INDEX

INTRODUCTION

Raisins are a good source of carbohydrates and contain various other nutrients like Potassium, fiber, and iron. Raisins help to aid our digestion and reduce stomach issues, they contain tartaric acid Research has shown that this contains anti-inflammatory properties which help in intestinal function and also help to regulate the balance of bacteria in yogurt, another important use of Raisin they help in protecting the skin from sun damage and hydrates the skin from within this also helps in healing the skin tissues

In this project, I am going to discuss the Raisins which are grown in Turkey which Is the top-ranked country in growing grapes apparently Raisins are dried from the grapes when we talk the ratios of about 30% of the grape raw, and another 37% as dried 3% as wine and the rest all other products of the grapes

First, the types of raisins were collected but the original image of the raisin types mentioned in the project which are Besni and Kecimen are mostly one is black and other is white so it is much more difficult to process those images to train using machine learning algorithms so first the images are grey scaled and then they are converted into Binary images, later after using in complementary function black images were converted to white and white are converted to black later the noise is been cleared from those images later the monographical feature inference is applied on the images to

For the understanding purpose the below two are the types of Raisins and also the images added for each type as well

(i) Besni Raisin
(ii) Kecimen Raisin

fig:- Besni Raisin

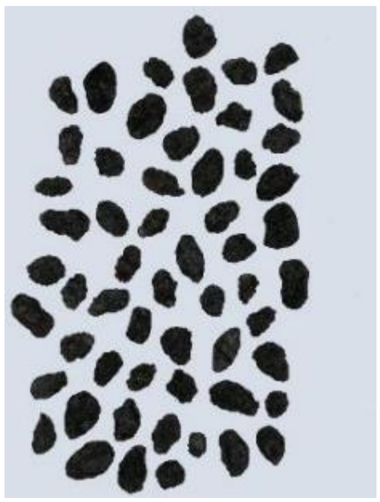


fig:-Kecimen Raisin

There is not much pre-processing work left since the dataset I used for this project has already been completed with image classification, data conversion to numerical features, and inclusion of numerical features rather than categorical features in the data set's excel file. The only thing that is missing is that, despite the numerical nature of the features, each column contains a lot of bias and the values are significantly different from one another, so there will be challenges. This is primarily used as a broad scaling tool for preprocessing.

DATASET

In this section, we will discuss in detail the data set and the features available in my dataset, and some other things

Firstly coming to the dataset I have picked the dataset from the UCI website and below is the link for that dataset

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

speaking further my dataset name is Raisin Dataset and in this data set all the features are in numerical except the label feature which is in the string coming to the detailed explanation of the features in the dataset I have a total of 8 features and below are the names of the features which is a string feature

Features ['Area',' MajorAxisLength', ' MinorAxisLength',' Eccentricity',' ConvexArea',' Extent',' Perimeter', 'class']

Let us discuss what is data available in some of the individual features

Area: Gives the number of pixels within the boundaries of the raisin grain.

MajorAxisLength: This feature gives the length of the main axis, which is the longest line of the raisin grain, simply called the length of the raisin grain.

MinorAxisLength: This feature is the reciprocal of the MinorAxisLength this feature gives the small line of the Raisin, simply called the Width of the raisin grain.

Eccentricity: This feature is basically the radius for the grain the eccentricity of a circle is zero. The eccentricity of an ellipse which is not a circle is greater than zero but less than 1, so when you take a glance at the values of Eccentricity in the Dataset they are below one which means the grains are not circular they are ellipse which is the general shape of the raisin.

ConvexArea: This feature gives the number of pixels that can be used to compute a shape factor known as "convexity" or "solidity", defined as the ratio of the area over the convex area. It can be obtained by using the 'solidity' parameter in region props, giving the grain the smallest region shell.

Extent: Gives the ratio of the region of the total pixel in the box.

Perimeter: Generally perimeter is the length of the outline of the shape, here in my dataset this refers to the length and width of the shape of the raisin grain.

Class: Last but not least this feature is the label in my dataset this feature is generally divided into two parts half of the dataset is given as Kecimen Raisin and the rest half of the records are given as Besni Raisin.

And in my data set feature Class is the Y and the label contains two types of values one is Kecimen and Besni there are a total of 900 columns or records and in this total columns some are divided into Kecimen Raisin and some are divided into Besni Raisin.

The libraries I used inside my project are as follows:

