  0  0 33  40  37 402]]
```

Classification report:

	precision	recall	f1-score	support
0	0.47	0.50	0.49	265
1	0.97	1.00	0.99	104
2	0.75	0.71	0.73	326
3	0.83	0.88	0.85	709
4	0.74	0.74	0.74	386
5	0.77	0.67	0.72	406
6	0.76	0.76	0.76	527
accuracy			0.75	2723
macro avg	0.76	0.75	0.75	2723
weighted avg	0.75	0.75	0.75	2723

```
In [68]: # Training Accuracy
y_pred_train=lgt_model.predict(x_train)
accu_score=accuracy_score(y_train, y_pred_train)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_train, y_pred_train)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_train, y_pred_train)
print('Classification report: \n', clf_report)
print('*'*30)
```

```
Accuracy score: 0.7521124173401911
*****
Confusion matrix:
[[ 555     5   316     0   143     0    38]
 [  0   418     0     0     0     0     0]
 [ 317    20   916     0    47     0     4]
 [  0     0    0 2465     0   221   151]
 [ 126     0     7   98 1139     1   171]
 [  49     0     0   316     2 1078   176]
 [  35     0     0   182   127   147 1618]]
*****
Classification report:
      precision    recall  f1-score   support
          0       0.51     0.53     0.52     1057
          1       0.94     1.00     0.97     418
          2       0.74     0.70     0.72     1304
          3       0.81     0.87     0.84     2837
          4       0.78     0.74     0.76     1542
          5       0.74     0.67     0.70     1621
          6       0.75     0.77     0.76     2109
          accuracy                           0.75     10888
         macro avg       0.75     0.75     0.75     10888
      weighted avg       0.75     0.75     0.75     10888
*****

```

2. KNN

```
In [69]: knn_clf=KNeighborsClassifier()
knn_clf
```

```
Out[69]: KNeighborsClassifier()
```

```
In [70]: knn_clf.fit(x_train, y_train)
```

```
Out[70]: KNeighborsClassifier()
```

```
In [71]: # Training Accuracy
y_pred_train=knn_clf.predict(x_train)
accu_score=accuracy_score(y_train, y_pred_train)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_train, y_pred_train)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_train, y_pred_train)
print('Classification report: \n', clf_report)
print('*'*30)
```

```
Accuracy score: 0.8170462894930198
*****
Confusion matrix:
[[ 707    1  245    1   78    0   25]
 [  2  416    0    0    0    0    0]
 [ 232    3 1017    0   47    1    4]
 [  0    0    0 2656    2   83   96]
 [ 78    0   59   39 1199    2  165]
 [  3    0    1 193   22 1201  201]
 [ 10    0    2 214   114   69 1700]]
*****
Classification report:
      precision    recall  f1-score   support
          0       0.69     0.67     0.68     1057
          1       0.99     1.00     0.99      418
          2       0.77     0.78     0.77     1304
          3       0.86     0.94     0.89     2837
          4       0.82     0.78     0.80     1542
          5       0.89     0.74     0.81     1621
          6       0.78     0.81     0.79     2109
          accuracy                           0.82     10888
          macro avg       0.83     0.81     0.82     10888
          weighted avg      0.82     0.82     0.82     10888
*****

```

In [72]: # Testing Accuracy

```
y_pred=knn_clf.predict(x_test)
accu_score=accuracy_score(y_test, y_pred)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_test, y_pred)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_test, y_pred)
print('Classification report: \n', clf_report)
print('*'*30)
```

Accuracy score: 0.7275064267352185

Confusion matrix:

```
[[138   0   86   0   35   0   6]
 [  0 104   0   0   0   0   0]
 [ 98   1 203   0   21   1   2]
 [  0   0   0 638   0   34   37]
 [ 29   0   27   9 273   0   48]
 [  0   0   1  71   2 249   83]
 [  0   0   0  57   72   22 376]]
```

Classification report:

	precision	recall	f1-score	support
0	0.52	0.52	0.52	265
1	0.99	1.00	1.00	104
2	0.64	0.62	0.63	326
3	0.82	0.90	0.86	709
4	0.68	0.71	0.69	386
5	0.81	0.61	0.70	406
6	0.68	0.71	0.70	527
accuracy			0.73	2723
macro avg	0.74	0.73	0.73	2723
weighted avg	0.73	0.73	0.73	2723

3. Decision Tree

In [80]: dt_clf=DecisionTreeClassifier(criterion='entropy', max_depth=8)

In [81]: dt_clf.fit(x_train, y_train)

Out[81]: DecisionTreeClassifier(criterion='entropy', max_depth=8)

In [82]: # Training Accuracy

