AMRITA SCHOOL OF ENGINEERING, COIMBATORE

BTech in CSE – IV Year – VII Semester

15CSE481 – Machine Learning and Data Mining Lab

Case Study - Group 17

DRY BEANS CLASSIFICATION

Team Members

Name	Registration No.
Parripati Divyasri	CB.EN.U4CSE18041
Harish K	CB.EN.U4CSE18501

TABLE OF CONTENTS

S.No	Topic	Page No.
1	Introduction	3
2	Objective	3
3	Problem Statement	4
4	Architecture Diagram	4
5	Related work and novelty	5
6	Data collection and preparation	6
7	Data modeling and Inference	15
8	Conclusion and future enhancements	22
9	References	22

1. INTRODUCTION

Dry bean (Phaseolus vulgaris) is the most important and the most produced pulse over the world. The seed is an important input in agricultural expense and dry beans take an important part in food technology. Most of the activities about dry bean seed technology is realized especially by private companies. Seed quality is definitely influential in crop production It is the key to bean cultivation in terms of yield and disease. Hence, seed classification is essential for both marketing and production. Moreover, the identification of bean varieties helps farmers to use seeds with basic standards for planting and marketing. In this study, we propose a novel ensemble-based machine learning model for classifying dry beans into one of the seven distinct types. Data is obtained from UCI Machine Learning Repository and used for training the constituent models for ensemble learning.

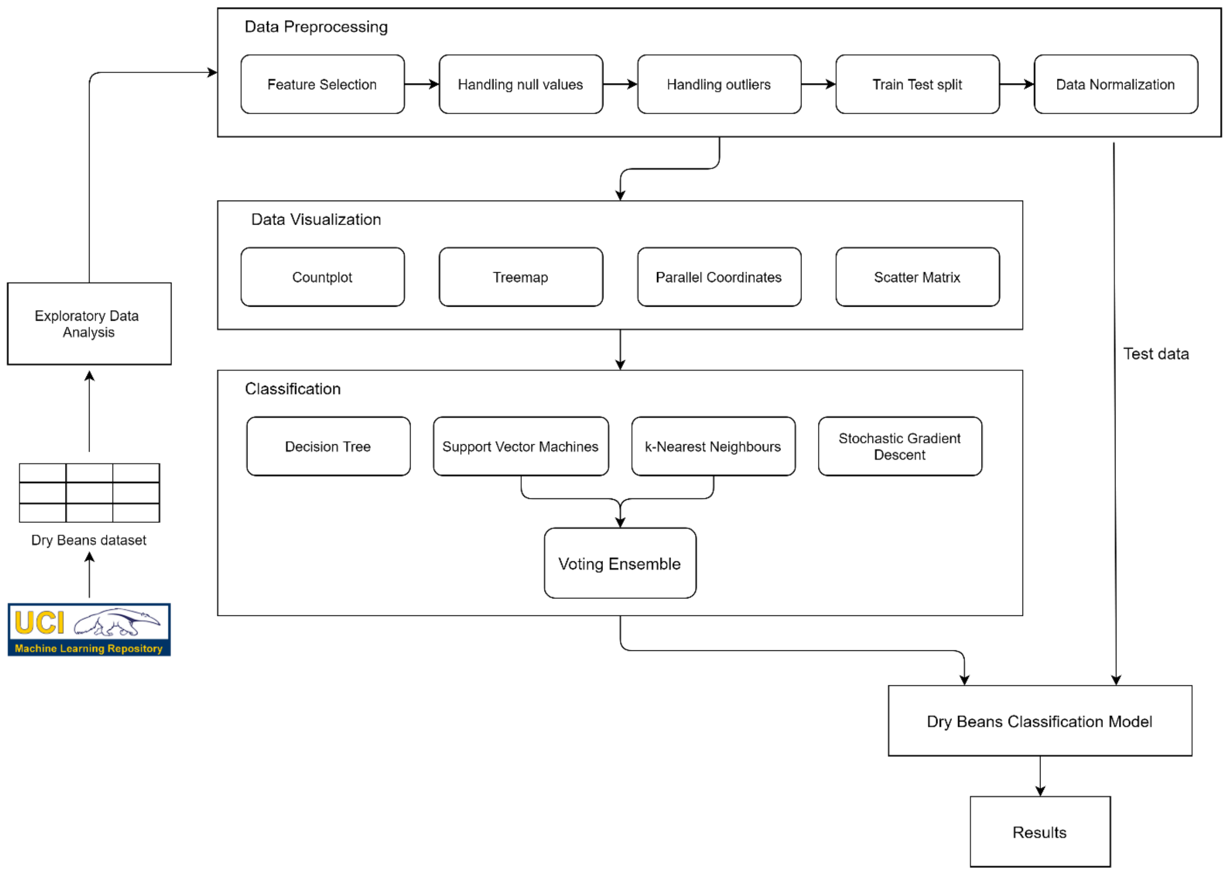
2. OBJECTIVE

To build and analyze multi-class classification models namely Support Vector Machine (SVM), k-Nearest Neighbors (kNN), Decision Tree (DT) and Stochastic Gradient Descent (SGD) and implement ensemble learning in the models to obtain improved performance.

3. PROBLEM STATEMENT

Dry Beans Classification is a multi-class classification problem which aims to build and analyze various classification models that distinguish seven different varieties of dry beans with several features in order to obtain uniform and accurate seed classification.

4. ARCHITECTURE DIAGRAM



5. RELATED WORKS AND NOVELTY

A computer vision system was developed by Koklu et al. (2020) to distinguish seven different registered varieties of dry beans with similar features in order to obtain uniform seed classification. For the classification model, images of 13,611 grains of 7 different registered dry beans were taken. Bean images obtained by computer vision system were subjected to segmentation and feature extraction stages, and a total of 16 features; 12 dimension and 4 shape forms, were obtained from the grains. Multilayer perceptron (MLP), Support Vector Machine (SVM), k-Nearest Neighbors (kNN), Decision Tree (DT) classification models were created with 10-fold cross validation and performance metrics were compared.

Kılıç et al. (2007) proposed a classification system for beans using computer vision system and artificial neural networks for color quantification of the samples. Testing of the ANN was performed with 371 samples. An overall correct classification rate of 90.6% was obtained.

From the literature survey, we can see that the dry beans samples were trained on individual models and the accuracies of the models obtained are in the range 90-93%. In addition, more emphasis was given to feature extraction from images than data processing.

In our case study, as we have numeric data in hand, we concentrate more on preparing the data for better training of the models. Moreover, ensemble technique based on voting is implemented for obtained better classification model.

6. DATA COLLECTION AND PREPARATION

The dataset used here is "Dry Bean Dataset" which is publicly available in UCI Machine Learning Repository.

Dataset Link

https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset

Dataset Description

This dataset has 13,611 instances of multivariate data containing both integer and real attributes. The dataset has total 17 attributes which are as follows:

1) Area: The area of bean zone and the number of pixels within its boundaries.

2)Perimeter: The length of bean's border.

3) Major Axis Length: It is the length between ends of the maximum length line that can be drawn from a bean

4) Minor Axis Length: It is the maximum length line that can be drawn from the bean while standing perpendicular to main axis.

5) Aspect Ratio: It defines a relationship between Major Axis Length and Minor Axis Length.

6) Eccentricity: Eccentricity of ellipse having the same moments as the region.

7) Convex Area: It is the number of pixels in the smallest convex polygon that can have the area of a bean seed.

8) Equivalent Diameter: It is the diameter of a circle having the same area as a bean seed area.

9) Extent: It is the ratio of the pixels in the bounding box to the pixels in bean area.

10) Solidity: It is also referred as Convexity. It is the ratio of the pixels in the convex shell to the pixels in beans.

11) Roundness: It is calculated with formula (4*π*Area)/(Perimeter^2)

12) Compactness: It measures the roundness of an object (Equivalent diameter/ Major axis length)

13) Shape Factor 1

14) Shape Factor 2

15) Shape Factor 3

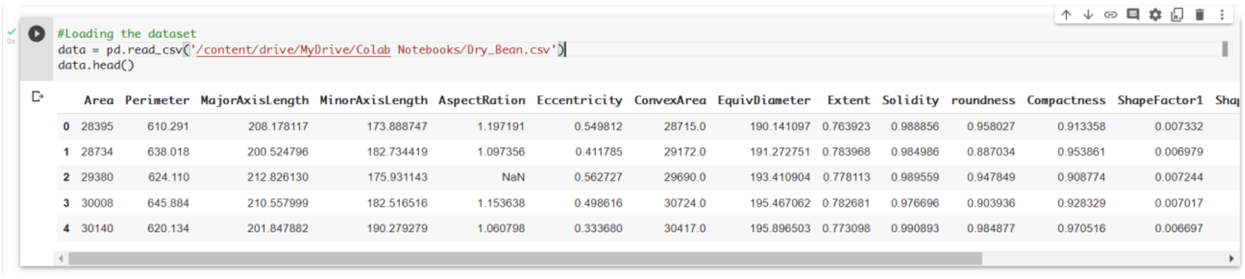
16) Shape Factor 4

17) Class: Output variable which contains the class of the bean. (Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira).