1. Pandas

2. Seaborn

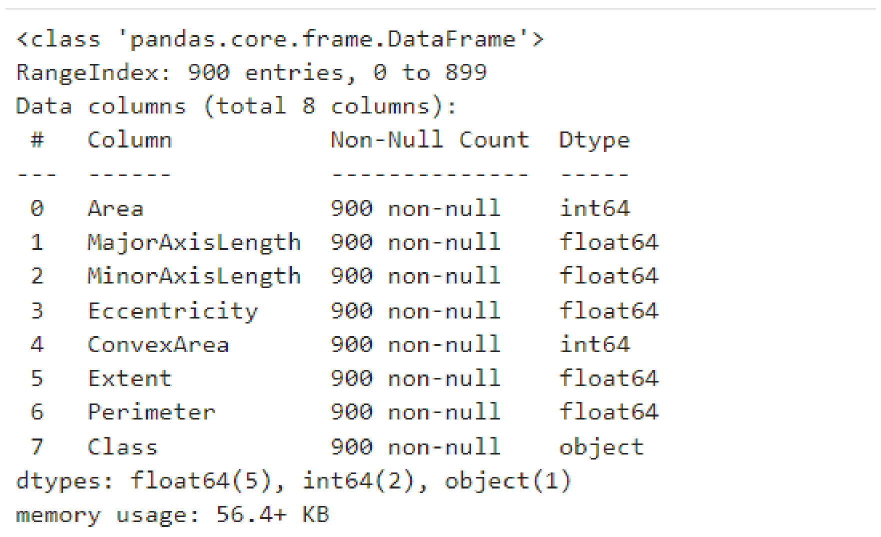
3. Matplotlib

4.StandardScaler

5. Tensorflow

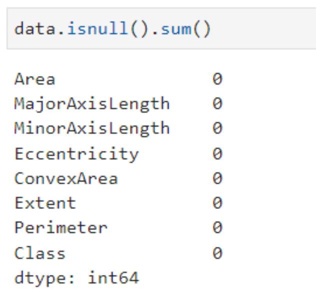
6. Keras

Below is the information on the dataset



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Area             900 non-null    int64  
 1   MajorAxisLength  900 non-null    float64
 2   MinorAxisLength  900 non-null    float64
 3   Eccentricity     900 non-null    float64
 4   ConvexArea       900 non-null    int64  
 5   Extent           900 non-null    float64
 6   Perimeter        900 non-null    float64
 7   Class            900 non-null    object 
dtypes: float64(5), int64(2), object(1)
memory usage: 56.4+ KB
```

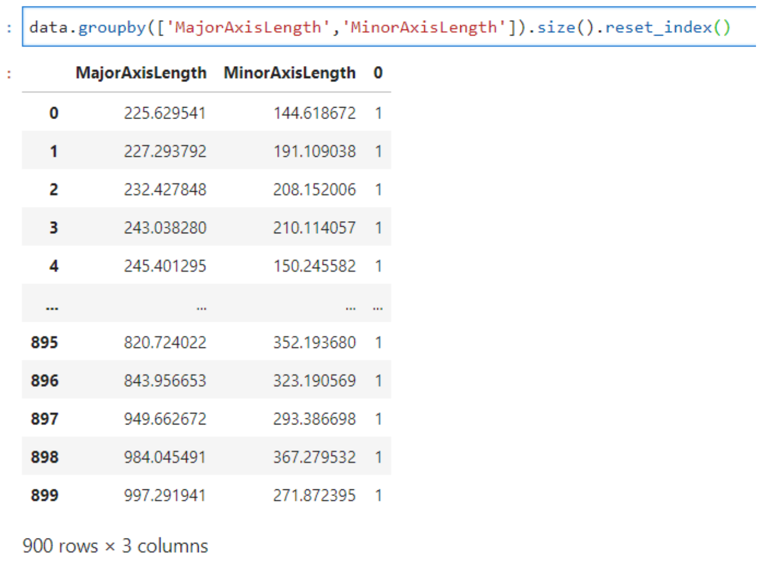
After loading the dataset I checked the availability of the null values in the dataset below is the information on the same



```
data.isnull().sum()
```

```
Area               0
MajorAxisLength    0
MinorAxisLength    0
Eccentricity       0
ConvexArea         0
Extent             0
Perimeter          0
Class              0
dtype: int64
```

As per the above image, there are no null values in my dataset so, I have moved further to check the distribution of the data, the below step is used to display the similar type of data features in the data set.

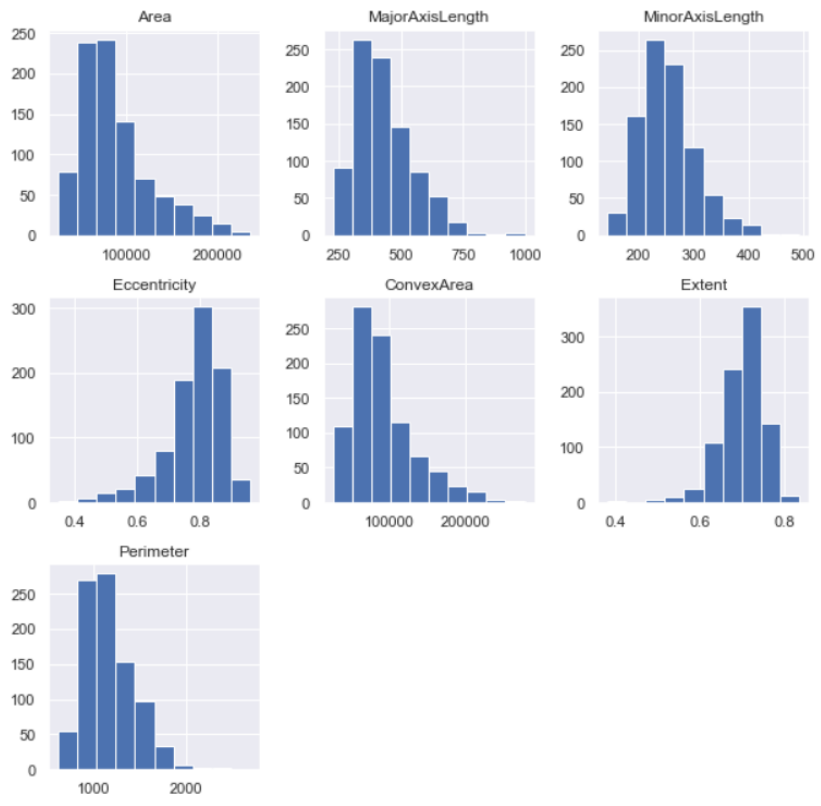


