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

Introduction

Images of 6 varieties of dry beans were provided(Koklu, 2020), for use in testing a classification model. 16 physical variables were recorded for each of the dry beans to be used to help with the classification processes the ground truth was known, as each was described by a specific class. Classes of the beans were determined by the Turkish Standards Institute and beans collected from certified seed producers. Using a variety of beans in agriculture will be helpful due to creating a more resilient agricultural systems, due to a wider variety of plants grown which can withstand extreme climate events such as drought and frosts.

Exploratory data analysis

Data was loaded into a Python data frame using the Pandas library. The initial rows were visualised using the .head() function.

```
import pandas as pd

data = pd.read_excel("C:\\Users\\Toshiba\\OneDrive - University of
Sussex\\Machine_Learning\\DryBeanDataset\\DryBeanDataset\\
Dry_Bean_Dataset.xlsx", sheet_name = "Dry_Beans_Dataset" )


data.head()
```

```
    Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRation  \
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

   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

   Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFactor4
Class
0     0.913358      0.007332      0.003147      0.834222      0.998724
SEKER
1     0.953861      0.006979      0.003564      0.909851      0.998430
SEKER
2     0.908774      0.007244      0.003048      0.825871      0.999066
SEKER
3     0.928329      0.007017      0.003215      0.861794      0.994199
SEKER
4     0.970516      0.006697      0.003665      0.941900      0.999166
SEKER
```

From the initial data we can see the ranges of values for each physical variable.

To check that the data did not have null values, all null values in the data frame were summed, no Null values were recorded.

```
data.isnull().sum(),
#zero NA values

(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,)
```

The data set was split into a training data set and a testing data set using the sklearn library, with 80% used for the training set and 20% used for the testing dataset. A random seed was chosen to ensure reproducibility of results.

```
import random
random.seed(10)
from sklearn.model_selection import train_test_split
```

```
df_EDA, df_TEST = train_test_split(data, test_size=0.2, random_state =
0)
df_EDA = df_EDA.reset_index()
df_TEST = df_TEST.reset_index()
df_EDA = df_EDA.drop('index', axis=1)
df_TEST = df_TEST.drop('index', axis=1)
```