Summary of the dataset

Dataset Characteristics	Multivariate
Attribute Characteristics	Integer, real
Number Of Instances	13611
Number Of Attributes	17
Missing values	Found
Associated Tasks	Classification

6.1. Loading the Dataset to Google Colab

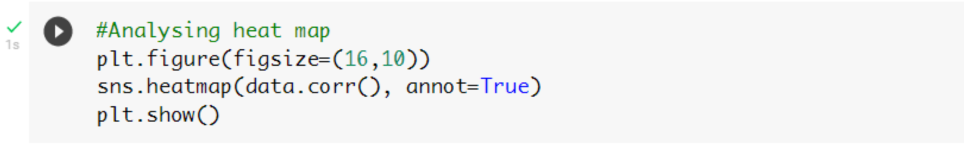


```
#Loading the dataset
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Dry_Bean.csv')
data.head()
```

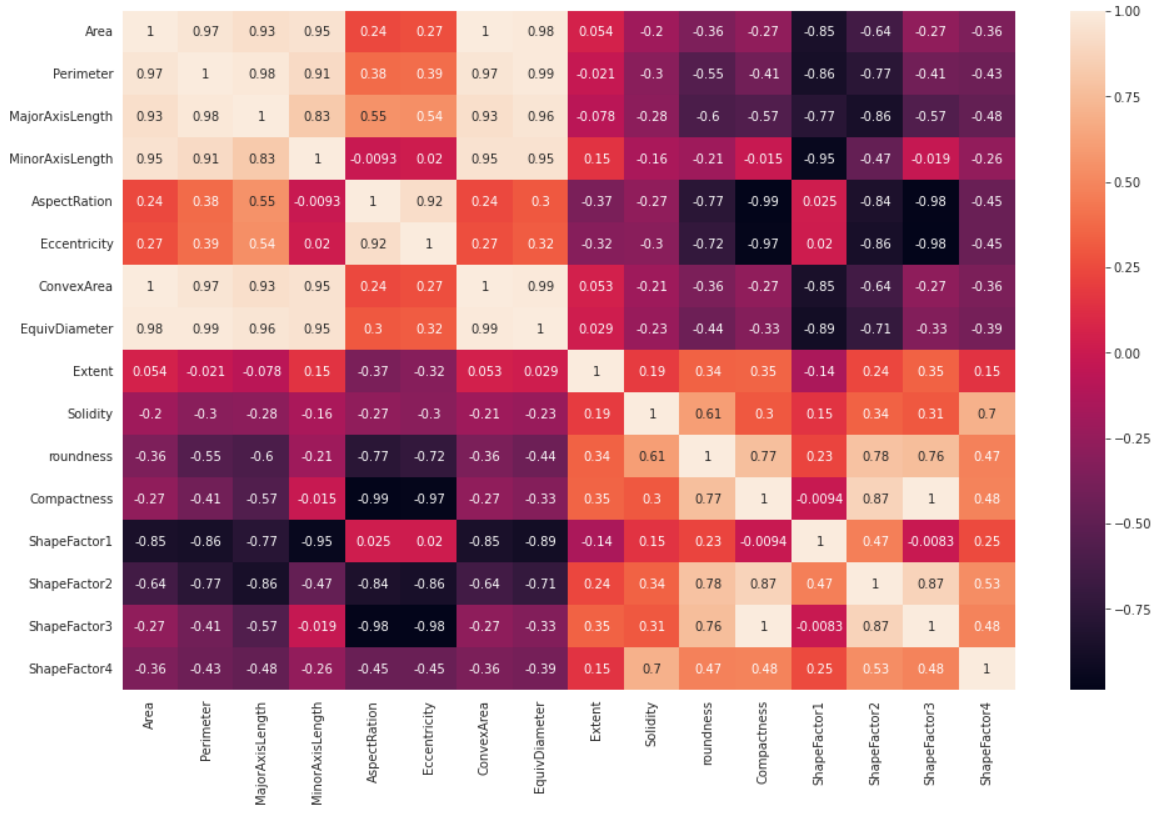
```
    Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRation  Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  roundness  Compactness  ShapeFactor1  Shap
0  28395    610.291       208.178117       173.888747      1.197191      0.549812     28715.0     190.141097  0.763923  0.988856   0.958027     0.913358      0.007332
1  28734    638.018       200.524796       182.734419      1.097356      0.411785     29172.0     191.272751  0.783968  0.984986   0.887034     0.953861      0.006979
2  29380    624.110       212.826130       175.931143           NaN      0.562727     29690.0     193.410904  0.778113  0.989559   0.947849     0.908774      0.007244
3  30008    645.884       210.557999       182.516516      1.153638      0.498616     30724.0     195.467062  0.782681  0.976696   0.903936     0.928329      0.007017
4  30140    620.134       201.847882       190.279279      1.060798      0.333680     30417.0     195.896503  0.773098  0.990893   0.984877     0.970516      0.006697
```

6.2. Feature Selection/Extraction

Correlation heatmap



```
#Analysing heat map
plt.figure(figsize=(16,10))
sns.heatmap(data.corr(), annot=True)
plt.show()
```



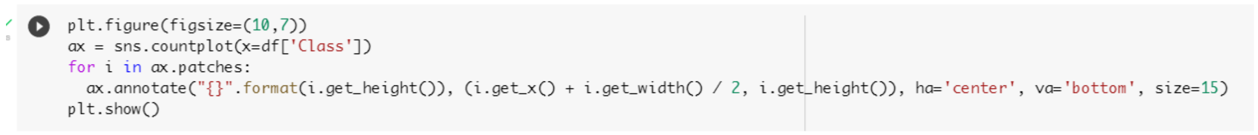
	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFactor3	ShapeFactor4
Area	1	0.97	0.93	0.95	0.24	0.27	1	0.98	0.054	-0.2	-0.36	-0.27	-0.85	-0.64	-0.27	-0.36
Perimeter	0.97	1	0.98	0.91	0.38	0.39	0.97	0.99	-0.021	-0.3	-0.55	-0.41	-0.86	-0.77	-0.41	-0.43
MajorAxisLength	0.93	0.98	1	0.83	0.55	0.54	0.93	0.96	-0.078	-0.28	-0.6	-0.57	-0.77	-0.86	-0.57	-0.48
MinorAxisLength	0.95	0.91	0.83	1	-0.0093	0.02	0.95	0.95	0.15	-0.16	-0.21	-0.015	-0.95	-0.47	-0.019	-0.26
AspectRation	0.24	0.38	0.55	-0.0093	1	0.92	0.24	0.3	-0.37	-0.27	-0.77	-0.99	0.025	-0.84	-0.98	-0.45
Eccentricity	0.27	0.39	0.54	0.02	0.92	1	0.27	0.32	-0.32	-0.3	-0.72	-0.97	0.02	-0.86	-0.98	-0.45
ConvexArea	1	0.97	0.93	0.95	0.24	0.27	1	0.99	0.053	-0.21	-0.36	-0.27	-0.85	-0.64	-0.27	-0.36
EquivDiameter	0.98	0.99	0.96	0.95	0.3	0.32	0.99	1	0.029	-0.23	-0.44	-0.33	-0.89	-0.71	-0.33	-0.39
Extent	0.054	-0.021	-0.078	0.15	-0.37	-0.32	0.053	0.029	1	0.19	0.34	0.35	-0.14	0.24	0.35	0.15
Solidity	-0.2	-0.3	-0.28	-0.16	-0.27	-0.3	-0.21	-0.23	0.19	1	0.61	0.3	0.15	0.34	0.31	0.7
roundness	-0.36	-0.55	-0.6	-0.21	-0.77	-0.72	-0.36	-0.44	0.34	0.61	1	0.77	0.23	0.78	0.76	0.47
Compactness	-0.27	-0.41	-0.57	-0.015	-0.99	-0.97	-0.27	-0.33	0.35	0.3	0.77	1	-0.0094	0.87	1	0.48
ShapeFactor1	-0.85	-0.86	-0.77	-0.95	0.025	0.02	-0.85	-0.89	-0.14	0.15	0.23	-0.0094	1	0.47	-0.0083	0.25
ShapeFactor2	-0.64	-0.77	-0.86	-0.47	-0.84	-0.86	-0.64	-0.71	0.24	0.34	0.78	0.87	0.47	1	0.87	0.53
ShapeFactor3	-0.27	-0.41	-0.57	-0.019	-0.98	-0.98	-0.27	-0.33	0.35	0.31	0.76	1	-0.0083	0.87	1	0.48
ShapeFactor4	-0.36	-0.43	-0.48	-0.26	-0.45	-0.45	-0.36	-0.39	0.15	0.7	0.47	0.48	0.25	0.53	0.48	1

Multicollinearity (high inter-correlation between variables) exists between the features Area, ConvexArea and EquivDiameter

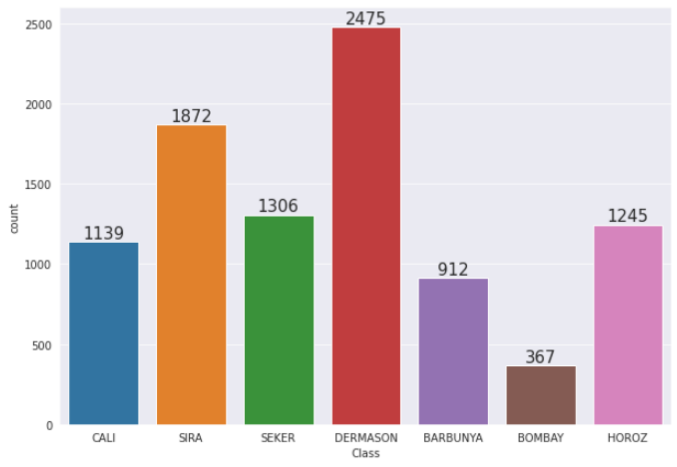
So, we drop two columns(ConvexArea, EquivDiameter)