```
data.groupby(['MajorAxisLength','MinorAxisLength']).size().reset_index()
```

	MajorAxisLength	MinorAxisLength	0
0	225.629541	144.618672	1
1	227.293792	191.109038	1
2	232.427848	208.152006	1
3	243.038280	210.114057	1
4	245.401295	150.245582	1
...	...	...	...
895	820.724022	352.193680	1
896	843.956653	323.190569	1
897	949.662672	293.386698	1
898	984.045491	367.279532	1
899	997.291941	271.872395	1

900 rows × 3 columns

Next, the below step is just to check the data distribution for each individual feature other than the label feature



After looking at the data distribution I found that there is much difference in the values for the features from the previous values which will bias the model so I applied a standard scaler on the data set to resize the distribution of values so that the mean of the observed values is 0 and the standard deviation is 1, Below is the steps on the same.

Applying standard scaler

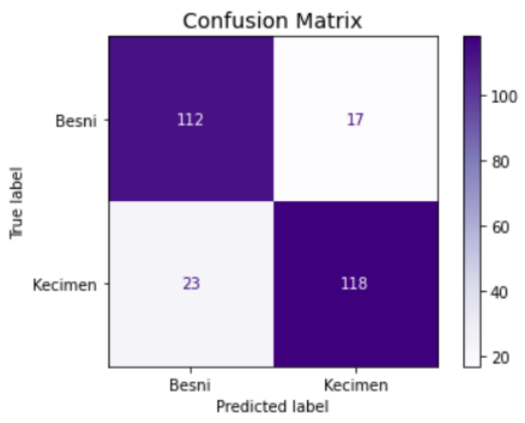
```
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Now let us discuss some visualizations that I have used, I have used a confusion matrix to find the overall accuracies for the models I have used

First, let us see the confusion matrix for the Logistic regression model

Confusion matrix for Logistic regression model

```
#confusion matrix for Logistic regression model
plot_confusion_matrix(binary_reg, X_test, y_test,cmap='Purples')
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```



Experimentally the model correctly predicted the number of correct predictions as true values are

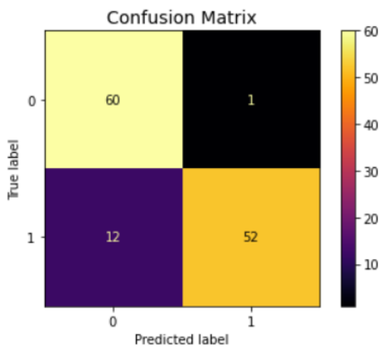
TP=112, whereas the number of predictions where the model correctly predicts the negative

class label as negative is TN=118

Next, we go to check the confusion matrix for SVM model

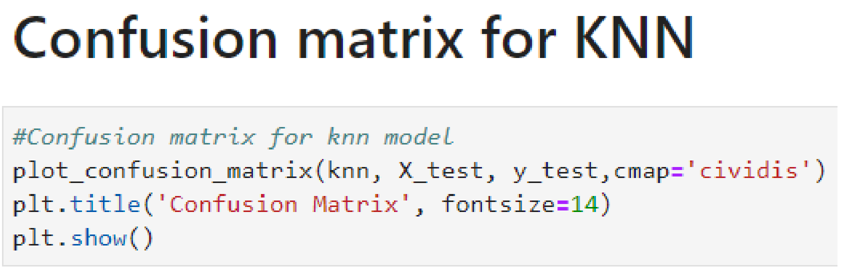
Confusion matrix for SVM model

```
plot_confusion_matrix(svm, X_test, y_test,cmap='inferno')
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```



Experimentally the model correctly predicted the number of correct predictions as true values are TP=60, whereas the number of predictions where the model correctly predicts the negative class label as negative is TN=52

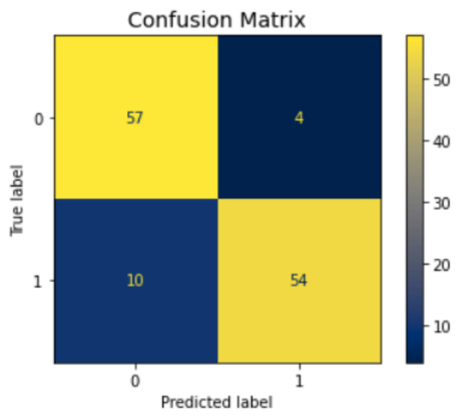
Next, we go to check the confusion matrix for KNN model



