INTRODUCTION TO DATA SCIENCE

PROJECT REPORT

ON

RICE GRAIN CLASSIFICATION USING SUPERVISED MACHINE LEARNING ALGORITHMS

Team Members:

Pavan Aditya - 21ucc113

Dhanush Kodavanti- 21ucs062

Ramgopal Reddy - 21ucs167

K. Sriman Narayana- 21ucs101

INTRODUCTION

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. Labeled datasets are used to train algorithms to classify data and predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross-validation process.

In this project, we have used various machine learning algorithms, such as support vector machines (SVM), logistic regression, k-nearest neighbors (KNN), and Naïve Bayes Classifier, to classify the type of rice grain based on various characteristics, or features, of the rice grain. The project is implemented in python and it reads a dataset containing the features and target variable, performs data exploration and visualization, and prepares the data for machine learning. The code then trains and evaluates the machine learning models and presents the results of these models.

Dataset Source

The Dataset used for this Project is sourced from the UCI Machine Learning Repository. Below is the URL for the source of the dataset.

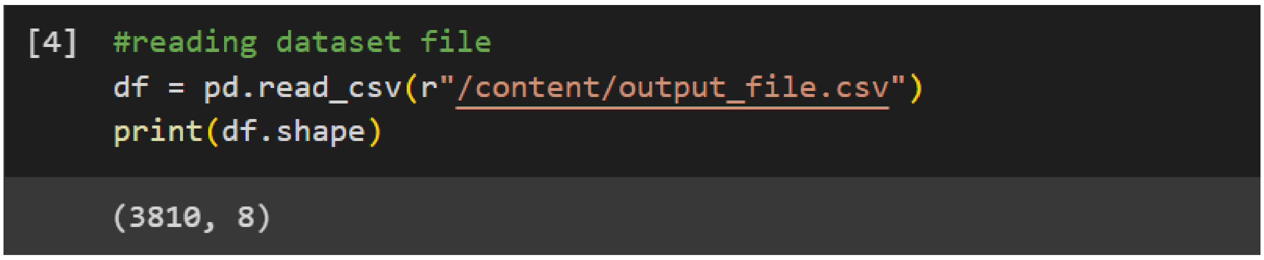
https://archive.ics.uci.edu/dataset/545/rice+cammeo+and+osmancik

Dataset Description

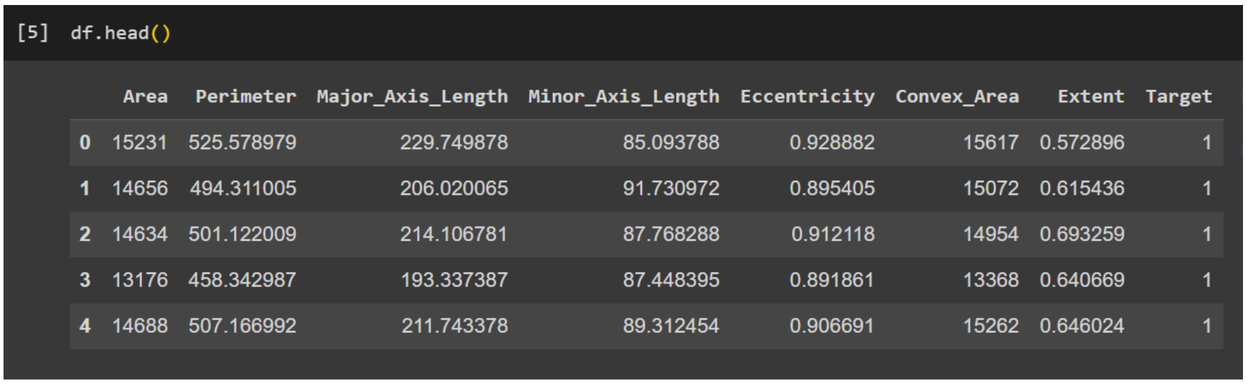
The chosen dataset is a labeled dataset with 3,810 entries each of which contains 7 input features and one class label.

The Attribute Information is as follows:

1. Area: Area of the rice grain
2. Perimeter: Perimeter of the rice grain
3. Major Axis Length: Longest Line that can be drawn on rice grain
4. Minor Axis Length: Shortest Line that can be drawn on rice grain
5. Eccentricity: It measures how round the ellipse, which has same moments as the rice grain, is
6. Convex Area: The area of the smallest convex shell of the region formed by rice grain.
7. Extent: The ratio of region formed by the rice grain to the bounding box
8. Class: There are two classes namely Cammeo and Osmancik

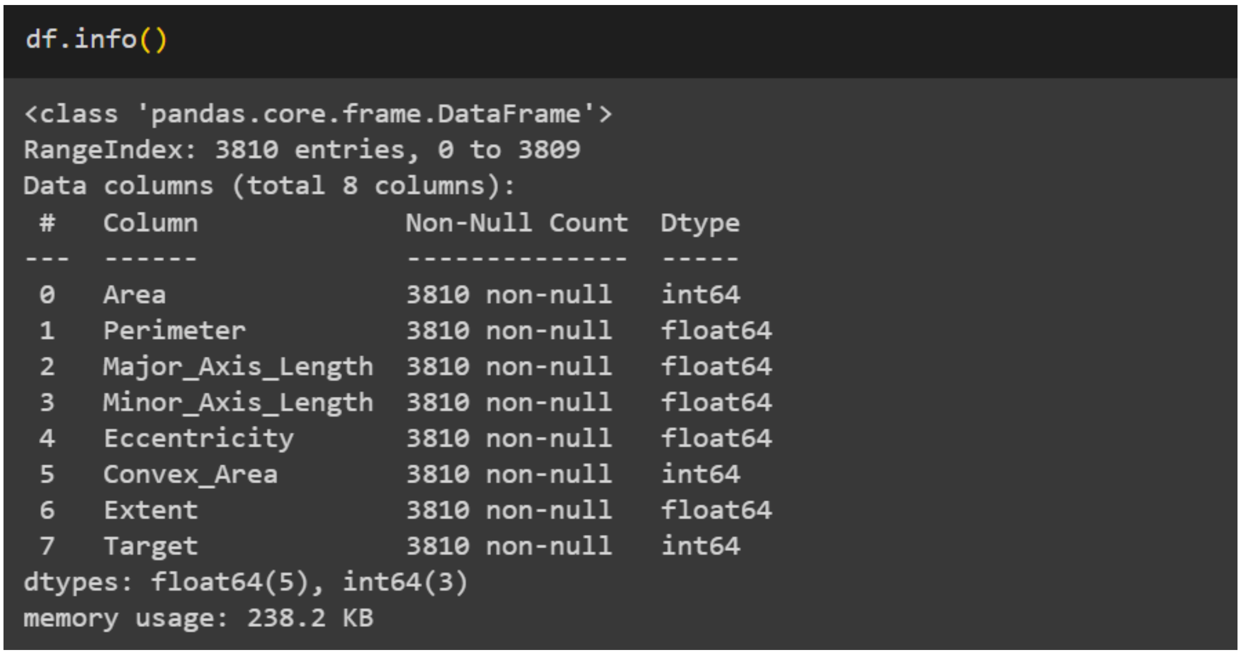


```
[4] #reading dataset file
    df = pd.read_csv(r"/content/output_file.csv")
    print(df.shape)

    (3810, 8)
```



```
[5] df.head()
```

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent	Target
0	15231	525.578979	229.749878	85.093788	0.928882	15617	0.572896	1
1	14656	494.311005	206.020065	91.730972	0.895405	15072	0.615436	1
2	14634	501.122009	214.106781	87.768288	0.912118	14954	0.693259	1
3	13176	458.342987	193.337387	87.448395	0.891861	13368	0.640669	1
4	14688	507.166992	211.743378	89.312454	0.906691	15262	0.646024	1

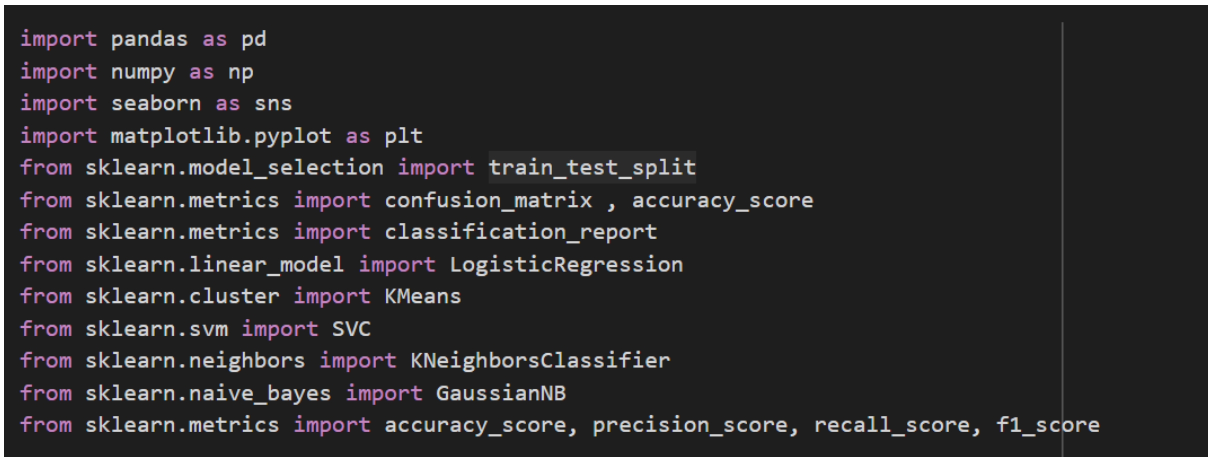


```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3810 entries, 0 to 3809
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Area               3810 non-null   int64  
 1   Perimeter          3810 non-null   float64
 2   Major_Axis_Length  3810 non-null   float64
 3   Minor_Axis_Length  3810 non-null   float64
 4   Eccentricity       3810 non-null   float64
 5   Convex_Area        3810 non-null   int64  
 6   Extent             3810 non-null   float64
 7   Target             3810 non-null   int64  
dtypes: float64(5), int64(3)
memory usage: 238.2 KB
```

LIBRARIES USED

All the libraries used in the code are present in the image below.

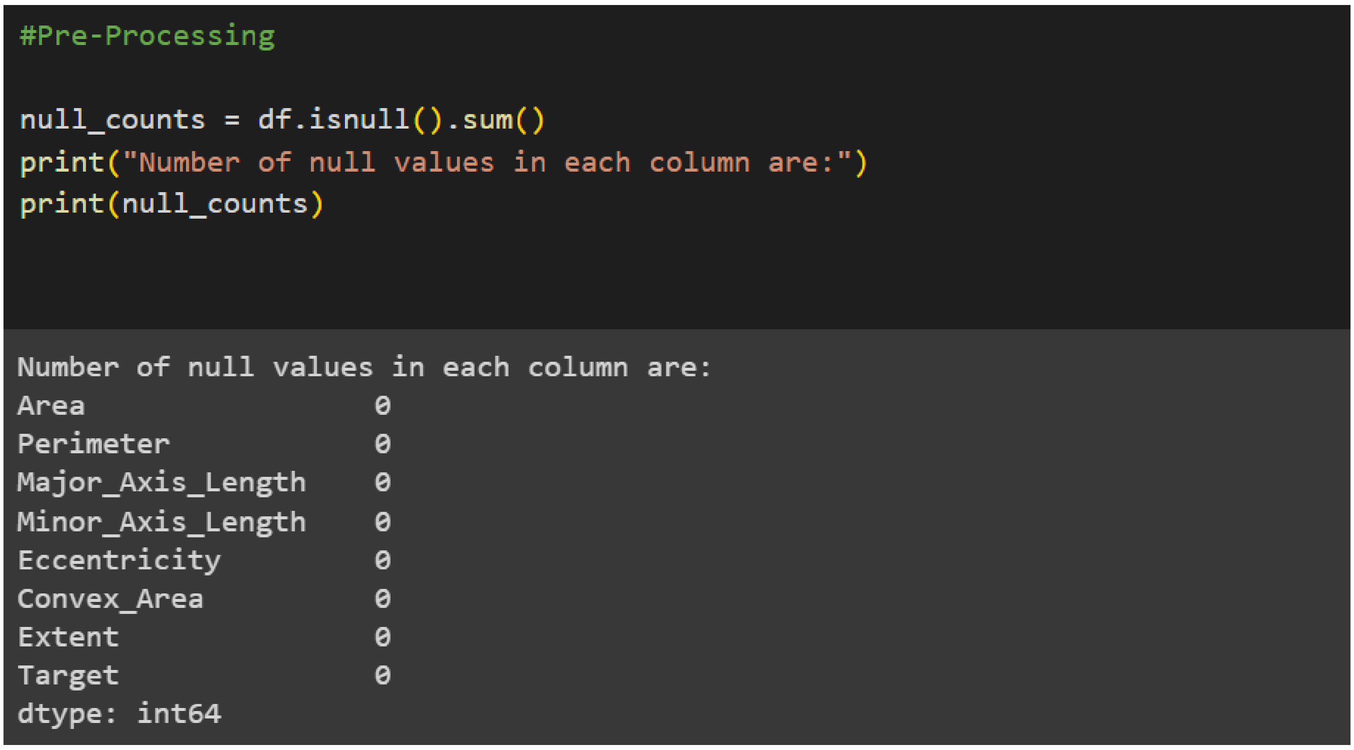


```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix , accuracy_score
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

DATA PRE-PROCESSING

As the first step of data pre-processing, we are trying to find any missing values present in the dataset and if found we will try to fill them with appropriate values. Luckily, the dataset doesn't have any missing values.

As the next step of data pre-processing we have converted the categorical class labels into numerical class labels by encoding class **Cammeo as "1"** and class **Osmancik as "0"** manually.



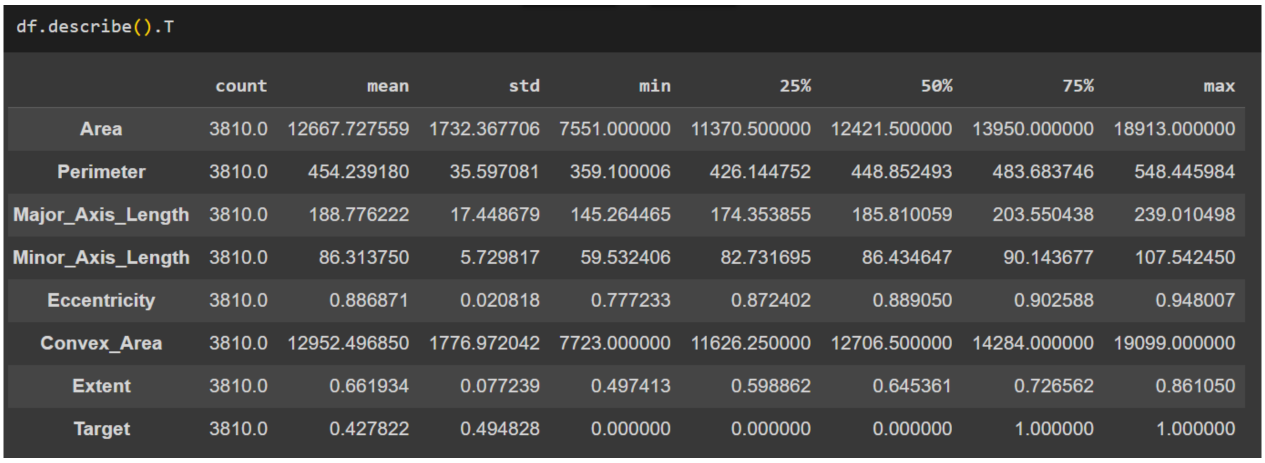
```
#Pre-Processing

null_counts = df.isnull().sum()
print("Number of null values in each column are:")
print(null_counts)
```

```
Number of null values in each column are:
Area                 0
Perimeter            0
Major_Axis_Length    0
Minor_Axis_Length    0
Eccentricity         0
Convex_Area          0
Extent               0
Target               0
dtype: int64
```

DESCRIPTIVE STATISTICS OF DATA

1. Non-Graphical Method

The describe () method in Pandas gives the descriptive analysis which includes count, mean, standard deviation, minimum, and maximum among others.