```
y_pred_train=dt_clf.predict(x_train)
accu_score=accuracy_score(y_train, y_pred_train)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_train, y_pred_train)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_train, y_pred_train)
print('Classification report: \n', clf_report)
print('*'*30)
```

Accuracy score: 0.9369948567229978

Confusion matrix:

[981	0	54	1	3	6	12]
[0	418	0	0	0	0	0]
[32	0	1255	0	14	1	2]
[0	0	0	2724	0	14	99]
[7	0	28	12	1475	0	20]
[3	0	0	52	0	1538	28]
[3	0	1	250	26	18	1811]]

Classification report:

	precision	recall	f1-score	support
0	0.96	0.93	0.94	1057
1	1.00	1.00	1.00	418
2	0.94	0.96	0.95	1304
3	0.90	0.96	0.93	2837
4	0.97	0.96	0.96	1542
5	0.98	0.95	0.96	1621
6	0.92	0.86	0.89	2109
accuracy			0.94	10888
macro avg	0.95	0.94	0.95	10888
weighted avg	0.94	0.94	0.94	10888

In [83]: # Testing Accuracy

```
y_pred=dt_clf.predict(x_test)
accu_score=accuracy_score(y_test, y_pred)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_test, y_pred)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_test, y_pred)
print('Classification report: \n', clf_report)
print('*'*30)
```

Accuracy score: 0.9177377892030848

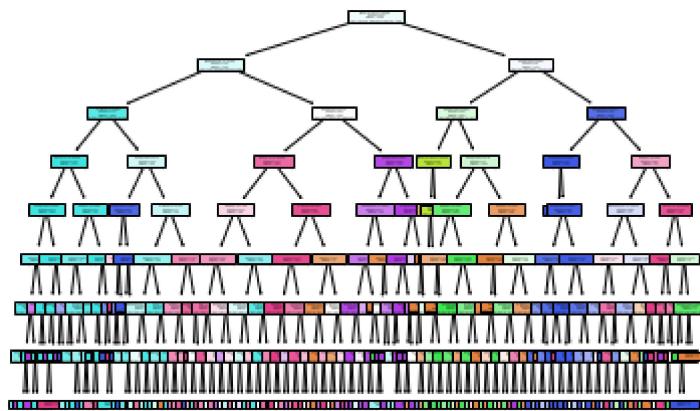
Confusion matrix:

```
[[241   0    8    0    6    4    6]
 [  0 104   0    0    0    0    0]
 [ 13   0 305   0    6    2    0]
 [  0   0    0 659   1    7   42]
 [  2   0    9    2 369   0    4]
 [  1   0    1   25   0 373   6]
 [  4   0    3   51   10   11 448]]
```

Classification report:

	precision	recall	f1-score	support
0	0.92	0.91	0.92	265
1	1.00	1.00	1.00	104
2	0.94	0.94	0.94	326
3	0.89	0.93	0.91	709
4	0.94	0.96	0.95	386
5	0.94	0.92	0.93	406
6	0.89	0.85	0.87	527
accuracy			0.92	2723
macro avg	0.93	0.93	0.93	2723
weighted avg	0.92	0.92	0.92	2723

```
In [84]: tree.plot_tree(dt_clf, filled=True, feature_names=x.columns, )
plt.figure(figsize=(50, 20))
plt.show()
plt.savefig('Decision_Tree_Dry_bean.png', dpi=500)
```



<Figure size 3600x1440 with 0 Axes>

4. Random forest

```
In [85]: rf_clf=RandomForestClassifier(random_state=13)
rf_clf.fit(x_train, y_train)
```

```
Out[85]: RandomForestClassifier(random_state=13)
```

In [86]: # Training Accuracy

```
y_pred_train=rf_clf.predict(x_train)
accu_score=accuracy_score(y_train, y_pred_train)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_train, y_pred_train)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_train, y_pred_train)
print('Classification report: \n', clf_report)
print('*'*30)
```

Accuracy score: 0.9997244673034533

Confusion matrix:

```
[[1057    0    0    0    0    0    0]
 [  0  418    0    0    0    0    0]
 [  0    0 1304    0    0    0    0]
 [  0    0    0 2837    0    0    0]
 [  0    0    0    0 1542    0    0]
 [  0    0    0    2    0 1619    0]
 [  0    0    0    0    1    0 2108]]
```

Classification report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1057
1	1.00	1.00	1.00	418
2	1.00	1.00	1.00	1304
3	1.00	1.00	1.00	2837
4	1.00	1.00	1.00	1542
5	1.00	1.00	1.00	1621
6	1.00	1.00	1.00	2109
accuracy			1.00	10888
macro avg	1.00	1.00	1.00	10888
weighted avg	1.00	1.00	1.00	10888

In [87]: # Testing Accuracy

```
y_pred=rf_clf.predict(x_test)
accu_score=accuracy_score(y_test, y_pred)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_test, y_pred)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_test, y_pred)
print('Classification report: \n', clf_report)
print('*'*30)
```

Accuracy score: 0.9397723099522586

Confusion matrix:

[245	0	9	0	2	4	5]
[0	104	0	0	0	0	0]
[12	0	306	0	6	2	0]
[0	0	0	673	0	6	30]
[0	0	5	3	374	0	4]
[1	0	0	10	0	387	8]
[2	0	3	36	8	8	470]]

Classification report:

	precision	recall	f1-score	support
0	0.94	0.92	0.93	265
1	1.00	1.00	1.00	104
2	0.95	0.94	0.94	326
3	0.93	0.95	0.94	709
4	0.96	0.97	0.96	386
5	0.95	0.95	0.95	406
6	0.91	0.89	0.90	527
accuracy			0.94	2723
macro avg	0.95	0.95	0.95	2723
weighted avg	0.94	0.94	0.94	2723

Hyperparameter tuning

```
In [88]: rf_model = RandomForestClassifier(random_state=10)
```

```
hyp = {  
    'n_estimators': np.arange(10,150),  
    'criterion':['gini','entropy'],  
    'max_depth':np.arange(5,15),  
    'min_samples_split':np.arange(5,20) ,  
    'min_samples_leaf': np.arange(4,15) ,  
    'max_features':['auto','log2'],  
    'random_state' : np.arange(1,10)  
}
```

```
rscv_rf_clf = RandomizedSearchCV(rf_model, hyp, cv=5)  
rscv_rf_clf.fit(x_train, y_train)  
rscv_rf_clf.best_params_
```

```
Out[88]: {'random_state': 2,  
          'n_estimators': 92,  
          'min_samples_split': 15,  
          'min_samples_leaf': 6,  
          'max_features': 'log2',  
          'max_depth': 11,  
          'criterion': 'entropy'}
```

In [89]: # Training Accuracy

```
y_pred_train=rscv_rf_clf.predict(x_train)
accu_score=accuracy_score(y_train, y_pred_train)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_train, y_pred_train)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_train, y_pred_train)
print('Classification report: \n', clf_report)
print('*'*30)
```

Accuracy score: 0.9494856722997795

Confusion matrix:

```
[[ 994    1   36     0     4     7   15]
 [  0  418    0     0     0     0     0]
 [ 14    1 1263     0    15     2     9]
 [  0    0     0 2724     2    28   83]
 [  5    0    16    11 1479     0    31]
 [  5    0     0    27     0 1553    36]
 [  4    0     1 159    19    19 1907]]
```

Classification report:

	precision	recall	f1-score	support
0	0.97	0.94	0.96	1057
1	1.00	1.00	1.00	418
2	0.96	0.97	0.96	1304
3	0.93	0.96	0.95	2837
4	0.97	0.96	0.97	1542
5	0.97	0.96	0.96	1621
6	0.92	0.90	0.91	2109
accuracy			0.95	10888
macro avg	0.96	0.96	0.96	10888
weighted avg	0.95	0.95	0.95	10888

In [90]: # Testing Accuracy

```
y_pred=rscv_rf_clf.predict(x_test)
accu_score=accuracy_score(y_test, y_pred)
print('Accuracy score: ', accu_score)
print('*'*30)
cnf_matrix=confusion_matrix(y_test, y_pred)
print('Confusion matrix:\n', cnf_matrix)
print('*'*30)
clf_report=classification_report(y_test, y_pred)
print('Classification report: \n', clf_report)
print('*'*30)
```

Accuracy score: 0.9357326478149101

Confusion matrix:

[244	0	11	0	2	2	6]
[0	104	0	0	0	0	0]
[10	0	306	0	7	1	2]
[0	0	0	669	0	7	33]
[0	0	5	2	372	0	7]
[1	0	0	11	0	385	9]
[2	0	3	41	6	7	468]]

Classification report:

	precision	recall	f1-score	support
0	0.95	0.92	0.93	265
1	1.00	1.00	1.00	104
2	0.94	0.94	0.94	326
3	0.93	0.94	0.93	709
4	0.96	0.96	0.96	386
5	0.96	0.95	0.95	406
6	0.89	0.89	0.89	527
accuracy			0.94	2723
macro avg	0.95	0.94	0.94	2723
weighted avg	0.94	0.94	0.94	2723

Creating pickle file

In [91]: import pickle

In [92]: with open('rf_clf.pkl', 'wb') as f:
pickle.dump(rscv_rf_clf, f)