The function .describe() was used to find the mean, standard deviation as well as quartile values for each physical descriptor, then mean value and standard deviation of each physical variable for each class was also found.

```
#describe( prints summary statisicts of dataframe, function takes 2
aguments a dataframe and a particpicant number
#returns summary statistics)

df_EDA.describe()
```

```
                Area     Perimeter  MajorAxisLength
MinorAxisLength  \
count   10888.000000  10888.000000     10888.000000     10888.000000

mean    53203.683780    856.280757       320.413307       202.571258

std     29577.127613    215.594309        86.131501        45.307357

min     20420.000000    524.736000       183.601165       122.512653

25%     36457.750000    704.743000       253.495671       176.001038

50%     44661.500000    794.889500       296.507842       192.535682

75%     61387.000000    977.813500       376.835376       217.272786

max    254616.000000   1985.370000       738.860153       460.198497


       AspectRation  Eccentricity     ConvexArea  EquivDiameter
Extent  \
count  10888.000000  10888.000000   10888.000000   10888.000000
10888.000000
mean       1.582320      0.750462   53924.566495     253.362929
0.749924
std        0.247216      0.092146   30029.120480      59.570012
0.049154
min        1.041964      0.280937   20684.000000     161.243764
0.555315
25%        1.430506      0.715069   36842.000000     215.451732
0.718664
50%        1.549491      0.763867   45170.000000     238.463389
```

```
0.760192
75%        1.706619      0.810344   62341.250000     279.571737
0.787088
max        2.430306      0.911423  263261.000000     569.374358
0.866195

           Solidity     roundness   Compactness  ShapeFactor1
ShapeFactor2  \
count  10888.000000  10888.000000  10888.000000  10888.000000
10888.000000
mean       0.987158      0.873345      0.800148      0.006556
0.001716
std        0.004691      0.059733      0.061853      0.001132
0.000597
min        0.919246      0.489618      0.640577      0.002778
0.000564
25%        0.985684      0.832035      0.762741      0.005885
0.001152
50%        0.988287      0.883283      0.801665      0.006642
0.001698
75%        0.990046      0.917093      0.834632      0.007267
0.002172
max        0.994677      0.990685      0.979432      0.010451
0.003665

       ShapeFactor3  ShapeFactor4
count  10888.000000  10888.000000
mean       0.644062      0.995107
std        0.099222      0.004354
min        0.410339      0.947687
25%        0.581774      0.993791
50%        0.642666      0.996421
75%        0.696610      0.997902
max        0.959287      0.999733
```

```
df_EDA.groupby("Class").mean().reset_index()
```

```
      Class           Area    Perimeter  MajorAxisLength
MinorAxisLength  \
0  BARBUNYA   69930.522024  1047.349992       370.454890
240.458533
1    BOMBAY  173358.609302  1584.979688       592.845434
374.258907
2      CALI   75708.078125  1058.876532       410.106367
236.554197
3  DERMASON   32131.562789   665.303222       246.603548
165.682955
4     HOROZ   53649.994148   919.992145       372.677151
184.100730
5     SEKER   39951.267717   728.373898       251.565457
```

```
202.030383
6      SIRA   44727.732354   796.351790       299.294379
190.844164

   AspectRation  Eccentricity     ConvexArea  EquivDiameter    Extent
\
0      1.545178      0.754907   71156.180881     297.583112  0.748740

1      1.585118      0.770539  175664.518605     468.806049  0.777703

2      1.734907      0.815108   76862.847656     309.882657  0.759281

3      1.490491      0.736595   32510.459623     201.717499  0.752805

4      2.027309      0.867638   54440.519506     260.734490  0.706736

5      1.245795      0.585563   40340.898243     225.138681  0.771959

6      1.569272      0.766988   45271.247276     238.330880  0.749583


   Solidity  roundness  Compactness  ShapeFactor1  ShapeFactor2
ShapeFactor3  \
0  0.982778   0.799761     0.804851      0.005353      0.001392
0.648908
1  0.987042   0.864626     0.792746      0.003442      0.000845
0.629366
2  0.984985   0.845844     0.756423      0.005455      0.001105
0.572597
3  0.988251   0.908154     0.819112      0.007754      0.002161
0.671646
4  0.985496   0.794254     0.700699      0.007008      0.001047
0.491533
5  0.990337   0.944295     0.896589      0.006331      0.002536
0.804682
6  0.987983   0.884770     0.797567      0.006718      0.001684
0.636715

   ShapeFactor4
0      0.995842
1      0.991998
2      0.990564
3      0.996912
4      0.992032
5      0.998366
6      0.995412

df_EDA.groupby("Class").std().reset_index()

      Class          Area   Perimeter  MajorAxisLength
MinorAxisLength  \
```

```
0  BARBUNYA  10273.397859   89.735243        32.258679
19.820577
1    BOMBAY  22943.224669  114.443093        52.355372
22.940427
2      CALI   9414.641566   67.672927        29.522713
14.700405
3  DERMASON   4708.702881   50.722785        20.827669
12.592851
4     HOROZ   7338.306600   70.123049        30.283569
13.396994
5     SEKER   4832.918502   48.306396        19.967373
11.093681
6      SIRA   4557.341303   44.396102        20.816588
9.173038

   AspectRation  Eccentricity    ConvexArea  EquivDiameter
Extent  \
0      0.126553      0.050235  10454.810728      21.978406  0.040424

1      0.117195      0.040523  23471.326093      30.824265  0.039245

2      0.091551      0.022682   9550.899040      19.171466  0.042298

3      0.096657      0.040545   4742.525956      14.876344  0.037418

4      0.134534      0.020953   7446.448816      18.084038  0.075732

5      0.081529      0.070272   4878.561565      13.424499  0.018633

6      0.096822      0.032834   4607.339119      12.148210  0.044359


   Solidity  roundness  Compactness  ShapeFactor1  ShapeFactor2
ShapeFactor3  \
0  0.004043   0.048295     0.033525      0.000444      0.000214
0.054465
1  0.005204   0.026468     0.030383      0.000205      0.000133
0.048402
2  0.005769   0.023632     0.020533      0.000338      0.000128
0.031266
3  0.002935   0.029426     0.026470      0.000600      0.000286
0.043380
4  0.006210   0.032315     0.023541      0.000504      0.000150
0.033508
5  0.003087   0.032178     0.028483      0.000339      0.000325
0.050684
6  0.002795   0.023907     0.024554      0.000325      0.000205
0.039120

   ShapeFactor4
```

```
0      0.002470
1      0.004503
2      0.004506
3      0.001876
4      0.006445
5      0.001578
6      0.002651
```

The BOMBAY bean class mean area is double any of the other dry bean mean areas, suggesting it will be easier to classify this class. Besides these significant variations between classes cannot be recognised from the table.

```
df_EDA.dtypes
```

```
Area                int64
Perimeter         float64
MajorAxisLength   float64
MinorAxisLength   float64
AspectRation      float64
Eccentricity      float64
ConvexArea          int64
EquivDiameter     float64
Extent            float64
Solidity          float64
roundness         float64
Compactness       float64
ShapeFactor1      float64
ShapeFactor2      float64
ShapeFactor3      float64
ShapeFactor4      float64
Class              object
dtype: object
```

The type of data was found for each physical descriptor, most were “float64” objects except the area and convex area which were “int64 objects”.

```
df_EDA["Class"].value_counts()
```