6.3. Data Visualization and Hypothesis

6.3.1. Count plot



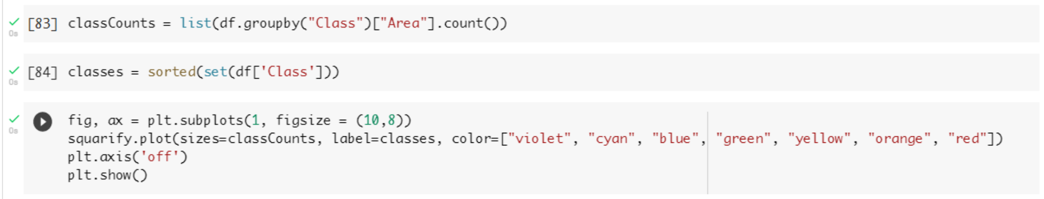
```
plt.figure(figsize=(10,7))
ax = sns.countplot(x=df['Class'])
for i in ax.patches:
  ax.annotate("{}".format(i.get_height()), (i.get_x() + i.get_width() / 2, i.get_height()), ha='center', va='bottom', size=15)
plt.show()
```



Inference

Here we can see that Dermason class has the most number of instances and Bombay class has the least number of instances

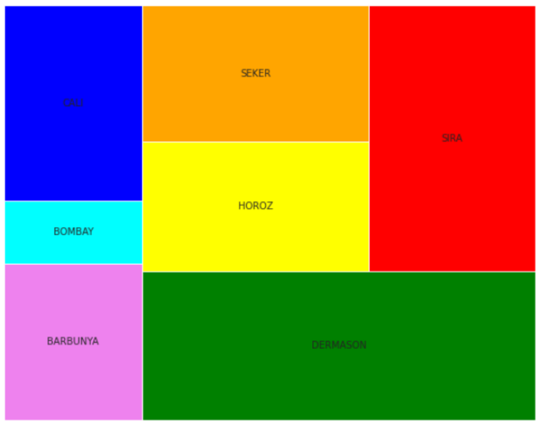
6.3.2Tree Map



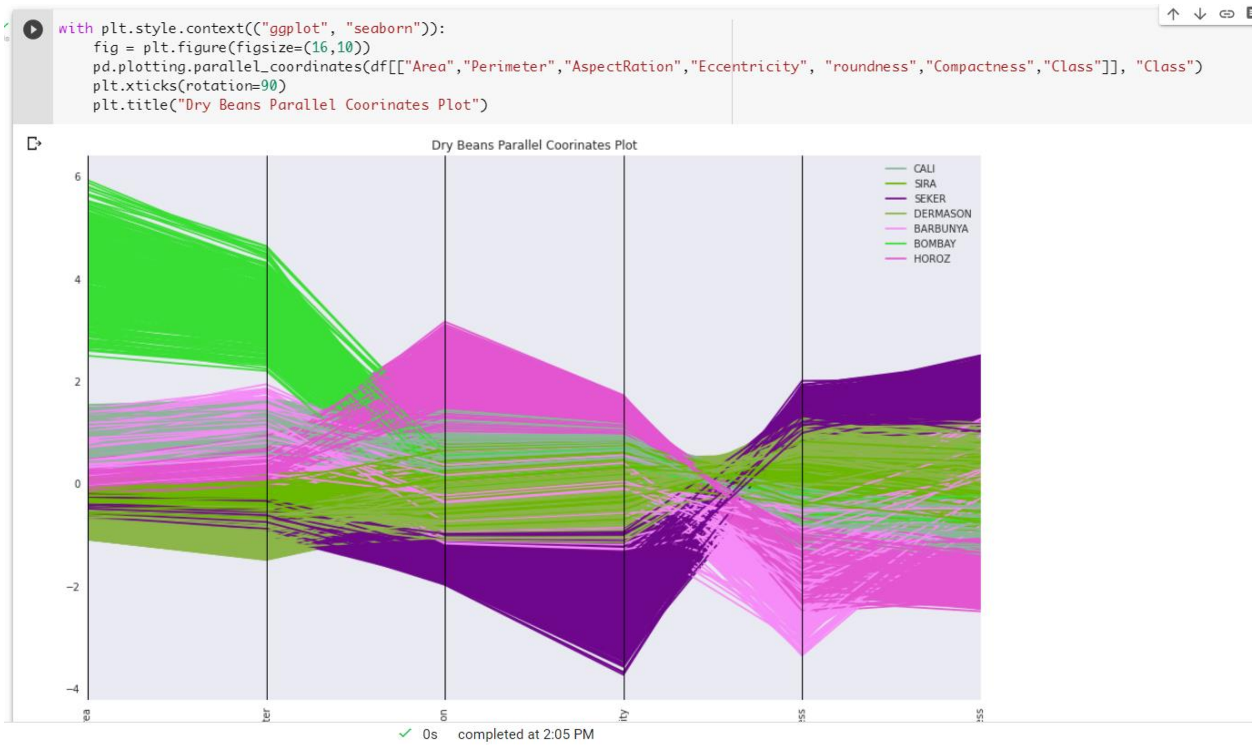
```
[83] classCounts = list(df.groupby("Class")["Area"].count())
```

```
[84] classes = sorted(set(df['Class']))
```

```
fig, ax = plt.subplots(1, figsize = (10,8))
squarify.plot(sizes=classCounts, label=classes, color=["violet", "cyan", "blue", "green", "yellow", "orange", "red"])
plt.axis('off')
plt.show()
```



6.3.3. Parallel Coordinates

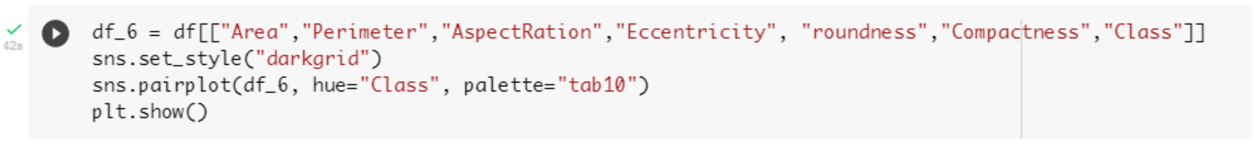


```
with plt.style.context(("ggplot", "seaborn")):
    fig = plt.figure(figsize=(16,10))
    pd.plotting.parallel_coordinates(df[["Area","Perimeter","AspectRation","Eccentricity", "roundness","Compactness","Class"]], "Class")
    plt.xticks(rotation=90)
    plt.title("Dry Beans Parallel Coorinates Plot")
```

