PREDICTING THE RAISIN TYPE :

KECIMEN OR BESNI

LNMIIT
The LNM Institute of
Information Technology

TABLE OF CONTENTS

Problem statement

We have collected our dataset from the given website, and the following steps have been performed on it:

Link to the dataset: UCI Machine Learning Repository: Raisin Dataset Data Set

1. Data pre-processing and its visualisation
2. Explain all the inferences we got from our data.
3. Explain what ML Classification Algorithms are being used and why
4. Implementing those algorithms
5. Output the result of the testing set and its visualisation

All of the tasks are performed with the help of pre-existing Python Libraries such as :

- matplotlib
- seaborn
- numpy
- Pandas
- Scikit_learn

Introduction to the dataset

Images of Kecimen and Besni raisin varieties grown in Turkey were obtained with CVS. A total of 900 raisin grains were used, including 450 pieces from both varieties. These images were subjected to various preprocessing stages, and 7 morphological features were extracted. These features have been classified using three different artificial intelligence techniques and based on them we will be training a model which will be able to predict the raisin type of new test data.

Our dataset consists of :

❖ Data Set Characteristics: Multivariate

❖ Area: Life

❖ Number of instances: 900

❖ Number of Attributes: 8

❖ Attribute characteristics: Integers, Real

❖ Associated Tasks: Classification, Regression

❖ Missing Values: No

❖ Date Donated: 2021-04-01

Attribute Description:

1.) **Area**: Gives the number of pixels within the boundaries of the raisin.

2.) **Perimeter**: It measures the environment by calculating the distance between the boundaries of the raisin and the pixels around it.

3.) **MajorAxisLength**: Gives the length of the main axis, which is the longest line that can be drawn on the raisin.

4.) **MinorAxisLength**: Gives the length of the small axis, which is the shortest line that can be drawn on the raisin.

5.) **Eccentricity**: It gives a measure of the eccentricity of the ellipse, which has the same moments as raisins.

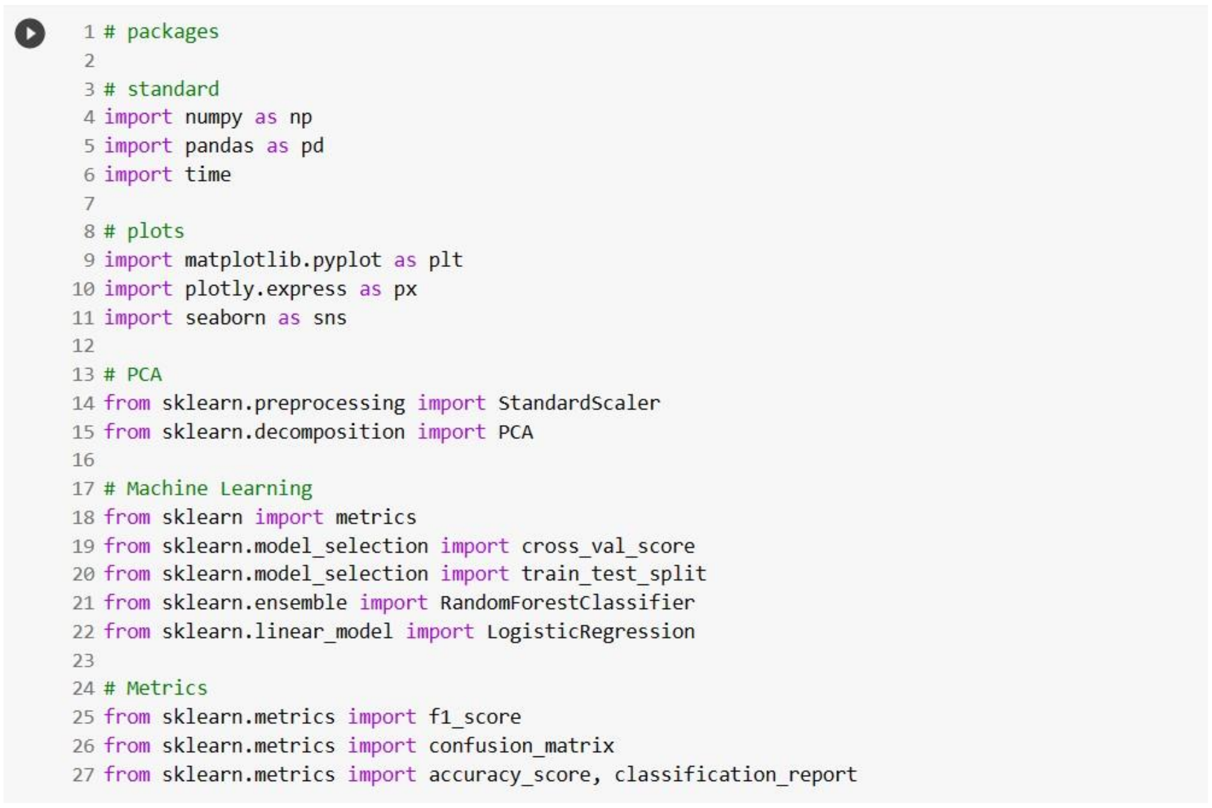
6.) **ConvexArea**: Gives the number of pixels of the smallest convex shell of the region formed by the raisin.

7.) **Extent**: Gives the ratio of the region formed by the raisin to the total pixels in the bounding box.

8.) **Class**: Kecimen and Besni raisin.

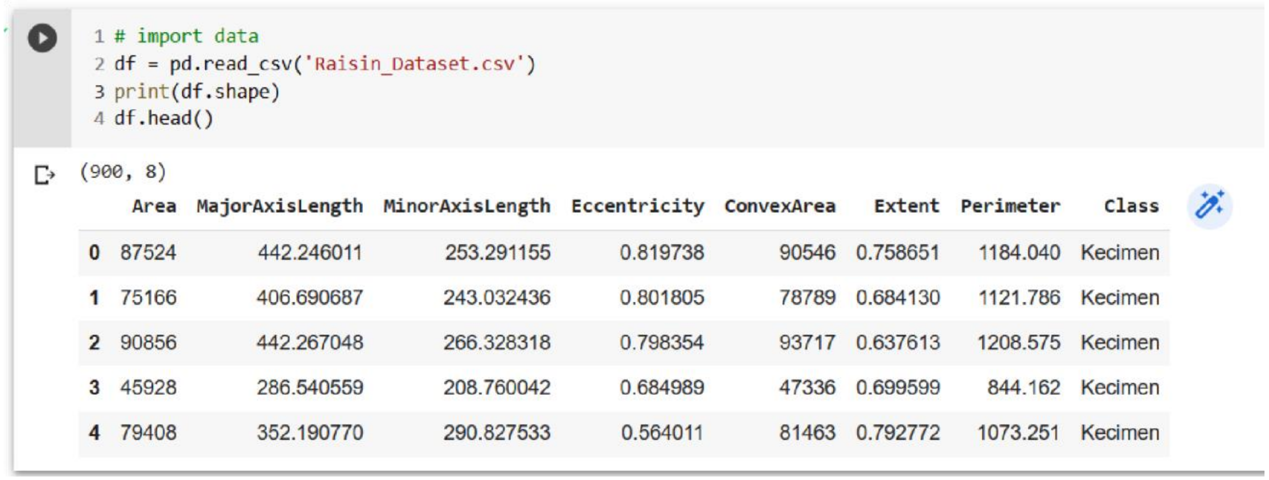
Data Analysis:

1) Importing all the Python libraries required in the code.



```python
# packages

# standard
import numpy as np
import pandas as pd
import time

# plots
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns

# PCA
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Machine Learning
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

# Metrics
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
```

2) Next, we read the dataset in a variable and output its first 5 rows, using pandas.read_csv()

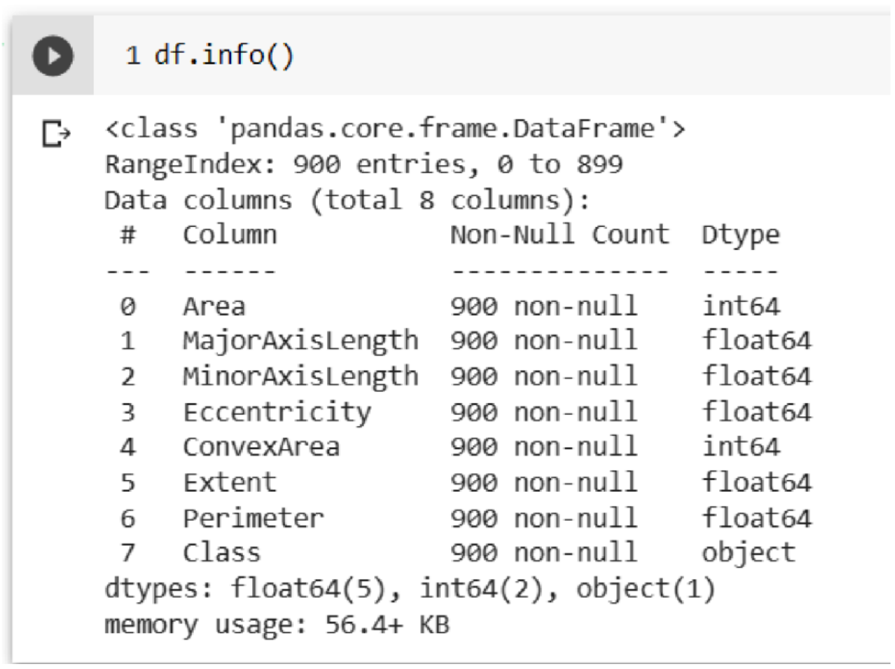


```python
# import data
df = pd.read_csv('Raisin_Dataset.csv')
print(df.shape)
df.head()
```

```
(900, 8)
```

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
0	87524	442.246011	253.291155	0.819738	90546	0.758651	1184.040	Kecimen
1	75166	406.690687	243.032436	0.801805	78789	0.684130	1121.786	Kecimen
2	90856	442.267048	266.328318	0.798354	93717	0.637613	1208.575	Kecimen
3	45928	286.540559	208.760042	0.684989	47336	0.699599	844.162	Kecimen
4	79408	352.190770	290.827533	0.564011	81463	0.792772	1073.251	Kecimen

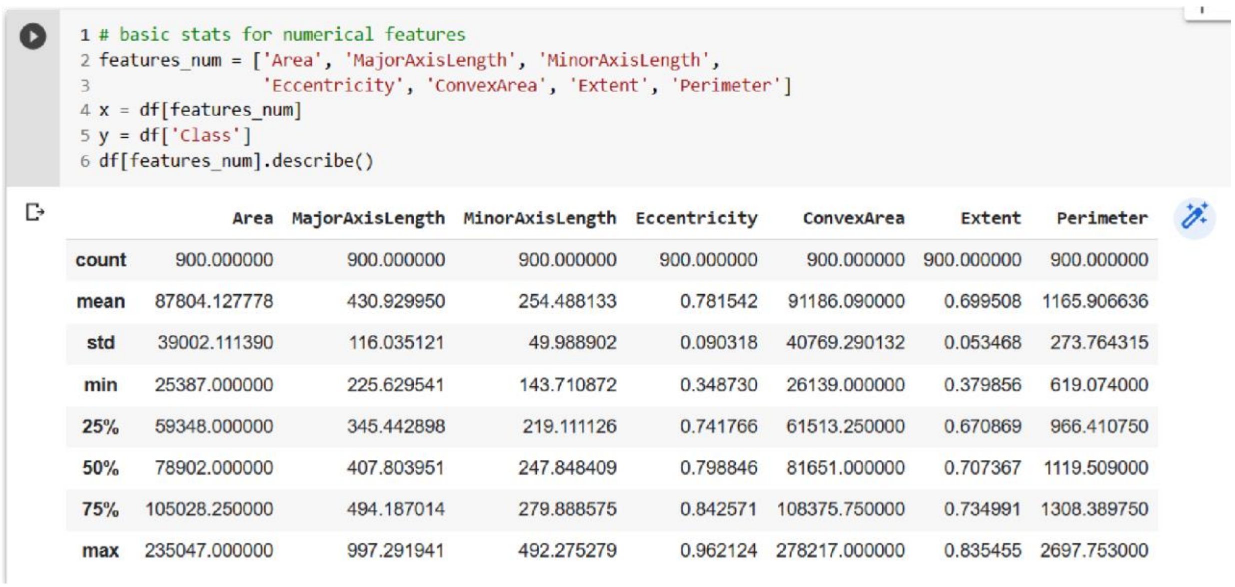
3) Using the DataFrame.info(), we find the information about our dataset, i.e., how many attributes are there, the datatype of each attribute, and whether is there any null value to it or not



```python
df.info()
```

```
<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
```