Confusion matrix for KNN

```
#Confusion matrix for knn model
plot_confusion_matrix(knn, X_test, y_test,cmap='cividis')
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```

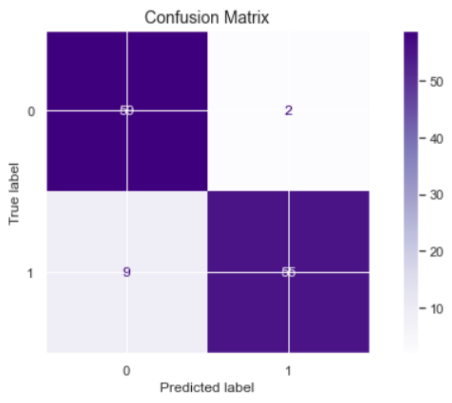
Experimentally the model correctly predicted the number of correct predictions as true values are TP=57, whereas the number of predictions where the model correctly predicts the negative class label as negative is TN=54



Next, we go to check the confusion matrix for the Ensemble model

Confusion matrix for Bagging in Decision tree

```
plot_confusion_matrix(bagging_classifier, X_test, y_test,cmap='Purples')
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```



Experimentally the model correctly predicted the number of correct predictions as true values are

TP=57, whereas the number of predictions where the model correctly predicts the negative

class label as negative is TN=54

As there are no null values in the data so I moved further ahead to split the data set into X and y for X I have used all the features [‘Area’,’ MajorAxisLength’,’ MinorAxisLength’,’ Eccentricity’,’ ConvexArea’,’ Extent’,’ Perimeter’] by dropping the label feature which is Class from the X, for y I have just used the feature ‘Class’ which is also label in my dataset the below is the code for the same

Splitting the dataset into X and y varaibles

```
from sklearn.model_selection import train_test_split
```

```
#Splitting of data into X and y
X = data.drop('Class',axis=1)
y = data['Class']
```

METHOD

Since my model is based on c_lassification I have trained models on the classification models the algorithms which I have used in the project are below

(i)Logistic regression model

(ii)SVM Model

(iii) 'K Nearest Neighbour model

(iv)Decision Tree model

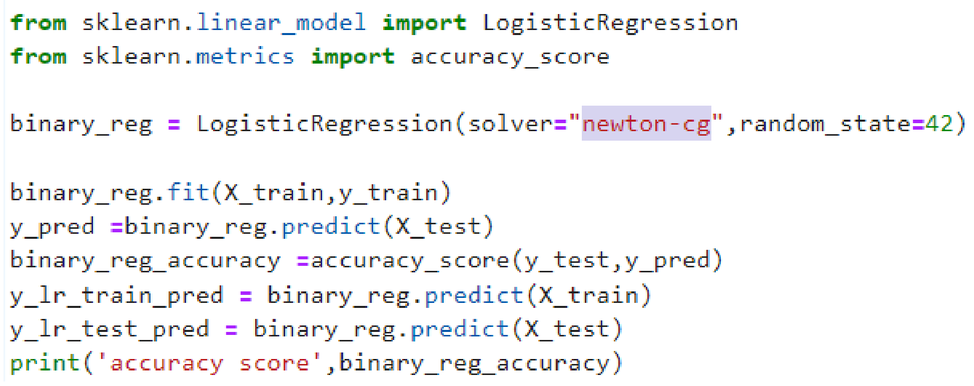
(v)Ensemble learning model

(vi)Neural network (MLP)

(i)Logistic regression model:

I have used Binary regression in the Logistic regression since it predicts binary outcomes such as YES or NO since my label is to predict either Kecimen Raisin or Besni Raisin took the solver as newton-cg to calculate the accuracy of this model I have used the y_test and y_pred as parameters and got the accuracy as 84 percentage.

Below is the code for the same

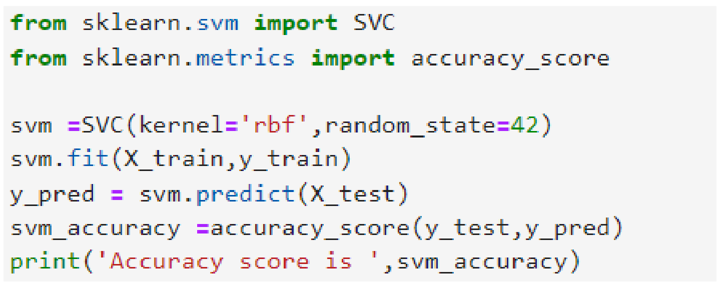


```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

binary_reg = LogisticRegression(solver="newton-cg",random_state=42)

binary_reg.fit(X_train,y_train)
y_pred =binary_reg.predict(X_test)
binary_reg_accuracy =accuracy_score(y_test,y_pred)
y_lr_train_pred = binary_reg.predict(X_train)
y_lr_test_pred = binary_reg.predict(X_test)
print('accuracy score',binary_reg_accuracy)
```

(ii)**SVM Model**:

The SVM model finds the line/hyperplane that separates these two classes the then it classifies the new point depending on whether it lies on the positive side or negative side coming to my case in my data set my feature label name is Class which contains only two types of values to be calculated they are either it has to be Kecimen Raisin or Besni Raisin so this Is the reason I have used SVM model to predict the accuracy of my dataset and I got the 91 per accuracy on this model