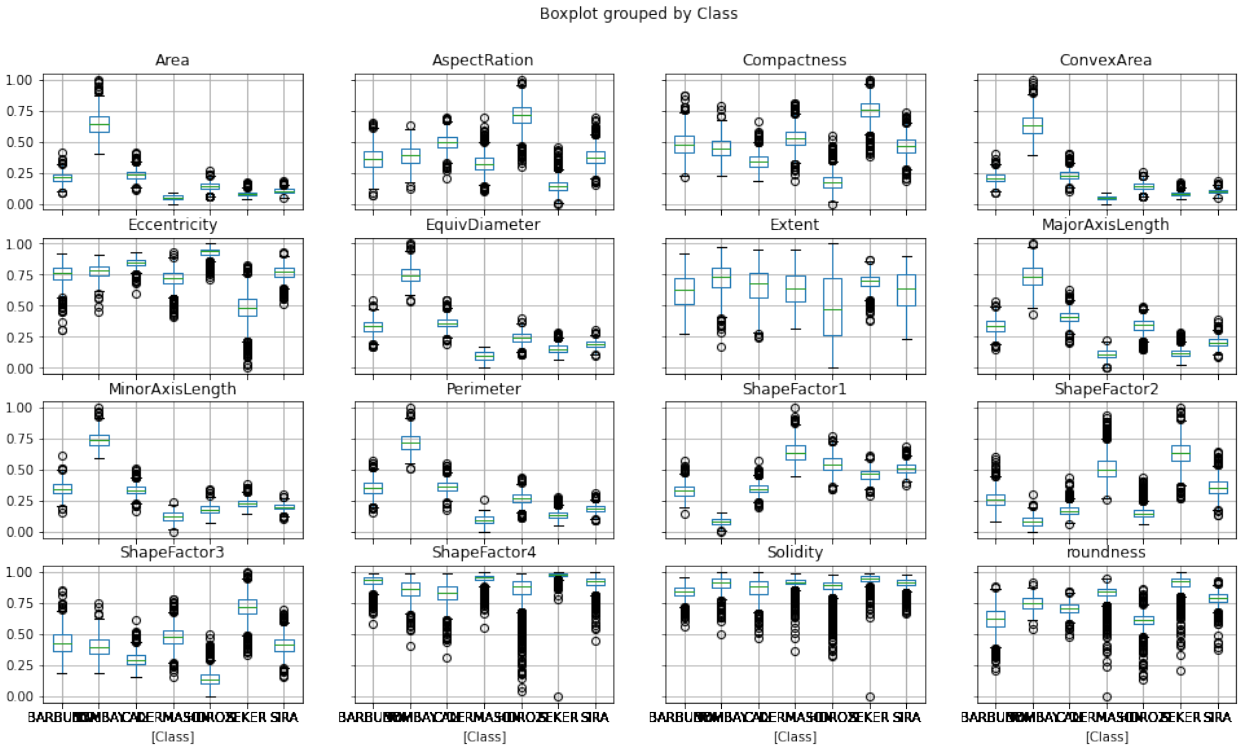
```
DERMASON    2811
SIRA        2111
SEKER       1651
HOROZ       1538
CALI        1280
BARBUNYA    1067
BOMBAY       430
Name: Class, dtype: int64
```

The data was normalised, and boxplots were created for each physical variable in each class, to help to visualise the spread of the data.

```
from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler object
scaler = MinMaxScaler()

df_EDA_2 = df_EDA.iloc[:,0:16]

# Normalize the values in the dataframe
df_normalized = pd.DataFrame(scaler.fit_transform(df_EDA_2),
columns=df_EDA_2.columns)
df_normalized["Class"] = df_EDA["Class"]
```

```
from matplotlib import pyplot as plt
df_normalized.boxplot(by = "Class", figsize = (16,9))
```

```
array([[<AxesSubplot:title={'center':'Area'}, xlabel='[Class]'>,
        <AxesSubplot:title={'center':'AspectRation'},
xlabel='[Class]'>,
        <AxesSubplot:title={'center':'Compactness'},
xlabel='[Class]'>,
        <AxesSubplot:title={'center':'ConvexArea'},
xlabel='[Class]'>],
       [<AxesSubplot:title={'center':'Eccentricity'},
xlabel='[Class]'>,
        <AxesSubplot:title={'center':'EquivDiameter'},
xlabel='[Class]'>,
        <AxesSubplot:title={'center':'Extent'}, xlabel='[Class]'>,
        <AxesSubplot:title={'center':'MajorAxisLength'},
xlabel='[Class]'>],
       [<AxesSubplot:title={'center':'MinorAxisLength'},
xlabel='[Class]'>,
        <AxesSubplot:title={'center':'Perimeter'}, xlabel='[Class]'>,
        <AxesSubplot:title={'center':'ShapeFactor1'},
xlabel='[Class]'>,
        <AxesSubplot:title={'center':'ShapeFactor2'},
xlabel='[Class]'>],
       [<AxesSubplot:title={'center':'ShapeFactor3'},
xlabel='[Class]'>,
        <AxesSubplot:title={'center':'ShapeFactor4'},
xlabel='[Class]'>,
        <AxesSubplot:title={'center':'Solidity'}, xlabel='[Class]'>,
        <AxesSubplot:title={'center':'roundness'},
xlabel='[Class]'>]],
      dtype=object)
```



From the boxplots, shape factor4 and solidity have the greatest number of outliers. Extent has the largest interquartile range. Visually it can be clearly seen that BOMBAY class is different from the other classes especially in area and perimeter.

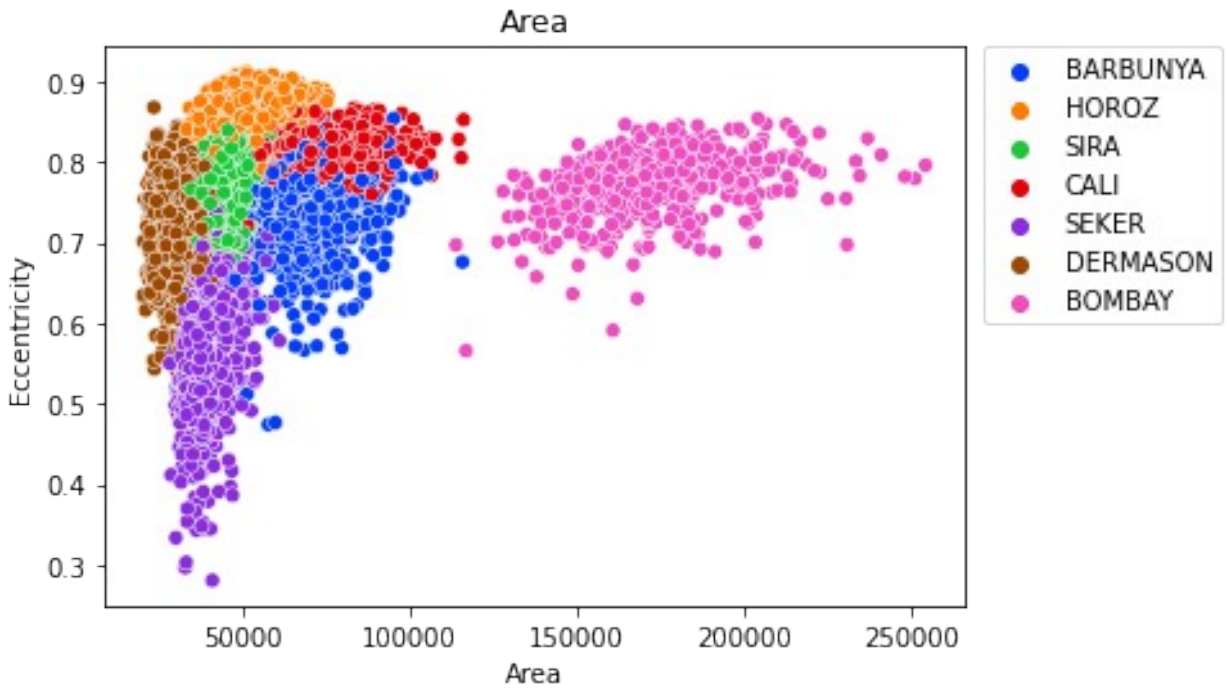
A heat map was created, to determine the extent to which different physical variables correlated with each other.

```
import seaborn as sns
plt.figure(figsize = (16,9))
plt.title('Peasons Correlation for all dependant variables')
sns.heatmap(df_EDA.corr(), cmap="PiYG", center=0, annot=True) #defualt
correlation paramter is pearson
```

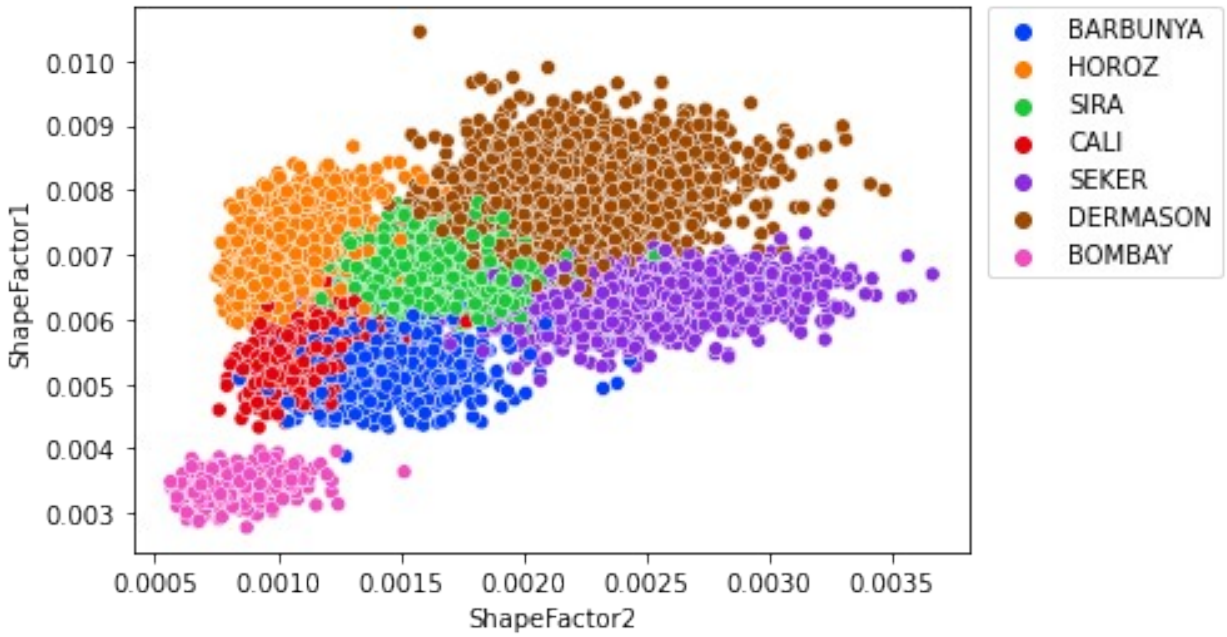
```
<AxesSubplot:title={'center':'Peasons Correlation for all dependant
variables'}>
```

A range of different physical variables were plotted in scatter plots, to visual the distribution and shape of varies clusters. Variables were chosen based on the box plots. Optimal clusters were obtained when there is a lack of outliers, a small value in the interquartile range and good separation between classes.

```
sns.scatterplot(data=df_EDA, x="Area", y="Eccentricity", hue="Class",
legend="full", palette=sns.color_palette('bright', n_colors=7))
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left',
borderaxespad=0)
plt.title('Area')
plt.xlabel('Area')
plt.ylabel('Eccentricity')
plt.show()
```




```
sns.scatterplot(data=df_EDA, x="ShapeFactor2", y="ShapeFactor1",
hue="Class", legend="full", palette=sns.color_palette('bright',
n_colors=7))
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left',
borderaxespad=0)

plt.xlabel('ShapeFactor2')
plt.ylabel('ShapeFactor1')
plt.show()
```



It becomes clear that the classes CALI and BARBUNYA are the most challenging to separate and have significant overlap between then in a range of physical characteristics. Bombay is a well separated cluster as predicted in the mean value table earlier. Apart from BOMBAY, there is little separation between the other classes, which may also present a challenge when classifying classes.

Methods

A random forest classifier(Decision tree model) was chosen to classify the data set. They combine multiple decision trees to make a prediction. Each tree is a random subset of the data. An advantage of this randomness is that it reduces issues with over-fitting Random forests are versatile, handle high-dimensional data well, and are effective for classification and regression tasks. random forests also handle outliers well, which may be useful due to the number of outliers seen in the exploratory data analysis. Other possible machine learning models that could have been used include, Multi layer perceptron, support vector machine and K-means clustering.

Firstly, the data was split into a training and testing set and the target class was extracted into a separate variable. The random forest classifier was imported from sklearn library.

```
import random
X = data
#Get Target data
y = data['Class']
print(y)
#Load X Variables into a Pandas Dataframe with columns
X = data.drop(['Class'], axis = 1)
random.seed(43)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=101)
```

```
0           SEKER
1           SEKER
2           SEKER
3           SEKER
4           SEKER
           ...
13606    DERMASON
13607    DERMASON
13608    DERMASON
13609    DERMASON
13610    DERMASON
Name: Class, Length: 13611, dtype: object
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rf = RandomForestClassifier(random_state = 20)

rf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators=10,
n_jobs=None,
            oob_score=False, random_state=20, verbose=0,
warm_start=False)
```

```
Y_pred = rf.predict(X_test)
accuracy_score(Y_pred, y_test)
```

```
0.9166360631656262
```

The default hyperparameters for the random forest classifier were then run on the data, to see how well the model performed. The default hyperparameters scored an accuracy of 91.7%(3sf), which is close to the optimal score of 93.1% achieved in the dry bean study by the support vector Machine model. Though higher than the accuracy of the decision tree model which was 87.9% (in the abstract it reports a value of 92.5% accuracy for the decision tree model - but then in table 12 of the paper it reports an accuracy of 87.9%).

```
import random

from sklearn.model_selection import RandomizedSearchCV
import numpy as np
```

```
param_grid = dict(max_features = np.arange(1,5,1),
                  n_estimators = np.arange(10,200,10),
                  bootstrap=[True, False],
                  max_depth=np.arange(1,5,1))
rf = RandomForestClassifier(random_state = 10)

random_search = RandomizedSearchCV(rf, param_distributions=param_grid,
random_state = 10)

random_search.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators='warn',
n_jobs=None,
            oob_score=False, random_state=10, verbose=0,
warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=None,
          param_distributions={'max_features': array([1, 2, 3, 4]),
'n_estimators': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90,
100, 110, 120, 130,
       140, 150, 160, 170, 180, 190]), 'bootstrap': [True, False],
'max_depth': array([1, 2, 3, 4])},
          pre_dispatch='2*n_jobs', random_state=10, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

```
print("The best parameters are %s with a score of %0.2f"
      % (random_search.best_params_, random_search.best_score_))
```

```
The best parameters are {'n_estimators': 30, 'max_features': 3,
'max_depth': 4, 'bootstrap': False} with a score of 0.89
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rf = RandomForestClassifier(n_estimators=30, max_features=3,
max_depth=4, bootstrap=False, random_state = 10)

rf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=False, class_weight=None,
criterion='gini',
            max_depth=4, max_features=3, 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, n_estimators=30,
```

```
n_jobs=None,
            oob_score=False, random_state=10, verbose=0,
warm_start=False)

Y_pred = rf.predict(X_test)
accuracy_score(Y_pred, y_test)

0.8876239441792141
```

A randomised search CV was then used with a variety of variable hyperparameter tested, including: “max features”, “n estimators”, “bootstrap” and “max depth”. The random search algorithm was run, and it outputted the best values for the hyperparameters as {'n_estimators': 30, 'max_features': 3, 'max_depth': 4, 'bootstrap': False} with an accuracy score of 0.89, however the accuracy score was lower than the default hyperparameters, which suggests the default hyperparameters are close to the optimal hyperparameter values.

A Grid search CV was then used with hyperparameter variables chosen close to the default values, to speed up computation time, only 3x3 variables were chosen with a 5-fold cross validation. From the grid search CV the optimal hyperparameters were given as {'min_samples_split': 10, n_estimators': 15} with and accuracy score of 92.0%. The process was the iteratively completed, with hyperparameters variable values tested as (n-1, n, n+1) from the optimal values given previously. This improved the accuracy of the classification to 91.7%, 92.0%, 92.1%, and 92.3%. At 92.3% accuracy, the performance did not improve further, suggesting a local maximum point of accuracy had been found. Since a Grid search CV was used, this unlike the randomised search gives the optimal outcome, from the given possibilities.

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [5,10,15],
    'min_samples_split': [2, 5, 10]
}


rf = RandomForestClassifier(random_state = 5)
grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs
= -1)

grid.fit(X_train, y_train)

GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators='warn',
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
```

```
       param_grid={'n_estimators': [5, 10, 15], 'min_samples_split':
[2, 5, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))

The best parameters are {'min_samples_split': 10, 'n_estimators': 15}
with a score of 0.92

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rf = RandomForestClassifier(min_samples_split=10,
n_estimators=15,random_state = 5)

rf.fit(X_train, y_train)

RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=10,
            min_weight_fraction_leaf=0.0, n_estimators=15,
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False)

Y_pred = rf.predict(X_test)
accuracy_score(Y_pred, y_test)

0.9173705471905986

from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [14,15,16],
    'min_samples_split': [9,10,11]
}


rf = RandomForestClassifier(random_state = 5)
grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs
= -1)


grid.fit(X_train, y_train)

GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators='warn',
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'n_estimators': [14, 15, 16], 'min_samples_split':
[9, 10, 11]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))

The best parameters are {'min_samples_split': 10, 'n_estimators': 16}
with a score of 0.92

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rf = RandomForestClassifier(min_samples_split=10,
n_estimators=16,random_state = 5)

rf.fit(X_train, y_train)

RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=10,
            min_weight_fraction_leaf=0.0, n_estimators=16,
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False)

Y_pred = rf.predict(X_test)
accuracy_score(Y_pred, y_test)

0.9199412412780023

from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [16,17,18],
    'min_samples_split': [9,10,11]
}


rf = RandomForestClassifier(random_state = 5)
grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs
= -1)

grid.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators='warn',
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'n_estimators': [16, 17, 18], 'min_samples_split':
[9, 10, 11]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))

The best parameters are {'min_samples_split': 9, 'n_estimators': 18}
with a score of 0.92

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rf = RandomForestClassifier(min_samples_split=9,
n_estimators=18,random_state = 5)

rf.fit(X_train, y_train)

RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=9,
            min_weight_fraction_leaf=0.0, n_estimators=18,
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False)

Y_pred = rf.predict(X_test)
accuracy_score(Y_pred, y_test)

0.9214102093279471

from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [18,19,20],
    'min_samples_split': [8,9,10]
}
```

```
rf = RandomForestClassifier(random_state = 5)

grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs
= -1)


grid.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators='warn',
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'n_estimators': [18, 19, 20], 'min_samples_split':
[8, 9, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))
```

```
The best parameters are {'min_samples_split': 9, 'n_estimators': 20}
with a score of 0.92
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rf = RandomForestClassifier(min_samples_split=9,
n_estimators=20,random_state = 5)

rf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=9,
            min_weight_fraction_leaf=0.0, n_estimators=20,
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False)
```

```
Y_pred = rf.predict(X_test)
accuracy_score(Y_pred, y_test)
```

```
0.9225119353654058
```

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [20,21,22],
    'min_samples_split': [8,9,10]
}


rf = RandomForestClassifier(random_state = 5)

grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs
= -1)


grid.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators='warn',
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'n_estimators': [20, 21, 22], 'min_samples_split':
[8, 9, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))
```

```
The best parameters are {'min_samples_split': 9, 'n_estimators': 20}
with a score of 0.92
```

```
from sklearn.model_selection import GridSearchCV
param_grid = {"max_leaf_nodes" : [2,5,10,15,20,None]}

rf = RandomForestClassifier(random_state = 5, min_samples_split=9,
n_estimators=20)

grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs
= -1)


grid.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True,
```

```
class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=9,
            min_weight_fraction_leaf=0.0, n_estimators=20,
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'max_leaf_nodes': [2, 5, 10, 15, 20, None]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))

The best parameters are {'max_leaf_nodes': None} with a score of 0.92

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rf = RandomForestClassifier(min_samples_split=9,
n_estimators=20,random_state = 5, max_depth = 50)

rf.fit(X_train, y_train)

RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=50, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=9,
            min_weight_fraction_leaf=0.0, n_estimators=20,
n_jobs=None,
            oob_score=False, random_state=5, verbose=0,
warm_start=False)

Y_pred = rf.predict(X_test)
accuracy_score(Y_pred, y_test)

0.9225119353654058

#df = pd.DataFrame({'Predicted': Y_pred, 'True Class': y_test})

# Specify the file path and name for the output XLSX file
#output_file = 'prediction_results.xlsx'

# Write the DataFrame to an XLSX file
#df.to_excel(output_file, index=False)
```

Evaluation

In the paper, the abstract and the results report different values (as discussed earlier)" Overall correct classification rates have been determined as 91.73%, 93.13%, 87.92% and 92.52% for MLP, SVM, kNN and DT, respectively.". For the decision tree algorithm, with this in mind I will assume the decision tree algorithms accuracy was 87.92% as reported in the table and make comparisons with the decision tree results.

From the sklearn library, a range of evaluation metrics, including, precision score, recall score and f1 score were imported as well as a confusion matrix.

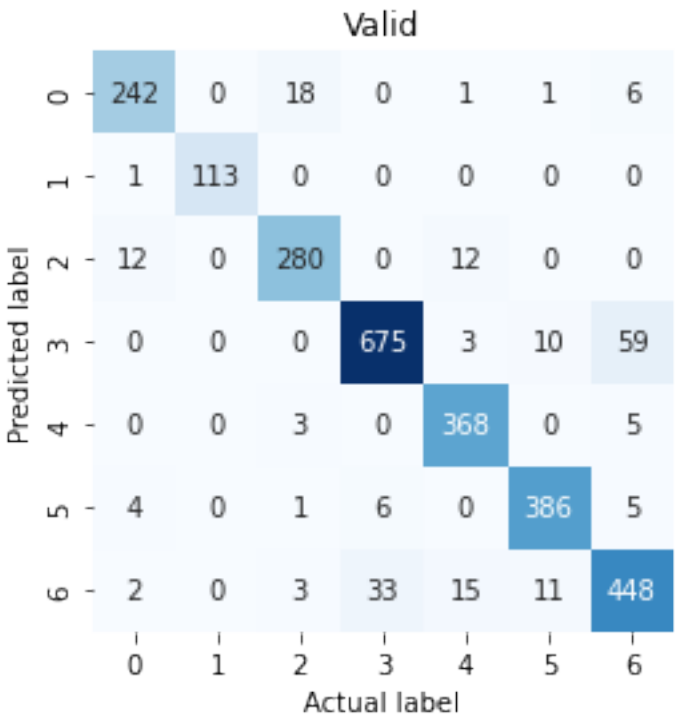
```
from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score, classification_report