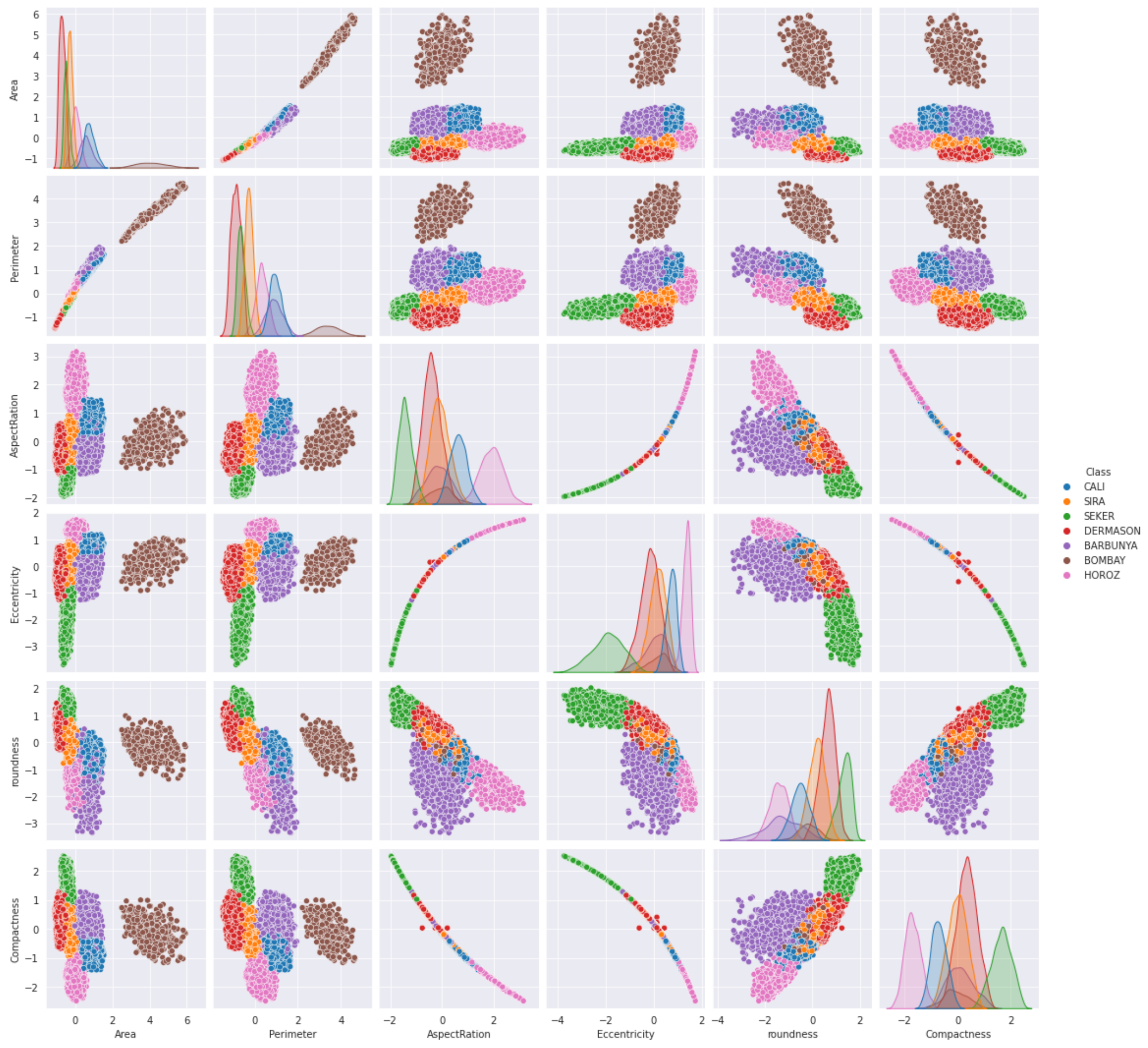
Inference

- The beans belonging to Bombay class has higher area and perimeter as compared to other classes
- The beans belonging to Horoz class has the highest AspectRation
- Seker class has the highest eccentricity
- Barbunya class has the lowest roundness
- Seker class has the highest compactness

6.3.4. Scatter Matrix



```
df_6 = df[["Area","Perimeter","AspectRation","Eccentricity", "roundness","Compactness","Class"]]
sns.set_style("darkgrid")
sns.pairplot(df_6, hue="Class", palette="tab10")
plt.show()
```



Inference

From the above scatter matrix, we can infer that:

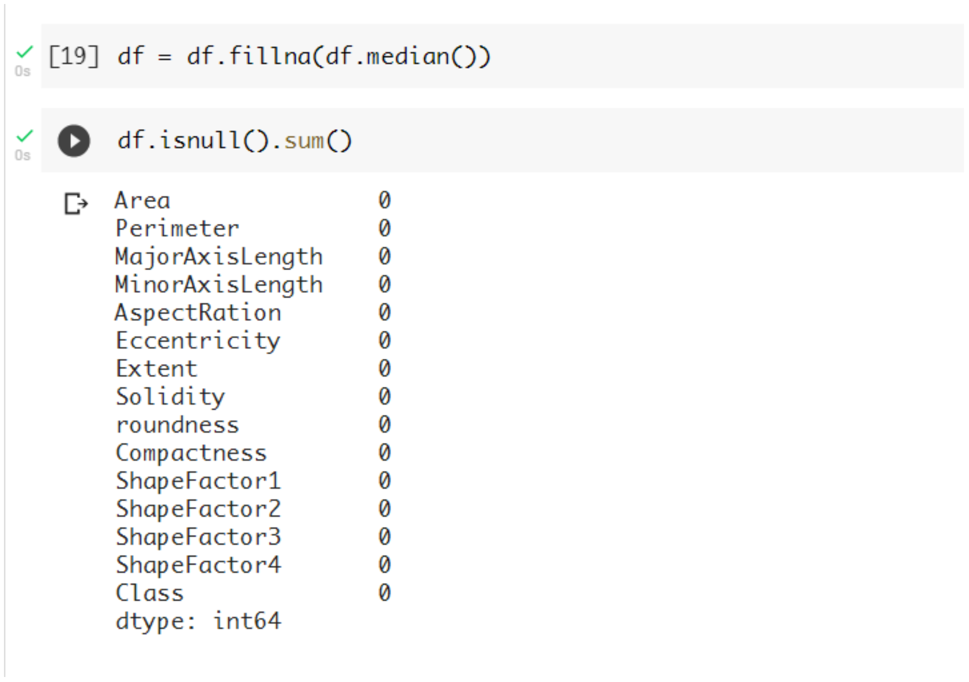
- BOMBAY class is linearly separable from the other classes
- There exists a positive correlation between AspectRation and Eccentricity whereas we see a negative correlation between Eccentricity and Compactness

The features - Area and Perimeter - show a positive correlation

6.4. Data Cleaning and preprocessing

6.4.1 Handling Null values

From the distribution analysis, we can see that the data is skewed for most of the features. Thus, we fill the null values with median.



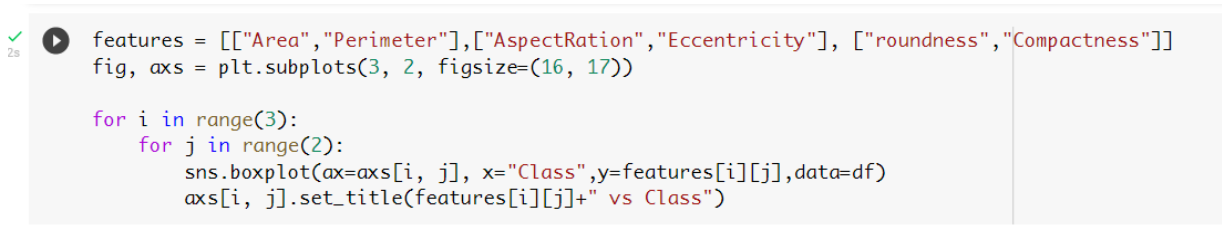
```
[19] df = df.fillna(df.median())
```

```
df.isnull().sum()
```

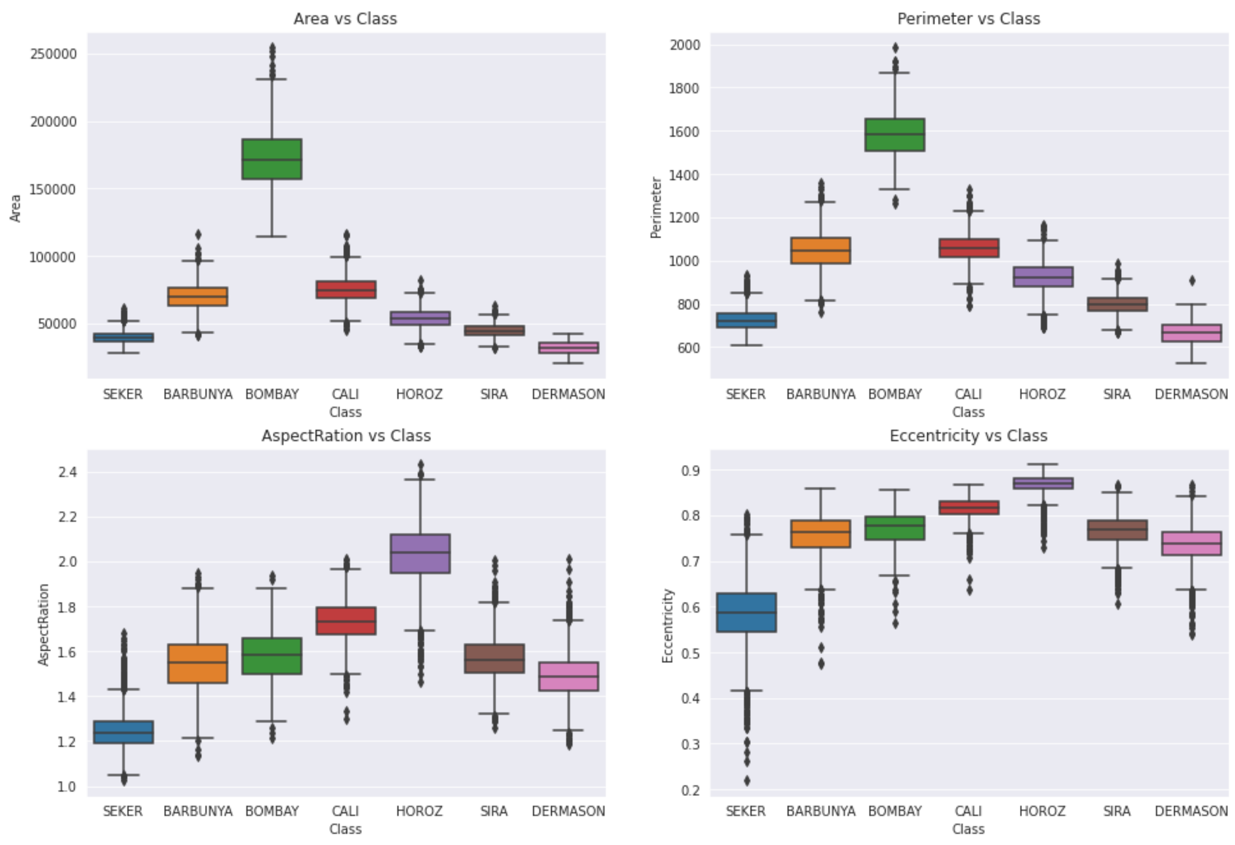
```
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    0
AspectRation       0
Eccentricity       0
Extent             0
Solidity           0
roundness          0
Compactness        0
ShapeFactor1       0
ShapeFactor2       0
ShapeFactor3       0
ShapeFactor4       0
Class              0
dtype: int64
```