Below is the code for the SVM model



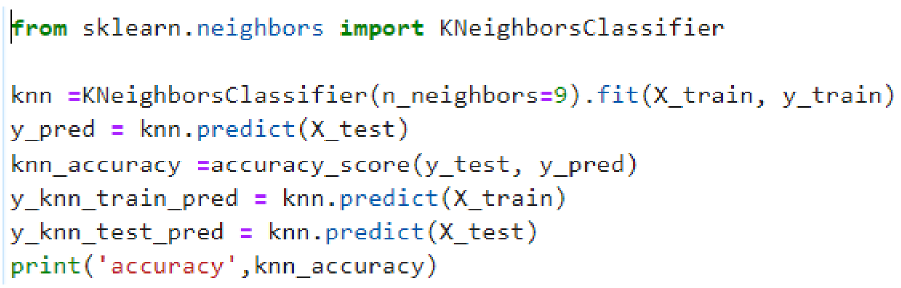
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

svm =SVC(kernel='rbf',random_state=42)
svm.fit(X_train,y_train)
y_pred = svm.predict(X_test)
svm_accuracy =accuracy_score(y_test,y_pred)
print('Accuracy score is ',svm_accuracy)
```

(iii) **K Nearest Neighbour model**:

Knn model is also known as the lazy learning algorithm there are some steps in this algorithm first it selects the number of k neighbors then it calculates the Euclidian distance then it takes that value as knn then among those neighbors counts the number of data points, In each category, then it assigns the new data points to that category for which the number of neighbors is maximum [1]

I used KNN because my data set is small and the data is also properly labeled because my prediction, in the end, is either Besni Raisin or Kecimen Raisin, I got the accuracy of 88 using this model

Below is the code for the KNN model

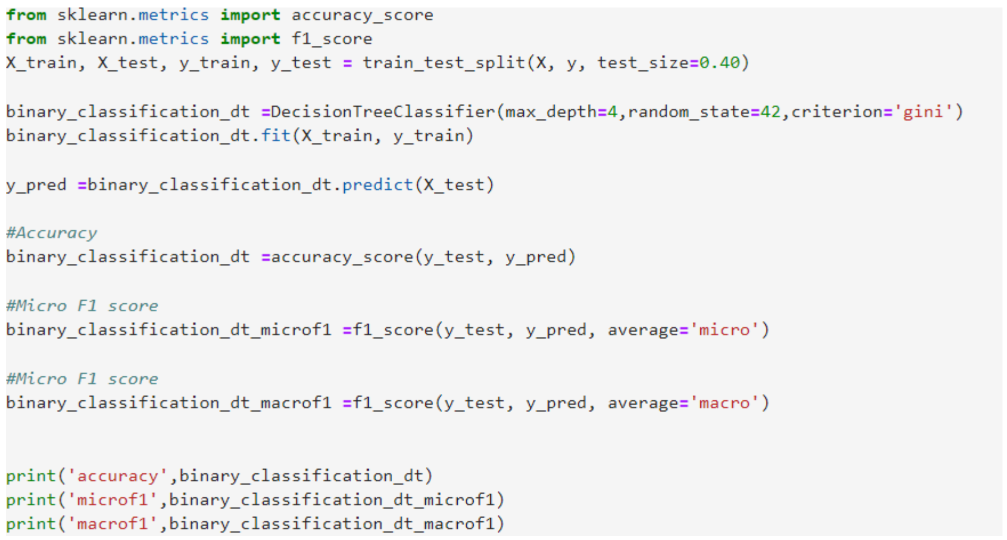


