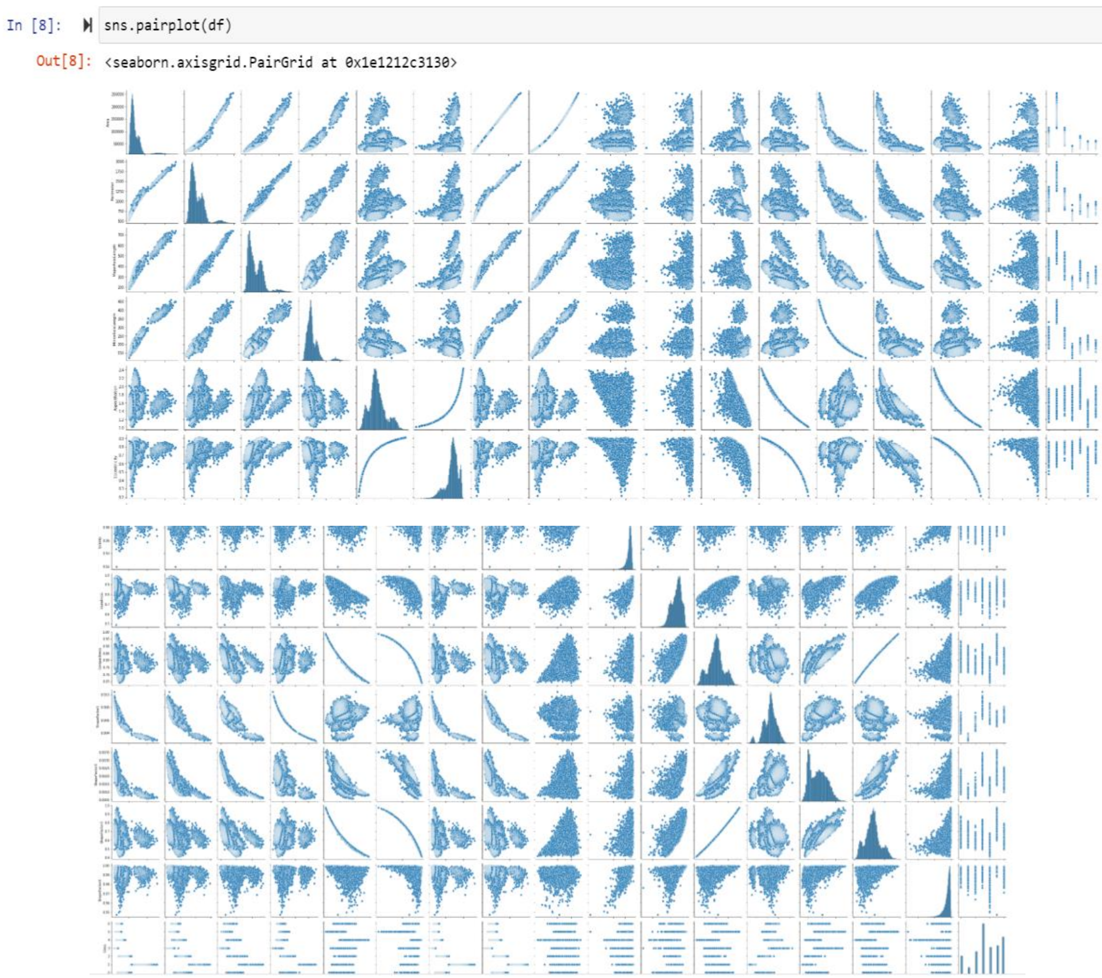
Exploratory Data Analysis (EDA):

After importing the data and we go to Exploratory Data analysis part which contains all the analytics:

1) Pair Plot



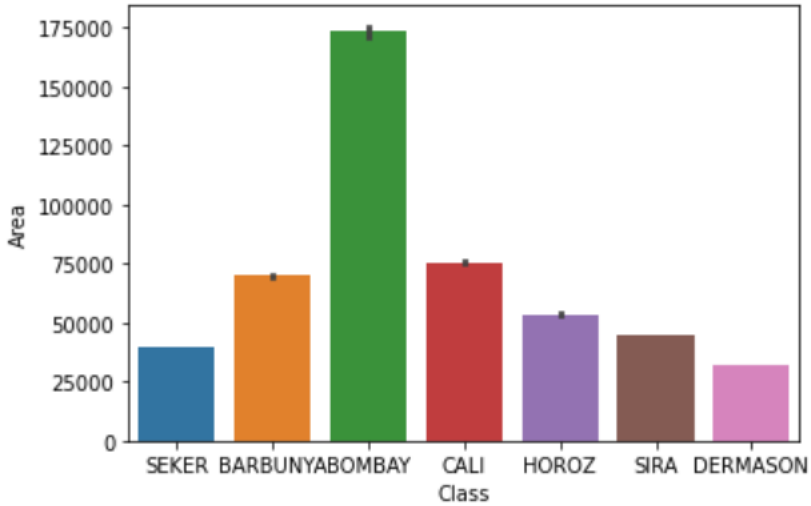
Code:- sns.pairplot(df)

Seaborn is a Python data visualization library based on matplotlib.

To plot multiple pairwise bivariate distributions in a dataset, we use the pairplot() function. This shows the relationship for (n, 2) combination of variable in a DataFrame as a matrix of plots and the diagonal plots are the univariate plots.

2) BarPlot between Class and Area

```
In [6]:  sns.barplot(x='Class', y='Area', data=df)
Out[6]: <AxesSubplot:xlabel='Class', ylabel='Area'>
```

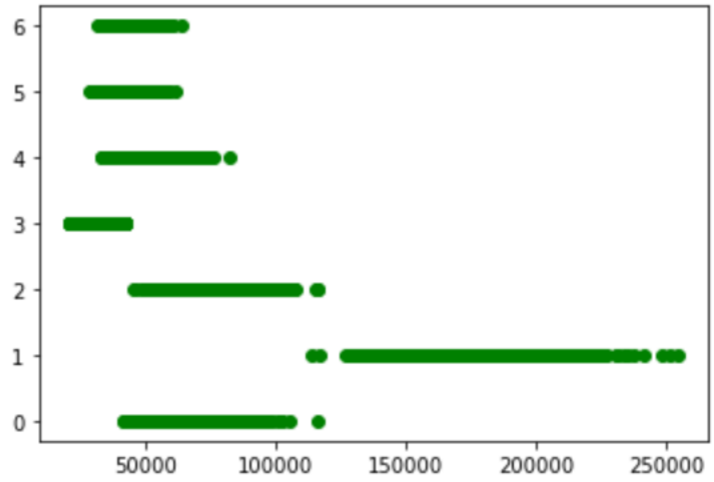


Code:- sns.barplot(x='Class', y='Area', data=df)

- A barplot is basically used to aggregate the categorical data according to some methods and by default it's the mean. It can also be understood as a visualization of the group by action.
- The above plot helps us to know whether our dataset is balanced or not.
- The X axis of the barplot represents the 7 Classes(dependent variable) given in the dataset and the Y axis of the bar plot represents the Area.
- We observe that the **dataset is imbalanced**. (unequal distribution of samples among classes)
- Most beans are classified as BOMBAY according to their area and least are classified as Dermason.

3) Scatterplot between Class and Area

```
In [9]:  plt.scatter(df['Area'],df['Class'],c='green')
Out[9]: <matplotlib.collections.PathCollection at 0x1e1325f4880>
```

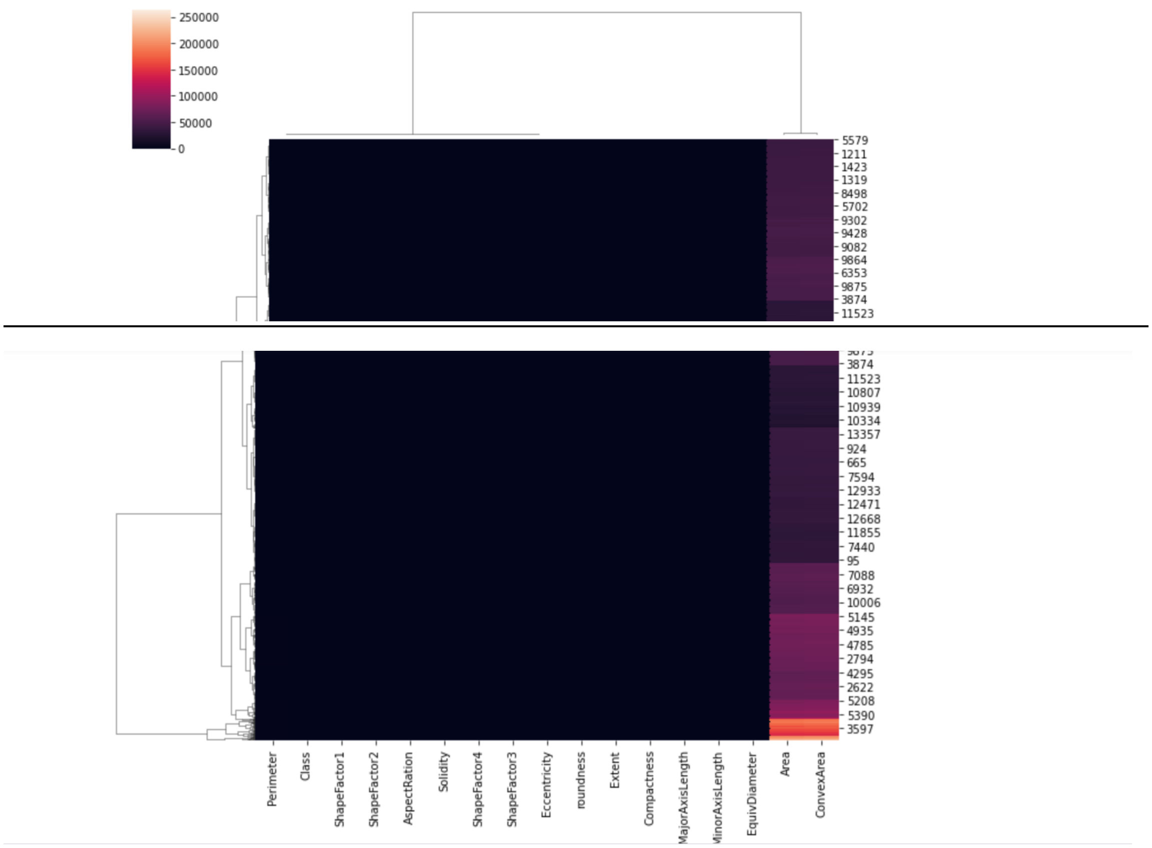


Code:- plt.scatter(df[‘Area’],df[‘Class’],c=’green’)

- **Scatterplot** can be used with several semantic groupings which can help to understand well in a graph. They can plot two-dimensional graphics that can be enhanced by mapping up to three additional variables while using the semantics of hue, size, and style parameters.
- The X axis of the scatterplot represents the different Classes(dependent variable) given in the dataset and the Y axis of the scatterplot represents the Area.

4) ClusterMAP

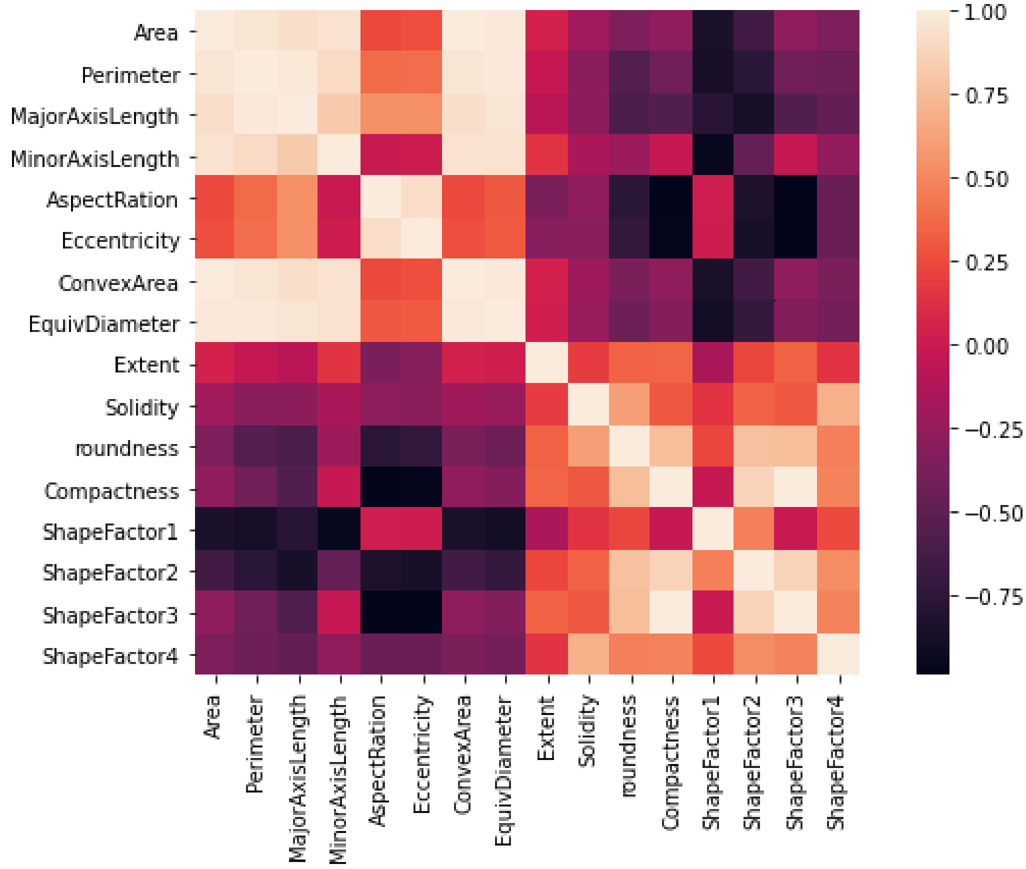
```
In [11]:  sns.clustermap(df)

C:\Users\Admin\anaconda3\lib\site-packages\seaborn\matrix.py:659: UserWarning: Clustering large matrix with scipy. Installin
g `fastcluster` may give better performance.
  warnings.warn(msg)

Out[11]: <seaborn.matrix.ClusterGrid at 0x1e129f8cc10>
```



Code:- sns.clustermap(df)

- Heat map- it is a graphical representation of data where values are represented using colors. Variation in the intensity of color depicts how data is clustered or varies over space.
- The *clustermap()* function of *seaborn* plots a hierarchically-clustered heat map of the given matrix dataset. It returns a clustered grid index.

5) HeatMap with all features



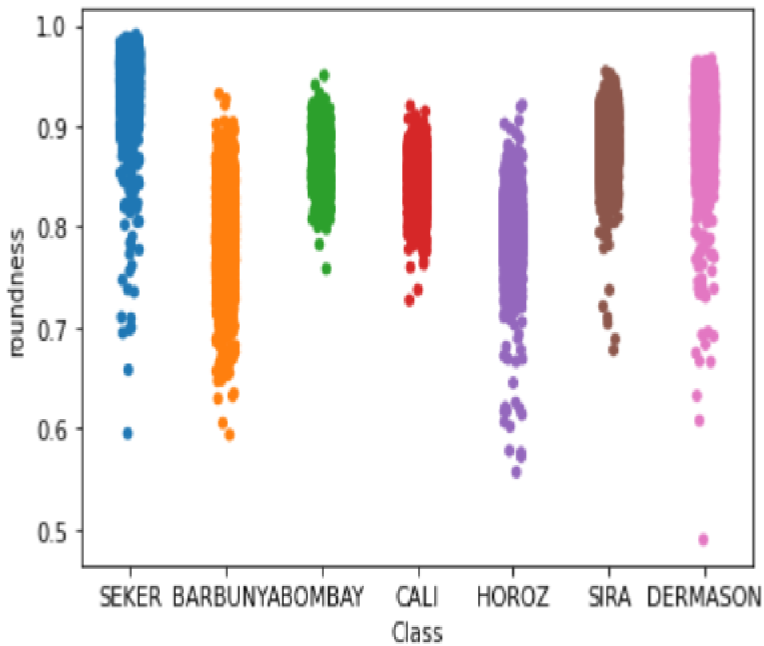
Code:- allfeat=df.corr()

```
allfeat=df.corr()
sns.heatmap(allfeat, square=True)
```

- **Heatmap** is defined as a graphical representation of data using colors to visualize the value of the matrix
- This plot shows the correlation between the variables.
- The scale on the right will show the correlation coefficient.
- We can conclude that Compactness and Eccentricity columns are closely correlated with correlation coefficient of around -0.75

6) StripPlot between Class and Solidity

```
In [7]:  sns.stripplot(x="Class", y="roundness", data=df)

Out[7]: <AxesSubplot:xlabel='Class', ylabel='roundness'>
```

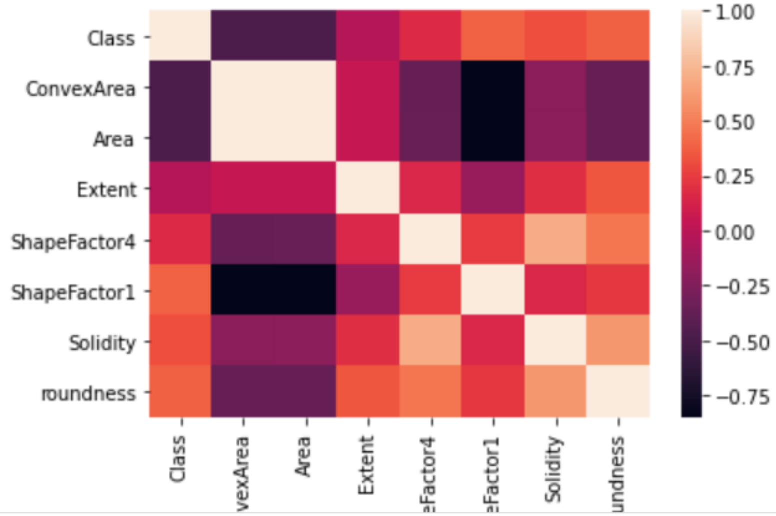


Code:- sns.stripplot(x="Class", y="roundness", data=df)

- A strip plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where you want to show all observations along with some representation of the underlying distribution.
- We draw a scatterplot where one variable is categorical.
- We see that SEEKER class has bean attributes that are rounder than the rest.

7) HeatMap with fine tuned data

Out[15]: <AxesSubplot:>



Code:- new_corr=new_df.corr()

sns.heatmap(new_corr)

```
new_df=df.filter(['Class','ConvexArea','Area',
                  'Extent',
                  'ShapeFactor4',
                  'ShapeFactor1',
                  'Solidity',
                  'roundness'])
```

- **Heatmap** is defined as a graphical representation of data using colors to visualize the value of the matrix
- This plot shows the correlation between the variables.
- We can see that area and convexarea have high co-relation with class.
- The purpose of fine-tuning is to choose the correct parametres for modelling.

Data Preparation :

- df=pd.read_csv('Dry_Bean_Dataset.csv')

  Check for missing values :

- df.isnull()

  Fill in the missing values :

- df.fillna(0)

Out[5]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Com
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.141097	0.763923	0.988856	0.958027	
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.272751	0.783968	0.984986	0.887034	
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.410904	0.778113	0.989559	0.947849	
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.467062	0.782681	0.976696	0.903936	
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.896503	0.773098	0.990893	0.984877	
...	...	...	...	...	...	...	...	...	...	...	...	
13606	42097	759.696	288.721612	185.944705	1.552728	0.765002	42508	231.515799	0.714574	0.990331	0.916603	
13607	42101	757.499	281.576392	190.713136	1.476439	0.735702	42494	231.526798	0.799943	0.990752	0.922015	
13608	42139	759.321	281.539928	191.187979	1.472582	0.734065	42569	231.631261	0.729932	0.989899	0.918424	
13609	42147	763.779	283.382636	190.275731	1.489326	0.741055	42667	231.653248	0.705389	0.987813	0.907906	
13610	42159	772.237	295.142741	182.204716	1.619841	0.786693	42600	231.686223	0.788962	0.989648	0.888380	

13611 rows × 17 columns

One Hot Encoding/Label encoding :

- Since the dry beans dataset has the class attribute in the form of strings , we label encode it to make it suitable for our machine learning model.

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df['Class']= label_encoder.fit_transform(df['Class'])
```

Preparing Machine Learning Model:

Train Test Split

X=df.iloc[:,:-1]

y=df.iloc[:,-1]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

1- Decision Tree model

Training the model

from sklearn.tree import DecisionTreeClassifier

dTree = DecisionTreeClassifier(criterion="entropy",max_depth=6)

- Max_depth – 6

dTree.fit(X_train,y_train)

Testing :

pred=dTree.predict(X_test)

Accuracy :

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("ACCURACY--",accuracy_score(y_test,pred))

```
ACCURACY-- 0.9005876591576886
```

print(classification_report(y_test,pred))

```
In [73]:  print(classification_report(y_test,pred))

              precision    recall  f1-score   support

           0       0.90      0.83      0.86       395
           1       1.00      1.00      1.00       161
           2       0.88      0.92      0.90       479
           3       0.89      0.91      0.90      1043
           4       0.95      0.95      0.95       588
           5       0.91      0.93      0.92       619
           6       0.86      0.83      0.85       799

    accuracy                           0.90      4084
   macro avg       0.91      0.91      0.91      4084
weighted avg       0.90      0.90      0.90      4084
```

2- Support Vector Machine

Training the Model

from sklearn.svm import SVC

model = SVC(kernel='linear')

model.fit(X_train,y_train)

Testing :

pred2=model.predict(X_test)

Accuracy :

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(accuracy_score(y_test,pred2))

```
0.9165034280117532
```

print(classification_report(y_test,pred2))

```
              precision    recall  f1-score   support

           0       0.93      0.88      0.90       395
           1       1.00      1.00      1.00       161
           2       0.91      0.94      0.92       479
           3       0.91      0.91      0.91      1043
           4       0.96      0.95      0.96       588
           5       0.94      0.93      0.93       619
           6       0.86      0.88      0.87       799

    accuracy                           0.92      4084
   macro avg       0.93      0.93      0.93      4084
weighted avg       0.92      0.92      0.92      4084
```

3-K-Nearest Neighbors

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

from sklearn.metrics import f1_score

```
agg_acc = np.zeros((11))


for n in range(1,12):
    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    agg_acc[n-1] = metrics.accuracy_score(y_test, yhat)


print(agg_acc)
knnbest = agg_acc.argmax()+1
print("Best Accuracy is with K ",knnbest)
print("Best Accuracy is ",agg_acc.max())
plt.plot(range(1,12),agg_acc)
plt.xlabel('K')
plt.ylabel('CV Accuracy')
plt.show()
```

- The above code determines at which K value , we have the best fit and accuracy.

Testing : bestknn = KNeighborsClassifier(n_neighbors = 1).fit(X_train,y_train)

y_pred=bestknn.predict(X_test)

Accuracy :

- print("ACCURACY--",accuracy_score(y_test,y_pred))
- print(classification_report(y_test,y_pred))
- print("F1-SCORE--",f1_score(y_test, y_pred, average='weighted'))

```
ACCURACY-- 0.7627326150832517
              precision    recall  f1-score   support

           0       0.62      0.48      0.54       395
           1       1.00      1.00      1.00       161
           2       0.66      0.77      0.71       479
           3       0.85      0.83      0.84      1043
           4       0.79      0.79      0.79       588
           5       0.70      0.70      0.70       619
           6       0.76      0.79      0.77       799

    accuracy                           0.76      4084
   macro avg       0.77      0.77      0.76      4084
weighted avg       0.76      0.76      0.76      4084

F1-SCORE-- 0.7608550579020791
```

4- Naïve Bayes

```
# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
# making predictions on the testing set
y_pred3 = gnb.predict(X_test)
```

Accuracy :

```
print("ACCURACY--",accuracy_score(y_test,y_pred3))
print(classification_report(y_test,y_pred3))
print("F1-SCORE--",f1_score( y_test, y_pred3, average='weighted'))
```

```
ACCURACY-- 0.7627326150832517
              precision    recall  f1-score   support

           0       0.62      0.48      0.54       395
           1       1.00      1.00      1.00       161
           2       0.66      0.77      0.71       479
           3       0.85      0.83      0.84      1043
           4       0.79      0.79      0.79       588
           5       0.70      0.70      0.70       619
           6       0.76      0.79      0.77       799

    accuracy                           0.76      4084
   macro avg       0.77      0.77      0.76      4084
weighted avg       0.76      0.76      0.76      4084

F1-SCORE-- 0.7608550579020791
```

Machine Learning Model Chart

Serial Number	ML Algorithm Used	Accuracy Score (approx. 4 decimal places)
1	Support Vector Machine	0.9165
2	Decision Tree	0.9005
3	KNN	0.7601
4	Naïve Bayes	0.7608