target_names = ['BARBUNYA', 'BOMBAY', 'CALI', 'DERMASON', 'HOROZ',
'SEKER','SIRA']
print(classification_report(y_test, Y_pred,
target_names=target_names))
```

```
              precision    recall  f1-score   support

    BARBUNYA       0.90      0.93      0.91       261
      BOMBAY       0.99      1.00      1.00       113
        CALI       0.92      0.92      0.92       305
    DERMASON       0.90      0.95      0.92       714
       HOROZ       0.98      0.92      0.95       399
       SEKER       0.96      0.95      0.95       408
        SIRA       0.88      0.86      0.87       523

   micro avg       0.92      0.92      0.92      2723
   macro avg       0.93      0.93      0.93      2723
weighted avg       0.92      0.92      0.92      2723
```

To evaluate the performance of the random forest model, the results of the testing Classes were compared with the true class (ground truth), since the classification was supervised. As discussed in the report, a range of performance measures were utilised, including precision score, recall score and f1 score.

The values of the accuracy and the F1 score were the same suggesting False positive and False negative results did not impact the accuracy score of the model.

```
import seaborn as sns
rf_cm = confusion_matrix(y_test, Y_pred)
rf_cm_plt=sns.heatmap(rf_cm.T, square=True, annot=True, fmt='d',
cbar=False, cmap="Blues")
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Valid");
```



The plot of the Confusion matrix shows the results of the classification. Class BOMBAY achieved the highest accuracy with 113/114 of the bean varieties correctly predicted, this is to be expected as seen from the boxplots in the exploratory data analysis, the BOMBAY class was very different from the other varieties of bean, especially in area and perimeter-based metrics. Bean variety SIRA had the poorest classification results with only 448/542 of the SIRA beans classified correctly. This is like the results reported in the study, with SIRA bean variety being the most challenging to classify.

Discussion and conclusions

The performance of the random forest model for this study was 92.3%, which was stronger than the decision tree model at 87.9%, and like the KNN classifier at 92.5%. Due to computational time the ability to find a global maximum point in the grid search CV algorithm was not possible, nonetheless a local maximum point was found, near the default hyperparameters given by the sklearn library. The previous study used 10 times cross fold validation, whereas for this study the value of 5 was chosen to speed up computation calculation. Use of more powerful random forest classifiers such as XGBOOST(a hybrid random forest model)(anon, 2023), may also be investigated in future to see if it could achieve a better accuracy than what was achieved with the random forest classifier.

Multi Layer perceptron - Extra task

```
import random
```

```
X = data
#Get Target data
y = data['Class']
print(y)
#Load X Variables into a Pandas Dataframe with columns
X = data.drop(['Class'], axis = 1)
#Normalise data
X = scaler.fit_transform(X)
random.seed(43)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=101)
```

```
0           SEKER
1           SEKER
2           SEKER
3           SEKER
4           SEKER
           ...
13606    DERMASON
13607    DERMASON
13608    DERMASON
13609    DERMASON
13610    DERMASON
Name: Class, Length: 13611, dtype: object
```

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(random_state = 20)
mlp
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto',
beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=20, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

```
# Test default hyperparameters
mlp.fit(X_train,y_train)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto',
beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=20, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

```
from sklearn.metrics import accuracy_score
Y_pred = mlp.predict(X_test)
Y_pred
accuracy_score(Y_pred, y_test)
```

```
0.9214102093279471
```

```
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50)],
    'activation': ['identity', 'logistic', 'tanh', 'relu'],
    'max_iter': [200, 300, 400]
}
mlp = MLPClassifier()

random_search = RandomizedSearchCV(mlp,
param_distributions=param_grid, random_state = 10, n_jobs = -1)

random_search.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=MLPClassifier(activation='relu', alpha=0.0001,
batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'hidden_layer_sizes': [(50,), (100,),
(50, 50), (100, 50)], 'activation': ['identity', 'logistic', 'tanh',
'relu'], 'max_iter': [200, 300, 400]},
          pre_dispatch='2*n_jobs', random_state=10, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

```
print("The best parameters are %s with a score of %0.2f"
      % (random_search.best_params_, random_search.best_score_))
```

```
The best parameters are {'max_iter': 200, 'hidden_layer_sizes': (50,
50), 'activation': 'tanh'} with a score of 0.93
```

```
mlp = MLPClassifier(random_state = 20, max_iter = 400,
hidden_layer_sizes =  (100, 50), activation ='tanh')
mlp.fit(X_train,y_train)
```

```
MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto',
beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
```

```
       hidden_layer_sizes=(100, 50), learning_rate='constant',
       learning_rate_init=0.001, max_iter=400, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=20, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)

mlp.fit(X_train,y_train)
```

```
MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto',
beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100, 50), learning_rate='constant',
       learning_rate_init=0.001, max_iter=400, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=20, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

```
Y_pred = mlp.predict(X_test)
Y_pred
accuracy_score(Y_pred, y_test)
```

```
0.9261843554902681
```

```
#How does the hyperparameter max_iter = 200 (defult setting) effect
accurarcy?
mlp = MLPClassifier(random_state = 20, max_iter = 200,
hidden_layer_sizes =  (100, 50), activation ='tanh')
mlp.fit(X_train,y_train)
```

```
MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto',
beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100, 50), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=20, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

```
from sklearn.metrics import accuracy_score
Y_pred = mlp.predict(X_test)
Y_pred
accuracy_score(Y_pred, y_test)
```

```
0.9283878075651855
```

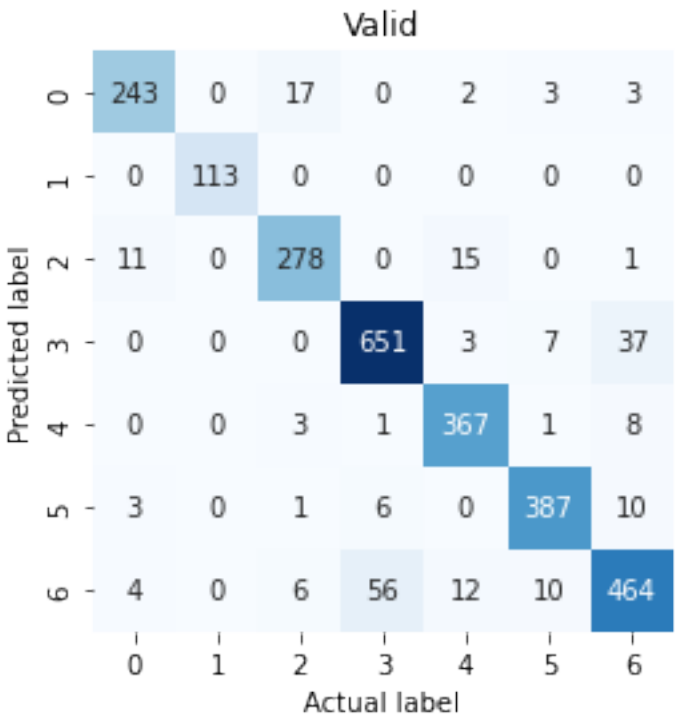
```
#How does the hyperparameter early stopping = True effect accuracy
(Validation fraction = 0.1)?
mlp = MLPClassifier(random_state = 20, max_iter = 200,
hidden_layer_sizes =  (100, 50), activation ='tanh', early_stopping =
True)
mlp.fit(X_train,y_train)
```

```
MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto',
beta_1=0.9,
       beta_2=0.999, early_stopping=True, epsilon=1e-08,
       hidden_layer_sizes=(100, 50), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=20, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)

from sklearn.metrics import accuracy_score
Y_pred = mlp.predict(X_test)
Y_pred
accuracy_score(Y_pred, y_test)

0.9173705471905986

# Hpw does value of alpha effect accuracy?
a = [0.0001, 0.001, 0.01, 0.1, 1, 10]
for i in a:
    mlp = MLPClassifier(random_state = 20, max_iter = 200,
hidden_layer_sizes =  (100, 50), activation ='tanh',alpha = i)
    mlp.fit(X_train,y_train)
    Y_pred = mlp.predict(X_test)
    print(accuracy_score(Y_pred, y_test))

0.9283878075651855
0.927653323540213
0.9203084832904884
0.9192067572530297
0.9155343371281675
0.8229893499816379

mlp = MLPClassifier(random_state = 20, max_iter = 200,
hidden_layer_sizes =  (100, 50), activation ='tanh',alpha = 0.1)
mlp.fit(X_train,y_train)
Y_pred = mlp.predict(X_test)
print(accuracy_score(Y_pred, y_test))

0.9192067572530297

#df = pd.DataFrame({'Predicted': Y_pred, 'True Class': y_test})

# Specify the file path and name for the output XLSX file
#output_file = 'prediction_results_MLP.xlsx'

# Write the DataFrame to an XLSX file
#df.to_excel(output_file, index=False)

from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score, classification_report
```

```
#Evaluation table
target_names = ['BARBUNYA', 'BOMBAY', 'CALI', 'DERMASON', 'HOROZ',
'SEKER','SIRA']
print(classification_report(y_test, Y_pred,
target_names=target_names))
```

```
              precision    recall  f1-score   support

    BARBUNYA       0.91      0.93      0.92       261
      BOMBAY       1.00      1.00      1.00       113
        CALI       0.91      0.91      0.91       305
    DERMASON       0.93      0.91      0.92       714
       HOROZ       0.97      0.92      0.94       399
       SEKER       0.95      0.95      0.95       408
        SIRA       0.84      0.89      0.86       523

   micro avg       0.92      0.92      0.92      2723
   macro avg       0.93      0.93      0.93      2723
weighted avg       0.92      0.92      0.92      2723
```

```
#Martix of correct scores
import seaborn as sns
rf_cm = confusion_matrix(y_test, Y_pred)
rf_cm_plt=sns.heatmap(rf_cm.T, square=True, annot=True, fmt='d',
cbar=False, cmap="Blues")
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Valid");
```



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. (https://www.sciencedirect.com/science/article/pii/S0168169919311573)

2. https://xgboost.readthedocs.io/en/stable/python/python_intro.html (accessed 01/05/2023)