```
from sklearn.neighbors import KNeighborsClassifier

knn =KNeighborsClassifier(n_neighbors=9).fit(X_train, y_train)
y_pred = knn.predict(X_test)
knn_accuracy =accuracy_score(y_test, y_pred)
y_knn_train_pred = knn.predict(X_train)
y_knn_test_pred = knn.predict(X_test)
print('accuracy',knn_accuracy)
```

(iv)**Decision Tree model**:

A decision tree is the most powerful and popular classification prediction it is a graphical representation of all possible decisions based on certain conditions in this model we basically start with the root of the tree and we compare values of the root attribute with the record attributes on comparison basis we follow that branch and then jump to next node coming to my model this decision tree root node is y label which is Class feature which is either Besni or Kecimen raisin then coming to its child nodes this take the other features from the X And then at the end of the tree this will show either this is Besni Raisin or Kecimen Raisin, from the below code I got the accuracy as 88 %

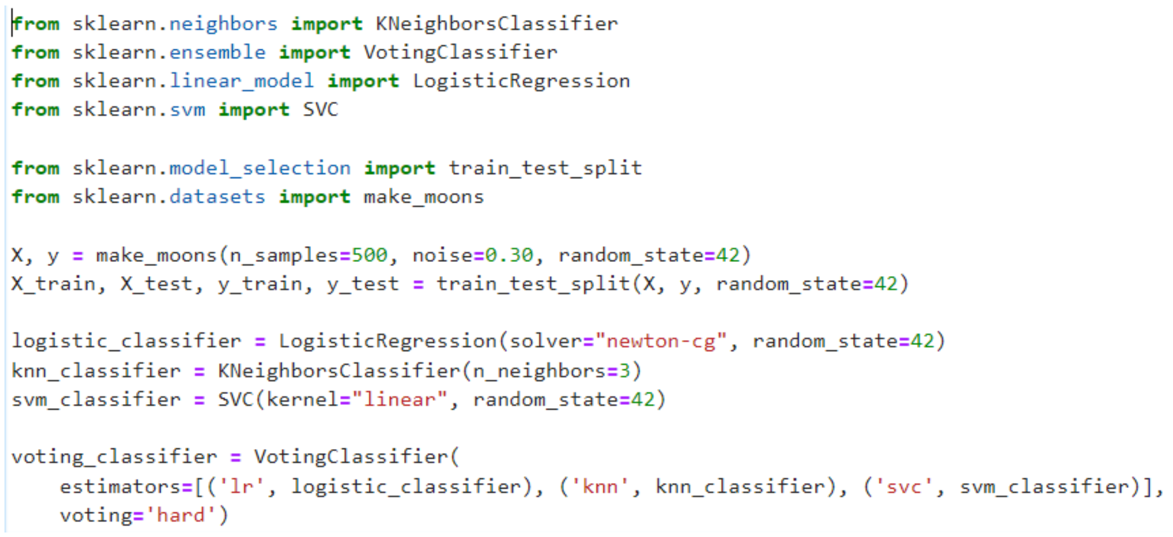
Below is the code for the same



```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40)

binary_classification_dt =DecisionTreeClassifier(max_depth=4,random_state=42,criterion='gini')
binary_classification_dt.fit(X_train, y_train)

y_pred =binary_classification_dt.predict(X_test)

#Accuracy
binary_classification_dt =accuracy_score(y_test, y_pred)

#Micro F1 score
binary_classification_dt_microf1 =f1_score(y_test, y_pred, average='micro')

#Micro F1 score
binary_classification_dt_macrof1 =f1_score(y_test, y_pred, average='macro')


print('accuracy',binary_classification_dt)
print('microf1',binary_classification_dt_microf1)
print('macrof1',binary_classification_dt_macrof1)
```

(v)**Ensemble learning**:

In this model I compared the three models together they are KNN, Logistic regression, and SVM and I got the accuracies for logistic regression which is 86% and for KNN is 89%, and for SVC it is 85% below is the code for the same



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

logistic_classifier = LogisticRegression(solver="newton-cg", random_state=42)
knn_classifier = KNeighborsClassifier(n_neighbors=3)
svm_classifier = SVC(kernel="linear", random_state=42)

voting_classifier = VotingClassifier(
    estimators=[('lr', logistic_classifier), ('knn', knn_classifier), ('svc', svm_classifier)],
    voting='hard')
```

(vi)**Neural network (MLP):**

There are many neural networks models among those MLP is the widely used one the sequence of neurons in MLP is in layers along with those there are many hidden layers also available inside the MLP there are two main layers one is the Input layer and another one is the Output layer, the input layer contains the information of the problem that needs to be predicted the output layer is the layer where the output of the information is present inside of it, The parameters that I used inside the MLP are Optimizer, Loss, metrics, Batch size, Validation split, Epochs let us discuss on these parameters in detail

Validation_split: Validation split helps to improve the model performance by fine-tuning the model after each epoch. The test set informs us about the final accuracy of the model after completing the training phase. In my model, I took the Validation_split as 0.2 which worked well with my dataset to get good accuracies.

Loss: A loss function is a function that compares the target and predicted output values, and measures how well the neural network models the training data. When training, we aim to minimize this loss between the predicted and target outputs. Generally, if your data is Categorical data you can use categorical_crossentropy, since my data is numerical so I have used sparse_categorical_crossentropy as my loss.

Batch_size: The batch size is the number of samples that will be passed through to the network at one time.

Optimizer: An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate. Thus, it helps in reducing the overall loss and improving accuracy. In my model, I have given the optimizer as adam because why not after using this optimizer for 100 epochs for the first epoch my loss was 2.8 something later at the end of the epoch it came down to 0.8 which is a great change according to me else I could have used SGD for the optimizer.

Below is the code for the same

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
mlp = Sequential([

      # dense layer 1
    Dense(256, activation='relu'),

    # dense layer 2
    Dense(128, activation='relu'),

     # dense layer 3
    Dense(128, activation='relu'),

     # dense layer 4
    Dense(128, activation='relu'),

      # output layer
    Dense(10, activation='sigmoid'),
])
```

```
mlp.compile(optimizer='adam',
      loss='sparse_categorical_crossentropy',
      metrics=['accuracy'])
```