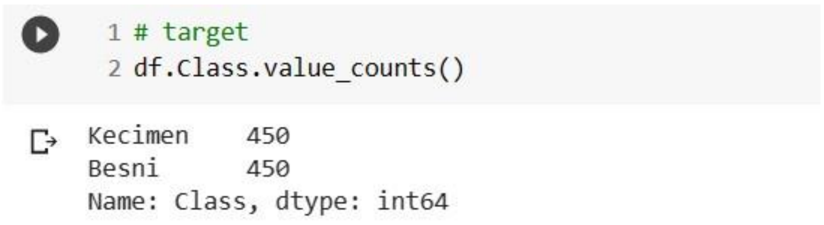
4) We also use DataFrame.describe( ), which is used to view basic, statistical details about our dataset such as min-max values, standard, deviation, mean, etc.



```python
# basic stats for numerical features
features_num = ['Area', 'MajorAxisLength', 'MinorAxisLength',
                'Eccentricity', 'ConvexArea', 'Extent', 'Perimeter']
x = df[features_num]
y = df['Class']
df[features_num].describe()
```

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter
count	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000
mean	87804.127778	430.929950	254.488133	0.781542	91186.090000	0.699508	1165.906636
std	39002.111390	116.035121	49.988902	0.090318	40769.290132	0.053468	273.764315
min	25387.000000	225.629541	143.710872	0.348730	26139.000000	0.379856	619.074000
25%	59348.000000	345.442898	219.111126	0.741766	61513.250000	0.670869	966.410750
50%	78902.000000	407.803951	247.848409	0.798846	81651.000000	0.707367	1119.509000
75%	105028.250000	494.187014	279.888575	0.842571	108375.750000	0.734991	1308.389750
max	235047.000000	997.291941	492.275279	0.962124	278217.000000	0.835455	2697.753000

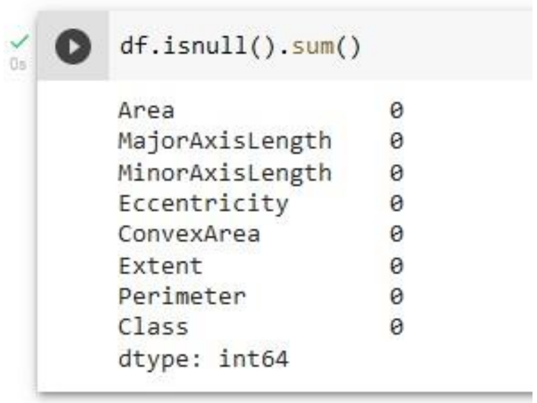
5) We have used the variable “class” as the target column and added it to the data frame. We’ve used 2 labels (“Kecimen”, “Besni”) to assign our Raisin Type.



```
# target
df.Class.value_counts()
```

```
Kecimen     450
Besni       450
Name: Class, dtype: int64
```

6) We use DataFrame.isnull().sum() to check if there are any null values in the dataset or not.



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

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

Inference:

In our dataset, we can observe the following things:

○ All of our data is numeric.

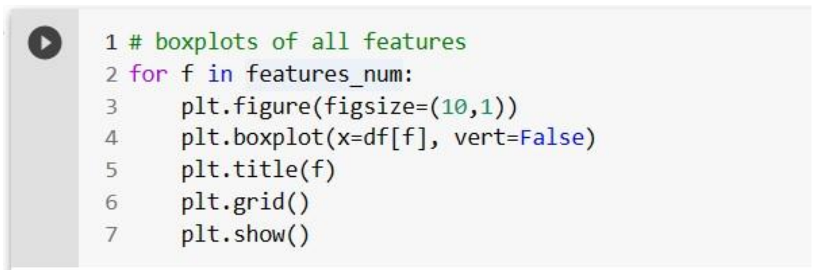
○ No missing or null values are present in our dataset.

○ There isn't any unnecessary attribute in our dataset, that we need to remove before we start visualisation.

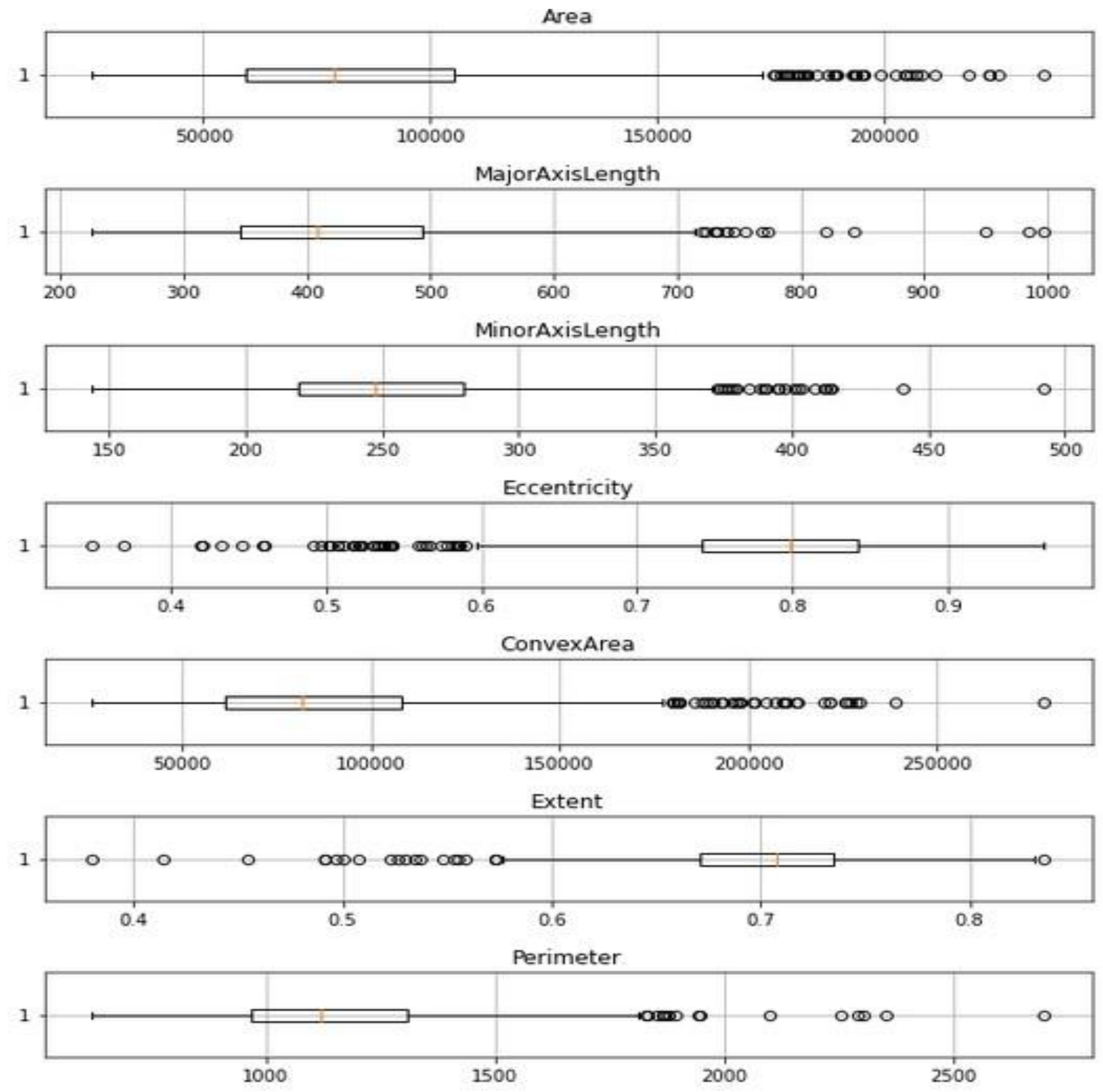
Data Visualisation:

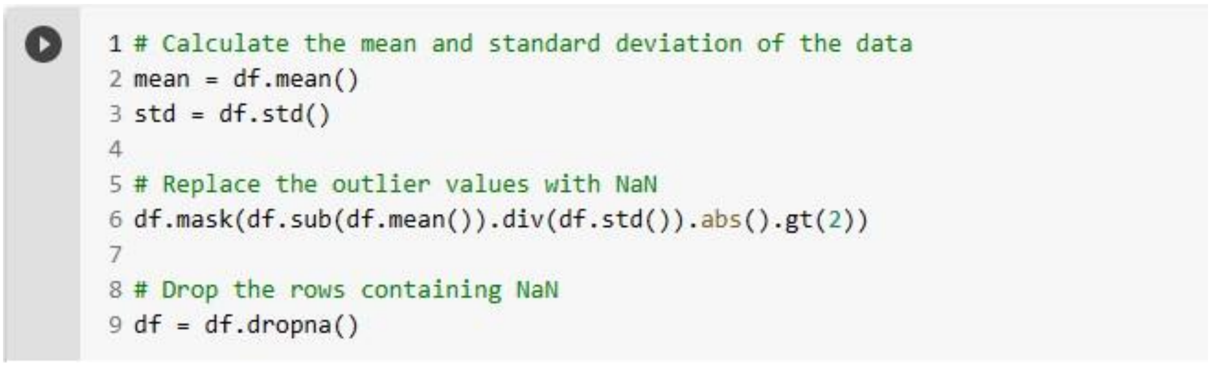
Box Plots

For every feature present in our dataset, we plot a box plot for each one of them.



```
# boxplots of all features
for f in features_num:
    plt.figure(figsize=(10,1))
    plt.boxplot(x=df[f], vert=False)
    plt.title(f)
    plt.grid()
    plt.show()
```




```python
# Calculate the mean and standard deviation of the data
mean = df.mean()
std = df.std()

# Replace the outlier values with NaN
df.mask(df.sub(df.mean()).div(df.std()).abs().gt(2))

# Drop the rows containing NaN
df = df.dropna()
```

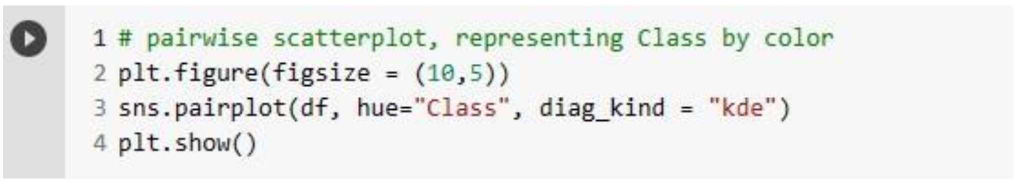
Inference:

- Box plot is used in explanatory data analysis.
- Box plots visually show the distribution of numerical data and skewness through displaying the data quartiles (or percentiles) and averages.

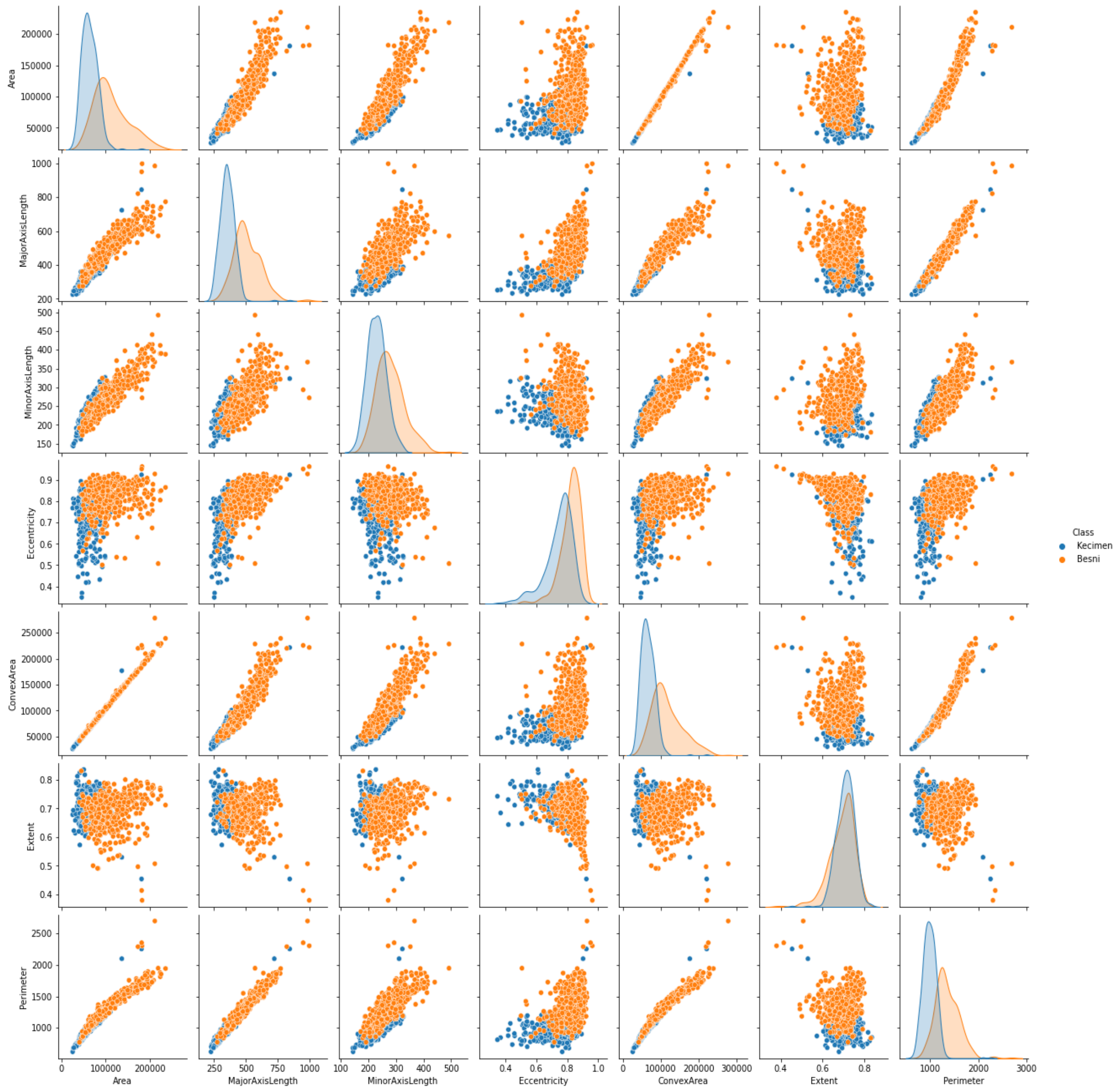
- It can be used to see outliers and also minimum and maximum range of attributes.
- By seeing the outliers, we have tried to handle them and substituted all the outliers with NaN and dropped the respective rows of data.