6.4.2. Handling Outliers

Plotting outliers using boxplot

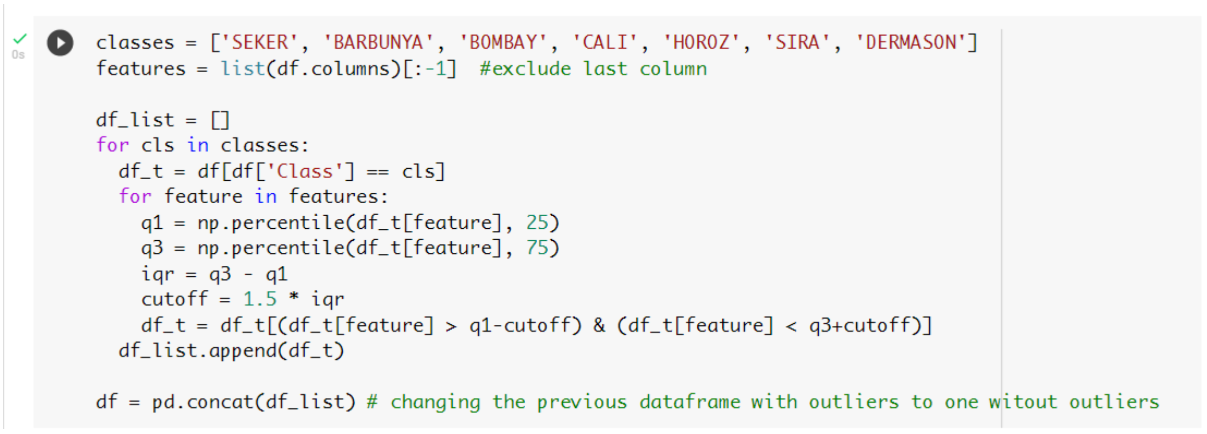


```
features = [["Area","Perimeter"],["AspectRation","Eccentricity"], ["roundness","Compactness"]]
fig, axs = plt.subplots(3, 2, figsize=(16, 17))

for i in range(3):
    for j in range(2):
        sns.boxplot(ax=axs[i, j], x="Class",y=features[i][j],data=df)
        axs[i, j].set_title(features[i][j]+" vs Class")
```

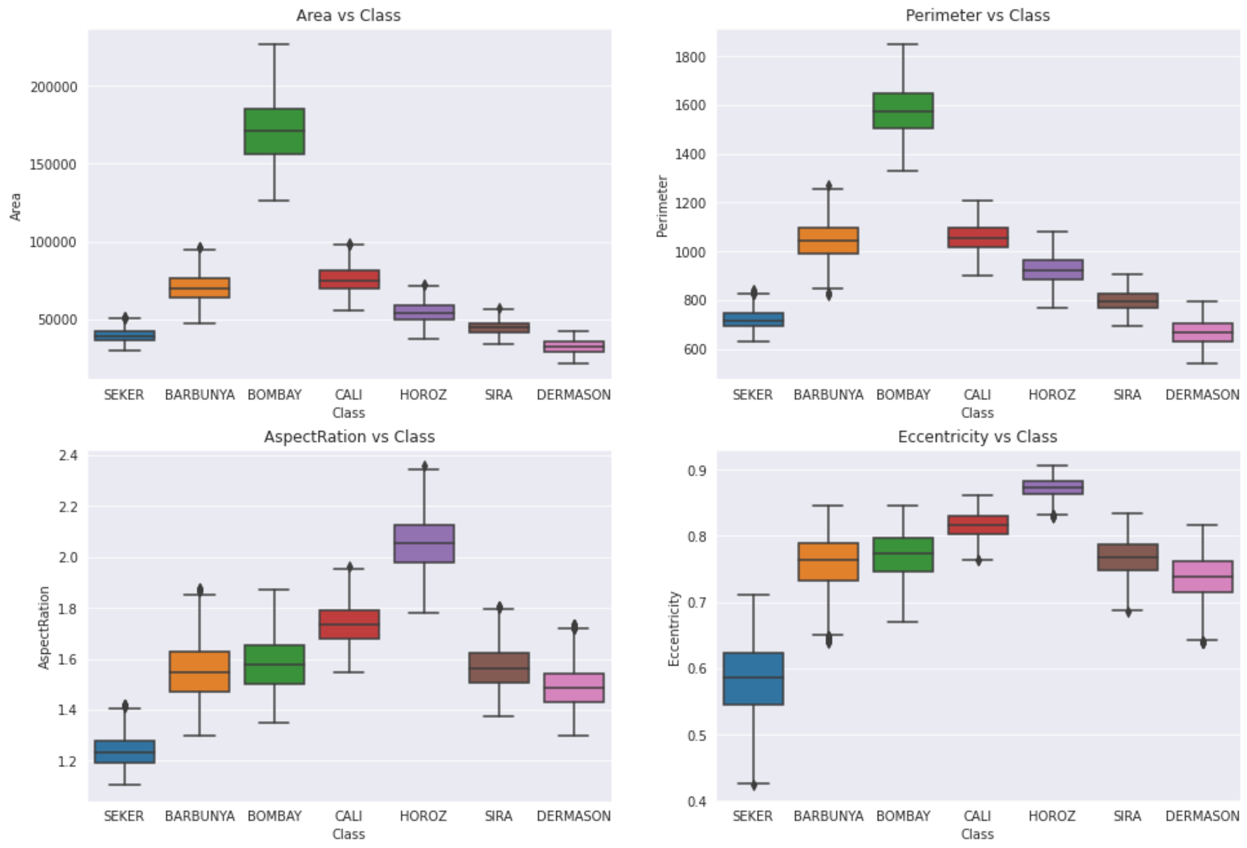


The boxplot shows that all the features of each class contains outliers that needs to be handled

Removing Outliers



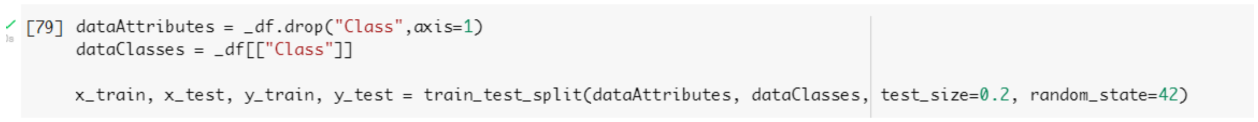
```
classes = ['SEKER', 'BARBUNYA', 'BOMBAY', 'CALI', 'HOROZ', 'SIRA', 'DERMASON']
features = list(df.columns)[:-1]  #exclude last column

df_list = []
for cls in classes:
  df_t = df[df['Class'] == cls]
  for feature in features:
    q1 = np.percentile(df_t[feature], 25)
    q3 = np.percentile(df_t[feature], 75)
    iqr = q3 - q1
    cutoff = 1.5 * iqr
    df_t = df_t[(df_t[feature] > q1-cutoff) & (df_t[feature] < q3+cutoff)]
  df_list.append(df_t)

df = pd.concat(df_list) # changing the previous dataframe with outliers to one witout outliers
```



1966 instances have been removed in the process of outlier removal

6.4.3. Splitting data into training and testing sets

Here we split 60% of the data for training and rest 40% for testing

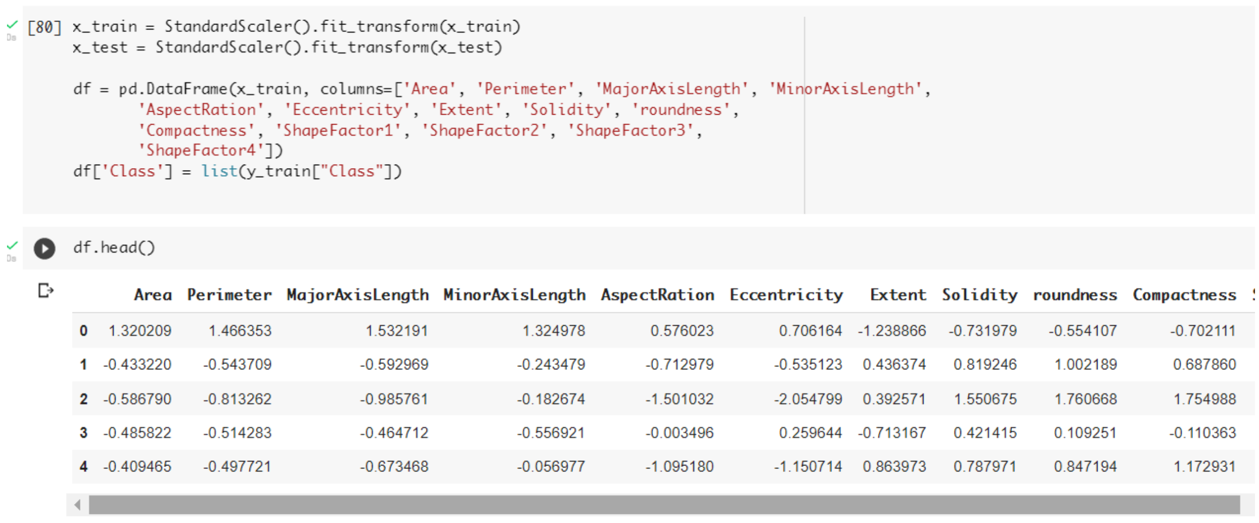


```
[79] dataAttributes = _df.drop("Class",axis=1)
     dataClasses = _df[["Class"]]

     x_train, x_test, y_train, y_test = train_test_split(dataAttributes, dataClasses, test_size=0.2, random_state=42)
```

6.4.4 Normalization using standard scaler

The scale of the features could affect the learning model's accuracy. Thus there is a necessity to change the values of the features in the dataset to use a common scale.

Here we achieve this using Standard Scaler



```
[80] x_train = StandardScaler().fit_transform(x_train)
     x_test = StandardScaler().fit_transform(x_test)

     df = pd.DataFrame(x_train, columns=['Area', 'Perimeter', 'MajorAxisLength', 'MinorAxisLength',
             'AspectRation', 'Eccentricity', 'Extent', 'Solidity', 'roundness',
             'Compactness', 'ShapeFactor1', 'ShapeFactor2', 'ShapeFactor3',
             'ShapeFactor4'])
     df['Class'] = list(y_train["Class"])
```

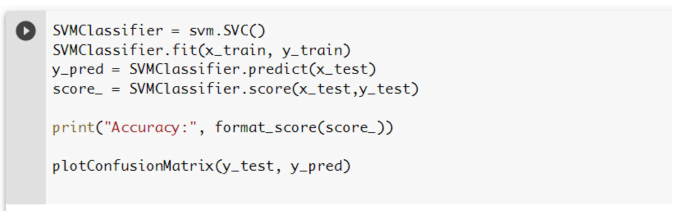
```
df.head()
```

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	Extent	Solidity	roundness	Compactness
0	1.320209	1.466353	1.532191	1.324978	0.576023	0.706164	-1.238866	-0.731979	-0.554107	-0.702111
1	-0.433220	-0.543709	-0.592969	-0.243479	-0.712979	-0.535123	0.436374	0.819246	1.002189	0.687860
2	-0.586790	-0.813262	-0.985761	-0.182674	-1.501032	-2.054799	0.392571	1.550675	1.760668	1.754988
3	-0.485822	-0.514283	-0.464712	-0.556921	-0.003496	0.259644	-0.713167	0.421415	0.109251	-0.110363
4	-0.409465	-0.497721	-0.673468	-0.056977	-1.095180	-1.150714	0.863973	0.787971	0.847194	1.172931

Preprocessing is required to remove null values, outliers and to standardize the data so that we can ensure no bias in the dataset. Outliers should be removed because outliers badly affect mean and standard deviation of the dataset. These may statistically give erroneous results. We should standardize the data to make sure that data is internally consistent that is, each data type has the same content and format.

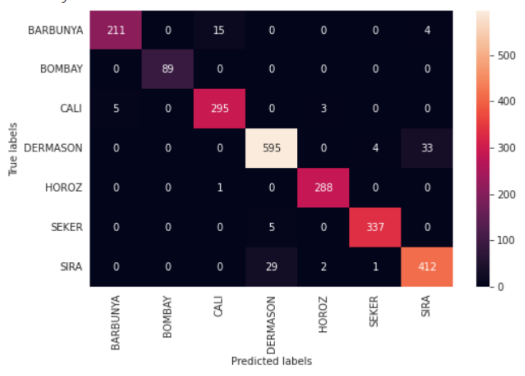
7. DATA MODELING AND INFERENCE

7.1. Support Vector Machine

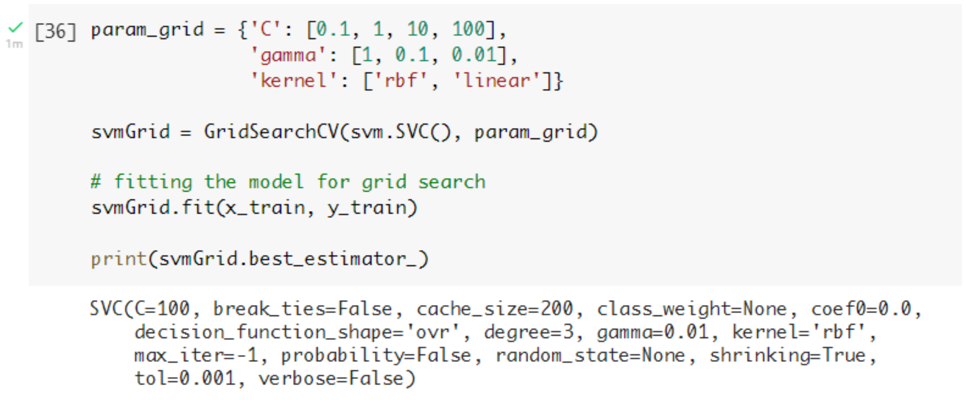


```
SVMClassifier = svm.SVC()
SVMClassifier.fit(x_train, y_train)
y_pred = SVMClassifier.predict(x_test)
score_ = SVMClassifier.score(x_test,y_test)

print("Accuracy:", format_score(score_))

plotConfusionMatrix(y_test, y_pred)
```

```
Accuracy: 95.6204
```

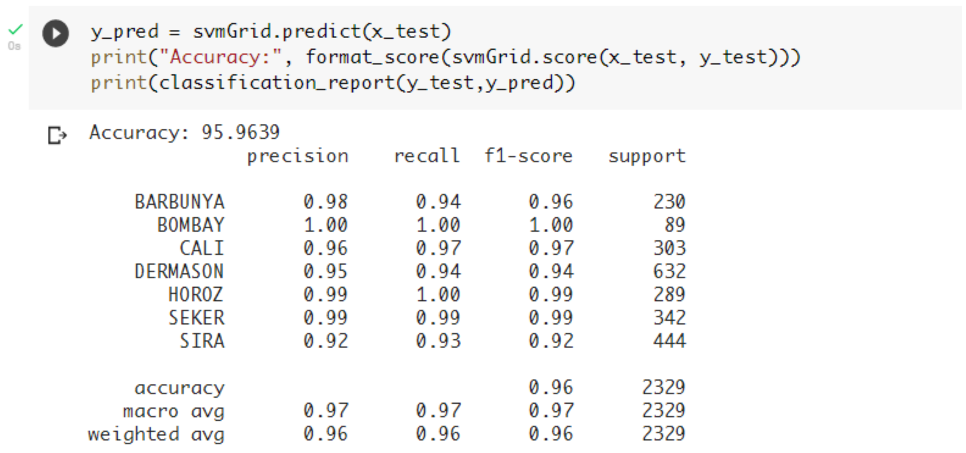


Hyperparameter Tuning



```
[36] param_grid = {'C': [0.1, 1, 10, 100],
                   'gamma': [1, 0.1, 0.01],
                   'kernel': ['rbf', 'linear']}

     svmGrid = GridSearchCV(svm.SVC(), param_grid)

     # fitting the model for grid search
     svmGrid.fit(x_train, y_train)

     print(svmGrid.best_estimator_)
```

```
SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

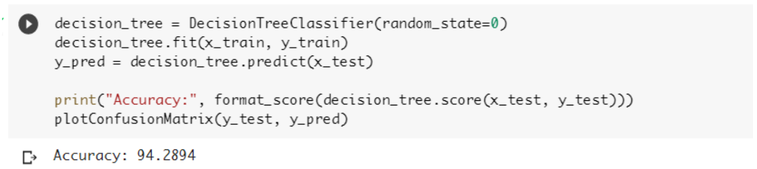


```
y_pred = svmGrid.predict(x_test)
print("Accuracy:", format_score(svmGrid.score(x_test, y_test)))
print(classification_report(y_test,y_pred))
```

```
Accuracy: 95.9639
              precision    recall  f1-score   support

    BARBUNYA       0.98      0.94      0.96       230
      BOMBAY       1.00      1.00      1.00        89
        CALI       0.96      0.97      0.97       303
    DERMASON       0.95      0.94      0.94       632
       HOROZ       0.99      1.00      0.99       289
       SEKER       0.99      0.99      0.99       342
        SIRA       0.92      0.93      0.92       444

    accuracy                           0.96      2329
   macro avg       0.97      0.97      0.97      2329
weighted avg       0.96      0.96      0.96      2329
```

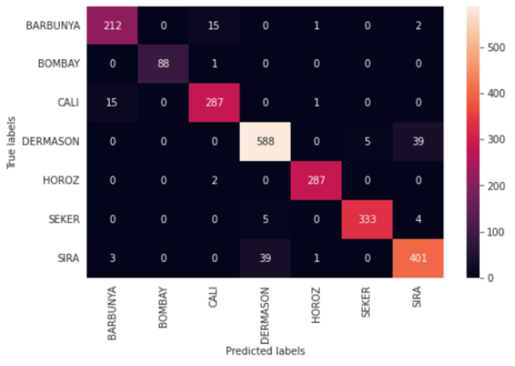
Here, we can see that the accuracy of SVM classifier has improved from 95.62% to 95.96%

7.2. Decision Tree



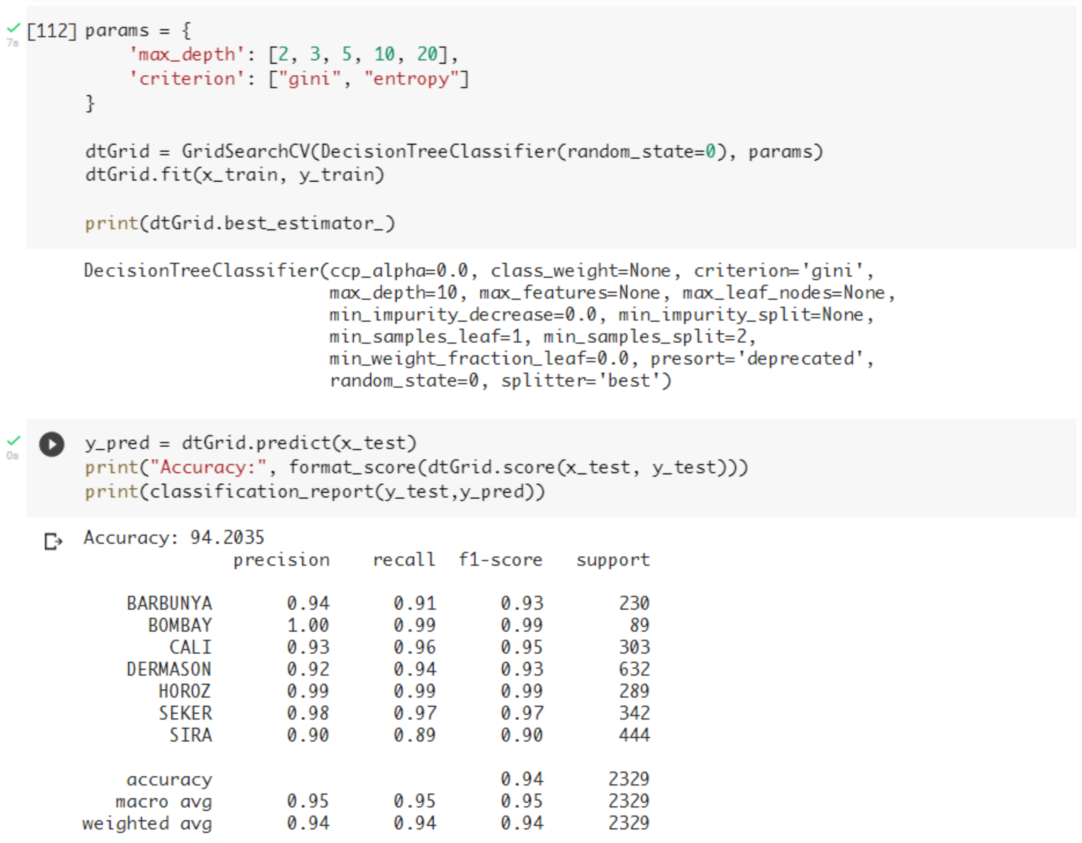
```
decision_tree = DecisionTreeClassifier(random_state=0)
decision_tree.fit(x_train, y_train)
y_pred = decision_tree.predict(x_test)

print("Accuracy:", format_score(decision_tree.score(x_test, y_test)))
plotConfusionMatrix(y_test, y_pred)
```

```
Accuracy: 94.2894
```



	BARBUNYA	BOMBAY	CALI	DERMASON	HOROZ	SEKER	SIRA
BARBUNYA	212	0	15	0	1	0	2
BOMBAY	0	88	1	0	0	0	0
CALI	15	0	287	0	1	0	0
DERMASON	0	0	0	588	0	5	39
HOROZ	0	0	2	0	287	0	0
SEKER	0	0	0	5	0	333	4
SIRA	3	0	0	39	1	0	401

Hyperparameter Tuning



```
[112] params = {
          'max_depth': [2, 3, 5, 10, 20],
          'criterion': ["gini", "entropy"]
      }

      dtGrid = GridSearchCV(DecisionTreeClassifier(random_state=0), params)
      dtGrid.fit(x_train, y_train)

      print(dtGrid.best_estimator_)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

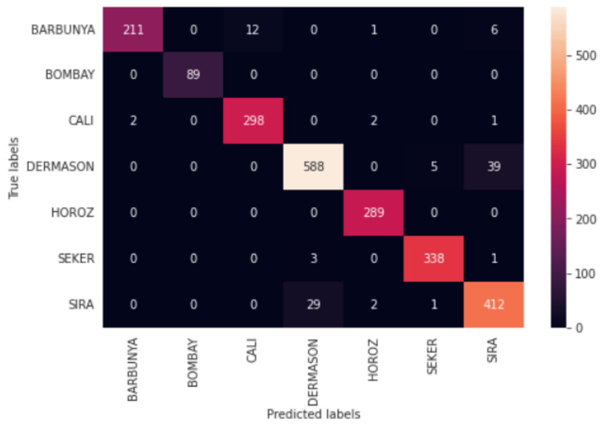
```
y_pred = dtGrid.predict(x_test)
print("Accuracy:", format_score(dtGrid.score(x_test, y_test)))
print(classification_report(y_test,y_pred))
```

```
Accuracy: 94.2035
              precision    recall  f1-score   support

    BARBUNYA       0.94      0.91      0.93       230
      BOMBAY       1.00      0.99      0.99        89
        CALI       0.93      0.96      0.95       303
    DERMASON       0.92      0.94      0.93       632
       HOROZ       0.99      0.99      0.99       289
       SEKER       0.98      0.97      0.97       342
        SIRA       0.90      0.89      0.90       444

    accuracy                           0.94      2329
   macro avg       0.95      0.95      0.95      2329
weighted avg       0.94      0.94      0.94      2329
```

7.3. K-Nearest Neighbours Classifier

```
model = KNeighborsClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

print("Accuracy:", format_score(model.score(x_test, y_test)))
plotConfusionMatrix(y_test, y_pred)
```

```
Accuracy: 95.5346
```

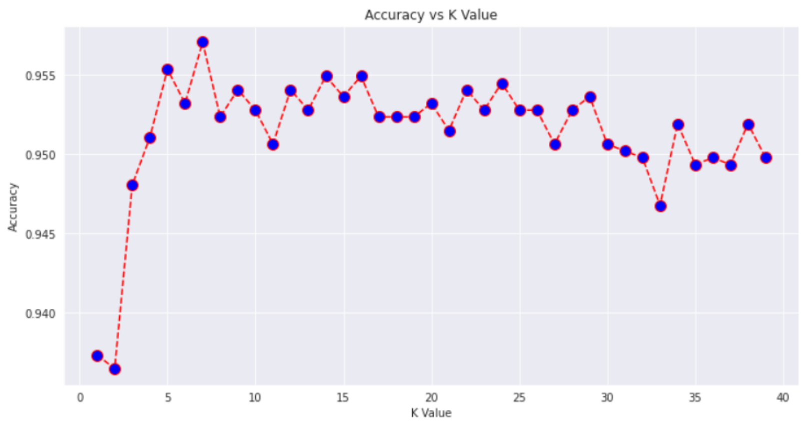


Hyperparameter Tuning

Finding the best value for the hyperparameter k

```
accuracy = []
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    accuracy.append(knn.score(x_test, y_test))
```

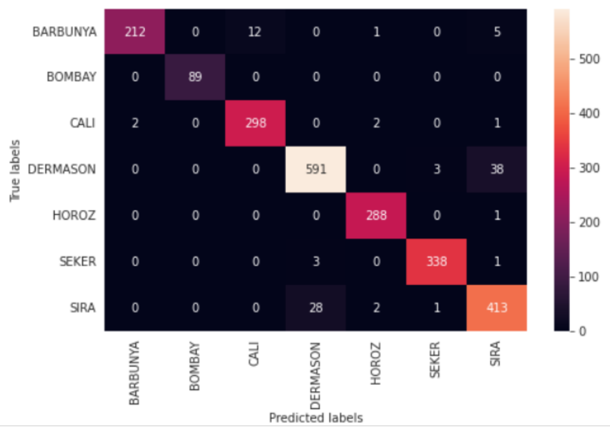
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), accuracy, color='red', linestyle='dashed', marker='o',markerfacecolor='blue', markersize=10)
plt.title('Accuracy vs K Value')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.show()
```



From the plot above, we can see that the highest accuracy is achieved at k=7

```
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train, y_train)

y_pred = knn.predict(x_test)
print("Accuracy:", format_score(knn.score(x_test, y_test)))
plotConfusionMatrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

```
Accuracy: 95.7063
```



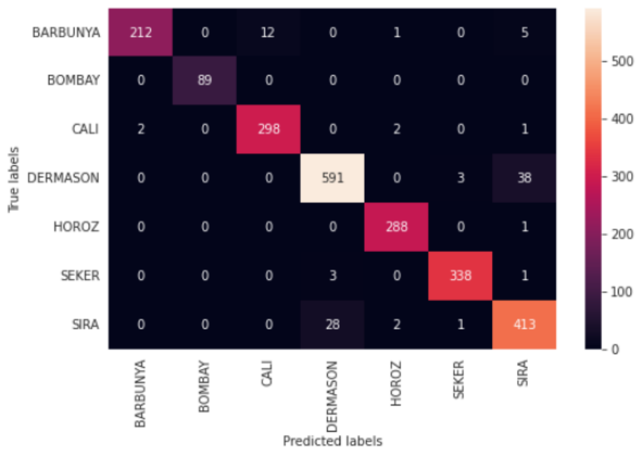
	BARBUNYA	BOMBAY	CALI	DERMASON	HOROZ	SEKER	SIRA
BARBUNYA	212	0	12	0	1	0	5
BOMBAY	0	89	0	0	0	0	0
CALI	2	0	298	0	2	0	1
DERMASON	0	0	0	591	0	3	38
HOROZ	0	0	0	0	288	0	1
SEKER	0	0	0	3	0	338	1
SIRA	0	0	0	28	2	1	413