```
raisin_net = mlp.fit(X_train, y_train, epochs=50,
          batch_size=1000,
          validation_split=0.2)
```

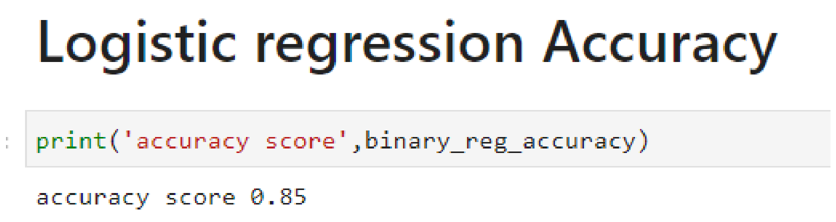
The above code raisin_net is used to get the accuracy for the MLP

EXPERIMENTAL RESULTS

In this Section, I would like to discuss the results I obtained on training each model the models I used in my project are Logistic regression, K Nearest Neighbours, Decision Tree, SVM, and Neural networks other than these models I have also used some others like Ensemble learning for three used models like SVM, K Nearest Neighbours, Logistic regression to test the combined accuracy and also Bagging in the Decision tree the below are the accuracies obtained for each model

(i)Accuracy for Logistic Regression:

For Logistic regression, I got an accuracy of 85 below is the snap for the same

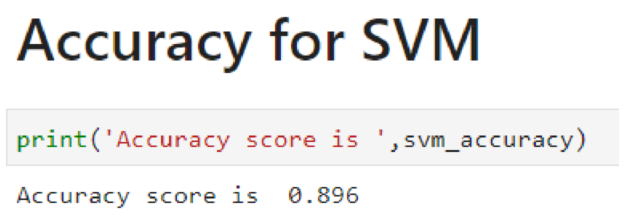

Logistic regression Accuracy

```
print('accuracy score',binary_reg_accuracy)
```

```
accuracy score 0.85
```

(ii)Accuracy for SVM Model:

For the SVM model, I got an accuracy of 89% and below is the snap for the same

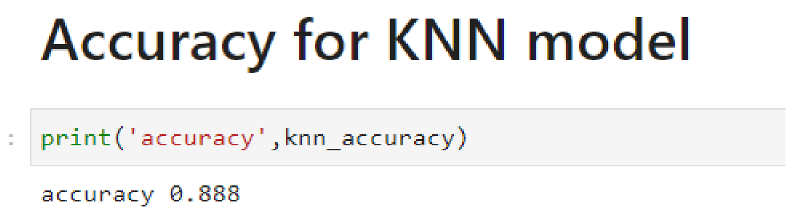

Accuracy for SVM

```
print('Accuracy score is ',svm_accuracy)
```

```
Accuracy score is  0.896
```

(iii)Accuracy for KNN Model:

For the KNN model, I got an accuracy of 88%, and below is the snap for the same

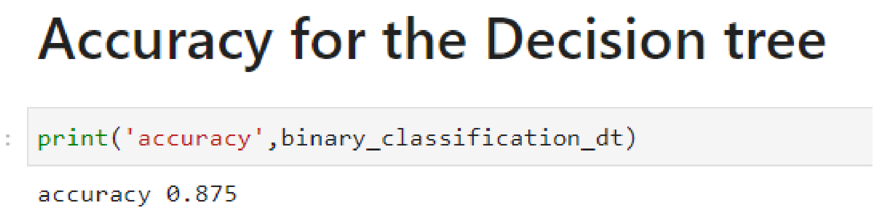

Accuracy for KNN model

```
print('accuracy',knn_accuracy)
```

```
accuracy 0.888
```

(iv)Accuracy for Decision tree Model:

For the Decision tree, I got an accuracy of 87%, and below is the snap for the same

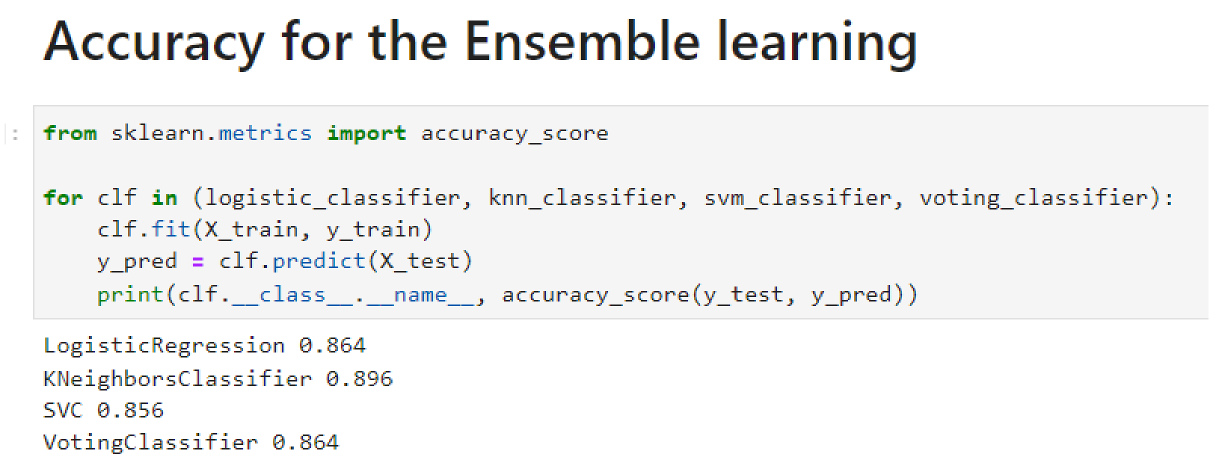

Accuracy for the Decision tree