Pair Plots:

Pairplot visualises given data to find the relationship between them where the variables can be continuous or categorical. The pair plot function creates a grid of Axes such that each variable in data will be shared on the y-axis across a single row and on the x-axis across a single column.



```
# pairwise scatterplot, representing Class by color
plt.figure(figsize = (10,5))
sns.pairplot(df, hue="Class", diag_kind = "kde")
plt.show()
```



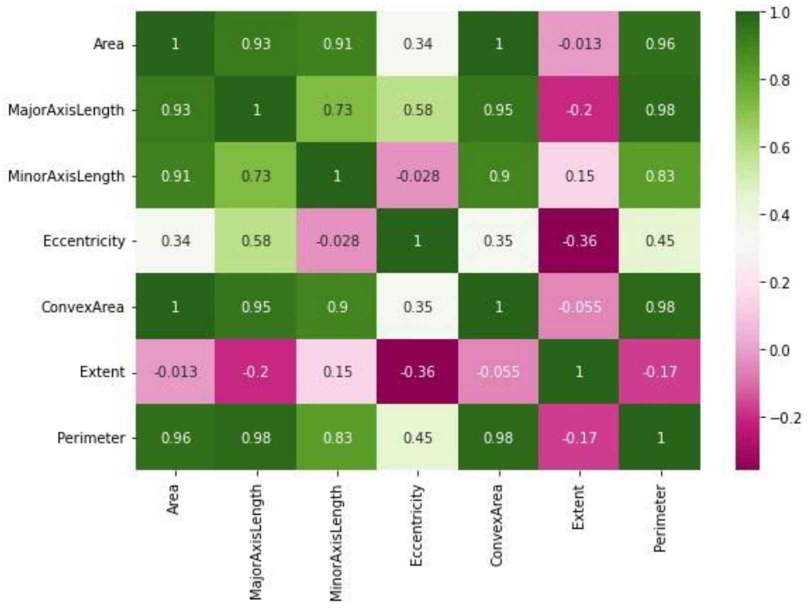
Inference:

- The blue squares represent “Kecimen” and the orange squares represent “Besni”.
- Each scatter plot can be studied thoroughly to gain more insights into attributes relationship

Correlation between Attributes:

We aim to find the pairwise correlation of columns.

```
# Let's see the correlation values on a heatmap

cor = df.corr()
plt.figure(figsize=(9,6))
sns.heatmap(data = cor, annot = True, cmap = 'PiYG')
plt.show()
```

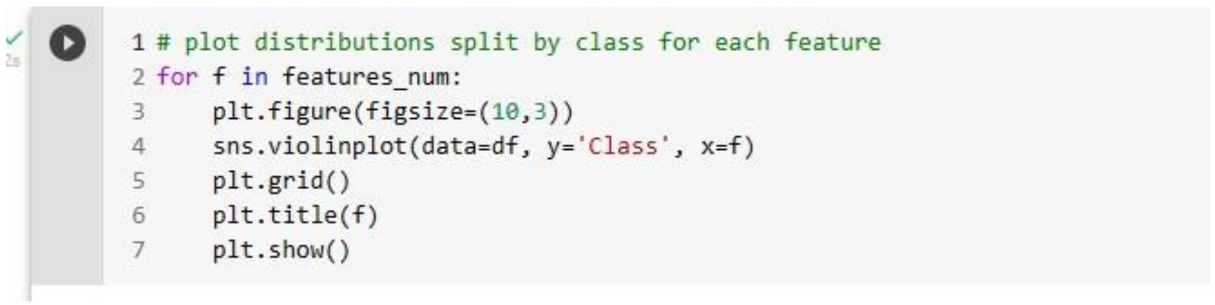


	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter
Area	1	0.93	0.91	0.34	1	-0.013	0.96
MajorAxisLength	0.93	1	0.73	0.58	0.95	-0.2	0.98
MinorAxisLength	0.91	0.73	1	-0.028	0.9	0.15	0.83
Eccentricity	0.34	0.58	-0.028	1	0.35	-0.36	0.45
ConvexArea	1	0.95	0.9	0.35	1	-0.055	0.98
Extent	-0.013	-0.2	0.15	-0.36	-0.055	1	-0.17
Perimeter	0.96	0.98	0.83	0.45	0.98	-0.17	1

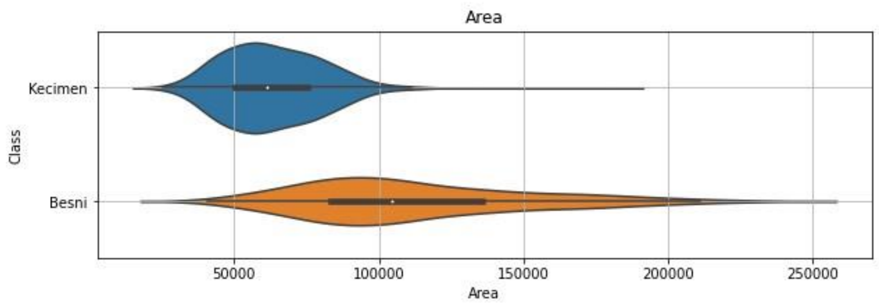
Inference:

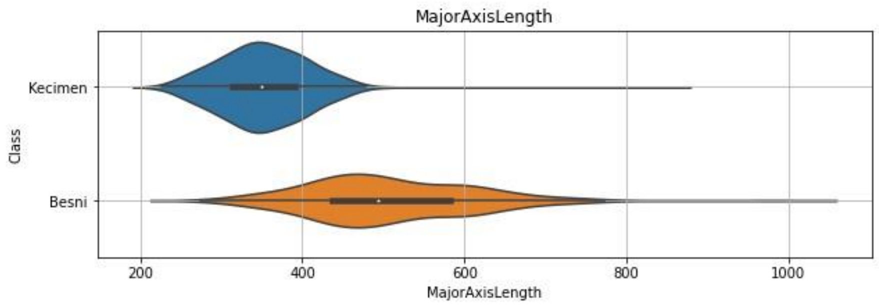
- Correlation t-test are done to test if two variables are linearly correlated, or in other words, whether there is a linear relationship between the two variables
- Correlation shows the strength of a relationship between two variables and is expressed numerically by the correlation coefficient. The correlation coefficient's values range between -1.0 and 1.0. A perfect positive correlation means that the correlation coefficient is exactly 1.

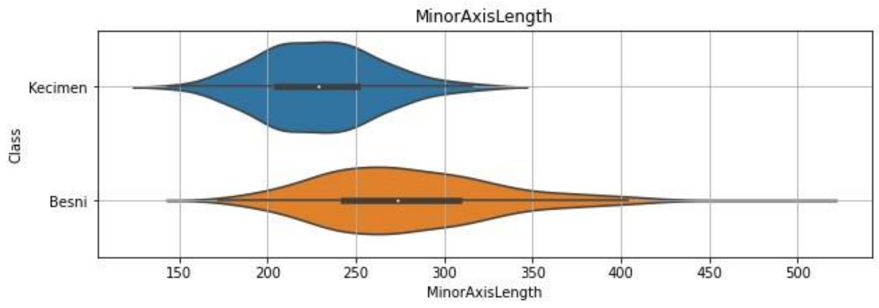
Feature Distribution Across Classes (Kecimen and Besni):

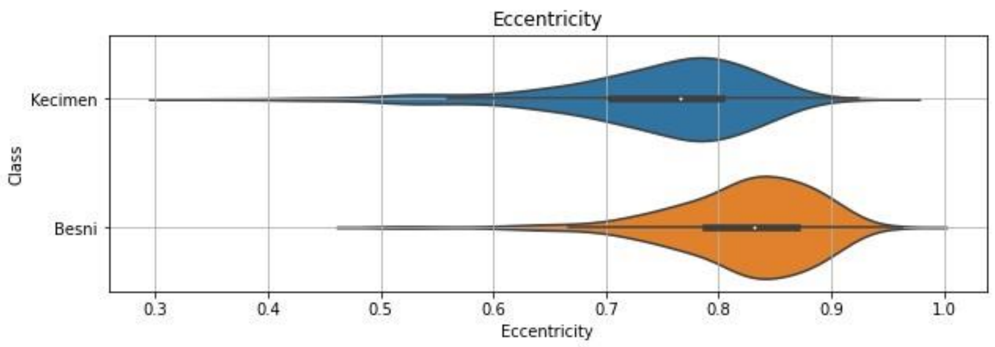


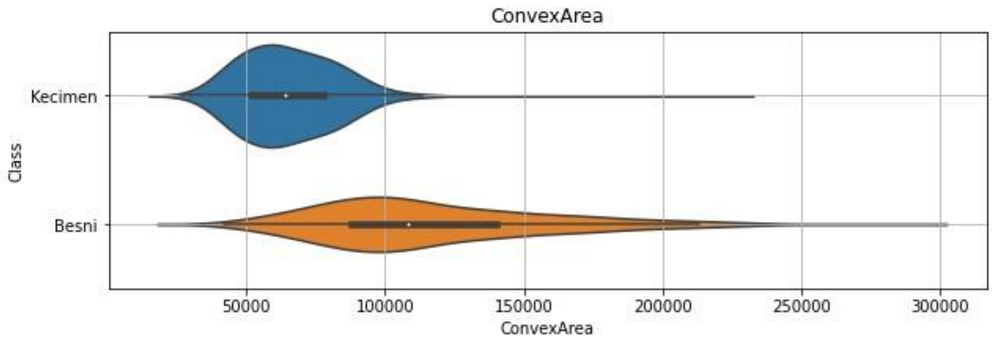
```python
# plot distributions split by class for each feature
for f in features_num:
    plt.figure(figsize=(10,3))
    sns.violinplot(data=df, y='Class', x=f)
    plt.grid()
    plt.title(f)
    plt.show()
```

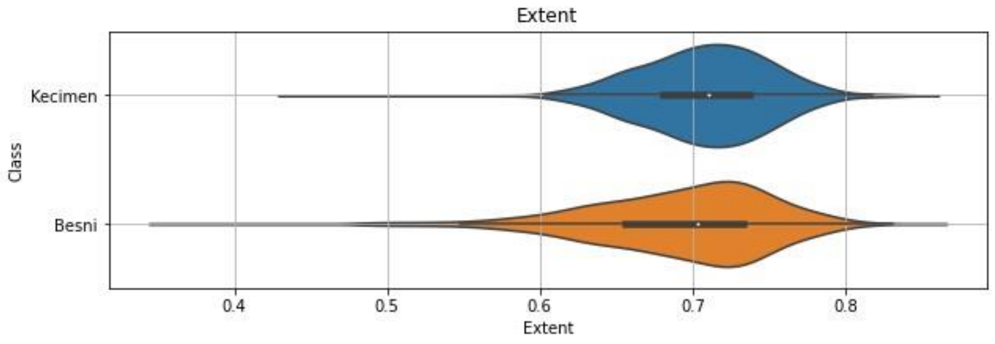


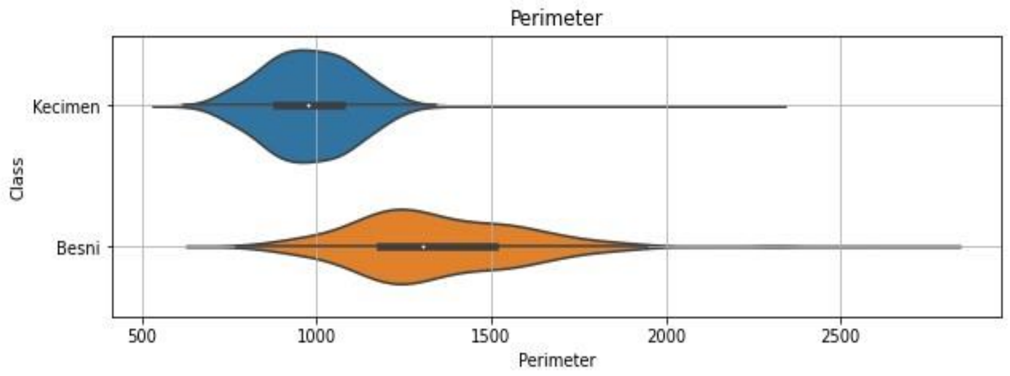












Inference:

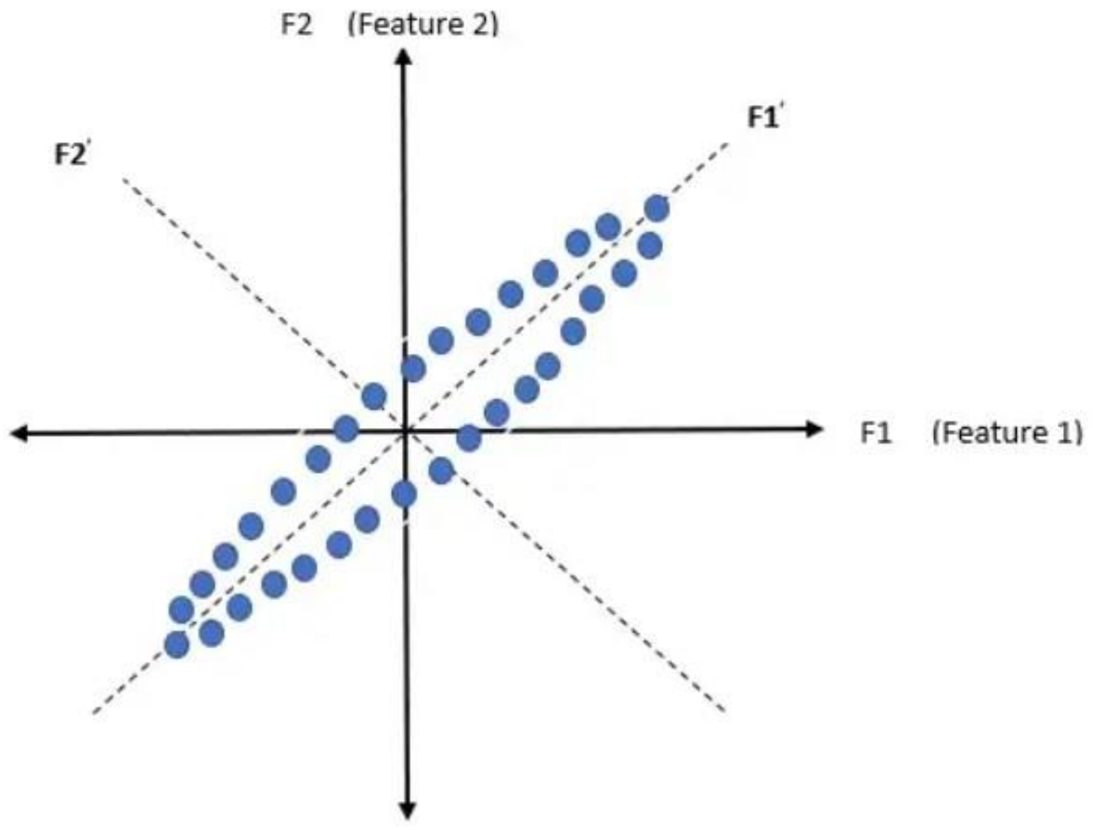
- The KDE plot gives the probability distribution of a continuous variable.
- Violin plots are a combination of a box plot and a KDE plot. The white dot on a violin plot gives the median of the range. The black bar in the centre of the plot gives the interquartile range of the attribute. As an inference, it is found that Besni has a higher interquartile range as compared to Kecimen.

Dimensionality Reduction - PCA:

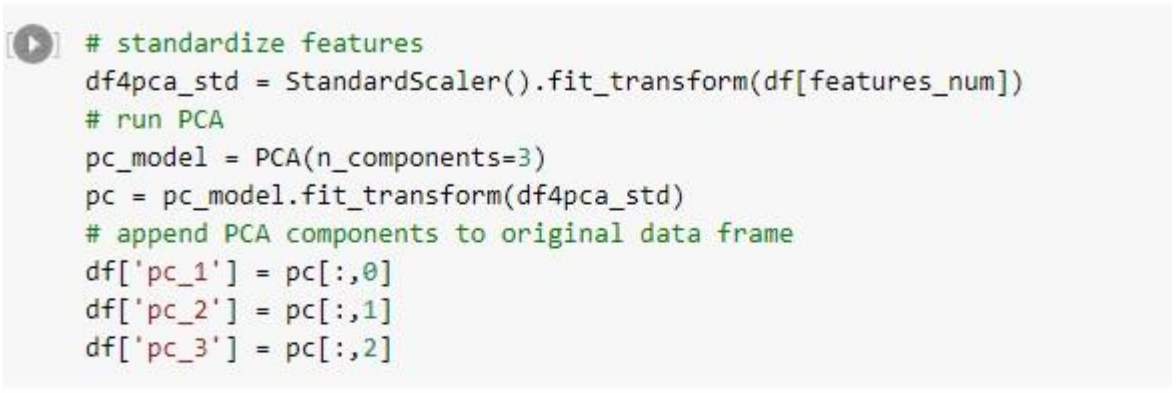
Dimensionality Reduction refers to reducing the number of input variables for the input variable for a dataset. If data is represented using rows and columns, such as in a spreadsheet, then the input variables are the columns that are fed as input to a model to predict the target variable.

PCA helps us to identify patterns in data based on the correlation between features. In a nutshell, PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions than the original one.

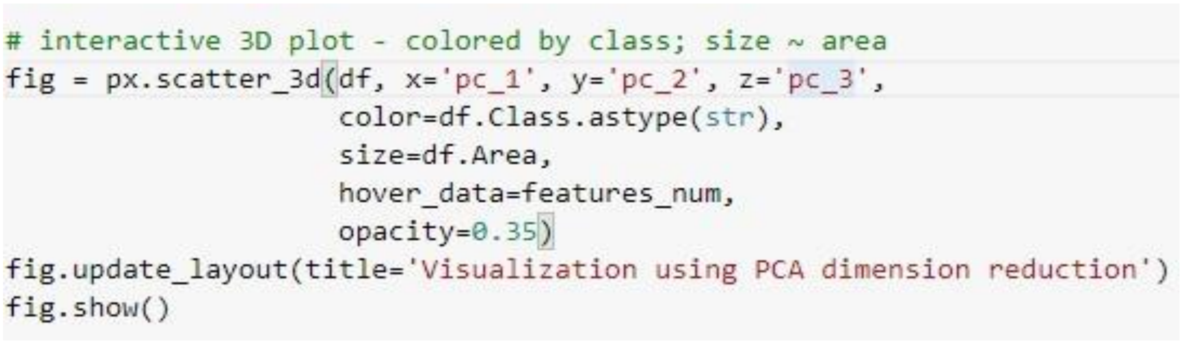
PCA performs linear orthogonal transformation on the data to find features F1’ and F2’ such that the variance on F1’ >> variance on F2’.



To find these new features, all PCA does is rotate the original axes in such a way that we get maximum spread (variance) on the new axes after projecting the data points on them. These new axes are a linear combination of the original axes and they are always perpendicular to each other.

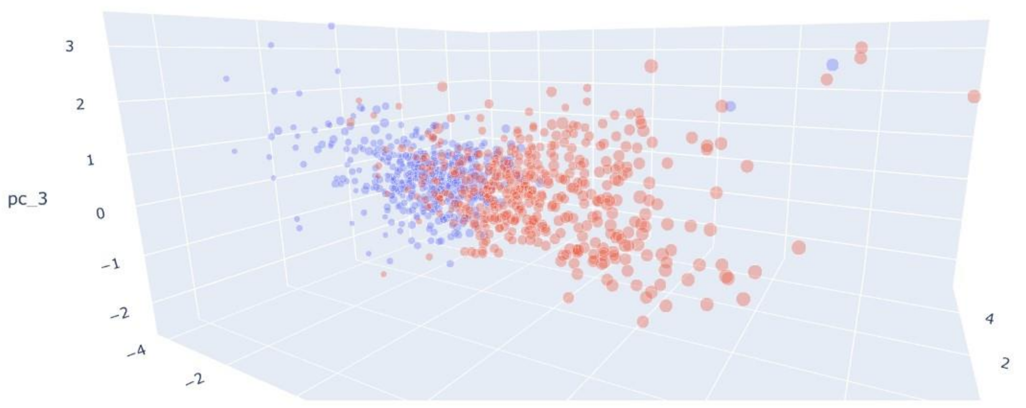


```
[ ] # standardize features
    df4pca_std = StandardScaler().fit_transform(df[features_num])
    # run PCA
    pc_model = PCA(n_components=3)
    pc = pc_model.fit_transform(df4pca_std)
    # append PCA components to original data frame
    df['pc_1'] = pc[:,0]
    df['pc_2'] = pc[:,1]
    df['pc_3'] = pc[:,2]
```

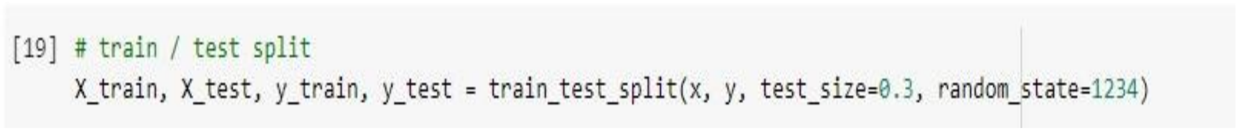


```
# interactive 3D plot - colored by class; size ~ area
fig = px.scatter_3d(df, x='pc_1', y='pc_2', z='pc_3',
                    color=df.Class.astype(str),
                    size=df.Area,
                    hover_data=features_num,
                    opacity=0.35)
fig.update_layout(title='Visualization using PCA dimension reduction')
fig.show()
```

Visualization using PCA dimension reduction



Splitting our Dataset into Training and Testing Subsets:



```
[19] # train / test split
     X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1234)
```

Training our Dataset on Different ML Classification Algorithms:

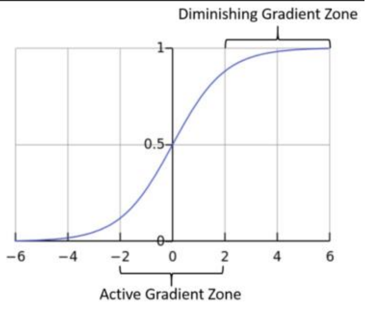
Logistic Regression

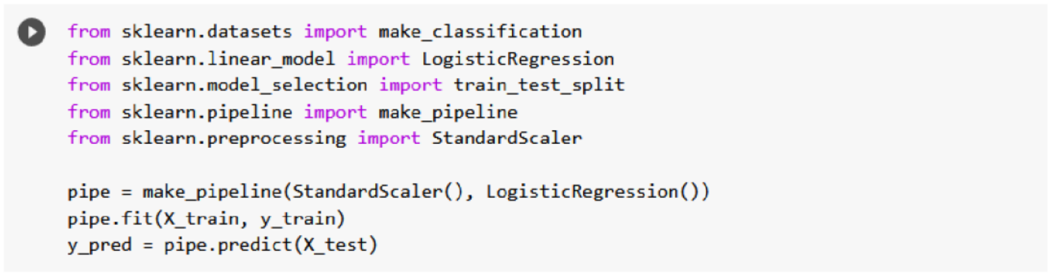
Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1.

In its fundamental structure, Logistic Regression is a model based on statistics. It is used when we have a categorical dependent variable(y), instead of a continuous.

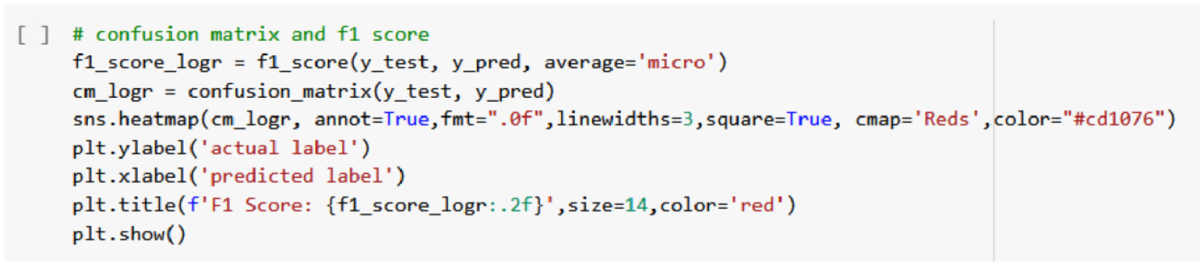
This model is used to calculate the probability of a certain class. It can also be used for multiclass attribute values as well. It basically follows the linear regression model, but the continuous output value is passed through a function called a “Sigmoid Function”, which is used to scale the value between 0 and 1. A threshold value was selected. For our problem which is a 2- Class, if the sigmoid function gives a value greater than the threshold, then class 1 is selected, otherwise class 0.

$$A = \frac{1}{1+e^{-x}}$$

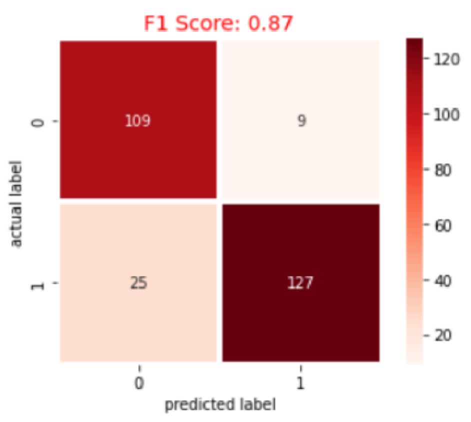


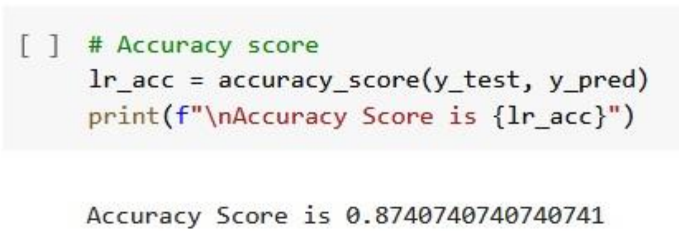


```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

pipe = make_pipeline(StandardScaler(), LogisticRegression())
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```



```
[ ]  # confusion matrix and f1 score
     f1_score_logr = f1_score(y_test, y_pred, average='micro')
     cm_logr = confusion_matrix(y_test, y_pred)
     sns.heatmap(cm_logr, annot=True,fmt=".0f",linewidths=3,square=True, cmap='Reds',color="#cd1076")
     plt.ylabel('actual label')
     plt.xlabel('predicted label')
     plt.title(f'F1 Score: {f1_score_logr:.2f}',size=14,color='red')
     plt.show()
```



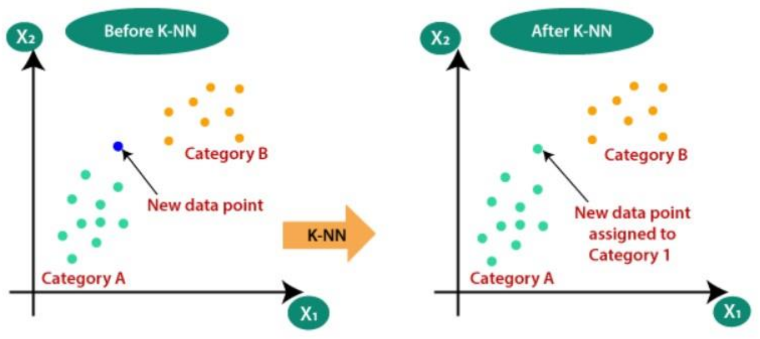


```
[ ]  # Accuracy score
     lr_acc = accuracy_score(y_test, y_pred)
     print(f"\nAccuracy Score is {lr_acc}")

     Accuracy Score is 0.8740740740740741
```

KNN - (k nearest neighbour)

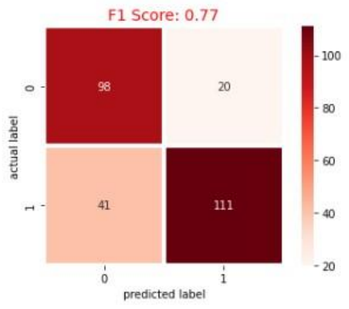
K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on the Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and puts the new case into the category that is most similar to the available categories. It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a good suite category by using K- NN algorithm. This algorithm can be used for Regression as well as for Classification but mostly it is used for Classification problems.

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories? To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



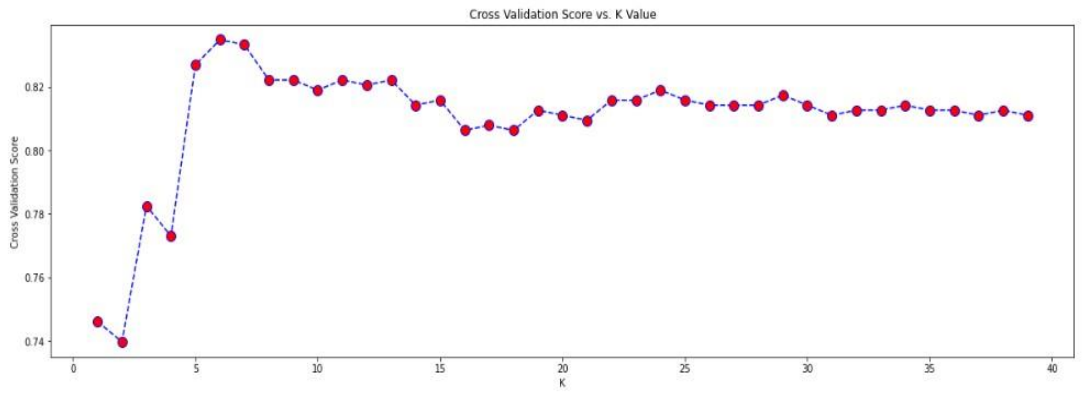
```
[23] from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier(n_neighbors=1)
     knn.fit(X_train, y_train)

     y_pred = knn.predict(X_test)
```

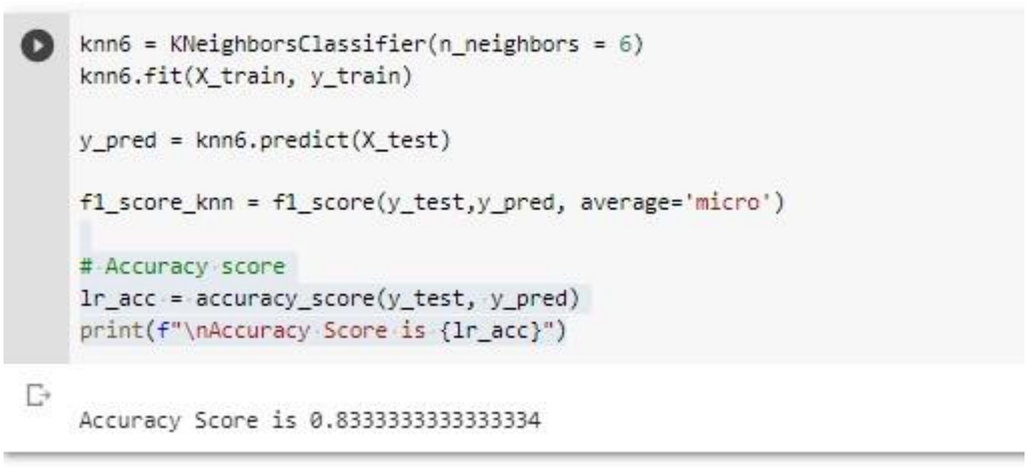
```
# confusion matrix and f1 score
f1_score_knn = f1_score(y_test,y_pred, average='micro')
cm_knn = confusion_matrix(y_test,y_pred)
sns.heatmap(cm_knn, annot=True,fmt=".0f",linewidths=3,square=True, cmap='Reds', color="#cd1076")
plt.ylabel('actual label')
plt.xlabel('predicted label')
plt.title(f'F1 Score: {f1_score_knn:.2f}',size=14,color='red')
plt.show()

# Accuracy score
lr_acc = accuracy_score(y_test, y_pred)
print(f"\nAccuracy Score is {lr_acc}")
```



```
Accuracy Score is 0.774074074074074
```

```
[26] # finding optimum k
error_rate = []

# specify the values of k to test
k_values = range(1, 40)

for i in range(1,40):
    knn_test = KNeighborsClassifier(n_neighbors=i, metric = 'minkowski')
    # evaluate the model using cross-validation
    cv_scores = cross_val_score(knn_test, X_train, y_train, cv=5)
    error_rate.append(cv_scores.mean())

# find the value of k with the highest mean score
plt.figure(figsize=(20,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
        marker='o',markerfacecolor='red', markersize=10)
plt.title('Cross Validation Score vs. K Value')
plt.xlabel('K')
plt.ylabel('Cross Validation Score')
plt.show()

# find the value of k with the highest mean score
bestKVal = k_values[error_rate.index(max(error_rate))]
print("\nBest value of k:", bestKVal)
```





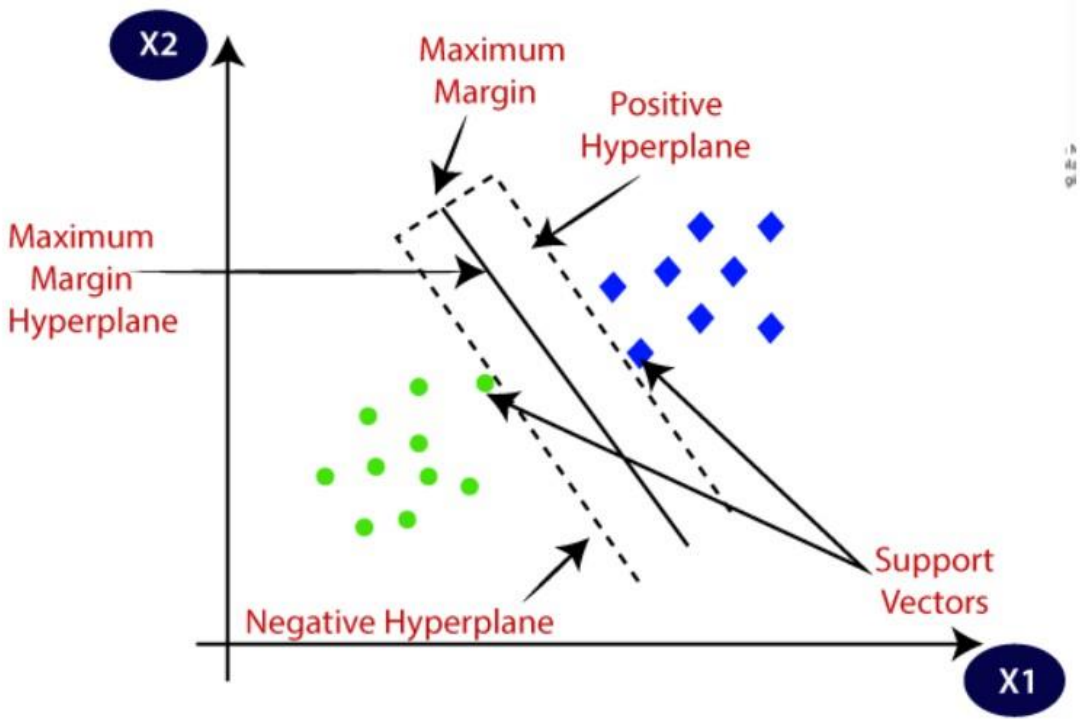
```
knn6 = KNeighborsClassifier(n_neighbors = 6)
knn6.fit(X_train, y_train)

y_pred = knn6.predict(X_test)

f1_score_knn = f1_score(y_test,y_pred, average='micro')

# Accuracy score
lr_acc = accuracy_score(y_test, y_pred)
print(f"\nAccuracy Score is {lr_acc}")
```

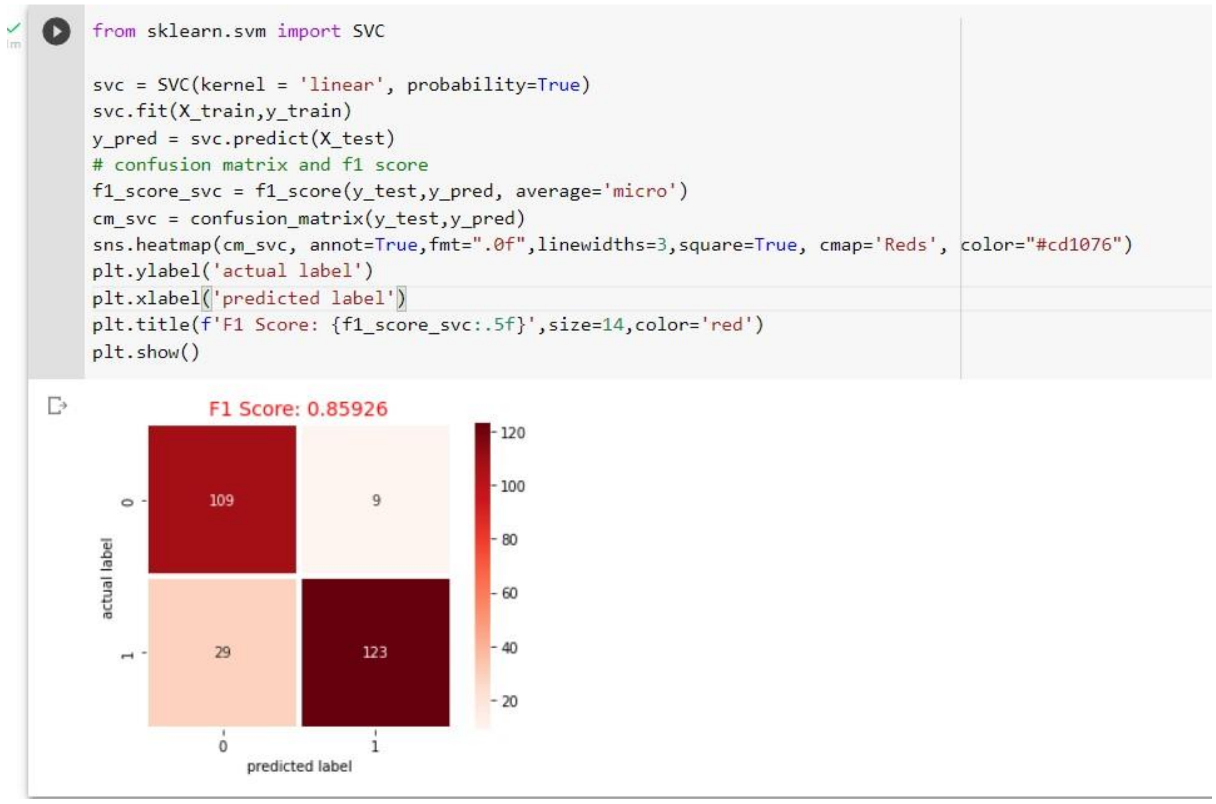
```
Accuracy Score is 0.8333333333333334
```

Support Vector Machine

Support Vector Machine uses a supervised learning algorithm. It is used to find a hyperplane that will separate the data classes. Now, there may be many hyperplanes which can separate the data, but SVM tries to find the best fit line for this separation. This classifier generally works well when there is a clear line of separation between the classes, and the dataset is not large enough

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



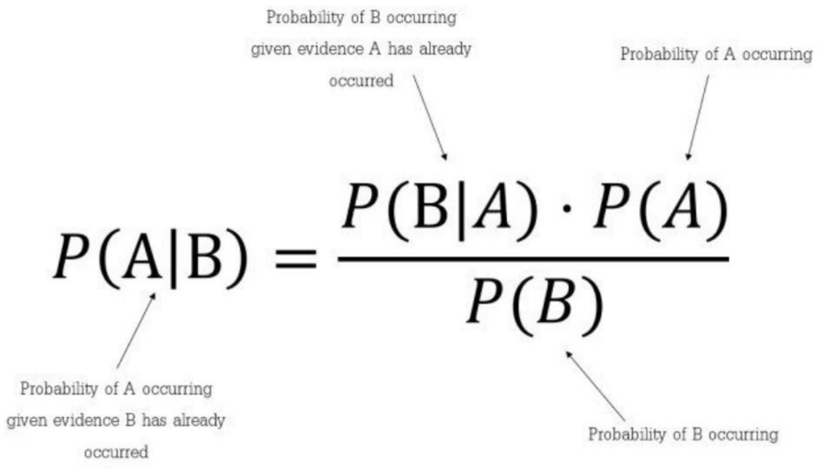


```
from sklearn.svm import SVC

svc = SVC(kernel = 'linear', probability=True)
svc.fit(X_train,y_train)
y_pred = svc.predict(X_test)
# confusion matrix and f1 score
f1_score_svc = f1_score(y_test,y_pred, average='micro')
cm_svc = confusion_matrix(y_test,y_pred)
sns.heatmap(cm_svc, annot=True,fmt=".0f",linewidths=3,square=True, cmap='Reds', color="#cd1076")
plt.ylabel('actual label')
plt.xlabel('predicted label')
plt.title(f'F1 Score: {f1_score_svc:.5f}',size=14,color='red')
plt.show()
```

Naive Bayes

Naive Bayes classifier is based on the Bayes theorem which we study in Probability. In ML, it takes an assumption that there is independence among the attributes X(i) when prediction for the class.

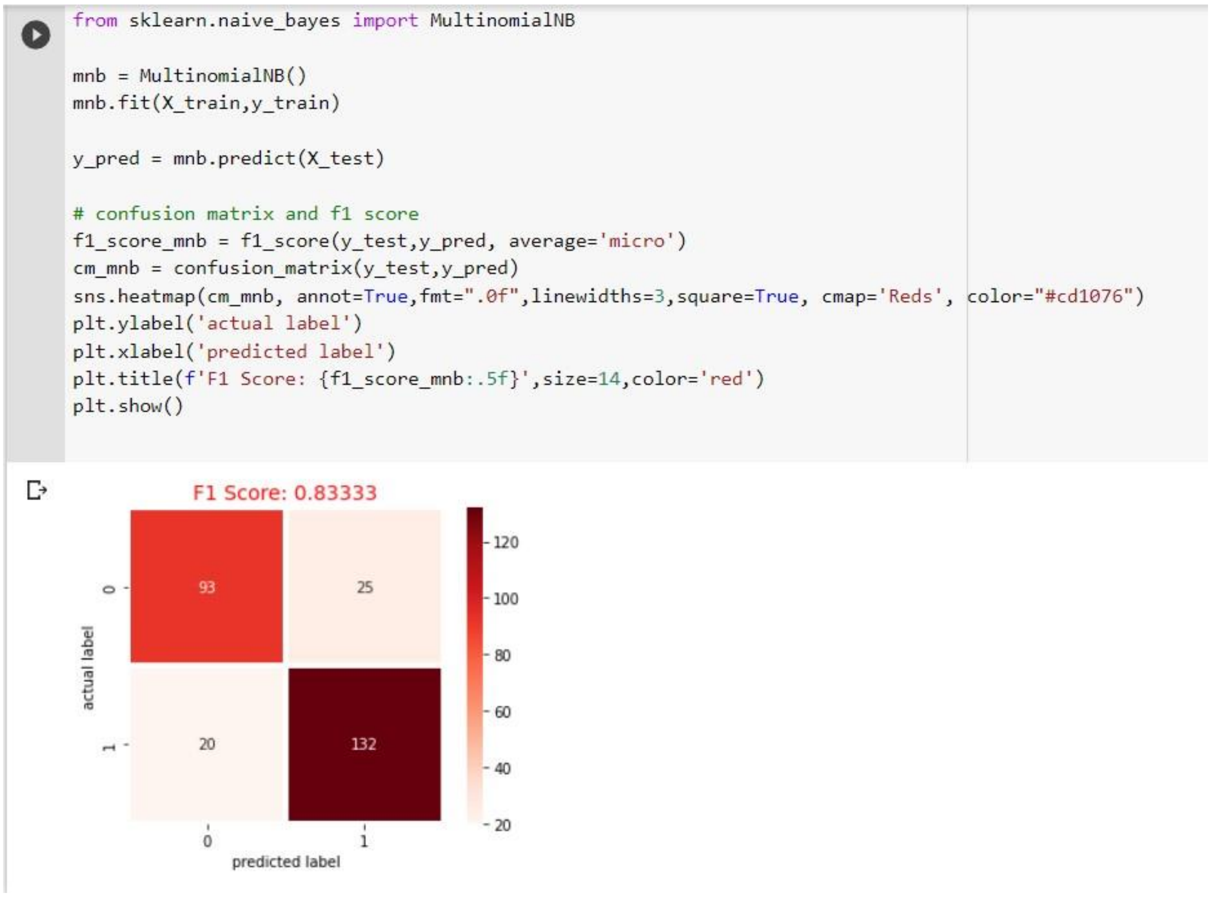
● The formula for calculating probability using Bayes Theorem.



$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Probability of B occurring given evidence A has already occurred
Probability of A occurring
Probability of A occurring given evidence B has already occurred
Probability of B occurring

The fundamental Naïve Bayes assumption is that each feature makes an independent equal contribution to the outcome.

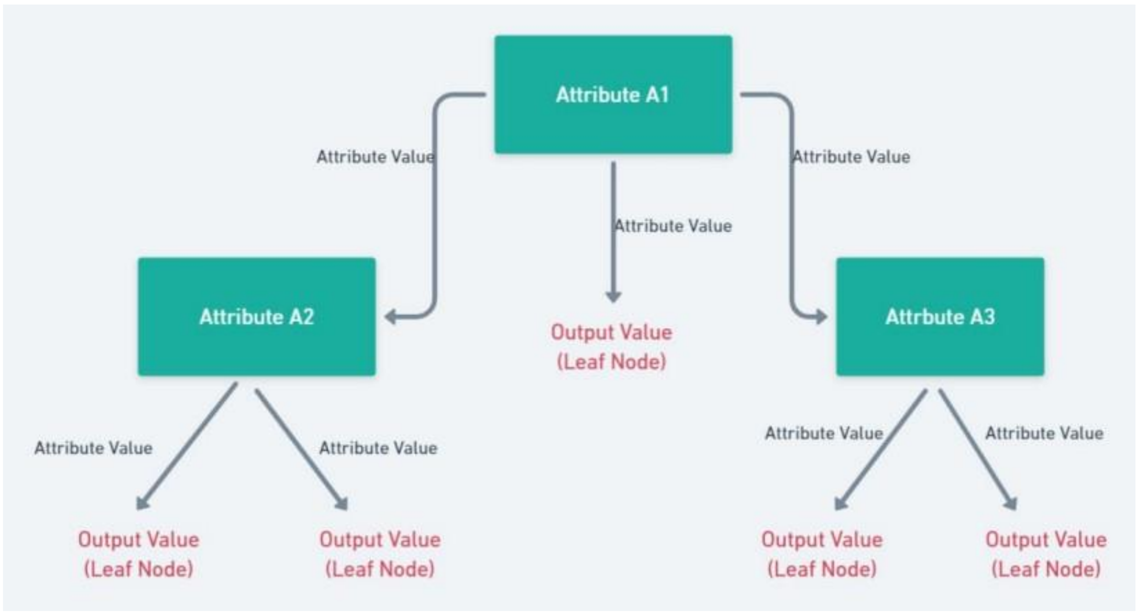


```
from sklearn.naive_bayes import MultinomialNB

mnb = MultinomialNB()
mnb.fit(X_train,y_train)

y_pred = mnb.predict(X_test)

# confusion matrix and f1 score
f1_score_mnb = f1_score(y_test,y_pred, average='micro')
cm_mnb = confusion_matrix(y_test,y_pred)
sns.heatmap(cm_mnb, annot=True,fmt=".0f",linewidths=3,square=True, cmap='Reds', color="#cd1076")
plt.ylabel('actual label')
plt.xlabel('predicted label')
plt.title(f'F1 Score: {f1_score_mnb:.5f}',size=14,color='red')
plt.show()
```

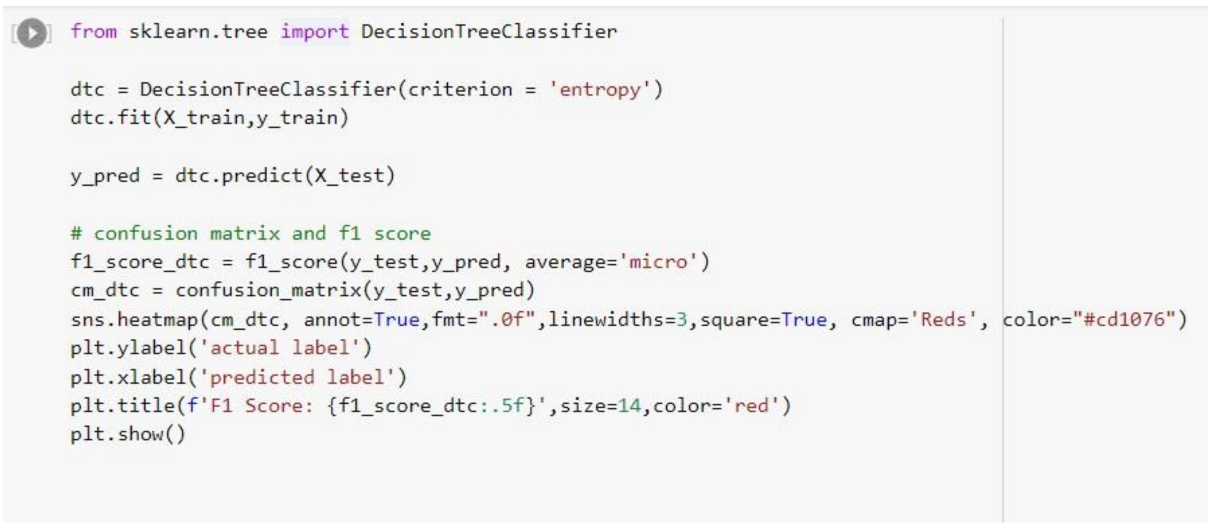
Decision Tree

This model forms a decision tree based on the input dataset. It helps to predict the class of a new input record. It's like a tree structure, with each node representing a condition on attribute values. Based on the condition output, we select our path(answer to conditions) and proceed to predict our class till we encounter a leaf node(output class variables).

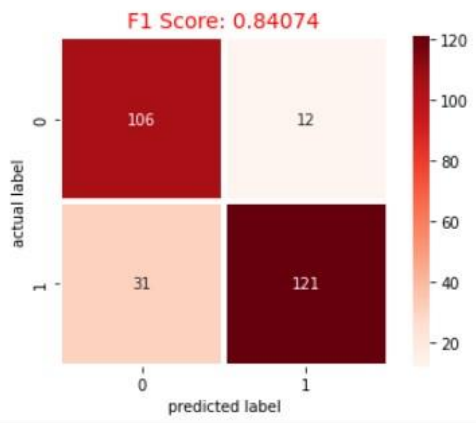
We've learnt that decision trees are made with the help of GINI Index, Entropy calculations etc which measure impurity, to try to determine what should be taken as the best split.



Structure of Decision Tree



```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(criterion = 'entropy')
dtc.fit(X_train,y_train)

y_pred = dtc.predict(X_test)

# confusion matrix and f1 score
f1_score_dtc = f1_score(y_test,y_pred, average='micro')
cm_dtc = confusion_matrix(y_test,y_pred)
sns.heatmap(cm_dtc, annot=True,fmt=".0f",linewidths=3,square=True, cmap='Reds', color="#cd1076")
plt.ylabel('actual label')
plt.xlabel('predicted label')
plt.title(f'F1 Score: {f1_score_dtc:.5f}',size=14,color='red')
plt.show()
```

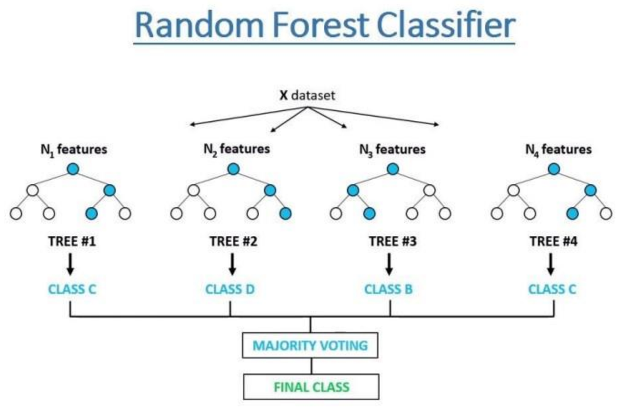


Random Forest Model

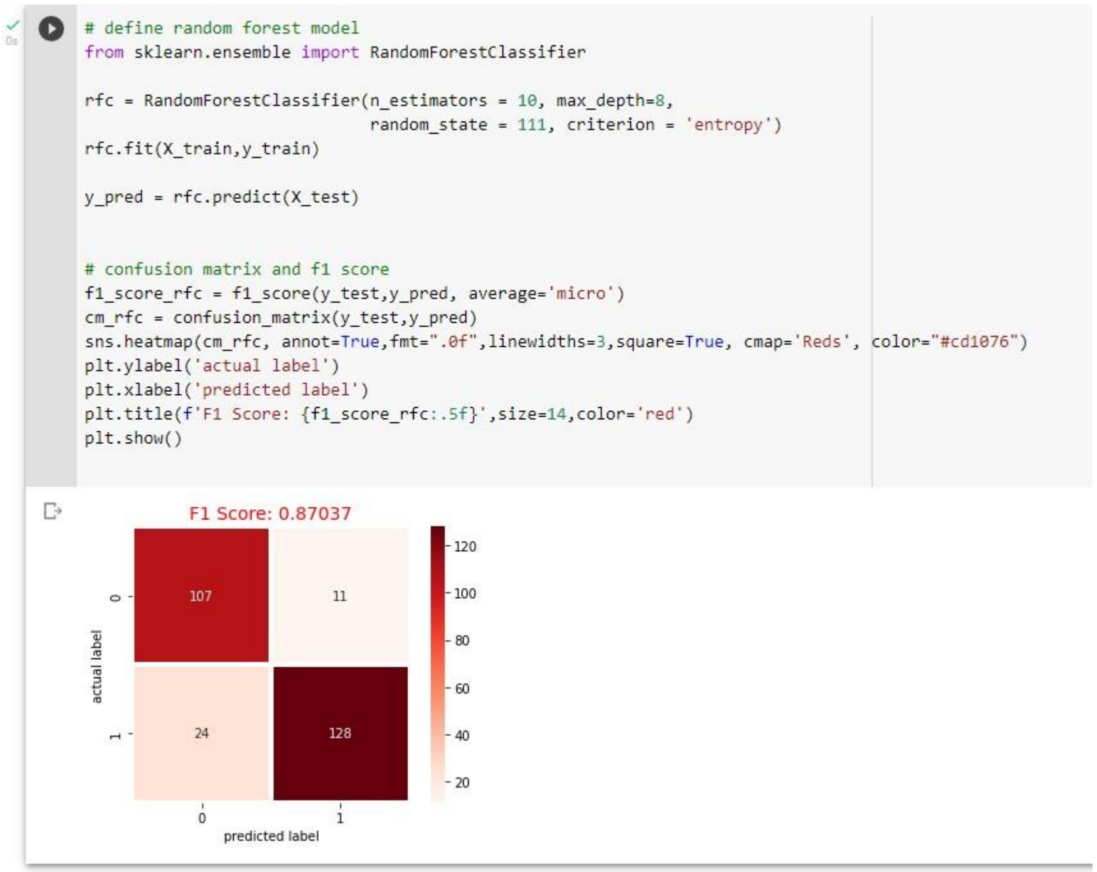
This classifier follows an ensemble learning. Instead of one, multiple decision trees work as an “ensemble”. It adds randomness to the ensemble by randomly creating a forest of decision trees.

Each decision tree produces an output and the final class for the input record is done by majority voting.

They are found to be more accurate than single decision trees, but have an extended time for training.



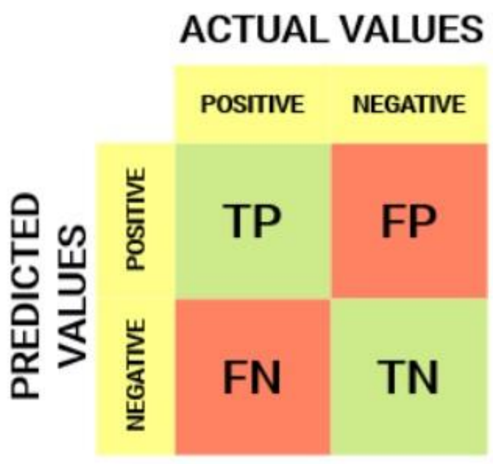
Structure of how Random Forest works



```
# define random forest model
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators = 10, max_depth=8,
                             random_state = 111, criterion = 'entropy')
rfc.fit(X_train,y_train)

y_pred = rfc.predict(X_test)


# confusion matrix and f1 score
f1_score_rfc = f1_score(y_test,y_pred, average='micro')
cm_rfc = confusion_matrix(y_test,y_pred)
sns.heatmap(cm_rfc, annot=True,fmt=".0f",linewidths=3,square=True, cmap='Reds', color="#cd1076")
plt.ylabel('actual label')
plt.xlabel('predicted label')
plt.title(f'F1 Score: {f1_score_rfc:.5f}',size=14,color='red')
plt.show()
```

ROC test

● Confusion Matrix



Sensitivity / True Positive Rate / Recall

$$Sensitivity = \frac{TP}{TP + FN}$$

False Negative Rate

$$FNR = \frac{FN}{TP + FN}$$

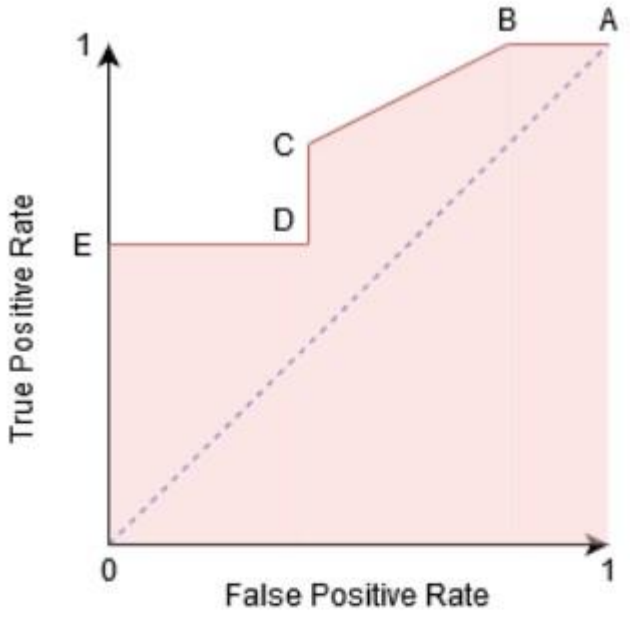
Specificity / True Negative Rate

$$Specificity = \frac{TN}{TN + FP}$$

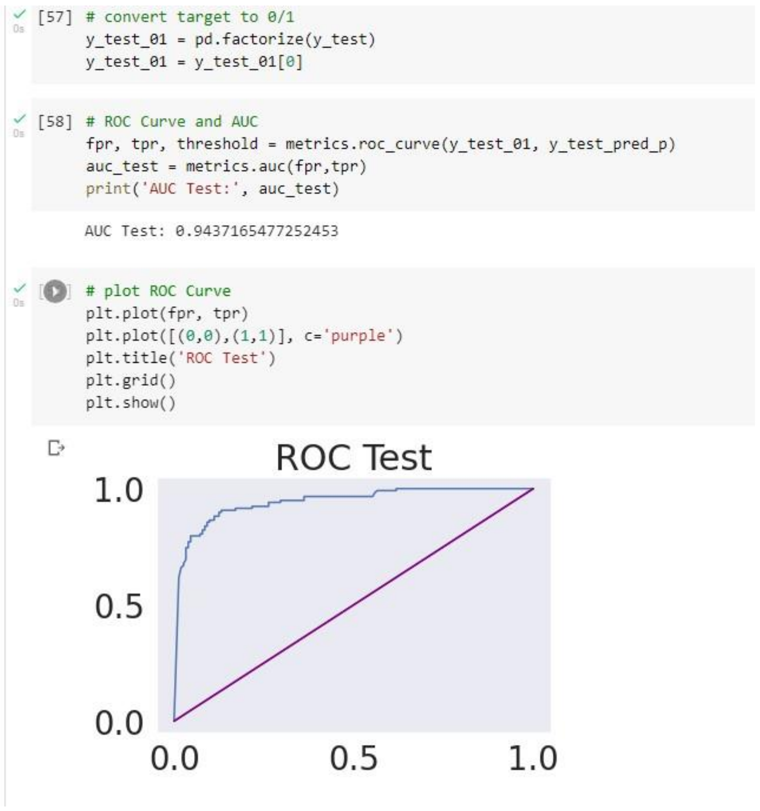
False Positive Rate

$$FPR = \frac{FP}{TN + FP} = 1 - Specificity$$

ROC plot



In this plot more is the area covered better the classifier used. All models below slow 1 are rejected and the classifier closer to the y=x line is the worst.



```
[57] # convert target to 0/1
     y_test_01 = pd.factorize(y_test)
     y_test_01 = y_test_01[0]
```

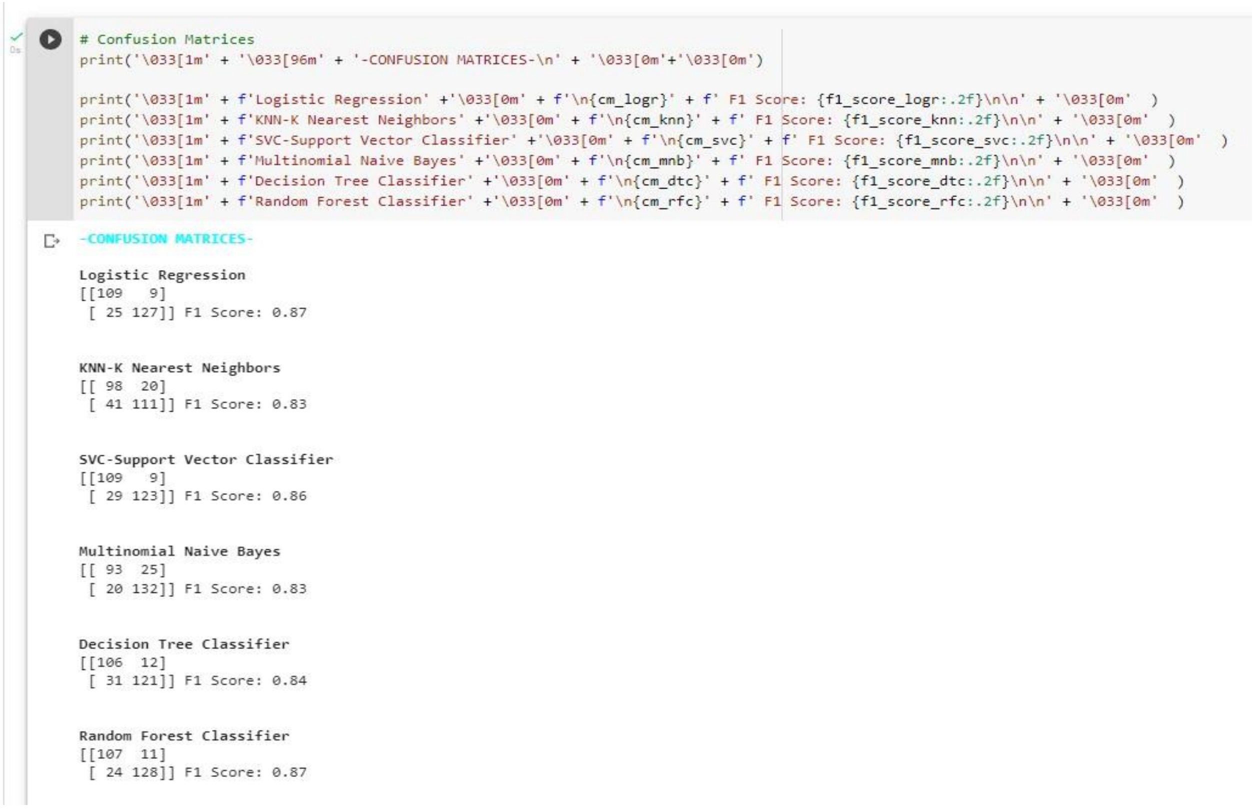
```
[58] # ROC Curve and AUC
     fpr, tpr, threshold = metrics.roc_curve(y_test_01, y_test_pred_p)
     auc_test = metrics.auc(fpr,tpr)
     print('AUC Test:', auc_test)
```

```
AUC Test: 0.9437165477252453
```

```
# plot ROC Curve
plt.plot(fpr, tpr)
plt.plot([(0,0),(1,1)], c='purple')
plt.title('ROC Test')
plt.grid()
plt.show()
```

Evaluating Different Trained Models:

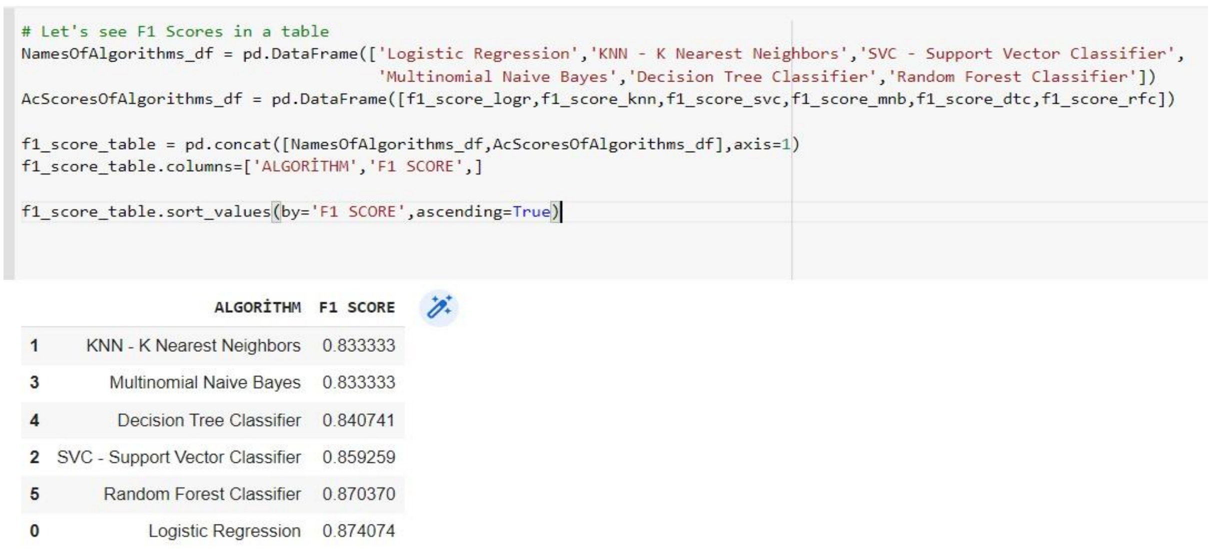
1) Comparison of Confusion Matrices of Different Models



```
# Confusion Matrices
print('\033[1m' + '\033[96m' + '-CONFUSION MATRICES-\n' + '\033[0m'+'\033[0m')

print('\033[1m' + f'Logistic Regression' +'\033[0m' + f'\n{cm_logr}' + f' F1 Score: {f1_score_logr:.2f}\n\n' + '\033[0m'  )
print('\033[1m' + f'KNN-K Nearest Neighbors' +'\033[0m' + f'\n{cm_knn}' + f' F1 Score: {f1_score_knn:.2f}\n\n' + '\033[0m'  )
print('\033[1m' + f'SVC-Support Vector Classifier' +'\033[0m' + f'\n{cm_svc}' + f' F1 Score: {f1_score_svc:.2f}\n\n' + '\033[0m'  )
print('\033[1m' + f'Multinomial Naive Bayes' +'\033[0m' + f'\n{cm_mnb}' + f' F1 Score: {f1_score_mnb:.2f}\n\n' + '\033[0m'  )
print('\033[1m' + f'Decision Tree Classifier' +'\033[0m' + f'\n{cm_dtc}' + f' F1 Score: {f1_score_dtc:.2f}\n\n' + '\033[0m'  )
print('\033[1m' + f'Random Forest Classifier' +'\033[0m' + f'\n{cm_rfc}' + f' F1 Score: {f1_score_rfc:.2f}\n\n' + '\033[0m'  )
```

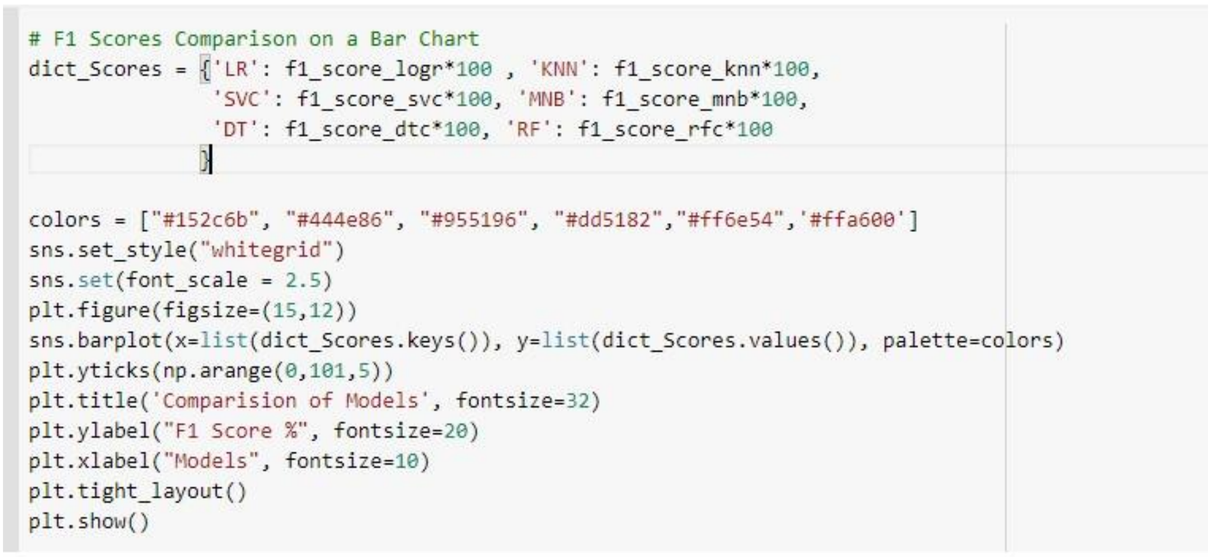
```
-CONFUSION MATRICES-

Logistic Regression
[[109   9]
 [ 25 127]] F1 Score: 0.87


KNN-K Nearest Neighbors
[[ 98  20]
 [ 41 111]] F1 Score: 0.83


SVC-Support Vector Classifier
[[109   9]
 [ 29 123]] F1 Score: 0.86


Multinomial Naive Bayes
[[ 93  25]
 [ 20 132]] F1 Score: 0.83


Decision Tree Classifier
[[106  12]
 [ 31 121]] F1 Score: 0.84


Random Forest Classifier
[[107  11]
 [ 24 128]] F1 Score: 0.87
```

2) Comparison of F1 scores of Different Models

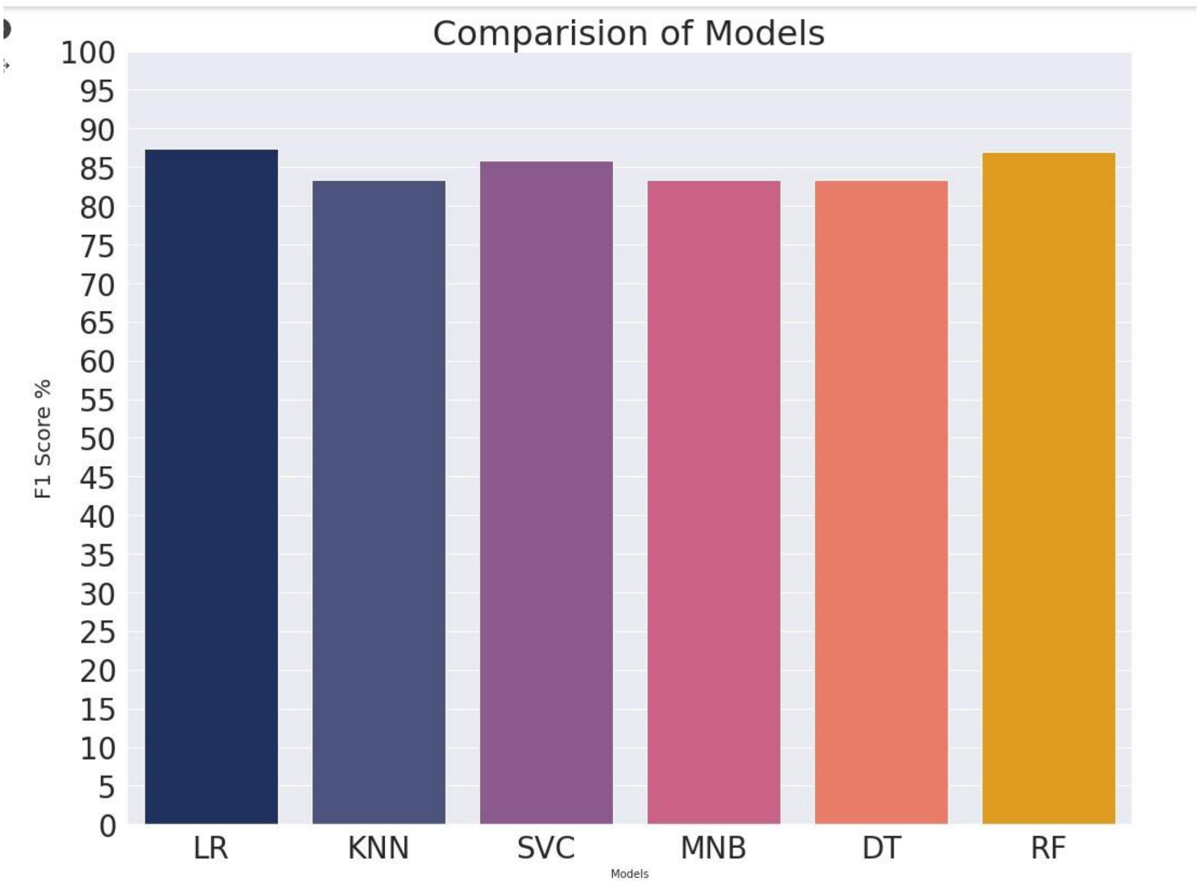


```
# Let's see F1 Scores in a table
NamesOfAlgorithms_df = pd.DataFrame(['Logistic Regression','KNN - K Nearest Neighbors','SVC - Support Vector Classifier',
                                     'Multinomial Naive Bayes','Decision Tree Classifier','Random Forest Classifier'])
AcScoresOfAlgorithms_df = pd.DataFrame([f1_score_logr,f1_score_knn,f1_score_svc,f1_score_mnb,f1_score_dtc,f1_score_rfc])

f1_score_table = pd.concat([NamesOfAlgorithms_df,AcScoresOfAlgorithms_df],axis=1)
f1_score_table.columns=['ALGORİTHM','F1 SCORE',]

f1_score_table.sort_values(by='F1 SCORE',ascending=True)
```

	ALGORİTHM	F1 SCORE
1	KNN - K Nearest Neighbors	0.833333
3	Multinomial Naive Bayes	0.833333
4	Decision Tree Classifier	0.840741
2	SVC - Support Vector Classifier	0.859259
5	Random Forest Classifier	0.870370
0	Logistic Regression	0.874074



```
# F1 Scores Comparison on a Bar Chart
dict_Scores = {'LR': f1_score_logr*100 , 'KNN': f1_score_knn*100,
               'SVC': f1_score_svc*100, 'MNB': f1_score_mnb*100,
               'DT': f1_score_dtc*100, 'RF': f1_score_rfc*100
              }

colors = ["#152c6b", "#444e86", "#955196", "#dd5182","#ff6e54",'#ffa600']
sns.set_style("whitegrid")
sns.set(font_scale = 2.5)
plt.figure(figsize=(15,12))
sns.barplot(x=list(dict_Scores.keys()), y=list(dict_Scores.values()), palette=colors)
plt.yticks(np.arange(0,101,5))
plt.title('Comparision of Models', fontsize=32)
plt.ylabel("F1 Score %", fontsize=20)
plt.xlabel("Models", fontsize=10)
plt.tight_layout()
plt.show()
```



Conclusion:

1) To summarise our results from the implementations, we get the following table:

Algorithm	**Accuracy**

Logistic Regression	**87.407%**
KNN	**83.334%**
Support Vector Machine	**85.926%**
Naive Bayes	**83.334%**
Decision Tree	**84.074%**
Random Forest Model	**87.037%**

2) From the results, it’s clear that **Logistic Regression Classifier** gives the best f1 score of **0.87407** on our dataset as compared to other ML classification algorithms because it is a **binary classifier** which divides our dataset into 2 classes and this sounds to be valid since in our dataset we have to **predict only between 2 labels - Kecimen and Besni.**