```
df.describe().T
```

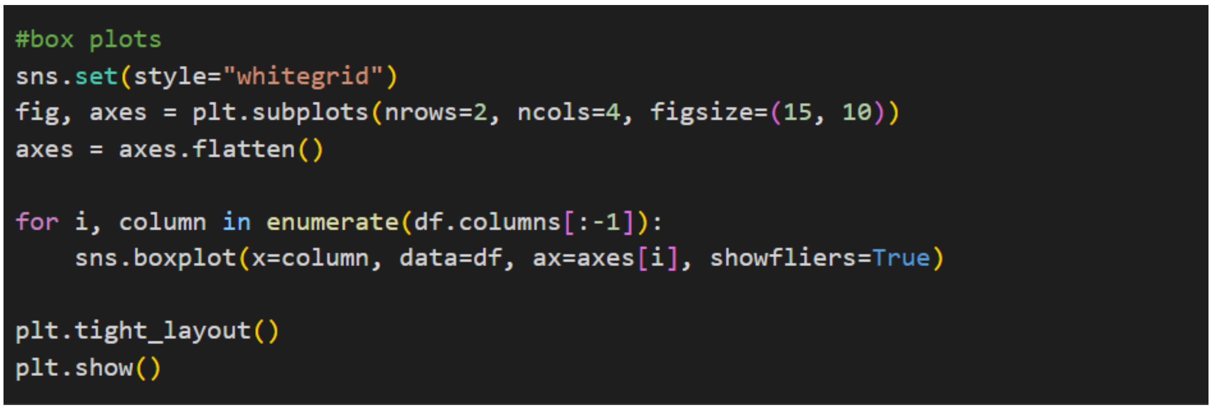
	count	mean	std	min	25%	50%	75%	max
Area	3810.0	12667.727559	1732.367706	7551.000000	11370.500000	12421.500000	13950.000000	18913.000000
Perimeter	3810.0	454.239180	35.597081	359.100006	426.144752	448.852493	483.683746	548.445984
Major_Axis_Length	3810.0	188.776222	17.448679	145.264465	174.353855	185.810059	203.550438	239.010498
Minor_Axis_Length	3810.0	86.313750	5.729817	59.532406	82.731695	86.434647	90.143677	107.542450
Eccentricity	3810.0	0.886871	0.020818	0.777233	0.872402	0.889050	0.902588	0.948007
Convex_Area	3810.0	12952.496850	1776.972042	7723.000000	11626.250000	12706.500000	14284.000000	19099.000000
Extent	3810.0	0.661934	0.077239	0.497413	0.598862	0.645361	0.726562	0.861050
Target	3810.0	0.427822	0.494828	0.000000	0.000000	0.000000	1.000000	1.000000

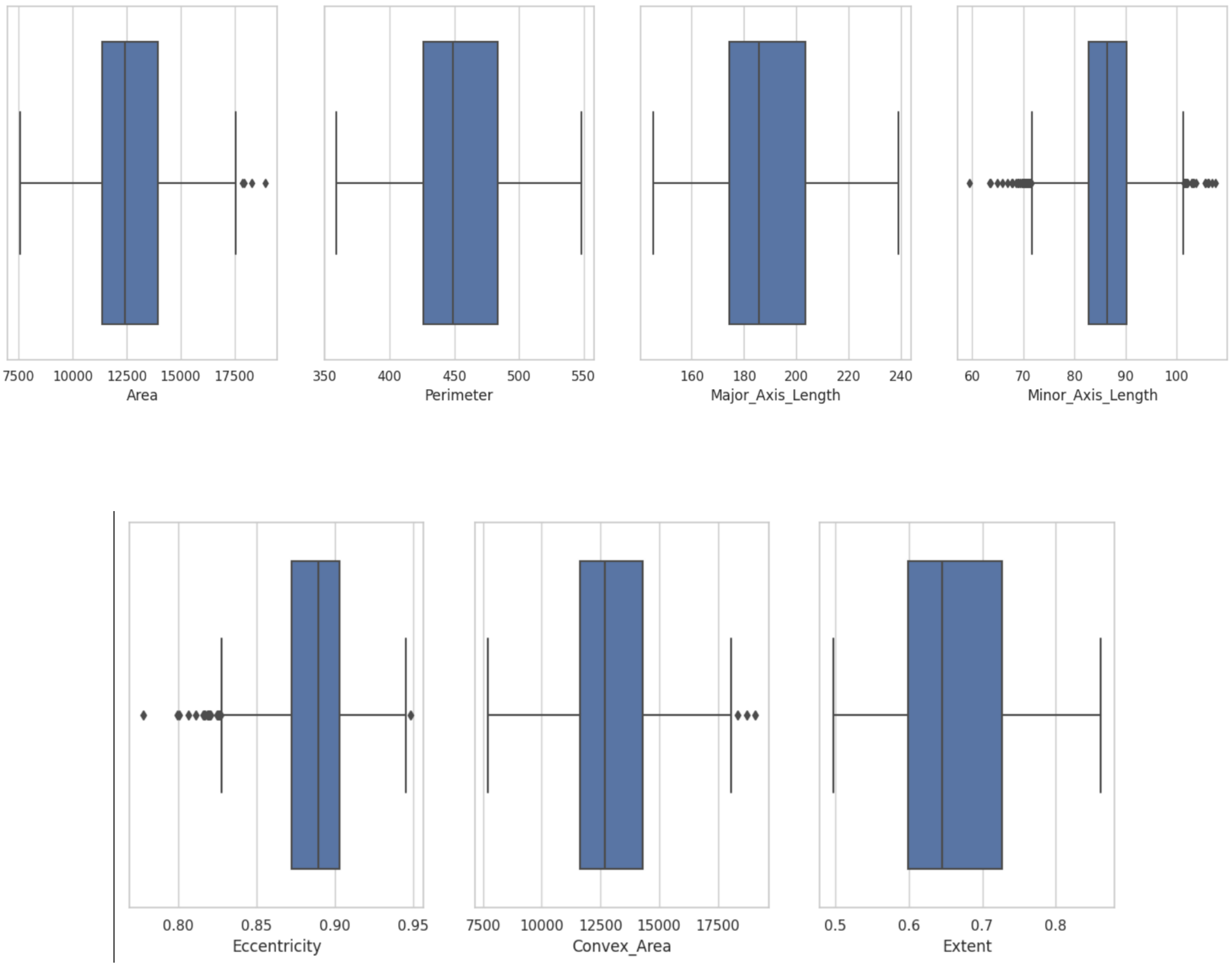
2. Graphical Method

- **Box Plots of input features**

We have used Box Plots to graphically represent the data and the reasons for using it over other graphical methods are as follows:

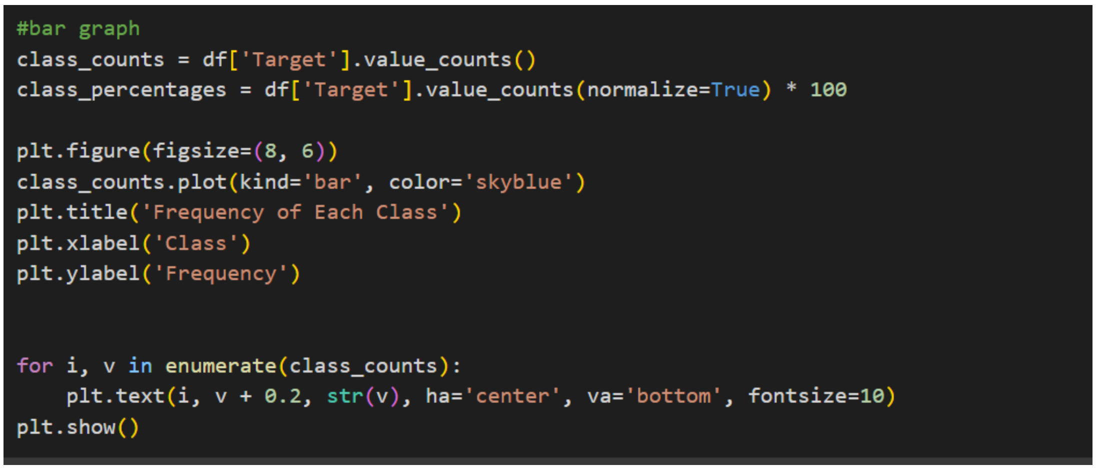
1. Outliers can be easily visualized.
2. Visual Indication of skewness of data is provided by box plots.
3. Just like our case when dealing with large datasets box plots are particularly useful as they efficiently summarize key statistics such as minimum, maximum, median, first quartile and third quartile in a compact form.
4. Easy to interpret


```
#box plots
sns.set(style="whitegrid")
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(15, 10))
axes = axes.flatten()

for i, column in enumerate(df.columns[:-1]):
    sns.boxplot(x=column, data=df, ax=axes[i], showfliers=True)

plt.tight_layout()
plt.show()
```

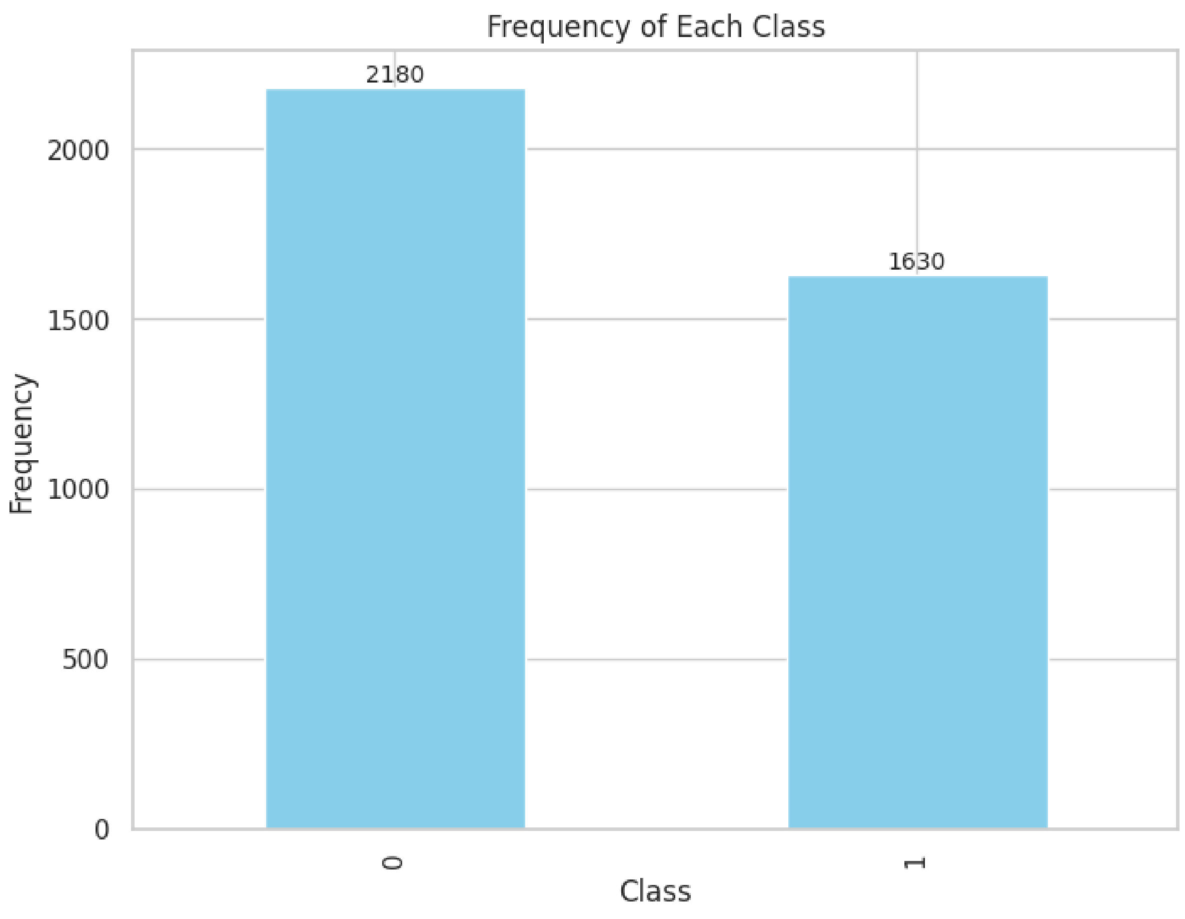


- **Bar Graph for Class Label**

Out of 3810 entries 2180 of them are labeled as class 0 whereas 1630 of them are labeled as 1. This ensures there is not a lot of **Class Imbalance** and the algorithm can train sufficiently on both the classes.