Here, we can see that the accuracy of **kNN** classifier has improved from 95.5% to 95.7%

7.4. Stochastic Gradient Descent

```
sgdClf = SGDClassifier(loss="hinge", penalty="l2", alpha =0.004, learning_rate = "optimal", random_state=0)
sgdClf.fit(x_train,y_train)

print("Accuracy:", format_score(sgdClf.score(x_test, y_test)))
plotConfusionMatrix(y_test, y_pred)
```

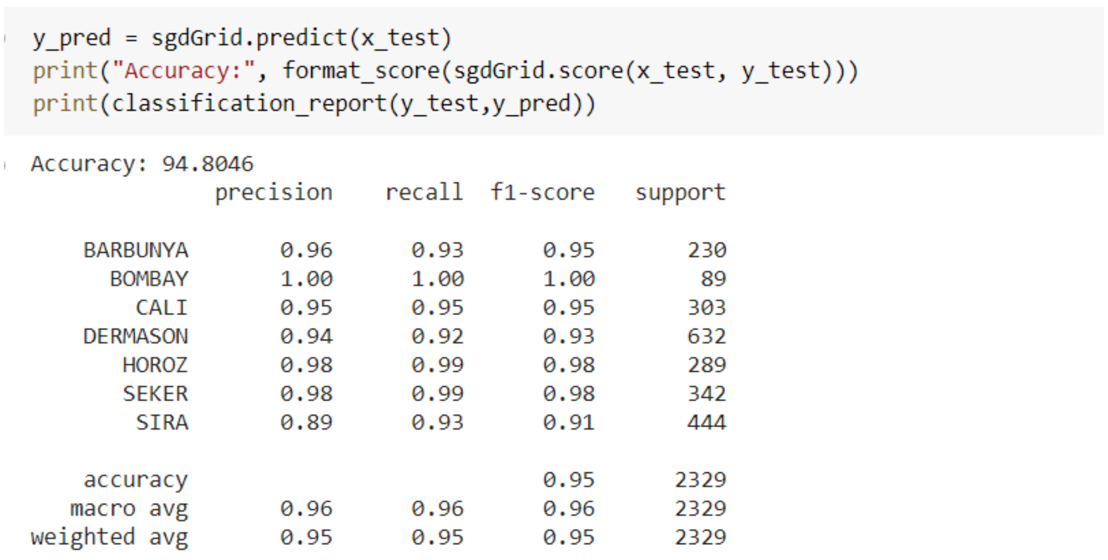
```
Accuracy: 93.4736
```



Hyperparameter Tuning

```
params = {
    "loss" : ["hinge", "log", "squared_hinge", "modified_huber"],
    "alpha" : [0.0001, 0.001, 0.01, 0.1],
    "penalty" : ["l2", "l1", "none"],
}

model = SGDClassifier(max_iter=1000)
sgdGrid = GridSearchCV(model, param_grid=params)
sgdGrid.fit(x_train, y_train)

print(sgdGrid.best_estimator_)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='modified_huber',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



```
y_pred = sgdGrid.predict(x_test)
print("Accuracy:", format_score(sgdGrid.score(x_test, y_test)))
print(classification_report(y_test,y_pred))
```

```
Accuracy: 94.8046
              precision    recall  f1-score   support

    BARBUNYA       0.96      0.93      0.95       230
      BOMBAY       1.00      1.00      1.00        89
        CALI       0.95      0.95      0.95       303
    DERMASON       0.94      0.92      0.93       632
       HOROZ       0.98      0.99      0.98       289
       SEKER       0.98      0.99      0.98       342
        SIRA       0.89      0.93      0.91       444

    accuracy                           0.95      2329
   macro avg       0.96      0.96      0.96      2329
weighted avg       0.95      0.95      0.95      2329
```

Here, we can see that the accuracy of **Stochastic Gradient Descent** classifier has improved from 93.47% to 94.8%

7.5. Inference

From the confusion matrices, it can be seen that we have a large proportion of correctly classified classes for the beans. Some notable observations include how there are many more instances of the DERMASON class of beans and thus we should not be surprised that there are far more incorrect classifications of the DERMASON class. In particular, there were some DERMASON beans which have been incorrectly classified as SIRA beans and vice versa. From these errors, we may assume that these two beans may be more similar to each other compared to the other beans.

Among the four models, SVM gives the highest accuracy of 95.96% followed by kNN which has an accuracy of 95.7%.

7.6. Voting Ensemble

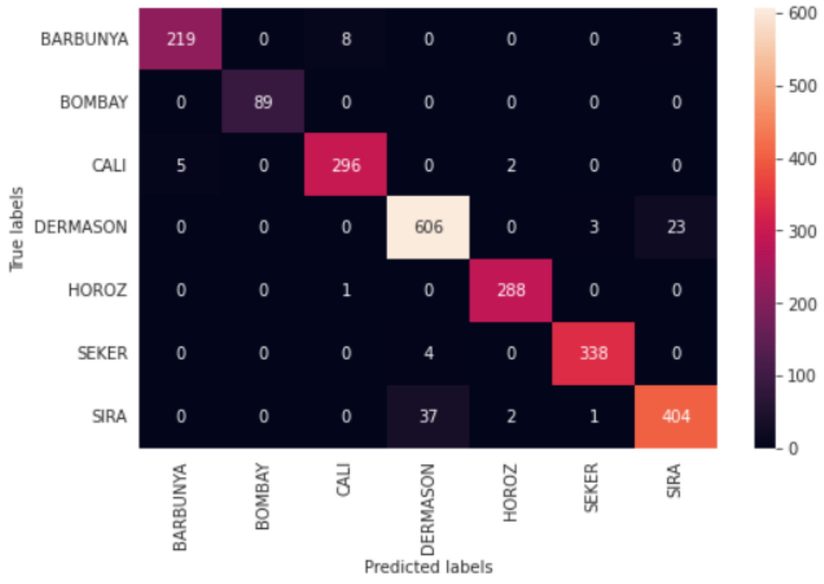
Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms.

It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data.

As the accuracies of SVM and kNN models are high, we use them as estimators in the Voting Classifier.

```
estimators = [
              ('svm', svmGrid.best_estimator_),
              ('knn', KNeighborsClassifier(n_neighbors=7))
]

votingClassifier = VotingClassifier(estimators=estimators)
votingClassifier.fit(x_train, y_train)
# y_pred = votingClassifier.predict(x_test)

y_pred = votingClassifier.predict(x_test)
print("Accuracy:", format_score(votingClassifier.score(x_test, y_test)))
plotConfusionMatrix(y_test, y_pred)
print(classification_report(y_test,y_pred))
```

```
Accuracy: 96.1786
```



	BARBUNYA	BOMBAY	CALI	DERMASON	HOROZ	SEKER	SIRA
BARBUNYA	219	0	8	0	0	0	3
BOMBAY	0	89	0	0	0	0	0
CALI	5	0	296	0	2	0	0
DERMASON	0	0	0	606	0	3	23
HOROZ	0	0	1	0	288	0	0
SEKER	0	0	0	4	0	338	0
SIRA	0	0	0	37	2	1	404

8. CONCLUSION AND FUTURE ENHANCEMENTS

The performance of the voting classifier is the highest achieved among all the models, with an accuracy of 96.12%. This shows that we can obtain improved accuracies with ensemble techniques. Thus, in this case study, we have implemented several multi-class classifiers and analyzed the performance of the models. We have also proposed an ensemble learning technique based on voting which is built on SVM and kNN models as estimators, which gives us the highest achieved accuracy. Although we have higher values of precision and recall, there exists a misclassification between the two classes “DERMASON” and “SIRA”. Moreover, the dataset on which the models have been trained, is skewed which was evident in the count plot earlier. Thus, the future work of this case study will be to find measure that can differentiate the two similar classes and also incorporating resampling techniques in order to remove skewness in the dataset for improved performance.

9. REFERENCES

[1] Murat Koklu, Ilker Ali Ozkan, Multiclass classification of dry beans using computer vision and machine learning techniques, Computers and Electronics in Agriculture, Volume 174, 2020, 105507, ISSN 0168-1699, https://doi.org/10.1016/j.compag.2020.105507

[2] Kıvanç Kılıç, İsmail Hakki Boyacı, Hamit Köksel, İsmail Küsmenoğlu, A classification system for beans using computer vision system and artificial neural networks, Journal of Food Engineering, Volume 78, Issue 3, 2007, Pages 897-904, ISSN 0260-774, https://doi.org/10.1016/j.jfoodeng.2005.11.030.

[3] Kim HC., Pang S., Je HM., Kim D., Bang SY. (2002) Support Vector Machine Ensemble with Bagging. In: Lee SW., Verri A. (eds) Pattern Recognition with Support Vector Machines. SVM 2002. Lecture Notes in Computer Science, vol 2388. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45665-1_31

[4] Ensemble Machine Learning Algorithms in Python with scikit-learn, https://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/