```
print('accuracy',binary_classification_dt)
```

```
accuracy 0.875
```

Accuracy for Ensemble Learning:

For Ensemble learning, I have a voting classifier for checking the accuracies for three models combining Linear Regression, KNN, and SVC which Is the classifier in the SVM model I got the accuracy obtained in combined is 86% Which is pretty much decent combining all three models



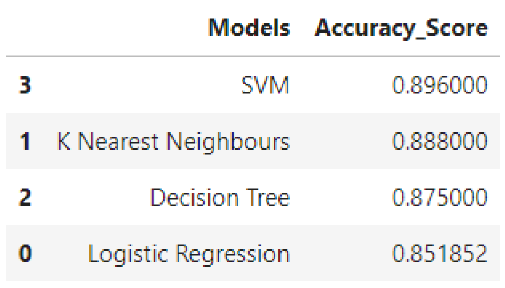
Accuracy for the Ensemble learning

```
from sklearn.metrics import accuracy_score

for clf in (logistic_classifier, knn_classifier, svm_classifier, voting_classifier):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.864
KNeighborsClassifier 0.896
SVC 0.856
VotingClassifier 0.864
```

Now let us talk about the visualization of the data

ANALYSIS FROM RESULT

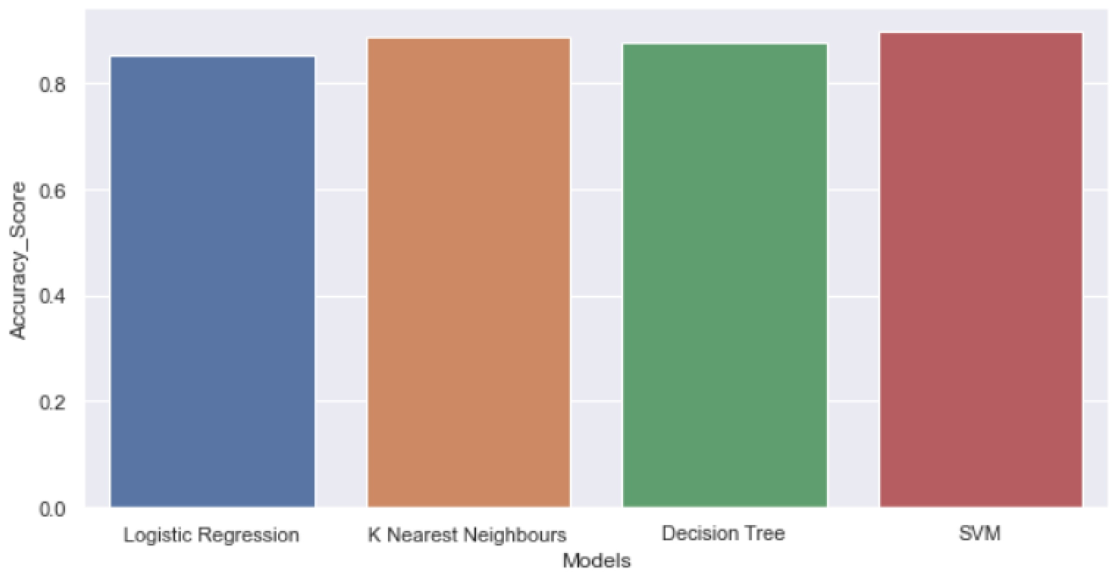


	Models	Accuracy_Score
3	SVM	0.896000
1	K Nearest Neighbours	0.888000
2	Decision Tree	0.875000
0	Logistic Regression	0.851852

From the above table, I can say that SVM got the highest accuracy of 89%, and the second highest accuracy is the KNN model with second highest accuracy which is 88%, in third place comes the Decision tree with 87% accuracy, and finally come to the Logistic regression with 85% accuracy

Here I can say SVM performed best with my dataset whereas Logistic regression works worst with my data set

Below is the heat map graph on all the accuracies



CHALLENGES AND FUTURE STEPS

There are no difficulties because the dataset has already been pre-processed and is numerical; nevertheless, because the data is different from the data set, I have scaled the data to make good sense. Forecasts for future actions If we had included additional varieties of raisins in the dataset, there might have been issues with the data as well as other factors. My data set only includes the two types of raisins that are grown in Turkey. Since I haven't performed any data pre-processing processes like image classification, there may be issues in the dataset if there are any string features. In the future, I'll try to do image classification in order to convert the dataset.

CONCLUSION

In conclusion, I have used a total of 5 machine learning algorithms for the Raisins dataset Performance measurements were made using statistical data derived from the confusion matrix derived from the classification outcome. With regard to average classification accuracy, the SVM method with regard to average classification accuracy, the SVM method algorithm with 89.6% and also To classify, calibrate, or employ in various stages of processing, automatic systems can be constructed using the data sets that have been created.

REFERENCES:

CINAR I., KOKLU M., and TASDEMIR S., (2020), Classification of Raisin Grains Using Machine Vision and Artificial Intelligence Methods. Gazi Journal of Engineering Sciences, vol. 6, no. 3, pp. 200-209, December 2020. DOI: https://dergipark.org.tr/tr/download/article-file/1227592

[1] https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning