Mini Project 2

Data Set Information:

Seven different types of dry beans were used in this project, taking into account the features such as form, shape, type, and structure by the market situation. Use best machine learning algorithm to classify the most well-known 7 types of beans in Turkey; Barbunya, Bombay, Cali, Dermason, Horoz, Seker and Sira, depending only on dimension and shape features of bean varieties with no external discriminatory features.

Features Information:

1. Area (A): The area of a bean zone and the number of pixels within its boundaries.
2. Perimeter (P): Bean circumference is defined as the length of its border.
3. Major axis length (L): The distance between the ends of the longest line that can be drawn from a bean.
4. Minor axis length (l): The longest line that can be drawn from the bean while standing perpendicular to the main axis.
5. Aspect ratio (K): Defines the relationship between L and l.
6. Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.
7. Convex area (C): Number of pixels in the smallest convex polygon that can contain the area of a bean seed.
8. Equivalent diameter (Ed): The diameter of a circle having the same area as a bean seed area.
9. Extent (Ex): The ratio of the pixels in the bounding box to the bean area.
10. Solidity (S): Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.
11. Roundness (R): Calculated with the following formula: (4piA)/(P^2)
12. Compactness (CO): Measures the roundness of an object: Ed/L
13. ShapeFactor1 (SF1)
14. ShapeFactor2 (SF2)
15. ShapeFactor3 (SF3)
16. ShapeFactor4 (SF4)
17. Class (Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira)

Importing Liberaries for Data Processing, Loading

In [1]:
```
import numpy as np
import pandas as pd
from sklearn import metrics # for calculating rootmean square
```

Importing Data Visualization Liberies

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing Machine Learning Liberaries

In [3]:
```
# Models to be used
from lazypredict.Supervised import LazyClassifier # for checking best model


# Evaluation of the model
from sklearn.metrics import classification_report, accuracy_score, average_precision_score, f1_score


# Preprocessing of the data
from sklearn.preprocessing import LabelEncoder, RobustScaler
from sklearn.model_selection import train_test_split
import joblib # to extract data
```

Loading the Data

In [4]:
```
df=pd.read_csv("data.csv")
```

Exploratory Data Analysis (EDA)

In [5]:
```
df.head() # previewing data
```

Out[5]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solid
0	28395	610.29	208.18	173.89	1.20	0.55	28715	190.14	0.76	0
1	28734	638.02	200.52	182.73	1.10	0.41	29172	191.27	0.78	0
2	29380	624.11	212.83	175.93	1.21	0.56	29690	193.41	0.78	0
3	30008	645.88	210.56	182.52	1.15	0.50	30724	195.47	0.78	0
4	30140	620.13	201.85	190.28	1.06	0.33	30417	195.90	0.77	0

In [6]:
```
df.size
```

Out[6]: 231387

In [7]:
```
df.shape
```

Out[7]: (13611, 17)

Description and Information About Dataset

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

Out[8]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Ex
count	13611.00	13611.00	13611.00	13611.00	13611.00	13611.00	13611.00	13611.00	1361
mean	53048.28	855.28	320.14	202.27	1.58	0.75	53768.20	253.06	
std	29324.10	214.29	85.69	44.97	0.25	0.09	29774.92	59.18	
min	20420.00	524.74	183.60	122.51	1.02	0.22	20684.00	161.24	
25%	36328.00	703.52	253.30	175.85	1.43	0.72	36714.50	215.07	
50%	44652.00	794.94	296.88	192.43	1.55	0.76	45178.00	238.44	
75%	61332.00	977.21	376.50	217.03	1.71	0.81	62294.00	279.45	
max	254616.00	1985.37	738.86	460.20	2.43	0.91	263261.00	569.37	

In [9]:
```
df.info() # for Columns /features data type and having null value or not
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13611 entries, 0 to 13610
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Area             13611 non-null  int64  
 1   Perimeter        13611 non-null  float64
 2   MajorAxisLength  13611 non-null  float64
 3   MinorAxisLength  13611 non-null  float64
 4   AspectRation     13611 non-null  float64
 5   Eccentricity     13611 non-null  float64
 6   ConvexArea       13611 non-null  int64  
 7   EquivDiameter    13611 non-null  float64
 8   Extent           13611 non-null  float64
 9   Solidity         13611 non-null  float64
 10  roundness        13611 non-null  float64
 11  Compactness      13611 non-null  float64
 12  ShapeFactor1     13611 non-null  float64
 13  ShapeFactor2     13611 non-null  float64
 14  ShapeFactor3     13611 non-null  float64
 15  ShapeFactor4     13611 non-null  float64
 16  Class            13611 non-null  object 
dtypes: float64(14), int64(2), object(1)
memory usage: 1.8+ MB
```

In [38]:
```
# lets check summery for all the Classes

print("Average Area : {0:.2f}".format(df['Area'].mean()))
print("Average Perimeter : {0:.2f}".format(df['Perimeter'].mean()))
print("Average MajorAxisLength : {0:.2f}".format(df['MajorAxisLength'].mean()))
print("Average MinorAxisLength: {0:.2f}".format(df['MinorAxisLength'].mean()))
print("Average AspectRation : {0:.2f}".format(df['AspectRation'].mean()))
print("Average Eccentricity : {0:.2f}".format(df['Eccentricity'].mean()))
print("Average ConvexArea: {0:.2f}".format(df['ConvexArea'].mean()))
print("Average EquivDiameter: {0:.2f}".format(df['EquivDiameter'].mean()))
print("Average Extent : {0:.2f}".format(df['Extent'].mean()))
print("Average Solidity : {0:.2f}".format(df['Solidity'].mean()))
print("Average roundness: {0:.2f}".format(df['roundness'].mean()))
print("Average Compactness : {0:.2f}".format(df['Compactness'].mean()))
print("Average ShapeFactor1: {0:.2f}".format(df['ShapeFactor1'].mean()))
print("Average ShapeFactor2: {0:.2f}".format(df['ShapeFactor2'].mean()))
print("Average ShapeFactor3 : {0:.2f}".format(df['ShapeFactor3'].mean()))
print("Average ShapeFactor4 : {0:.2f}".format(df['ShapeFactor4'].mean()))
```

```
Average Area : 50166.95
Average Perimeter : 847.33
Average MajorAxisLength : 318.31
Average MinorAxisLength: 198.67
Average AspectRation : 1.58
Average Eccentricity : 0.75
Average ConvexArea: 50869.02
Average EquivDiameter: 249.47
Average Extent : 0.75
Average Solidity : 0.99
Average roundness: 0.87
Average Compactness : 0.80
Average ShapeFactor1: 0.01
Average ShapeFactor2: 0.00
Average ShapeFactor3 : 0.64
Average ShapeFactor4 : 1.00
```

Data Preprocessing

Finding Null Values

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

Out[10]:
```
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    0
AspectRation       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
```

Finding Duplicate Data

In [11]:
```
df.drop_duplicates(inplace=True)
```

Finding Skewness in Data

In [12]:
```
df.skew().sort_values(ascending=True)
```

Out[12]:
```
ShapeFactor4      -2.76
Solidity          -2.55
Eccentricity      -1.06
Extent            -0.90
roundness         -0.65
ShapeFactor1      -0.53
Compactness        0.04
ShapeFactor3       0.24
ShapeFactor2       0.29
AspectRation       0.59
MajorAxisLength    1.37
Perimeter          1.63
EquivDiameter      1.95
MinorAxisLength    2.23
ConvexArea         2.94
Area               2.95
dtype: float64
```

Visualization of Data (Plot Analysis)

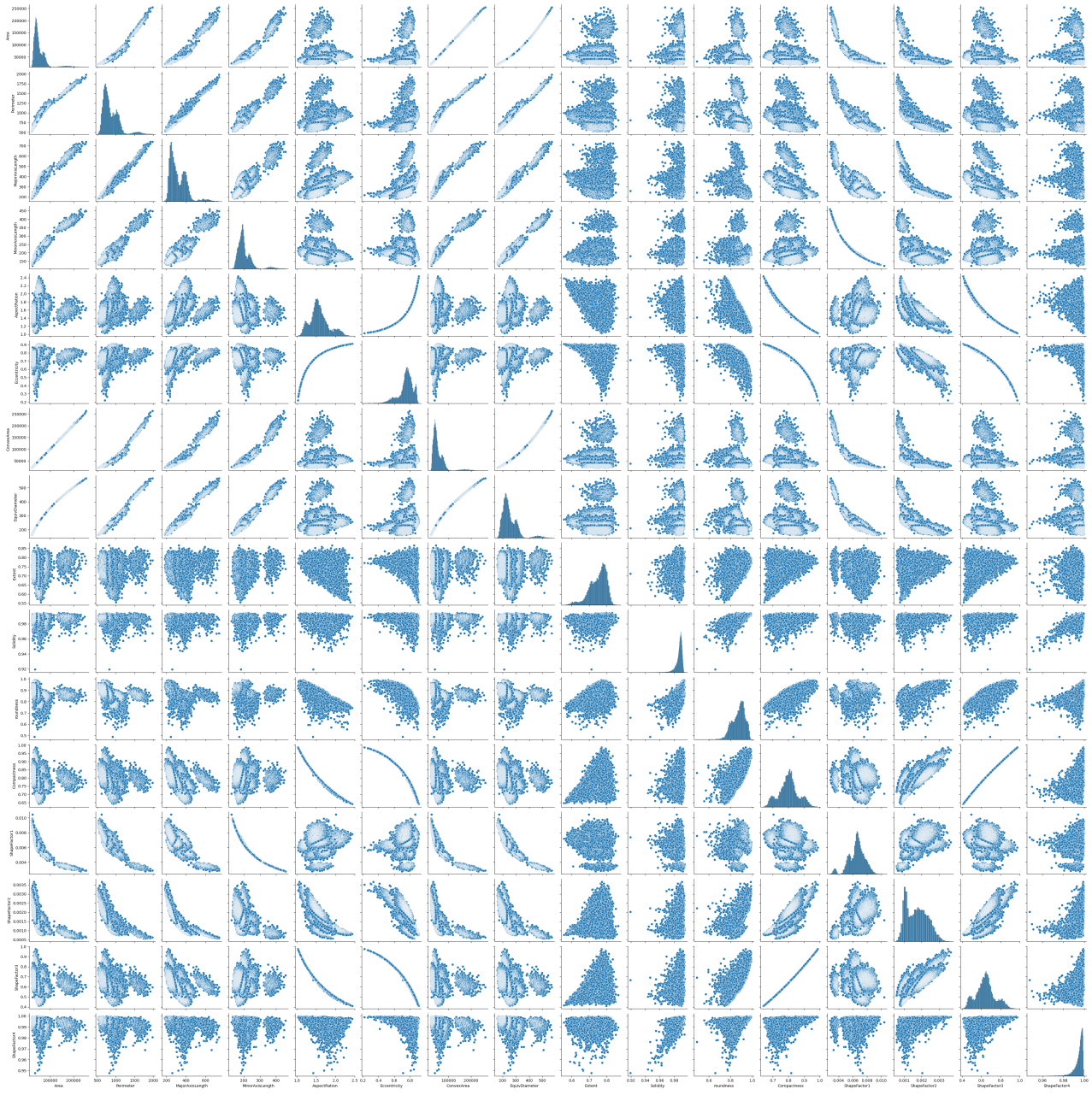
In [13]:
```
df_col = df.columns
df_col
```

Out[13]:
```
Index(['Area', 'Perimeter', 'MajorAxisLength', 'MinorAxisLength',
       'AspectRation', 'Eccentricity', 'ConvexArea', 'EquivDiameter', 'Extent',
       'Solidity', 'roundness', 'Compactness', 'ShapeFactor1', 'ShapeFactor2',
       'ShapeFactor3', 'ShapeFactor4', 'Class'],
      dtype='object')
```

Pair Plot

In [14]:
```
sns.pairplot(df)
```

Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x1681636fee0>
```



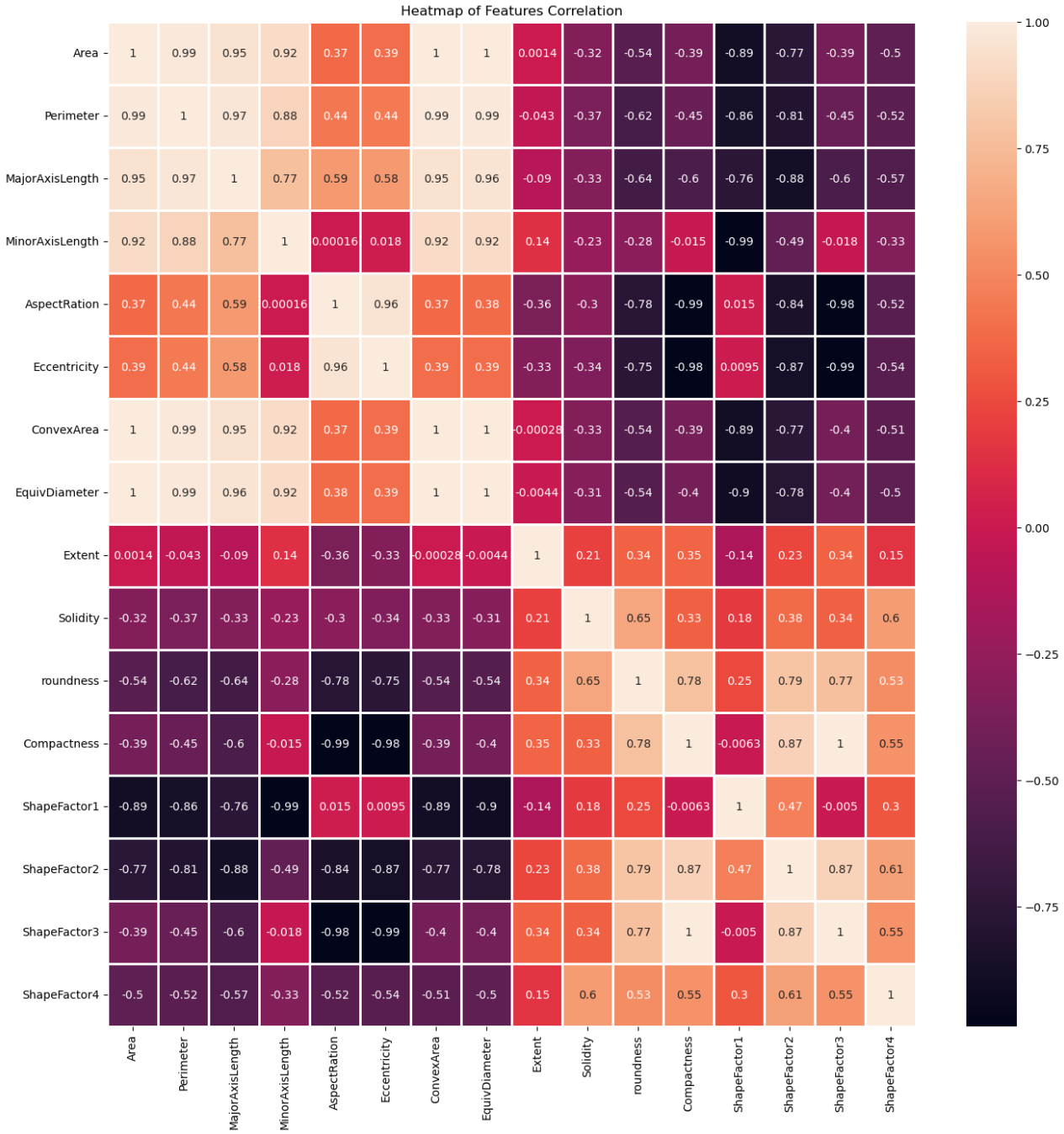
Heatmap

In [15]:
```
df.corr()
```

Out[15]:

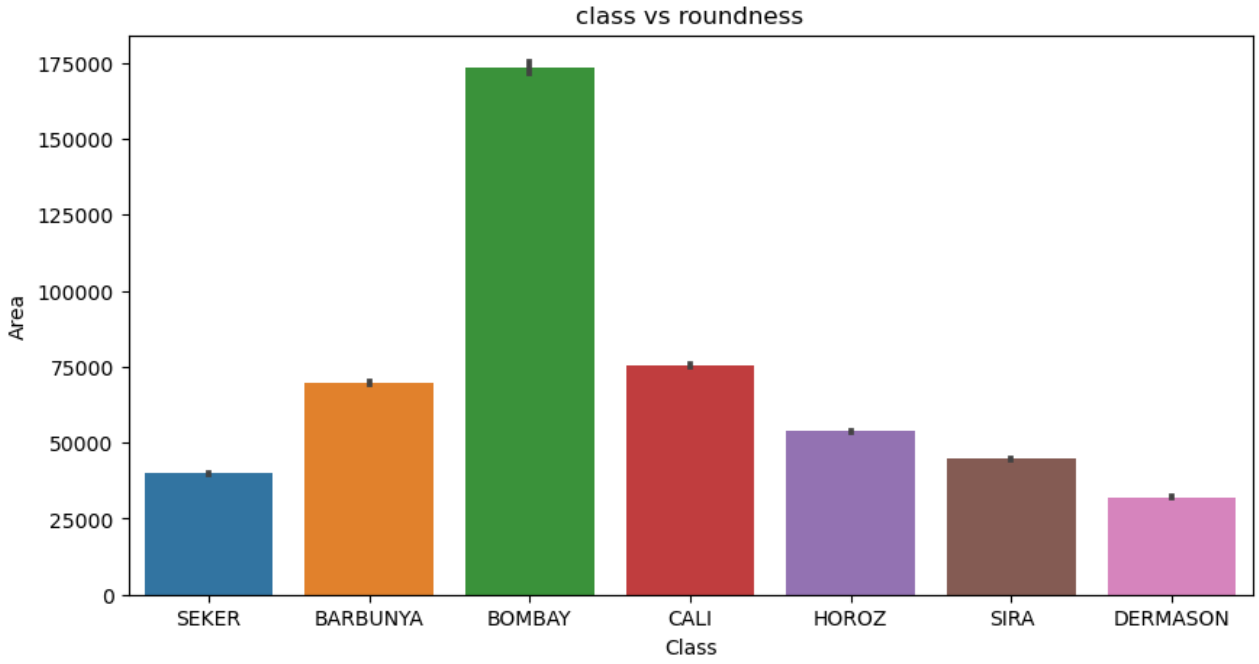
	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter
Area	1.00	0.97	0.93	0.95	0.24	0.27	1.00	0.98
Perimeter	0.97	1.00	0.98	0.91	0.39	0.39	0.97	0.99
MajorAxisLength	0.93	0.98	1.00	0.83	0.55	0.54	0.93	0.96
MinorAxisLength	0.95	0.91	0.83	1.00	-0.01	0.02	0.95	0.95
AspectRation	0.24	0.39	0.55	-0.01	1.00	0.92	0.25	0.31
Eccentricity	0.27	0.39	0.54	0.02	0.92	1.00	0.27	0.32
ConvexArea	1.00	0.97	0.93	0.95	0.25	0.27	1.00	0.99
EquivDiameter	0.98	0.99	0.96	0.95	0.31	0.32	0.99	1.00
Extent	0.05	-0.02	-0.08	0.15	-0.37	-0.32	0.05	0.03
Solidity	-0.20	-0.30	-0.28	-0.16	-0.27	-0.30	-0.21	-0.23
roundness	-0.36	-0.55	-0.60	-0.21	-0.76	-0.72	-0.36	-0.44
Compactness	-0.27	-0.41	-0.57	-0.02	-0.99	-0.97	-0.27	-0.33
ShapeFactor1	-0.85	-0.87	-0.78	-0.95	0.02	0.02	-0.85	-0.89
ShapeFactor2	-0.64	-0.77	-0.86	-0.48	-0.84	-0.86	-0.64	-0.71
ShapeFactor3	-0.27	-0.41	-0.57	-0.02	-0.98	-0.98	-0.28	-0.33
ShapeFactor4	-0.36	-0.43	-0.48	-0.27	-0.45	-0.45	-0.36	-0.39

In [52]:
```
corr=df.corr()
plt.subplots(figsize = (16, 16))
plt.title('Heatmap of Features Correlation')
hmap = sns.heatmap(corr, linewidth = 0.80, annot=True,linecolor='white',robust=True)
plt.show()
```



Bar Plot

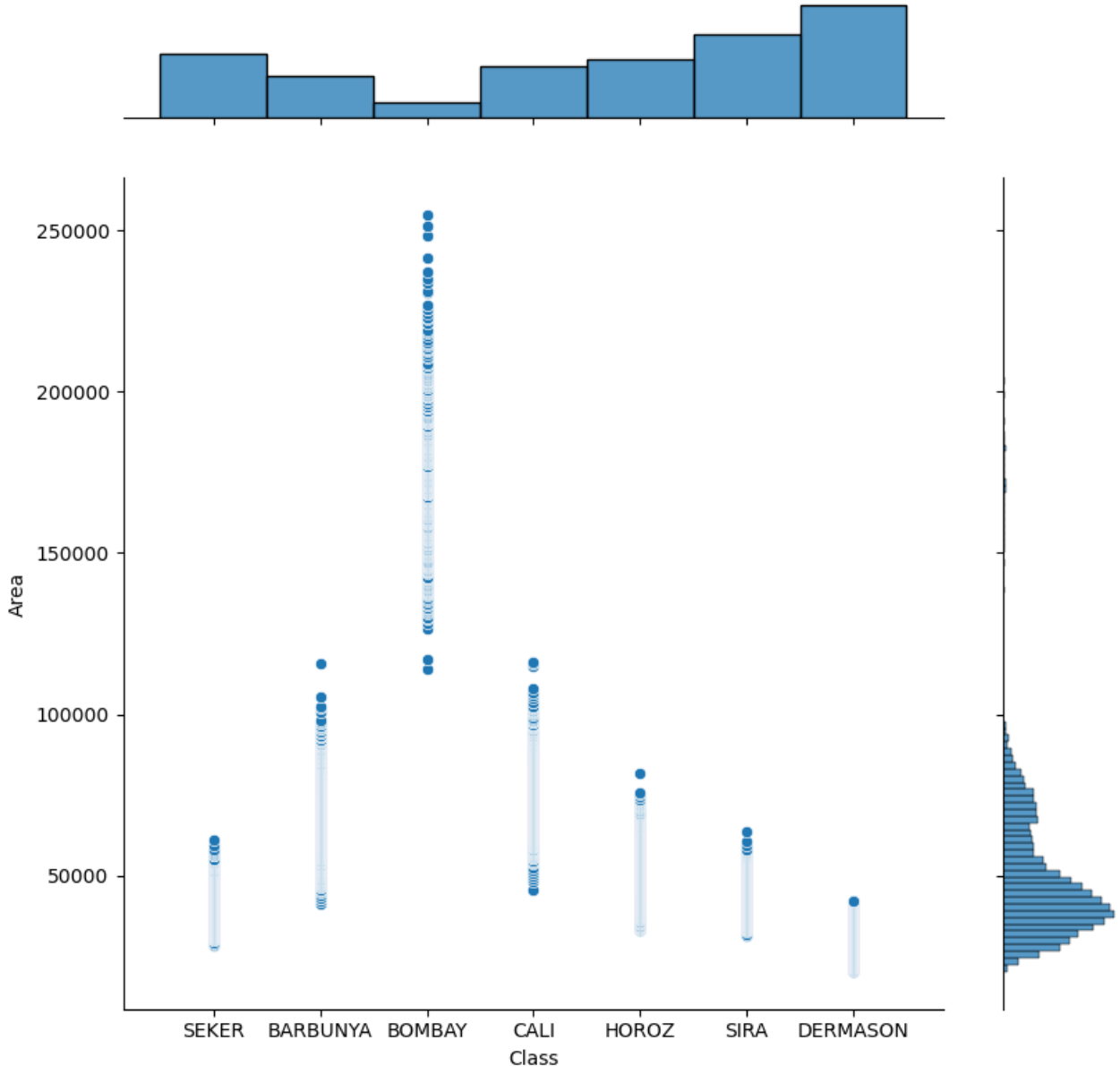
In [17]:
```
#bax plot for class vs roundness

plt.figure(figsize = (10, 5))
sns.barplot(x = 'Class', y = "Area", data = df)

plt.title('class vs roundness')
plt.show()
```



Scatter Plot

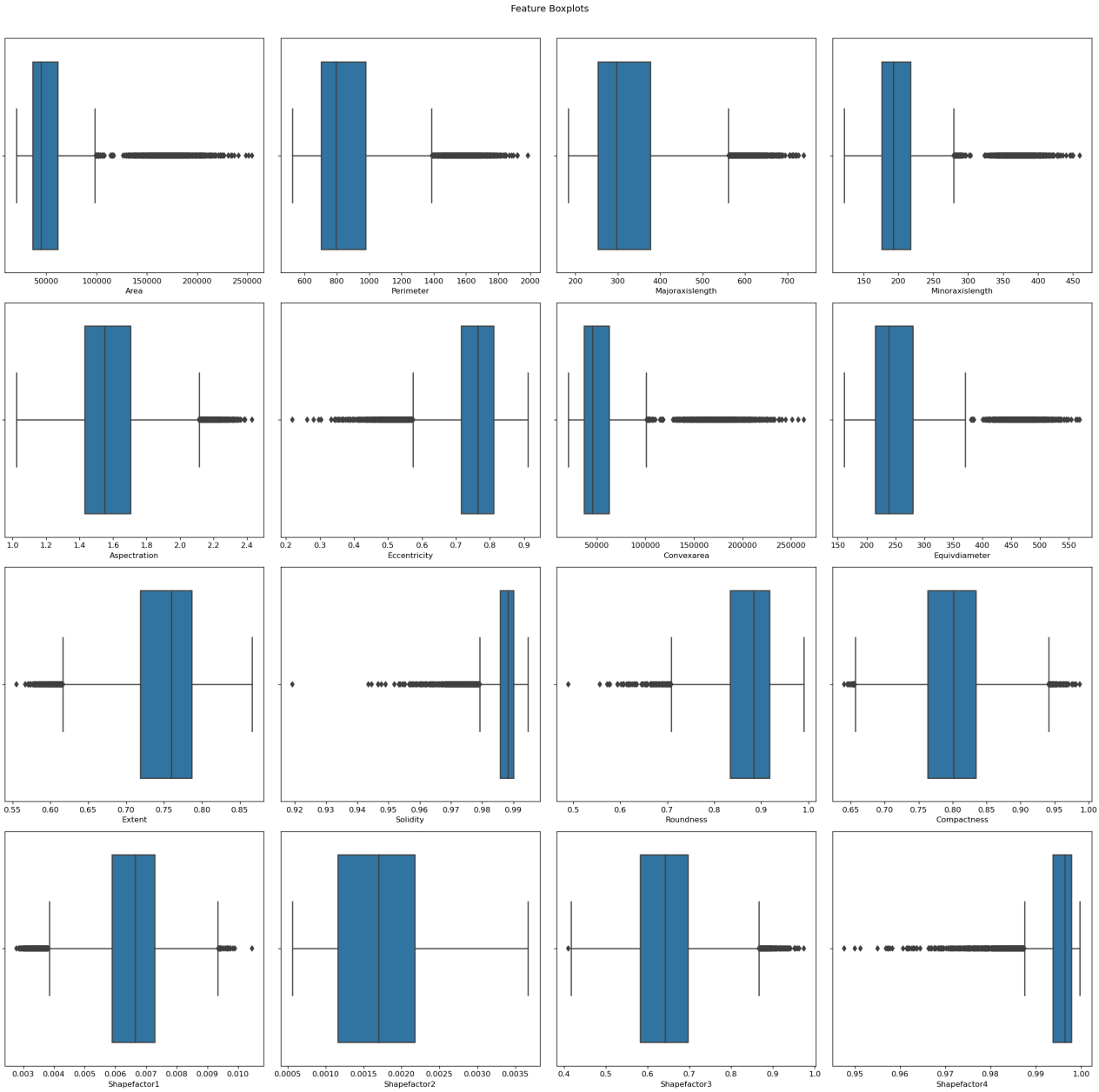
In [18]:
```
plt.figure(figsize = (12, 12))
sns.jointplot(data=df, x="Class", y="Area", kind='scatter',height=8,ratio=5,space=0.5)
plt.show()
```

```
<Figure size 1200x1200 with 0 Axes>
```



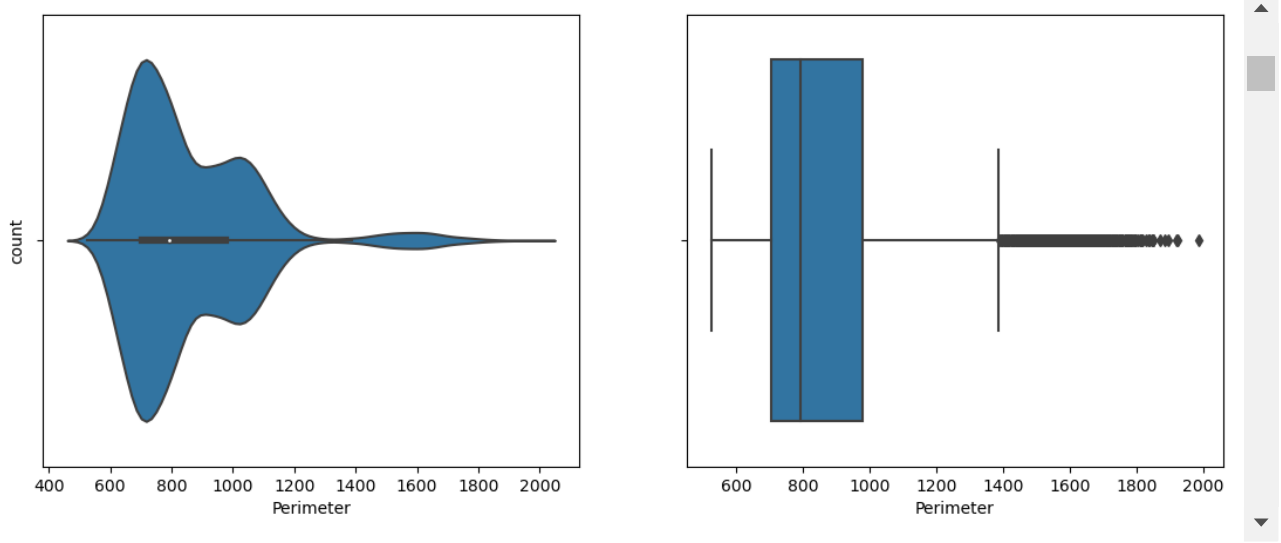
Box Plot

In [19]:
```
fig, axes = plt.subplots(4, 4, figsize=(20, 20), dpi=120)
for i, j in zip(corr[:8], axes.flatten()):
    sns.boxplot(data=df, x=i, ax=j)
    j.set_xlabel(f"{i.title().replace('_', ' ')}")
plt.suptitle(f"Feature Boxplots")
plt.tight_layout()
plt.subplots_adjust(top=0.95);
plt.show()
```



Outlier Detection

In [20]:
```
# Lets check outliers

for i in df:
    if i !='Class':
        print(i)
        print("Skewness : ", round(df[i].skew(),3))
        plt.figure(figsize=(13,5))
        plt.subplot(1,2,1)
        sns.violinplot(df[i])
        plt.ylabel('count')
        plt.subplot(1,2,2)
        sns.boxplot(x=df[i])
        plt.show()
```

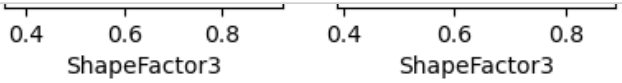


Treatment of Outliers

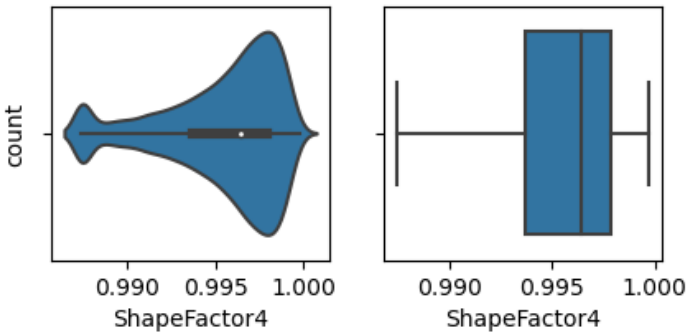
In [21]:
```
#Creating Function to remove Outliers

def remove_outliers(df):
    for col in df.columns:
        if col != 'Class':
            q25 = np.percentile(df[col] , 25)
            q75 = np.percentile(df[col] , 75)
            iqr = q75 - q25
            cut_off = iqr * 1.5
            lo = q25 - cut_off
            up = q75 + cut_off
            df[col] = df[col].clip(upper = up)
            df[col] = df[col].clip(lower=lo)
```

In [22]:
```
# Function calling

remove_outliers(df)
```

In [25]:
```
# After removing outliers

for i in df:
    if i !='Class':
        print(i)
        print("Skewness : ", round(df[i].skew(),3))
        plt.figure(figsize=(10,2))
        plt.subplot(1,4,1)
        sns.violinplot(df[i])
        plt.ylabel('count')
        plt.subplot(1,4,2)
        sns.boxplot(x=df[i])
        plt.show()
```



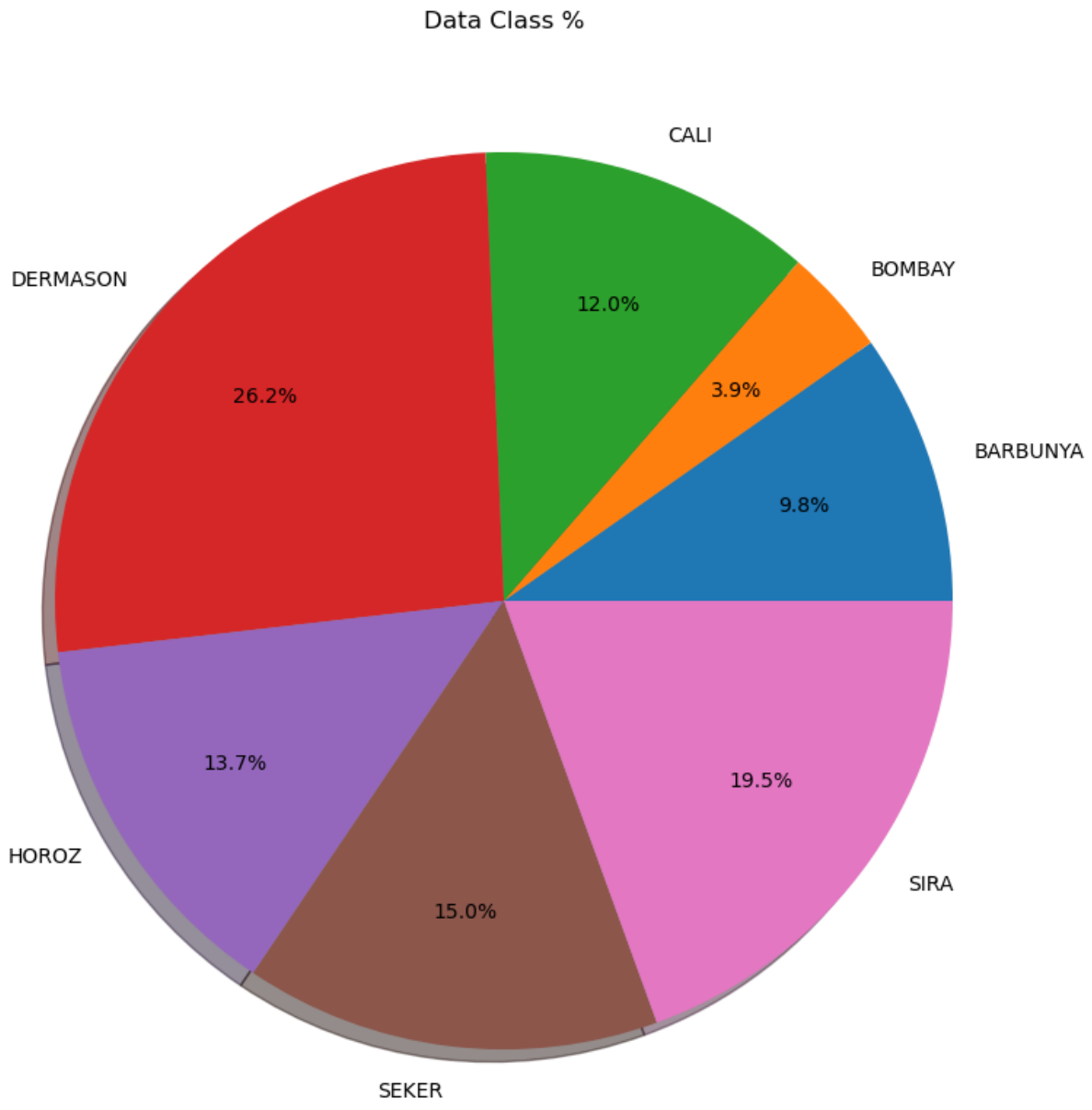

```
ShapeFactor4
Skewness :  -1.04
```



Insights: 1. Data is Already Imbalanced; 2. Data Contain Outlier That Require Handling; 3. Data Features Skewness Needed Workout

Lets Check How Unbalance the Data is

In [30]:
```
labels, counts = np.unique(df.Class, return_counts=True)
plt.figure(figsize = (10,10))
plt.pie(counts, autopct='%1.1f%%', labels=labels, pctdistance=0.7,shadow=True,counterclock=True,norma
plt.title('Data Class %')
plt.show()
```



Lets Try Balancing Data (SMOTE)

In [31]:
```
from imblearn.over_sampling import SMOTE


smote = SMOTE(random_state=42)
```

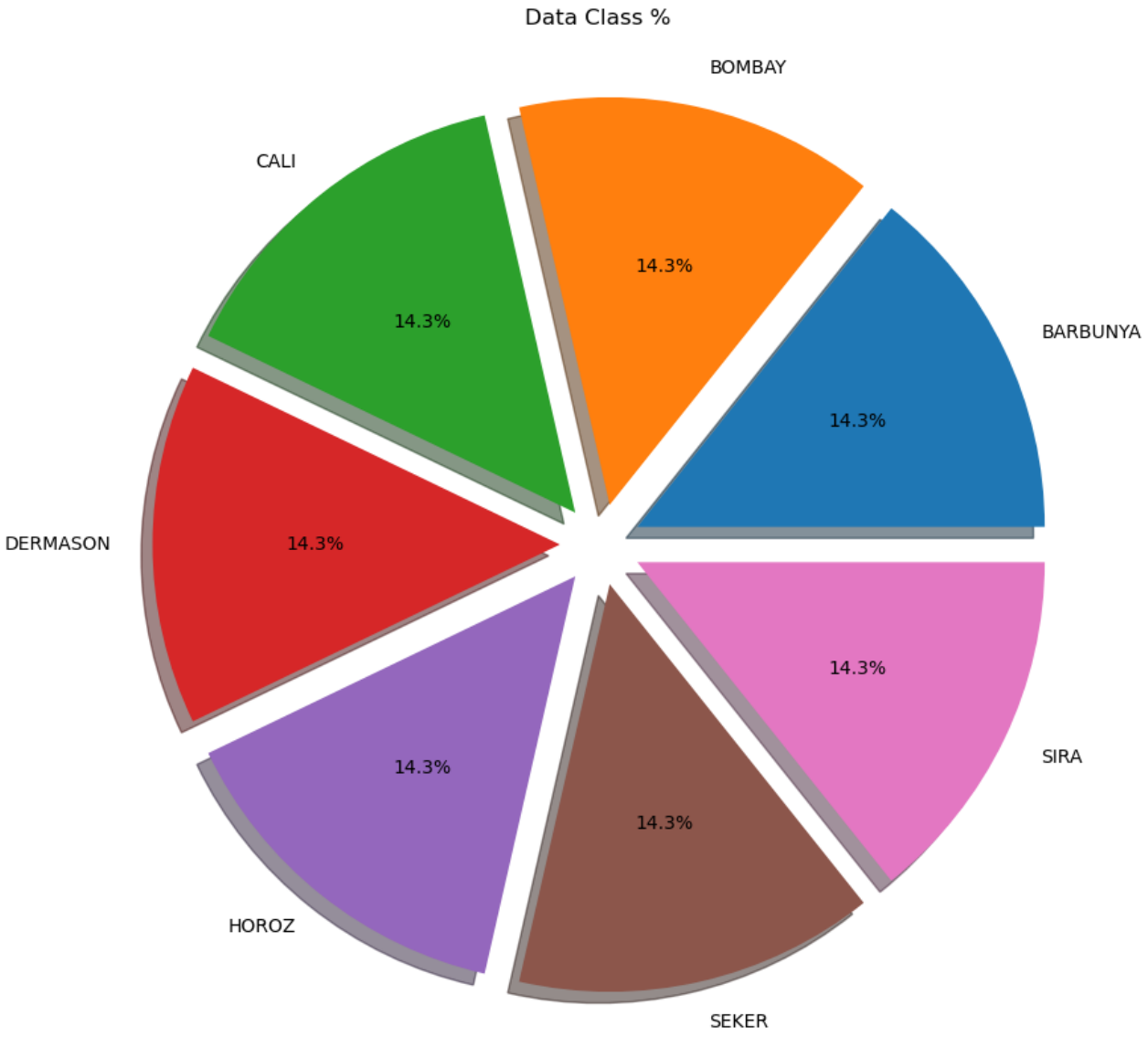
Defining Target Variable

In [32]:
```
X = df.drop('Class', axis=1)
y = df.Class
```

In [33]:
```
X_smote, y_smote = smote.fit_resample(X, y)
```

Lets Check New Data

In [34]:
```
labels, counts = np.unique(y_smote, return_counts=True)
myexplode=[0.10,0.10,0.1,0.1,0.1,0.1,0.1]
plt.figure(figsize=(10, 10))
plt.pie(counts, autopct='%1.1f%%', labels=labels,explode = myexplode,shadow=True)

plt.title('Data Class % ')
plt.show()
```



Data Cleaning

One-Hot-Encoding for df["Class"]

In [53]:
```
list(np.unique(y_smote))
```

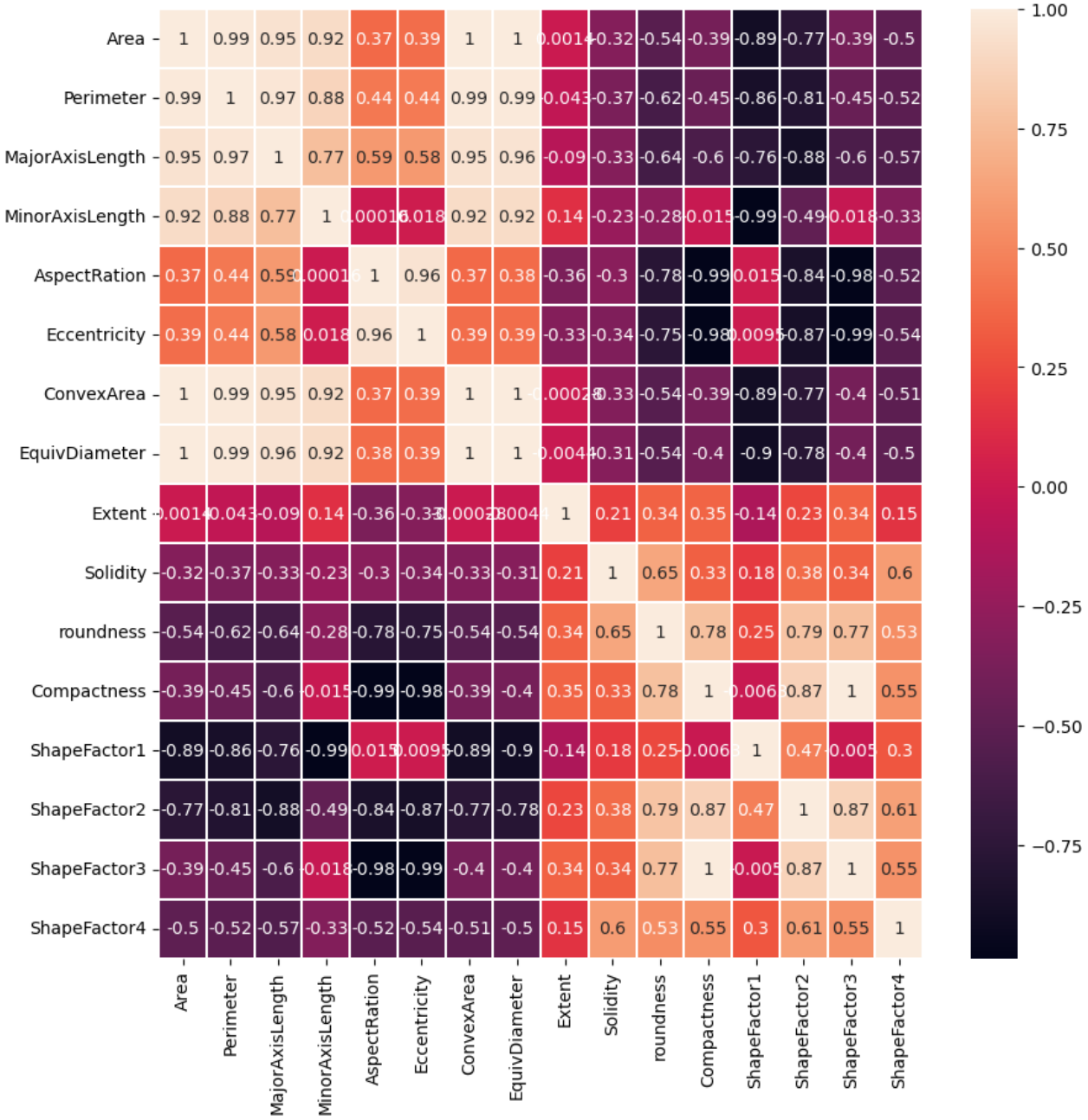
Out[53]:
```
['BARBUNYA', 'BOMBAY', 'CALI', 'DERMASON', 'HOROZ', 'SEKER', 'SIRA']
```

In [54]:
```
y_smote.replace(['BARBUNYA', 'BOMBAY', 'CALI', 'DERMASON', 'HOROZ', 'SEKER', 'SIRA'], [i for i in ran
```

Out[54]:
```
0        5
1        5
2        5
3        5
4        5
        ..
24817    6
24818    6
24819    6
24820    6
24821    6
Name: Class, Length: 24822, dtype: int64
```

Droping Highly Correlated Features/Columns

In [72]:
```
plt.figure(figsize=(10, 10))
sns.heatmap(data=df.corr(), annot=True,linewidth = 0.20,robust=True)
plt.show()
```



As per this Heat Map We can Say That Convex Area is Directly Propotional to Area,Perimeter,MajorAxisLength,MinorAxisLength and EquiDiameter Also and Shape 3 is Propotional to Eccentricity

Droping Unusual Features

In [42]:
```
X_smote.drop(['ConvexArea', 'EquivDiameter','ShapeFactor3'], axis=1, inplace=True)
```

In [56]:
```
X_smote.head()
```

Out[56]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	Extent	Solidity	roundness	Compactn
0	28395.00	610.29	208.18	173.89	1.20	0.57	0.76	0.99	0.96	0
1	28734.00	638.02	200.52	182.73	1.10	0.57	0.78	0.98	0.89	0
2	29380.00	624.11	212.83	175.93	1.21	0.57	0.78	0.99	0.95	0
3	30008.00	645.88	210.56	182.52	1.15	0.57	0.78	0.98	0.90	0
4	30140.00	620.13	201.85	190.28	1.06	0.57	0.77	0.99	0.98	0

Train Test And Split

In [57]:
```
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, random_state=42, shuffle=True,
```

Using Standard Scaler

In [58]:
```
from sklearn.preprocessing import StandardScaler # Importing Standered Scaler


ss = StandardScaler()
ss.fit_transform(X_train)
ss.transform(X_test)
```

Out[58]:
```
array([[-0.63129555, -0.40636467, -0.02562853, ...,  1.45382352,
        -0.86637853,  0.5658804 ],
       [-1.19720727, -1.13170831, -1.15469454, ...,  1.12673154,
         1.26306501,  0.46533576],
       [-0.6824093 , -0.68185232, -0.64308054, ...,  0.4836968 ,
         0.35221597,  0.69629351],
       ...,
       [ 0.22148337,  0.03778409, -0.05537979, ..., -0.46875489,
        -0.10192497,  0.7263045 ],
       [ 0.03395614,  0.06876152, -0.08046774, ..., -0.19480361,
        -0.21004302,  0.81040933],
       [-0.91330201, -0.92338955, -0.91270072, ...,  0.63991194,
         0.87336248,  0.4528637 ]])
```

Lets Check Which Model Will Be Better

In [46]:
```
from lazypredict.Supervised import LazyClassifier
# Creating the LazyClassifier object
clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
```

In [47]:
```
# Fitting the model on the training data
models, predictions = clf.fit(X_train, X_test, y_train, y_test)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 29/29 [05:1
0<00:00, 10.71s/it]
```

In [48]:
```
# Printing the performance metrics of the models
print(models)
```

```
                               Accuracy  Balanced Accuracy ROC AUC  F1 Score  \
Model
ExtraTreesClassifier               0.96               0.96    None      0.96
LGBMClassifier                     0.95               0.95    None      0.95
RandomForestClassifier             0.95               0.95    None      0.95
LabelSpreading                     0.95               0.95    None      0.95
LabelPropagation                   0.95               0.95    None      0.95
BaggingClassifier                  0.95               0.95    None      0.95
SVC                                0.95               0.95    None      0.95
KNeighborsClassifier               0.95               0.95    None      0.95
CalibratedClassifierCV             0.94               0.94    None      0.94
LinearSVC                          0.94               0.94    None      0.94
LogisticRegression                 0.94               0.94    None      0.94
SGDClassifier                      0.94               0.94    None      0.94
QuadraticDiscriminantAnalysis      0.94               0.94    None      0.94
NuSVC                              0.93               0.93    None      0.93
DecisionTreeClassifier             0.93               0.93    None      0.93
GaussianNB                         0.93               0.93    None      0.93
ExtraTreeClassifier                0.92               0.92    None      0.92
RidgeClassifier                    0.92               0.92    None      0.92
RidgeClassifierCV                  0.92               0.92    None      0.92
LinearDiscriminantAnalysis         0.92               0.92    None      0.92
Perceptron                         0.92               0.92    None      0.92
NearestCentroid                    0.91               0.91    None      0.91
PassiveAggressiveClassifier        0.88               0.88    None      0.89
AdaBoostClassifier                 0.73               0.73    None      0.69
BernoulliNB                        0.67               0.67    None      0.67
DummyClassifier                    0.14               0.14    None      0.03

                               Time Taken
Model
ExtraTreesClassifier                 1.80
LGBMClassifier                       3.00
RandomForestClassifier               5.40
LabelSpreading                     112.37
LabelPropagation                   108.07
BaggingClassifier                    3.21
SVC                                  4.66
KNeighborsClassifier                 0.85
CalibratedClassifierCV               6.84
LinearSVC                            1.29
LogisticRegression                   2.04
SGDClassifier                        0.33
QuadraticDiscriminantAnalysis        0.17
NuSVC                               52.99
DecisionTreeClassifier               0.52
GaussianNB                           0.14
ExtraTreeClassifier                  0.14
RidgeClassifier                      0.24
RidgeClassifierCV                    0.24
LinearDiscriminantAnalysis           0.40
Perceptron                           0.25
NearestCentroid                      0.14
PassiveAggressiveClassifier          0.32
AdaBoostClassifier                   4.49
BernoulliNB                          0.22
DummyClassifier                      0.12
```

Here You Can Find Diffrent Algorithm with their Accuracy and F1 Score and Time Taken Also Lets Try One of these aving Best Accuracy

Lets Try Decision Tree Classifier

In [59]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [60]:
```
# model

dtc = DecisionTreeClassifier(max_depth=8)

#fitting

dtc.fit(X_train,y_train)
```

Out[60]:
```
DecisionTreeClassifier(max_depth=8)
```

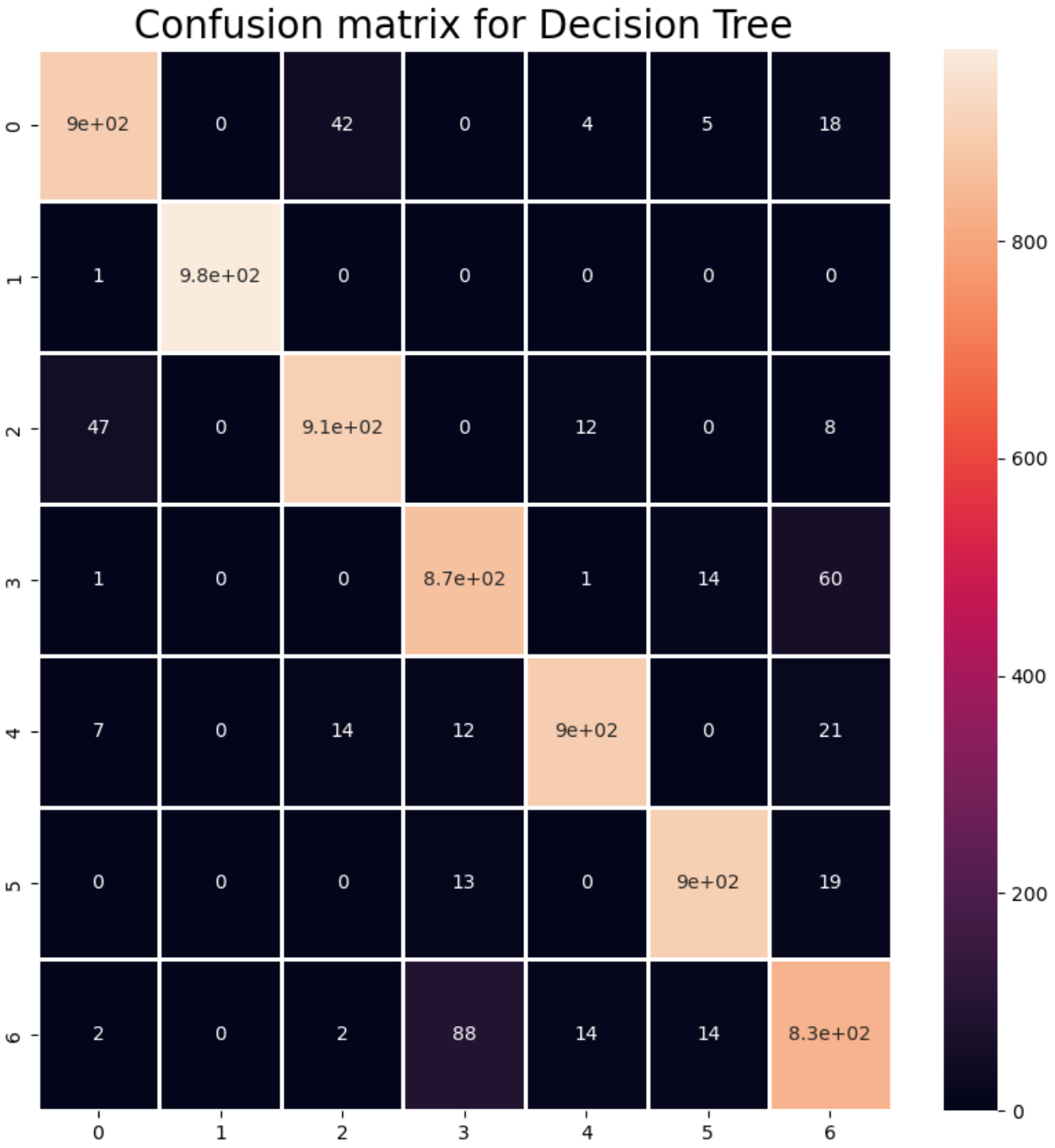
In [61]:
```
# predicting via Decision Tree Algorithm

y_pred=dtc.predict(X_test)

y_pred
```

Out[61]:
```
array(['HOROZ', 'DERMASON', 'SIRA', ..., 'BARBUNYA', 'BARBUNYA',
       'DERMASON'], dtype=object)
```

In [62]:
```
# compute accuracy on training set

dtc_train= dtc.score(X_train,y_train)

print("Training Data Accuracy by Decision Tree Algorithm is : " , round(dtc_train,2))

# compute accuracy on testing set

dtc_test= dtc.score(X_test,y_test)

print("Testing Data Accuracy by Decision Tree Algorithm is : " , round(dtc_test,2))
```

```
Training Data Accuracy by Decision Tree Algorithm is :  0.95
Testing Data Accuracy by Decision Tree Algorithm is :  0.94
```

In [71]:
```
# Lets find out our model performance

from sklearn.metrics import confusion_matrix

#Lets print the confusion metrix for this model

plt.rcParams['figure.figsize']=(10,10)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True,linewidths=1,linecolor='white',cbar=True)
plt.title("Confusion matrix for Decision Tree",fontsize=20)
plt.show()
```



In [64]:
```
X_smote.head()
```

Out[64]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	Extent	Solidity	roundness	Compactn
0	28395.00	610.29	208.18	173.89	1.20	0.57	0.76	0.99	0.96	0
1	28734.00	638.02	200.52	182.73	1.10	0.57	0.78	0.98	0.89	0
2	29380.00	624.11	212.83	175.93	1.21	0.57	0.78	0.99	0.95	0
3	30008.00	645.88	210.56	182.52	1.15	0.57	0.78	0.98	0.90	0
4	30140.00	620.13	201.85	190.28	1.06	0.57	0.77	0.99	0.98	0

In [66]:
```
# what is the process the model uses to generate its predictions, Lets Check?
new_check = dtc.predict((np.array([[28734,638.02,200.52,182.73,1.10,0.41,0.78,0.98,0.89,0.95,0.01,0.0
print("Provide Class : ",new_check)
```

```
Provide Class :  ['DERMASON']
```

In [67]:
```
# what is the process the model uses to generate its predictions, Lets Check?
new_check = dtc.predict((np.array([[14000,300,150,175.73,2.10,0.81,0.98,0.80,0.59,0.01,0.00,0.00,1.00
print("Provide Class : ",new_check)
```

```
Provide Class :  ['BARBUNYA']
```

In [68]:
```
# Lets print the classifiction report also
from sklearn.metrics import classification_report
cr =classification_report(y_test,y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

    BARBUNYA       0.94      0.93      0.93       968
      BOMBAY       1.00      1.00      1.00       977
        CALI       0.94      0.93      0.94       973
    DERMASON       0.88      0.92      0.90       944
       HOROZ       0.97      0.94      0.95       952
       SEKER       0.96      0.97      0.97       935
        SIRA       0.87      0.87      0.87       953

    accuracy                           0.94      6702
   macro avg       0.94      0.94      0.94      6702
weighted avg       0.94      0.94      0.94      6702
```

High precision indicates that the model is making less false positive predicions

While high recall signifies that the model is making lesser false negative predictions

We can conclude our model is working eficiently, recall and f1-score

Created by Bharat Bhushan Kulmani