```
#bar graph
class_counts = df['Target'].value_counts()
class_percentages = df['Target'].value_counts(normalize=True) * 100

plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color='skyblue')
plt.title('Frequency of Each Class')
plt.xlabel('Class')
plt.ylabel('Frequency')


for i, v in enumerate(class_counts):
    plt.text(i, v + 0.2, str(v), ha='center', va='bottom', fontsize=10)
plt.show()
```

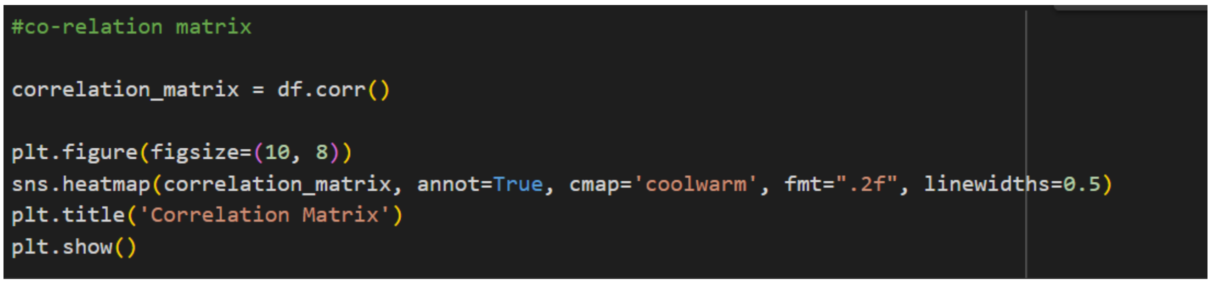


• Examining the relationship between input features using pairwise plots and correlation matrix

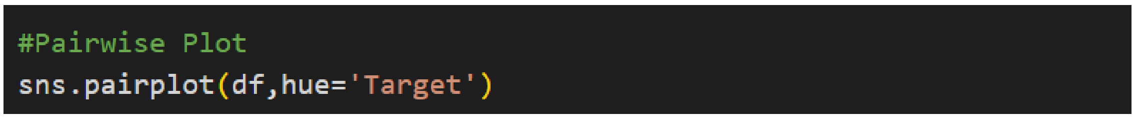
A correlation matrix is a statistical technique used to evaluate the relationship between two variables in a data set. The correlation Matrix is also known as the Dispersion Matrix where each cell represents the **correlation coefficient** between different features. The correlation coefficient of 1 is considered a strong relationship between features where an increase in one feature increases the other feature. A correlation coefficient of 0 is considered a neutral relationship between features where one feature is independent of the other and a correlation coefficient of -1 is considered a strong relationship between features where an increase in one feature decreases the other feature.

An effective way to familiarize with a dataset during data analysis is using a **pair plots** (also known as a scatter plot matrix). A pairs plot allows to see both the distribution of single variables and relationships between two variables in a dataset.

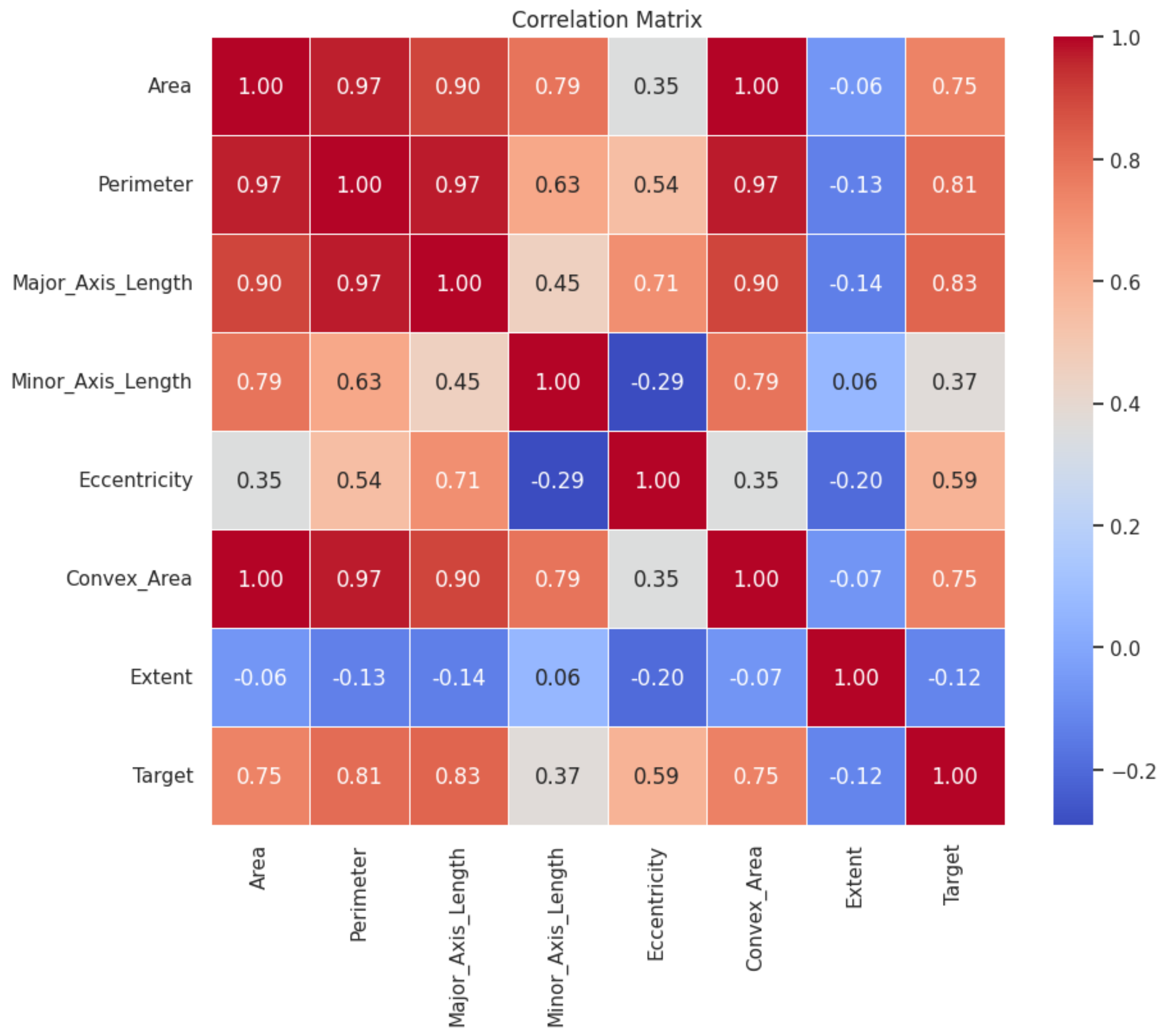
By default, this function will create a grid of Axes such that each numeric variable in data will by shared across the y-axes across a single row and the x-axes across a single column. The diagonal plots are treated differently: a **univariate distribution plot** is drawn to show the marginal distribution of the data in each column.

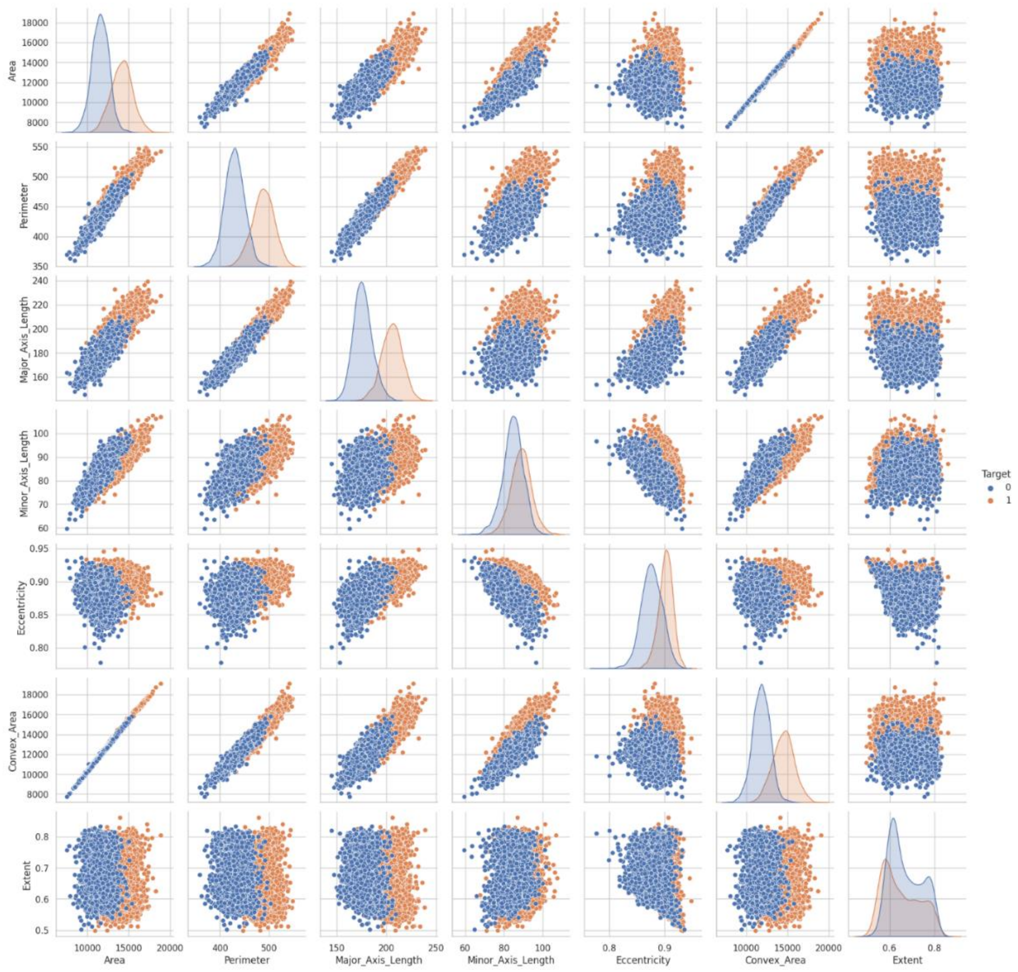


```
#co-relation matrix

correlation_matrix = df.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



```
#Pairwise Plot
sns.pairplot(df,hue='Target')
```





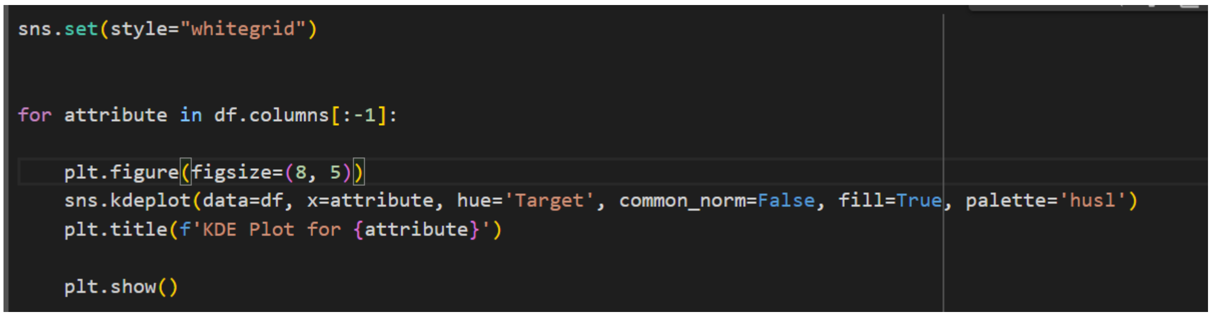
Inferences drawn from the Correlation matrix and pairwise plots:

1. Area and Perimeter are strongly correlated.
2. Area and Major_Axis_Length are strongly correlated.
3. Area and Extent are weakly correlated.
4. Perimeter and Major_Axis_Length are strongly correlated.
5. Perimeter and Convex_Area are strongly correlated.
6. Perimeter and Extent are weakly correlated.
7. Major_Axis_Length and Convex_Area are strongly correlated.
8. Major_Axis_Length and Extent are weakly correlated.
9. Minor_Axis_Length   and Extent are weakly correlated.
10. Eccentricity and Extent are weakly correlated.
11. Target and Extent are weakly correlated.

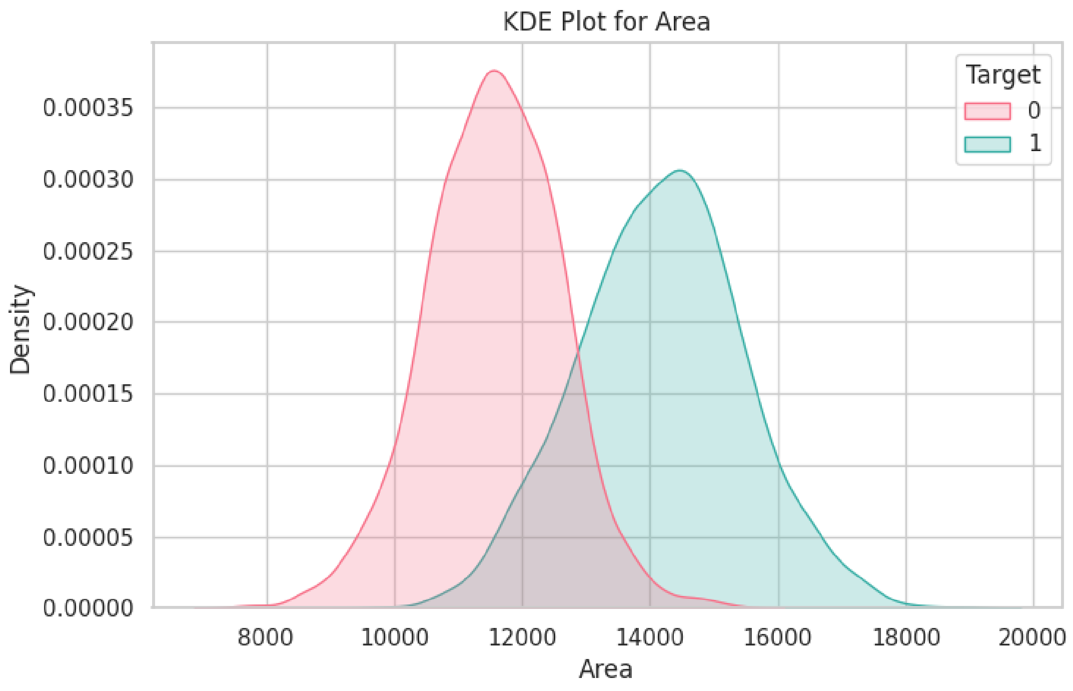
- **Analyzing the strength of each input feature in determining the class using KDE plots**

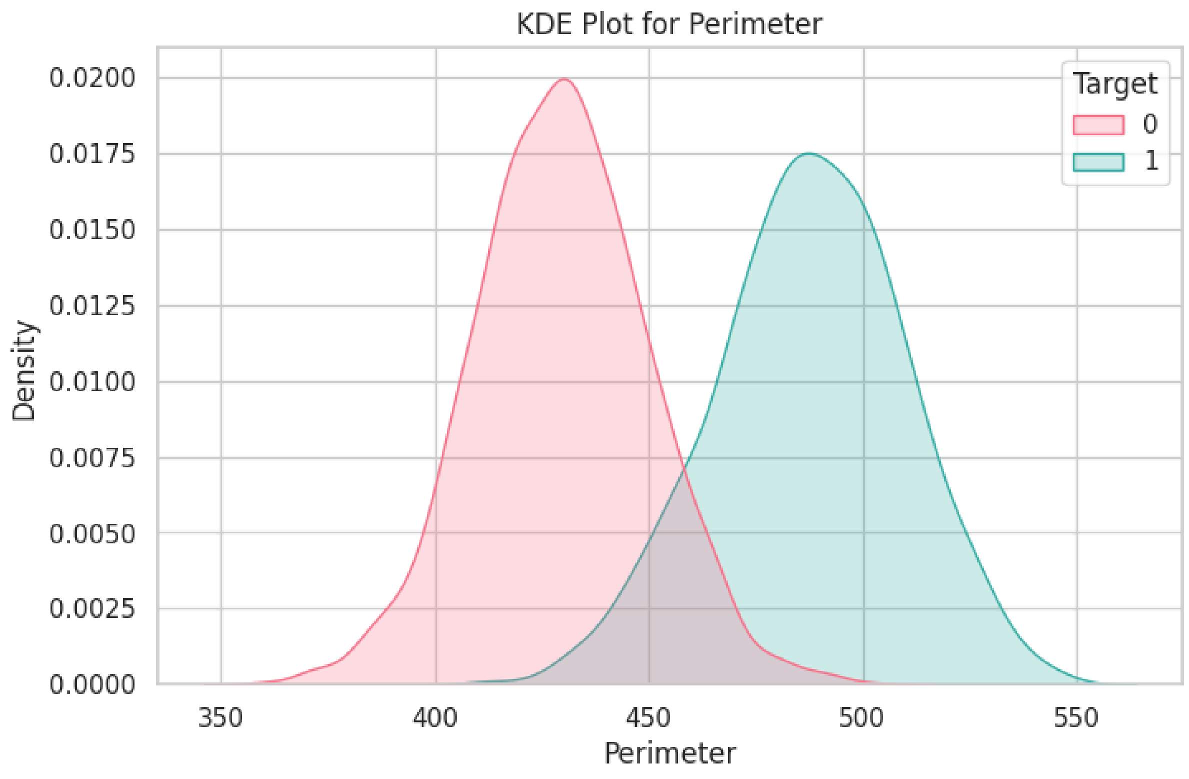
Kernel Density Estimation (KDE) plots are useful for visualizing the distribution of a variable or comparing distributions between different groups or classes. When you have KDE plots for different attributes separated by classes, you can infer several things:

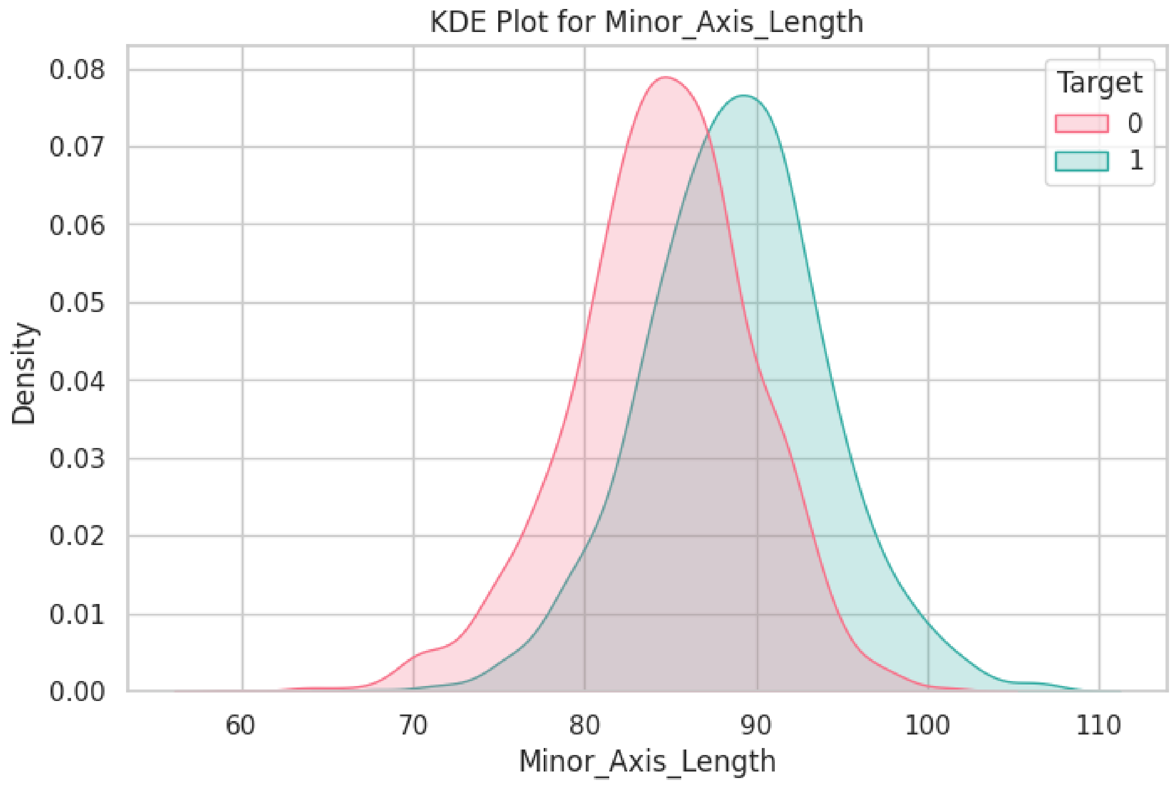
Long tails and unusual patterns in KDE indicate **outliers**. Check for overlap between the KDE plots of different classes. **Overlapping regions** indicate that the values of the corresponding attribute are not very distinctive in separating the classes. On the other hand, clear separation suggests that the attribute is informative for class discrimination.

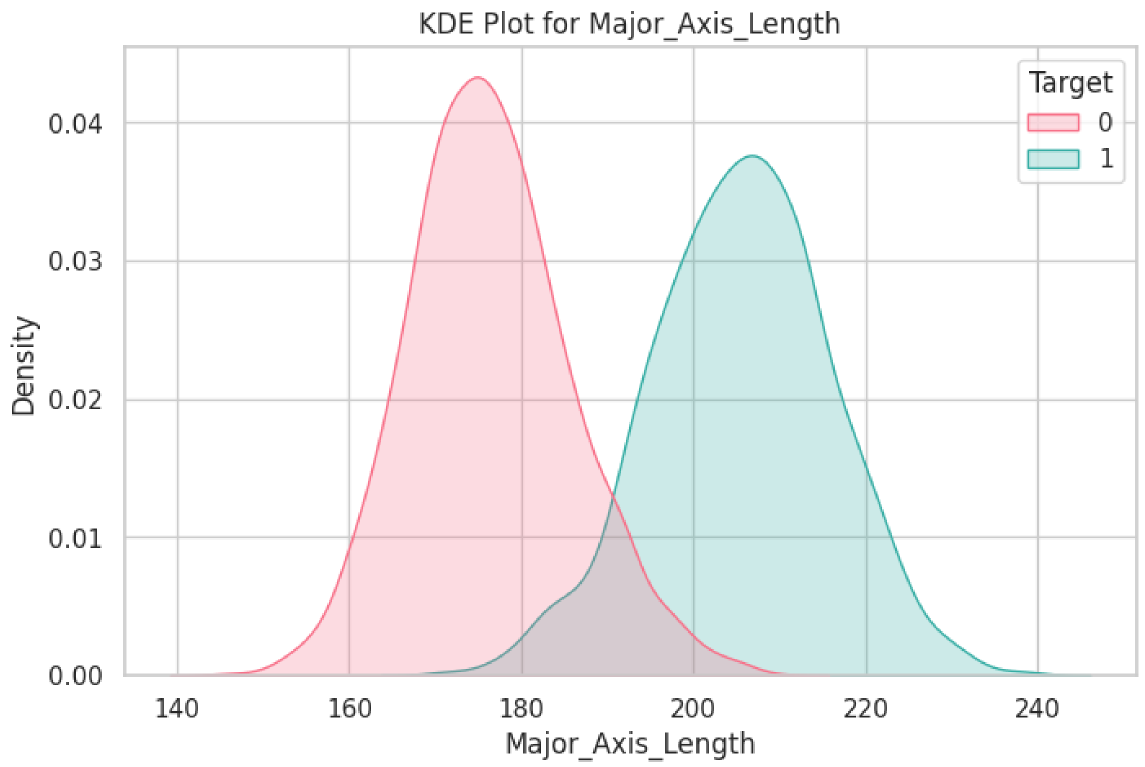


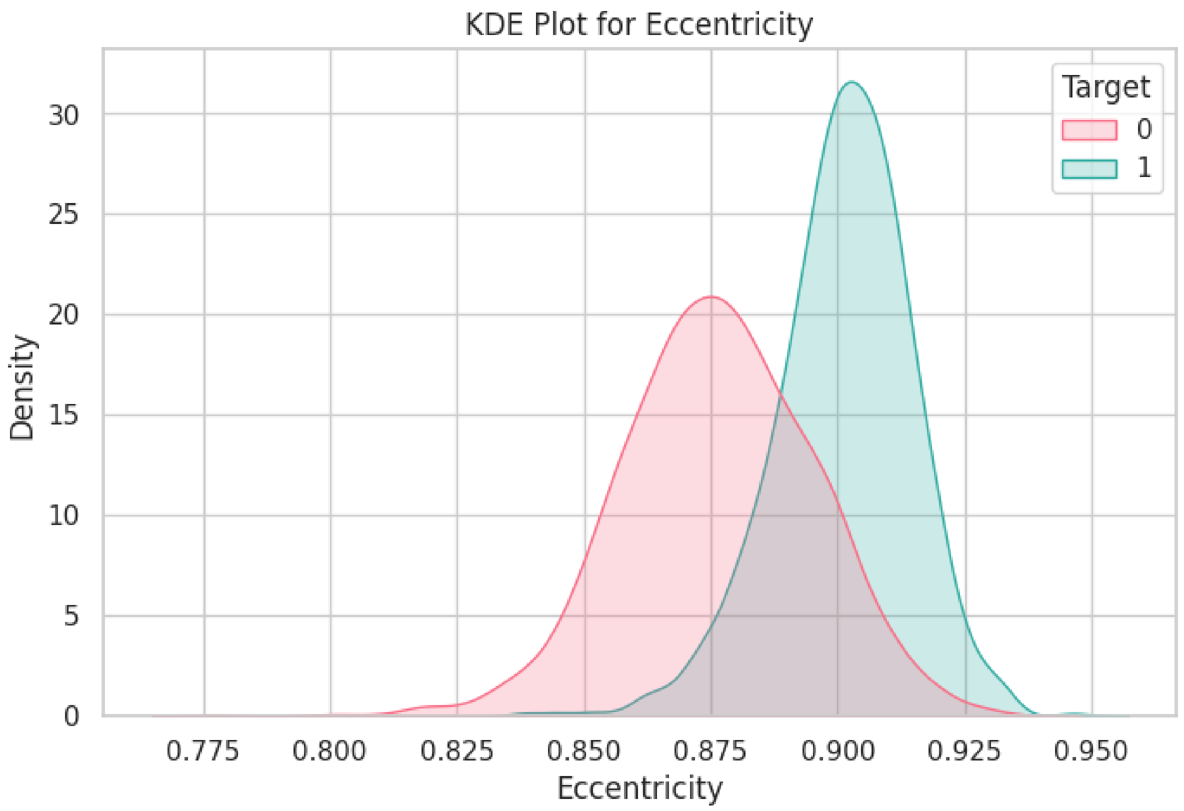
```
sns.set(style="whitegrid")


for attribute in df.columns[:-1]:

    plt.figure(figsize=(8, 5))
    sns.kdeplot(data=df, x=attribute, hue='Target', common_norm=False, fill=True, palette='husl')
    plt.title(f'KDE Plot for {attribute}')

    plt.show()
```

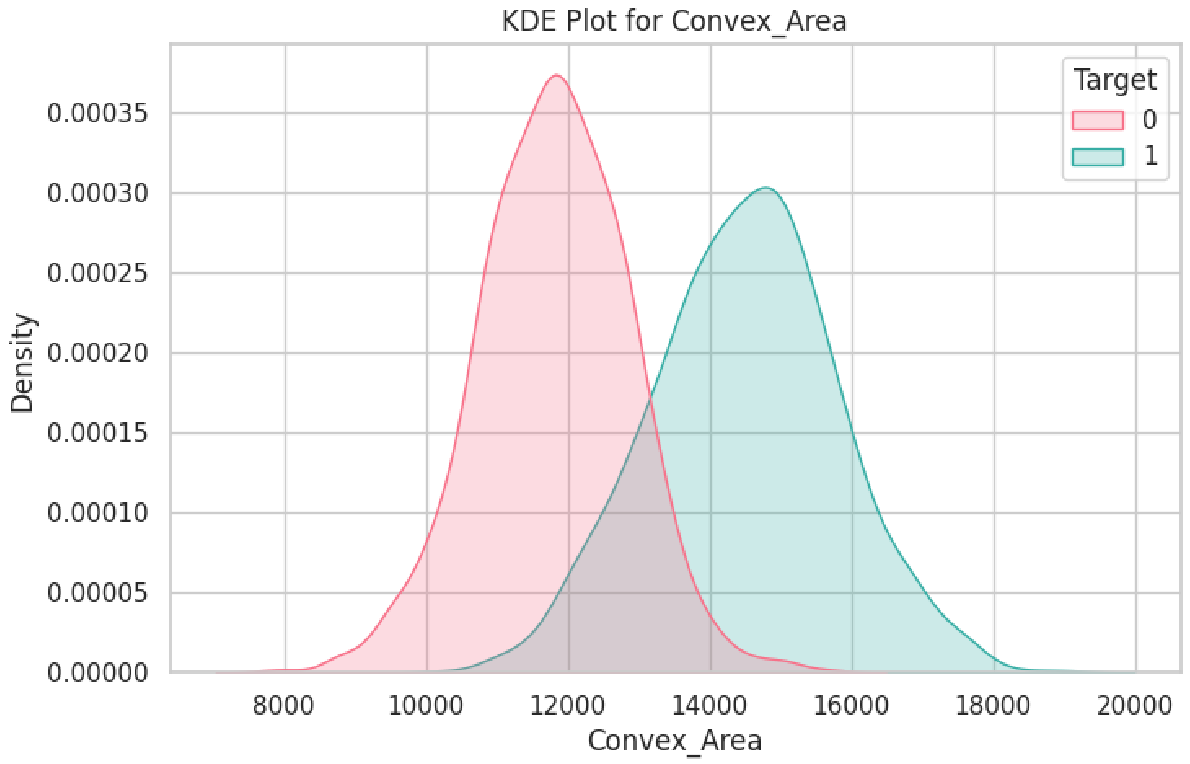


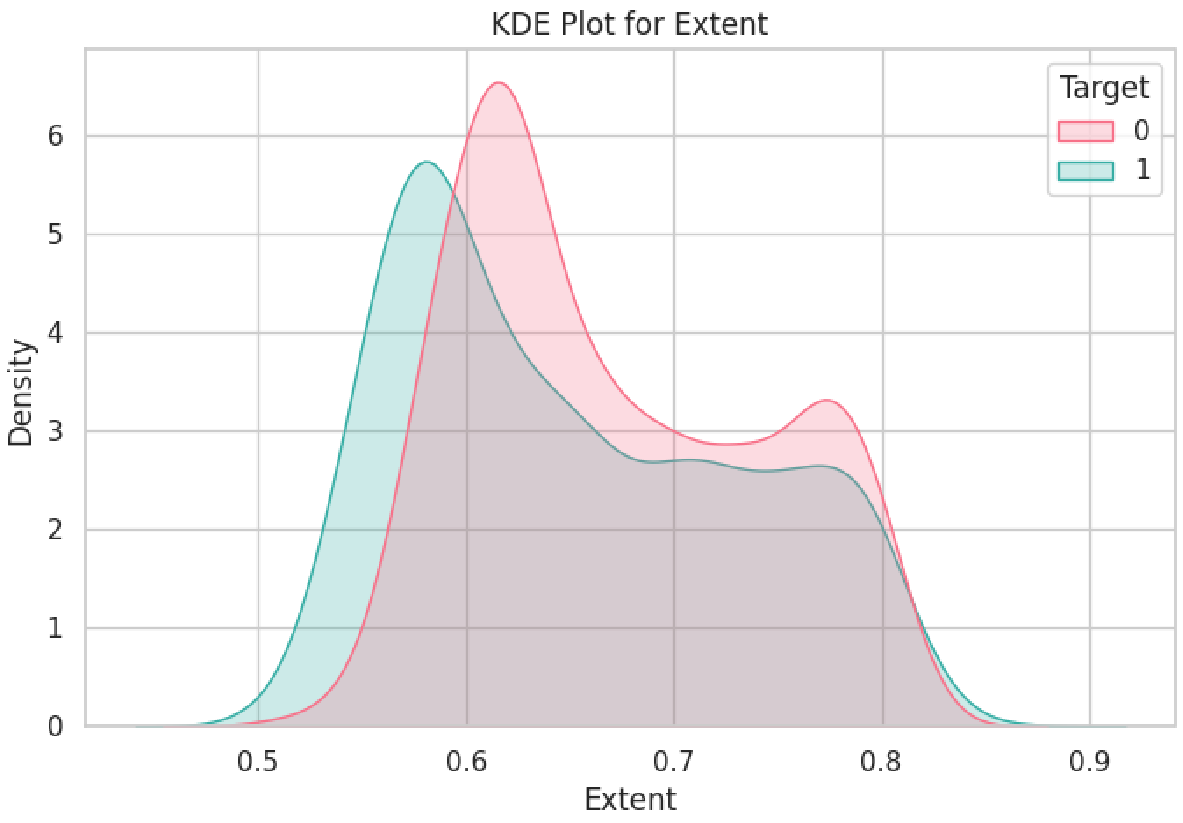

KDE Plot for Perimeter
Density
Perimeter
Target
0
1
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
350
400
450
500
550


KDE Plot for Minor_Axis_Length
Density
Minor_Axis_Length
Target
0
1
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
60
70
80
90
100
110


KDE Plot for Major_Axis_Length
Target
0
1
Density
0.00
0.01
0.02
0.03
0.04
140
160
180
200
220
240
Major_Axis_Length


KDE Plot for Eccentricity
Target
0
1
Density
0
5
10
15
20
25
30
0.775
0.800
0.825
0.850
0.875
0.900
0.925
0.950
Eccentricity


KDE Plot for Convex_Area
Density
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
8000
10000
12000
14000
16000
18000
20000
Convex_Area
Target
0
1


KDE Plot for Extent
Density
6
5
4
3
2
1
0
0.5
0.6
0.7
0.8
0.9
Extent
Target
0
1

- **Key Observation from KDE plots:**

Extent and **Minor Axis Lengths** have a large overlapping area between the classes indicating that their values do not have a lot of importance in determining the class. The same can be inferred from the pair plots as well.

MACHINE LEARNING MODELS FOR CLASSIFICATION

In this project we have used the following classification algorithms for classifying our dataset and tried to analyze their accuracies and find the best suitable classification algorithm for the chosen dataset.

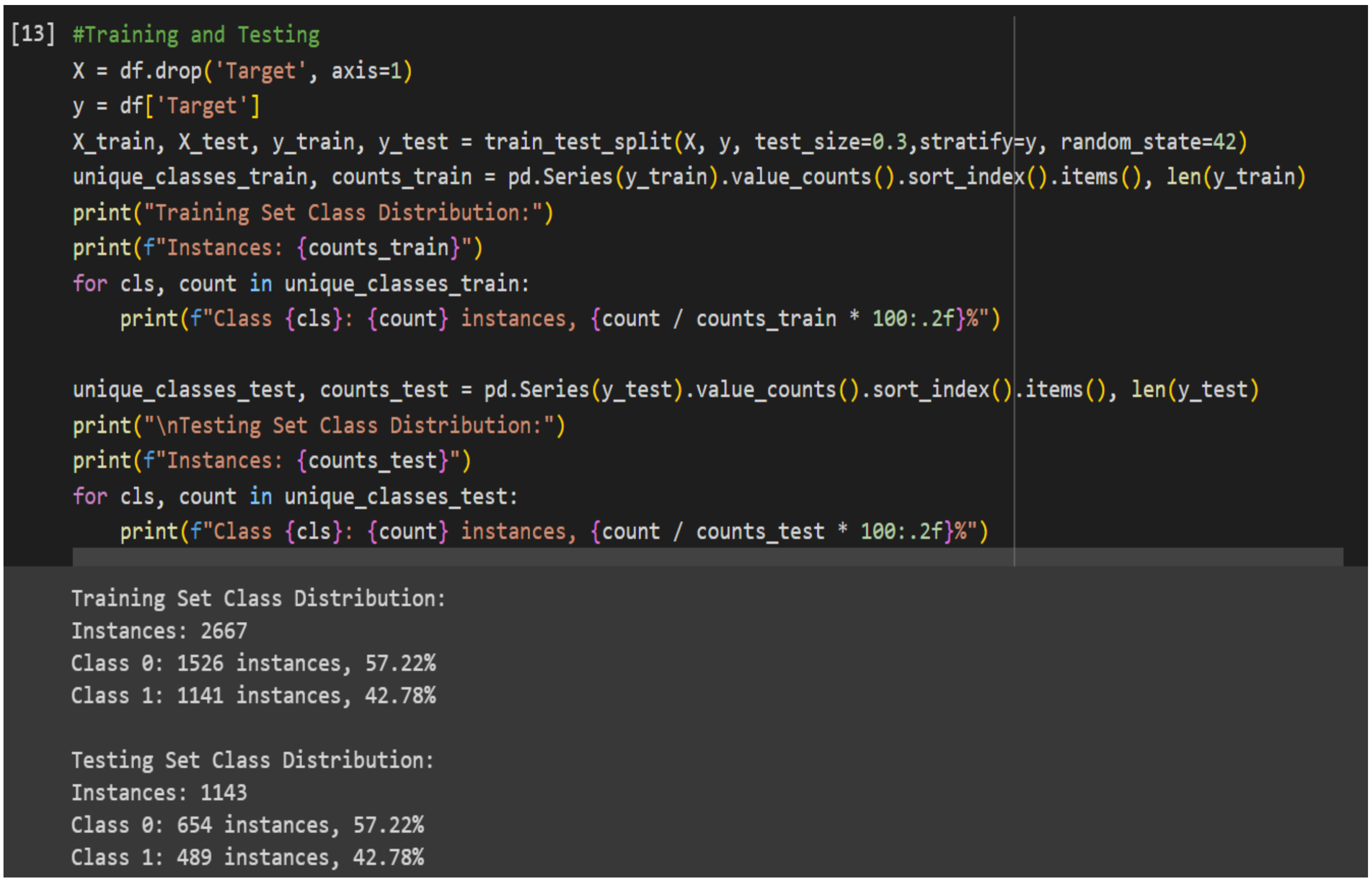
1. **Logistic Regression**
2. **Support Vector Machine**
3. **K-Nearest Neighbor**
4. **Naïve Bayes'**

The Performance Parameters that we have used to judge the effectiveness of a model are as follows:

1. **Accuracy** - measures the overall correctness of the model's predictions.
2. **Precision** - focuses on the proportion of true positive predictions among all positive predictions made by the model.
3. **Recall** - Recall (also known as sensitivity or true positive rate) measures the proportion of actual positive instances that the model correctly predicts.

4. **F1 Score** - It is the harmonic mean of precision and recall. F1 score= 2*precision*recall/(precision+recall).
5. **Support** – The proportion of true occurrence of a particular class within a dataset.

We have used **holdout method** to divide our data set into two parts namely training set and test set. In holdout method given data is randomly portioned into two independent sets. We have given **test_size parameter** as 0.3 implying test set should have 30% of entries from the whole dataset and training set to have remaining 70% of entries. The **stratify parameter** ensures that the class distribution in the training and testing sets is approximately the same as the class distribution in the original dataset.



```
[13] #Training and Testing
     X = df.drop('Target', axis=1)
     y = df['Target']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,stratify=y, random_state=42)
     unique_classes_train, counts_train = pd.Series(y_train).value_counts().sort_index().items(), len(y_train)
     print("Training Set Class Distribution:")
     print(f"Instances: {counts_train}")
     for cls, count in unique_classes_train:
         print(f"Class {cls}: {count} instances, {count / counts_train * 100:.2f}%")

     unique_classes_test, counts_test = pd.Series(y_test).value_counts().sort_index().items(), len(y_test)
     print("\nTesting Set Class Distribution:")
     print(f"Instances: {counts_test}")
     for cls, count in unique_classes_test:
         print(f"Class {cls}: {count} instances, {count / counts_test * 100:.2f}%")
```

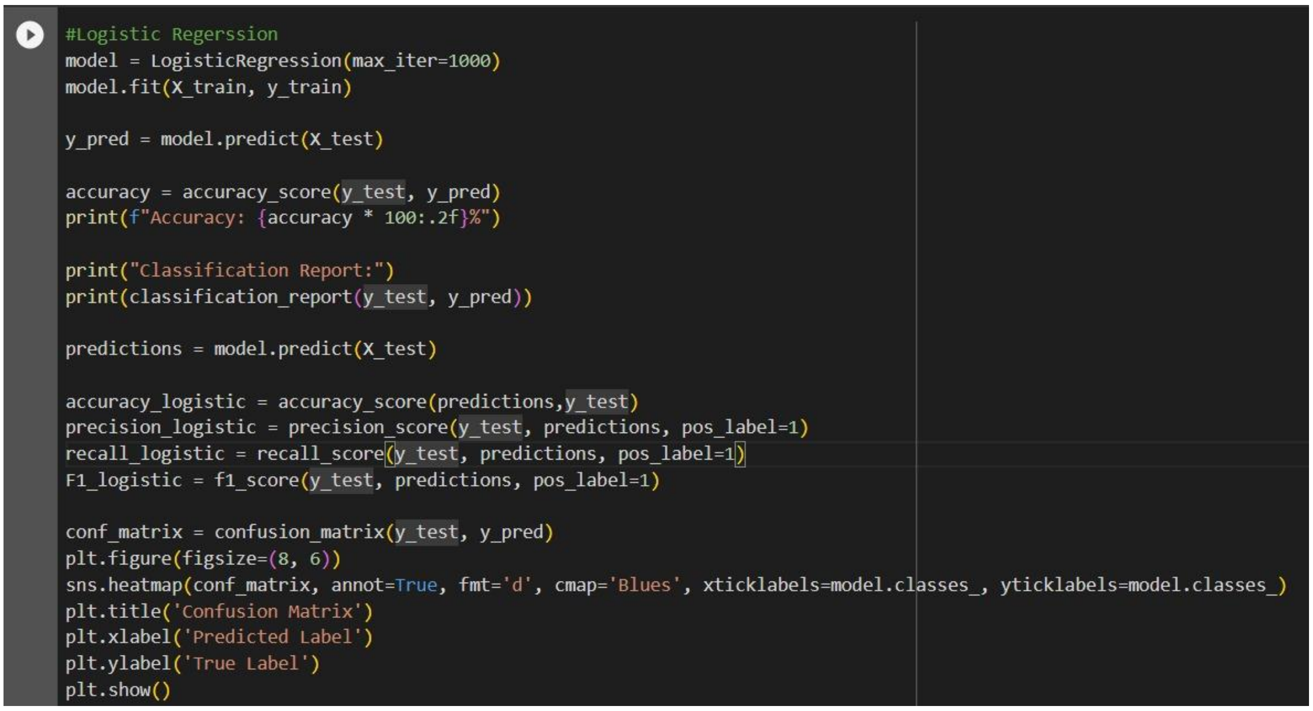
```
Training Set Class Distribution:
Instances: 2667
Class 0: 1526 instances, 57.22%
Class 1: 1141 instances, 42.78%

Testing Set Class Distribution:
Instances: 1143
Class 0: 654 instances, 57.22%
Class 1: 489 instances, 42.78%
```

1.Logistic Regression

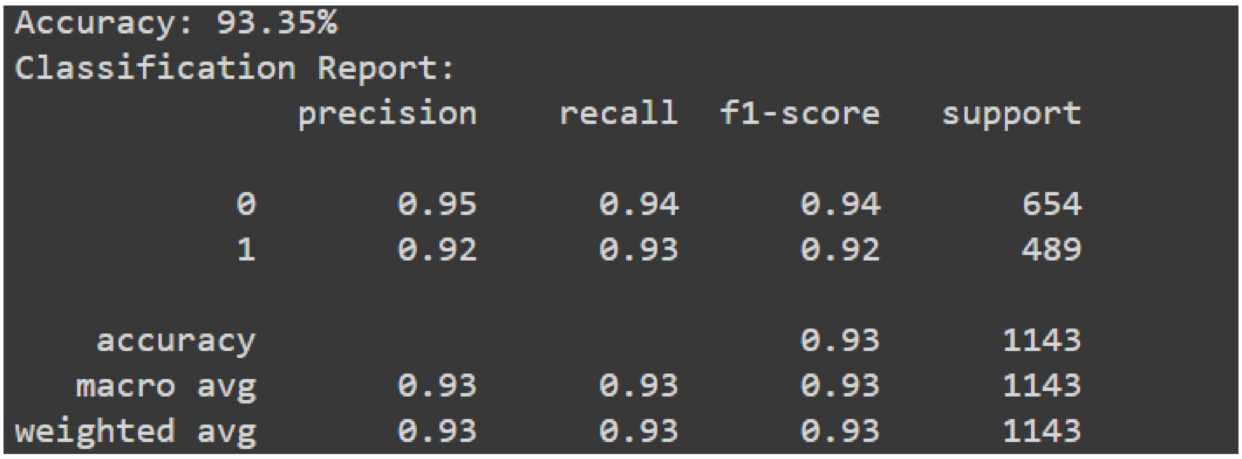
Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where it predicts the probability that an instance of belonging to a given class or not. It uses **sigmoid function** to assign those probabilities. The sigmoid function is a mathematical function used to map the predicted values to probabilities. It maps any real value into another value within a range of 0 and 1.

<u>Reasons for choosing Logistic Regression:</u>

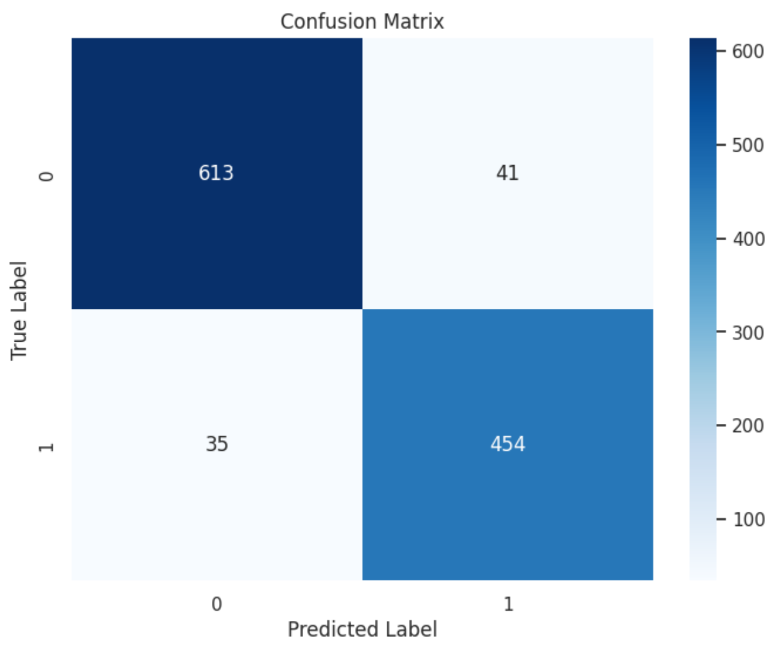
Logistic Regression has good performance for binary classification than multiclass classification and our dataset also has only two classes. Logistic Regression also has good performance for both linearly separable classes and non-linear separable classes and hence we have applied it for classification of our dataset.



```
#Logistic Regerssion
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("Classification Report:")
print(classification_report(y_test, y_pred))

predictions = model.predict(X_test)

accuracy_logistic = accuracy_score(predictions,y_test)
precision_logistic = precision_score(y_test, predictions, pos_label=1)
recall_logistic = recall_score(y_test, predictions, pos_label=1)
F1_logistic = f1_score(y_test, predictions, pos_label=1)

conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_, yticklabels=model.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



```
Accuracy: 93.35%
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.94      0.94       654
           1       0.92      0.93      0.92       489

    accuracy                           0.93      1143
   macro avg       0.93      0.93      0.93      1143
weighted avg       0.93      0.93      0.93      1143
```

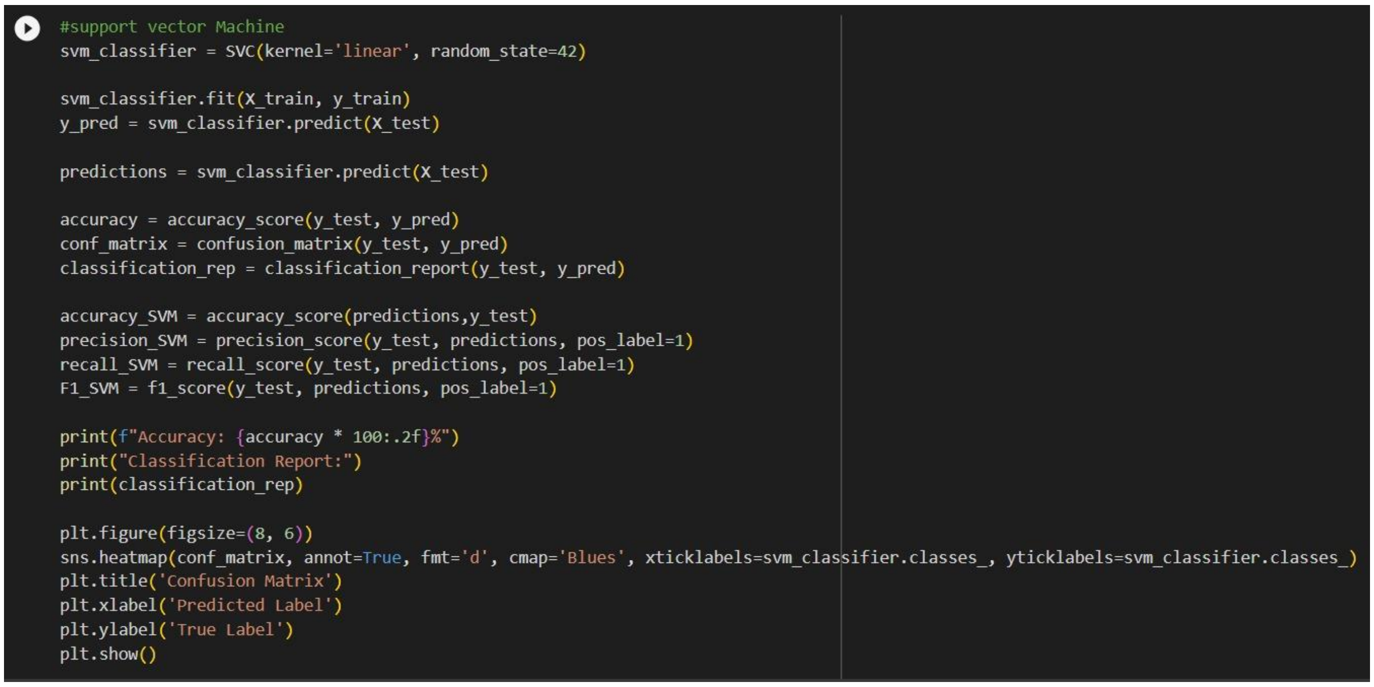


2.Support Vector Machine

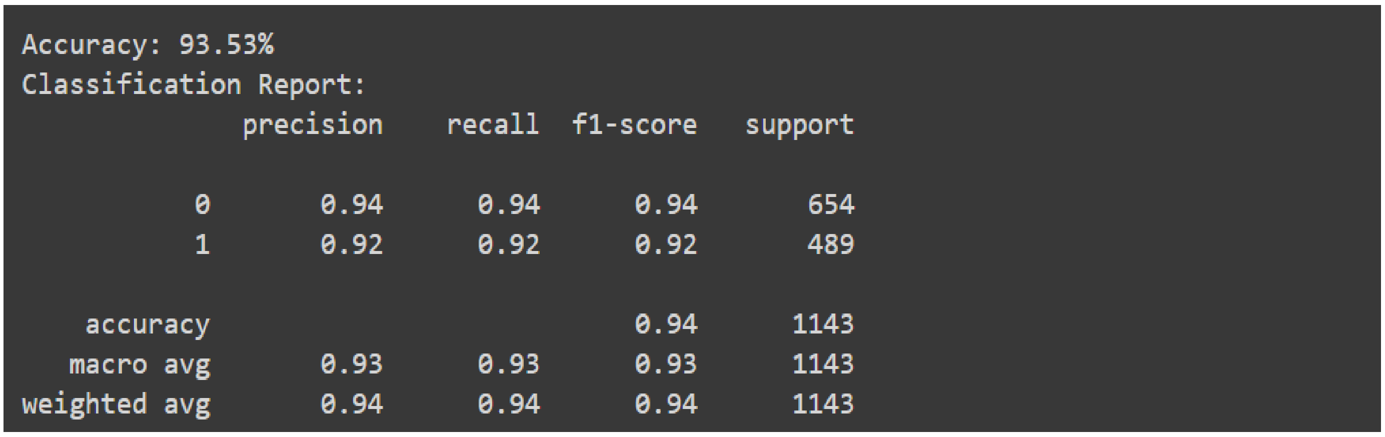
Support Vector Machine or SVM is a supervised and linear Machine Learning algorithm primarily used for solving classification problems as well as regression problems and is also referred to as Support Vector Classification. SVM also supports the kernel method called the kernel SVM, which allows us to tackle non-linearity. SVM chooses the extreme points/vectors that help create a hyperplane to differentiate between the different categories.

Reasons for choosing SVM:

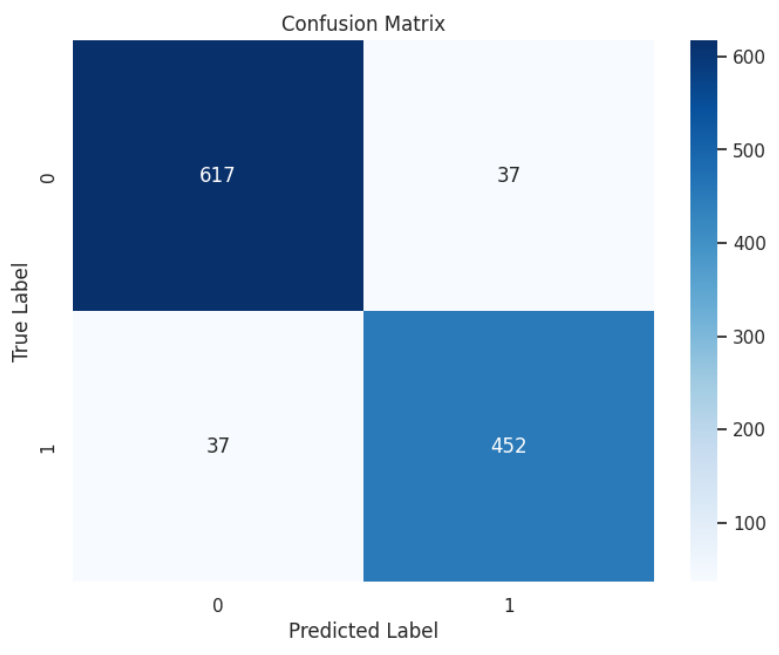
SVM is **less sensitive** to outliers and noise and from box plots we can observe that there are quite a few outliers in our dataset. SVM is also very effective in high dimensional spaces and hence we have used it for our dataset.



```
#support vector Machine
svm_classifier = SVC(kernel='linear', random_state=42)

svm_classifier.fit(X_train, y_train)
y_pred = svm_classifier.predict(X_test)

predictions = svm_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

accuracy_SVM = accuracy_score(predictions,y_test)
precision_SVM = precision_score(y_test, predictions, pos_label=1)
recall_SVM = recall_score(y_test, predictions, pos_label=1)
F1_SVM = f1_score(y_test, predictions, pos_label=1)

print(f"Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:")
print(classification_rep)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=svm_classifier.classes_, yticklabels=svm_classifier.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



```
Accuracy: 93.53%
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       654
           1       0.92      0.92      0.92       489

    accuracy                           0.94      1143
   macro avg       0.93      0.93      0.93      1143
weighted avg       0.94      0.94      0.94      1143
```

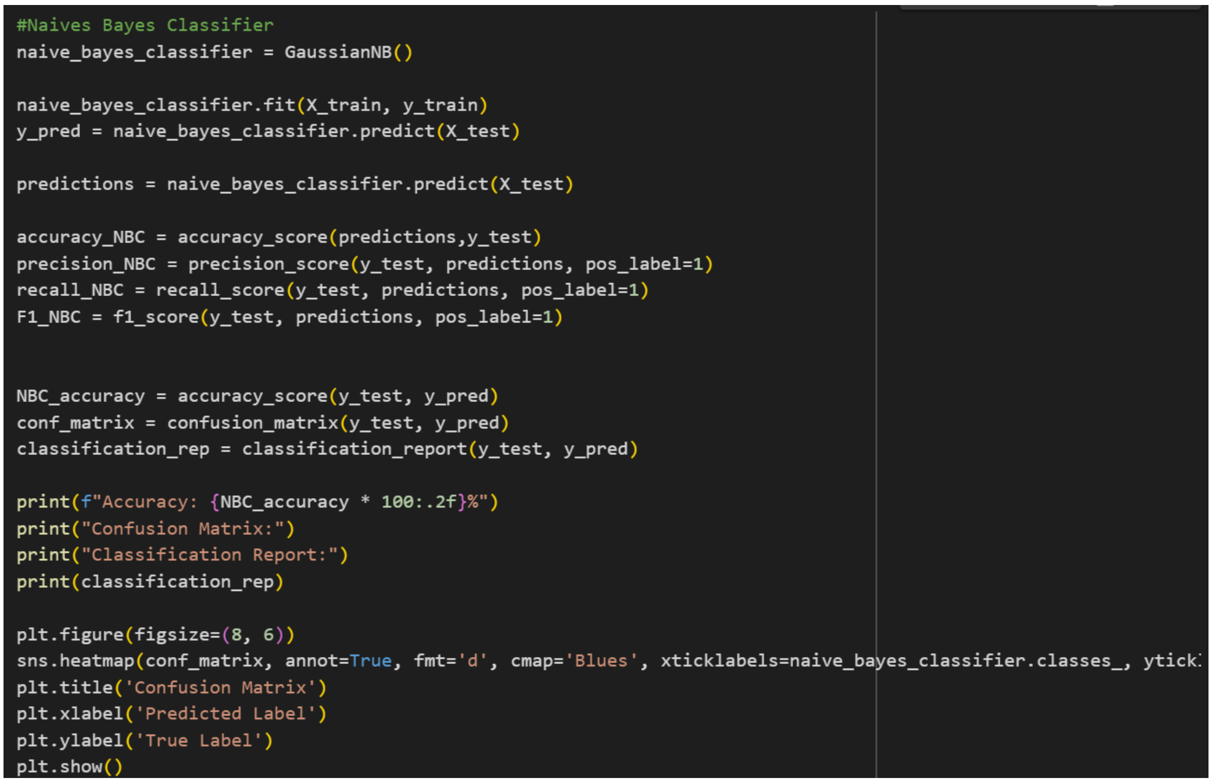


3.Naive Bayes’

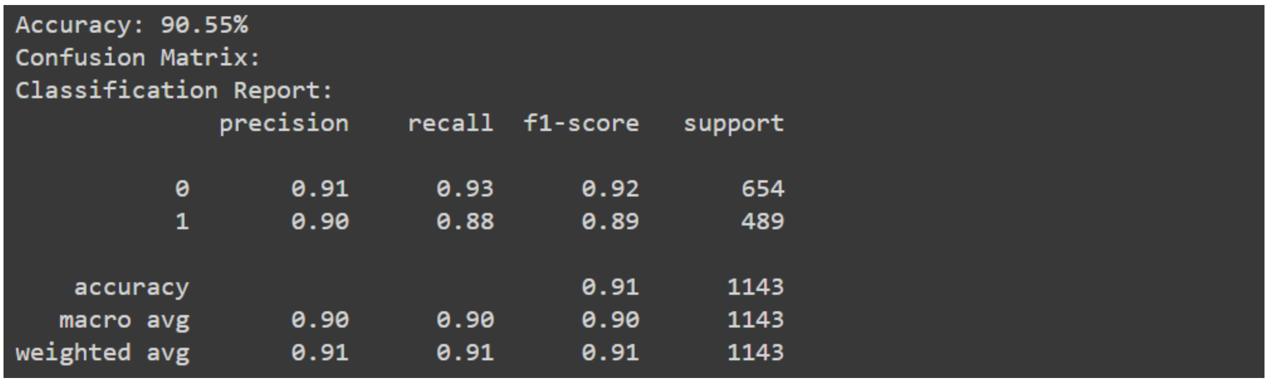
Naive Bayes is a Machine Learning model used for large data. The classifier works on the Bayes Theorem. Prediction of probabilities of every data set member in a class is made and the class with highest probability is chosen as the most suitable class. The assumption in distribution of $P(x_i | y)$ give rise to different NBM. For example, assuming Gaussian distribution will give rise to Gaussian Naive Bayes (GNB).

Reasons for choosing Naïve Bayes’:

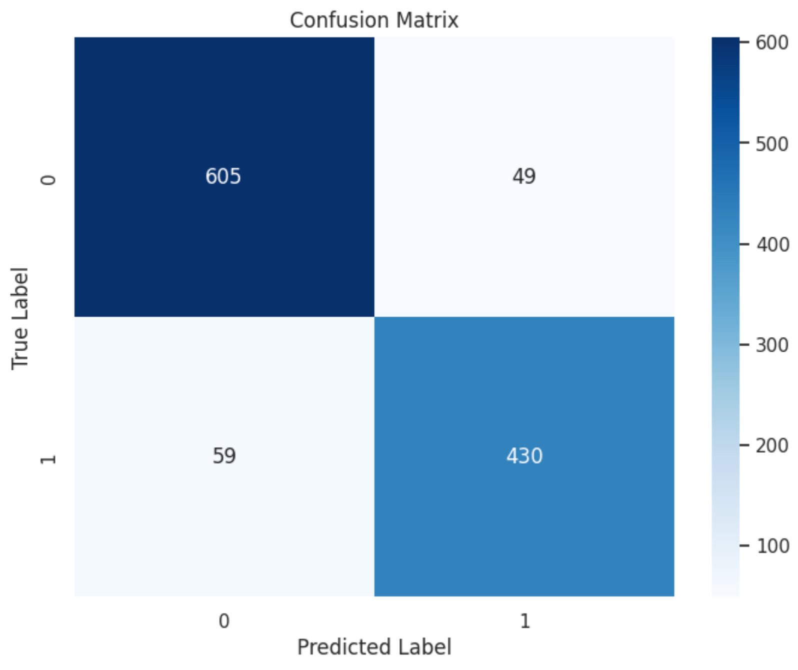
It requires a small amount of training data to learn the parameters. Can be trained relatively fast compared to sophisticated models. Naive Bayes is **less sensitive** to irrelevant features. Even if some features are not informative or are redundant, the classifier can still perform well and from KDE plots we have inferred that Extent and Minor Axis Length are less important features in our dataset.



```
#Naives Bayes Classifier
naive_bayes_classifier = GaussianNB()

naive_bayes_classifier.fit(X_train, y_train)
y_pred = naive_bayes_classifier.predict(X_test)

predictions = naive_bayes_classifier.predict(X_test)

accuracy_NBC = accuracy_score(predictions,y_test)
precision_NBC = precision_score(y_test, predictions, pos_label=1)
recall_NBC = recall_score(y_test, predictions, pos_label=1)
F1_NBC = f1_score(y_test, predictions, pos_label=1)


NBC_accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f"Accuracy: {NBC_accuracy * 100:.2f}%")
print("Confusion Matrix:")
print("Classification Report:")
print(classification_rep)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=naive_bayes_classifier.classes_, ytick
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



```
Accuracy: 90.55%
Confusion Matrix:
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       654
           1       0.90      0.88      0.89       489

    accuracy                           0.91      1143
   macro avg       0.90      0.90      0.90      1143
weighted avg       0.91      0.91      0.91      1143
```

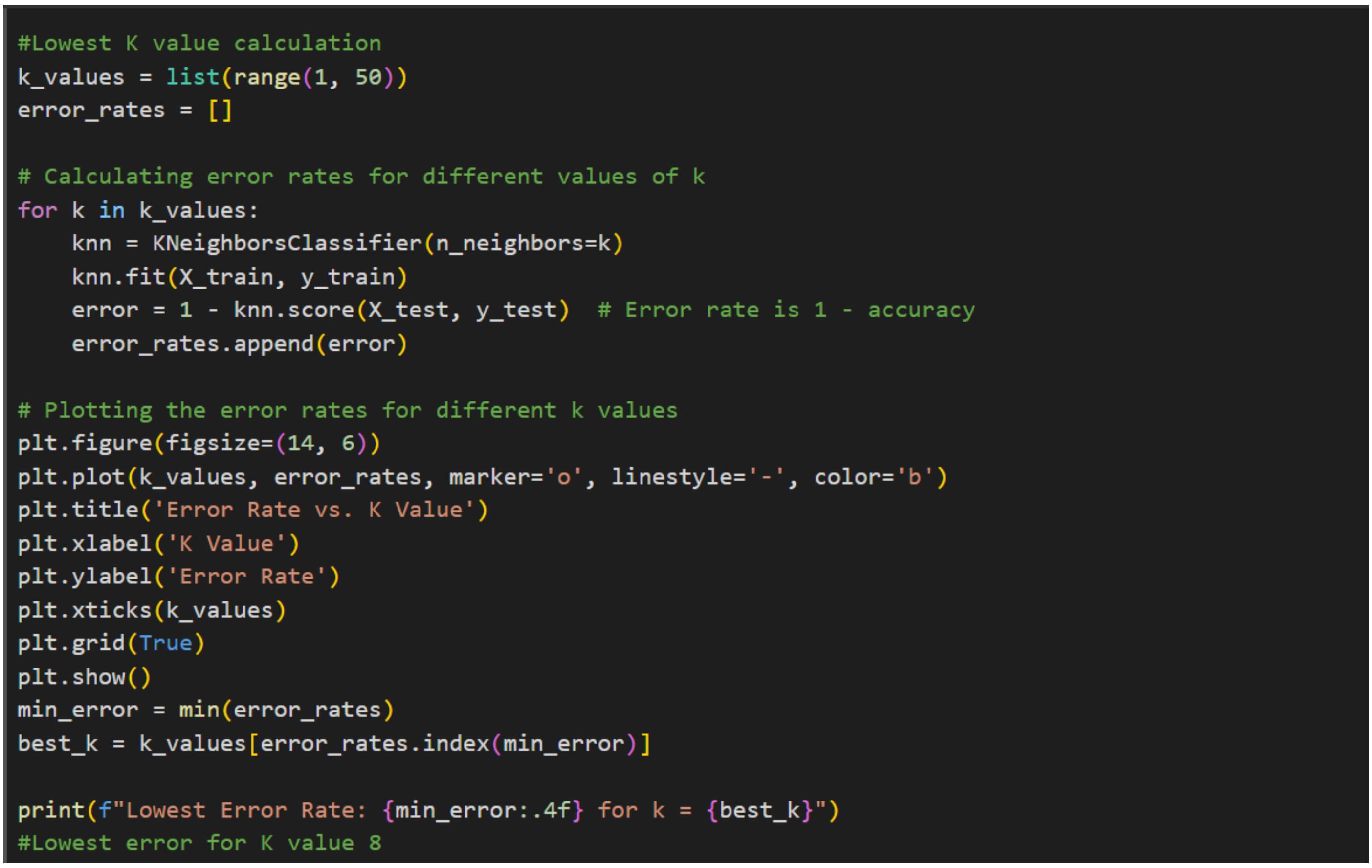


4. K-Nearest Neighbors

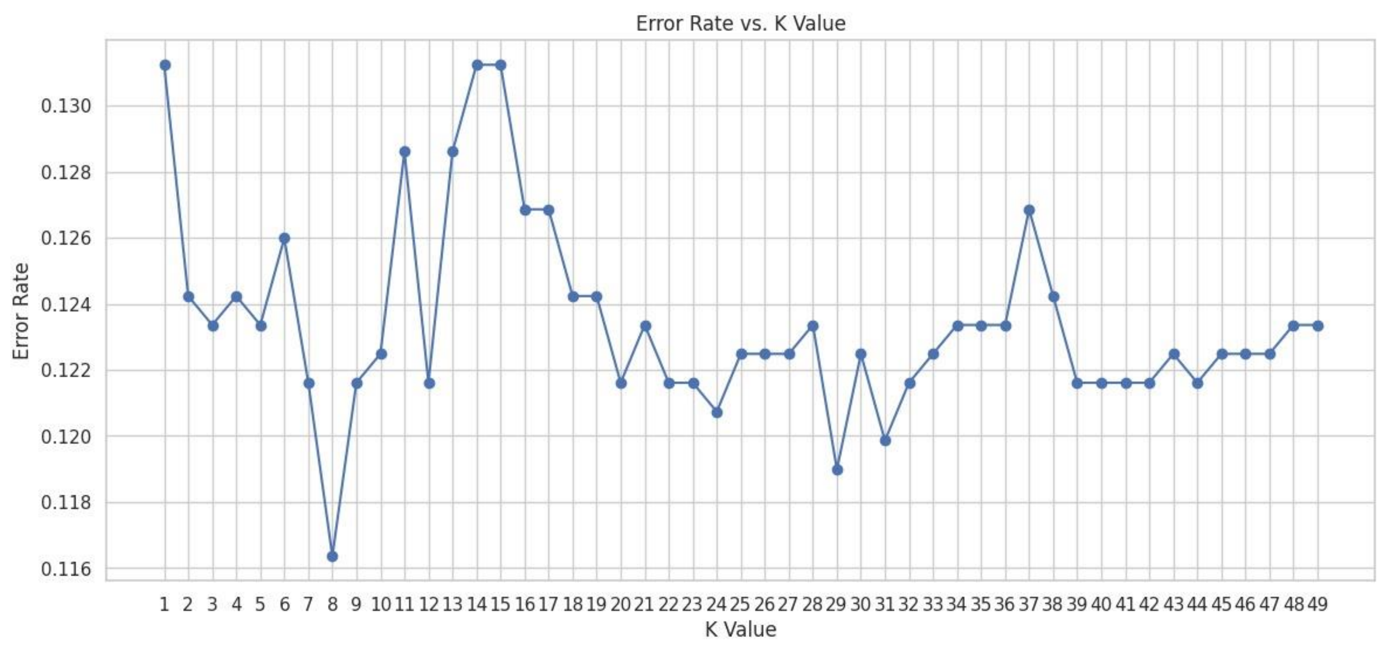
K-Nearest Neighbour is a simple Machine Learning algorithm based on Supervised Learning. It is mainly used for Classification problems but can also be used for Regression.

It is also called a **lazy learner algorithm** because it does not learn from the training set immediately; instead, it stores the dataset and acts on the dataset at the time of classification. K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data. One disadvantage of KNN is that we have to manually give a value for k and to find an optimal k value is a computationally intensive process as we have to try all possible k values and choose the one with least error rate. Below is the code implementation for the same.

Optimal Value of K for our Dataset is found to be 8 .

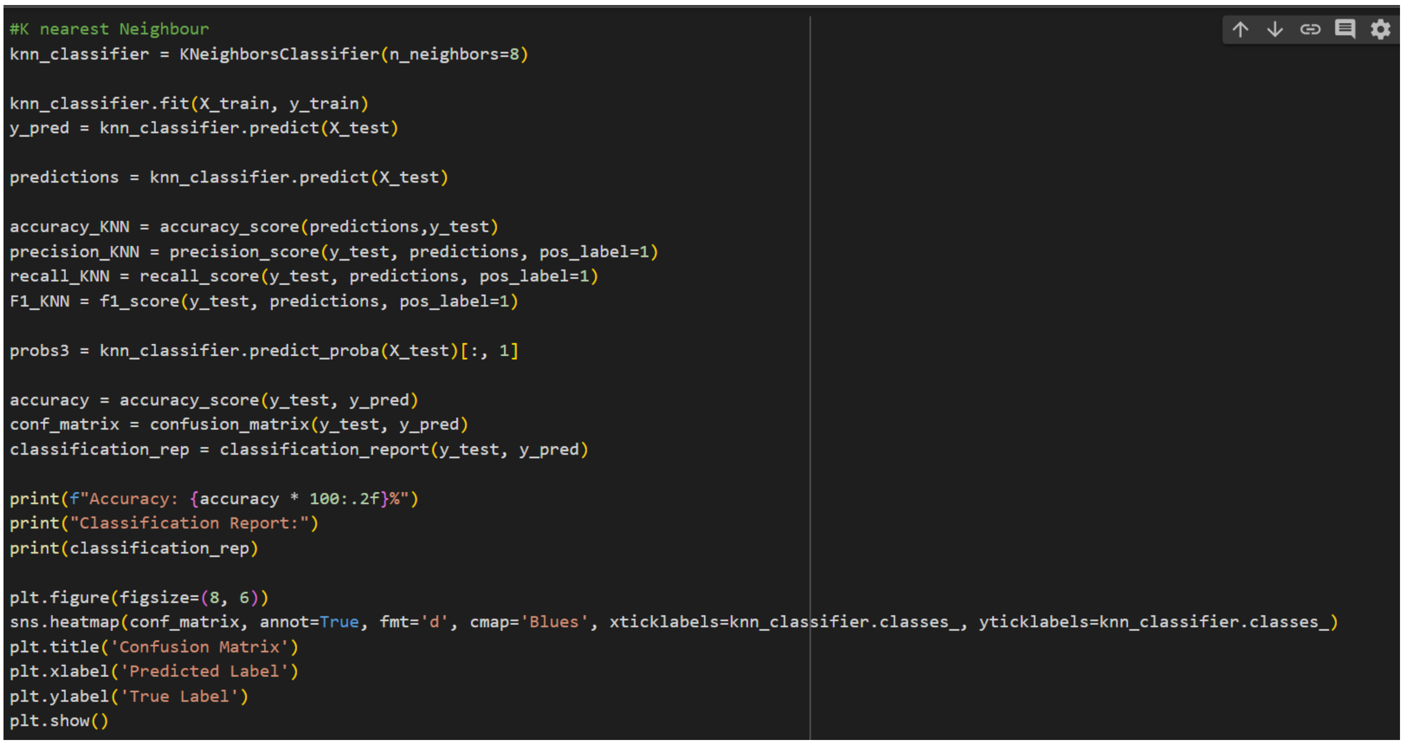


```
#Lowest K value calculation
k_values = list(range(1, 50))
error_rates = []

# Calculating error rates for different values of k
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    error = 1 - knn.score(X_test, y_test)  # Error rate is 1 - accuracy
    error_rates.append(error)

# Plotting the error rates for different k values
plt.figure(figsize=(14, 6))
plt.plot(k_values, error_rates, marker='o', linestyle='-', color='b')
plt.title('Error Rate vs. K Value')
plt.xlabel('K Value')
plt.ylabel('Error Rate')
plt.xticks(k_values)
plt.grid(True)
plt.show()
min_error = min(error_rates)
best_k = k_values[error_rates.index(min_error)]

print(f"Lowest Error Rate: {min_error:.4f} for k = {best_k}")
#Lowest error for K value 8
```

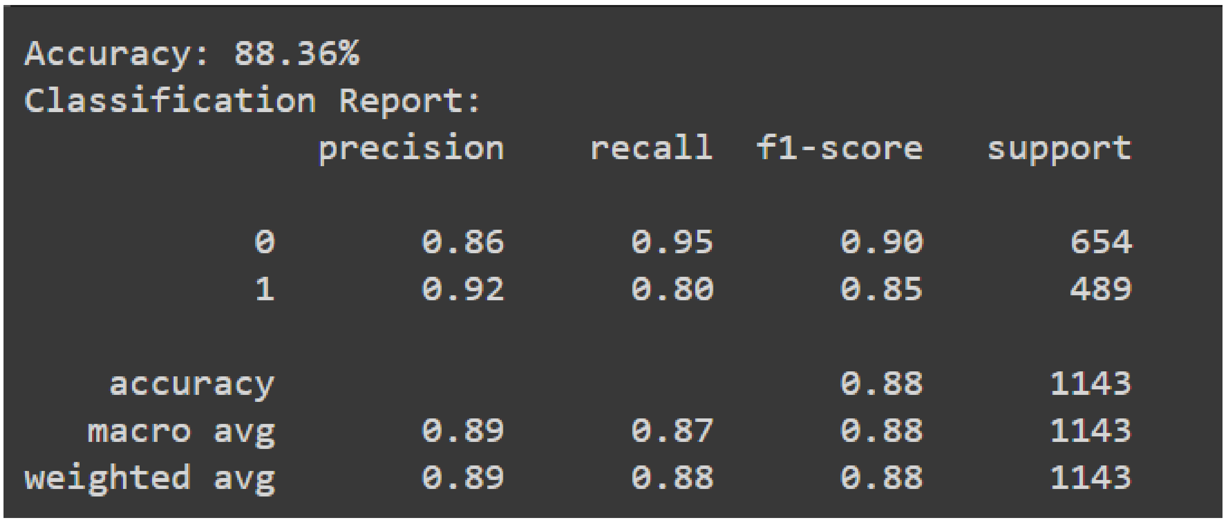


Reasons for choosing KNN:

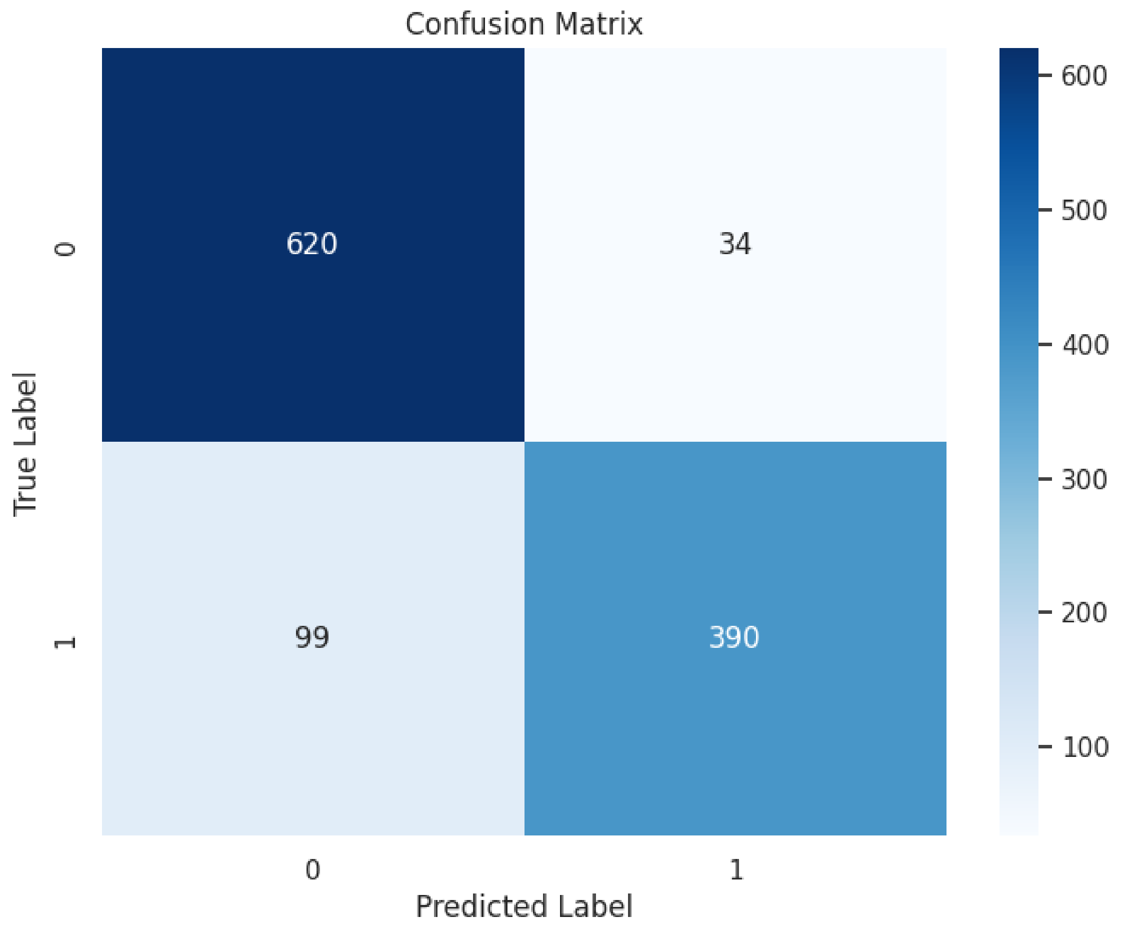
It can handle both numerical and categorical features, and can be adapted to different types of problems. Furthermore, KNN is able to capture **complex and nonlinear patterns** in the data, and is robust to noise and outliers if a large enough k is used.



```
#K nearest Neighbour
knn_classifier = KNeighborsClassifier(n_neighbors=8)

knn_classifier.fit(X_train, y_train)
y_pred = knn_classifier.predict(X_test)

predictions = knn_classifier.predict(X_test)

accuracy_KNN = accuracy_score(predictions,y_test)
precision_KNN = precision_score(y_test, predictions, pos_label=1)
recall_KNN = recall_score(y_test, predictions, pos_label=1)
F1_KNN = f1_score(y_test, predictions, pos_label=1)

probs3 = knn_classifier.predict_proba(X_test)[:, 1]

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:")
print(classification_rep)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=knn_classifier.classes_, yticklabels=knn_classifier.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



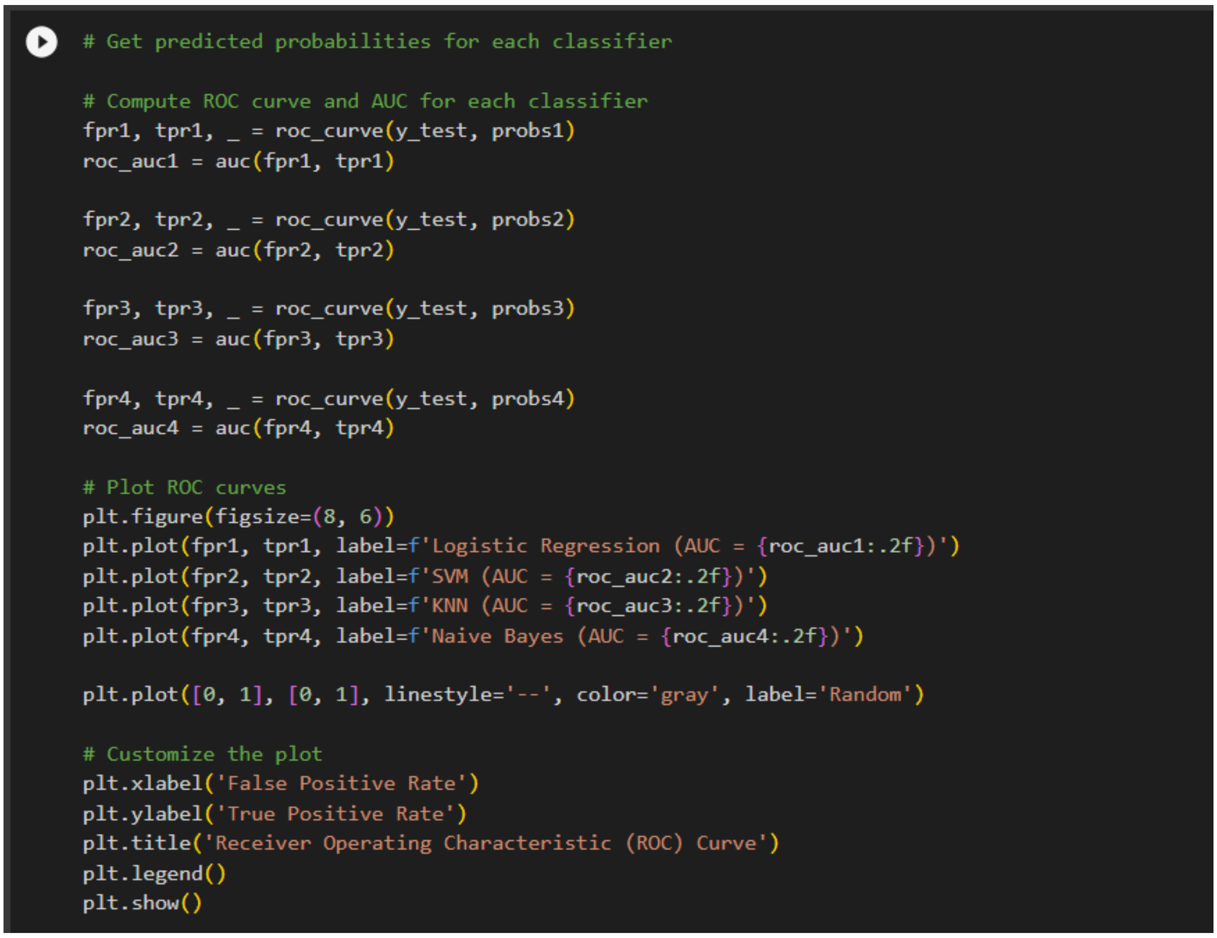
```
Accuracy: 88.36%
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.95      0.90       654
           1       0.92      0.80      0.85       489

    accuracy                           0.88      1143
   macro avg       0.89      0.87      0.88      1143
weighted avg       0.89      0.88      0.88      1143
```



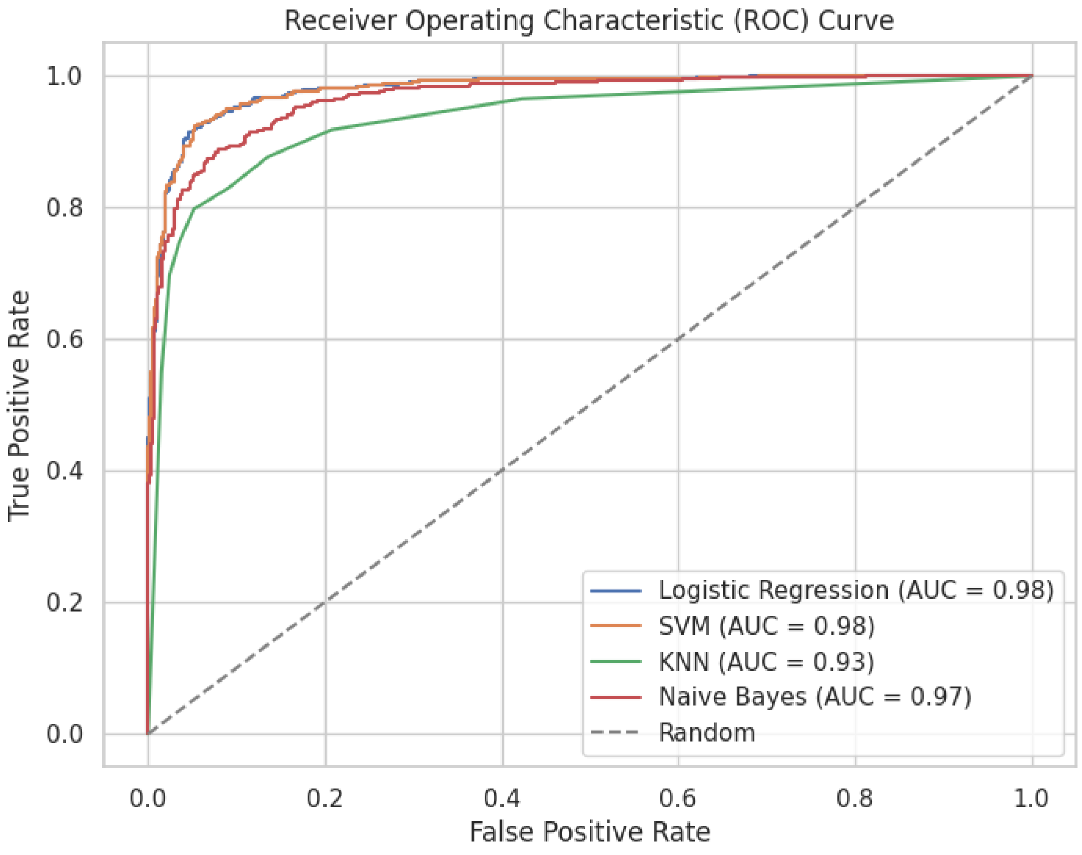
CONCLUSION

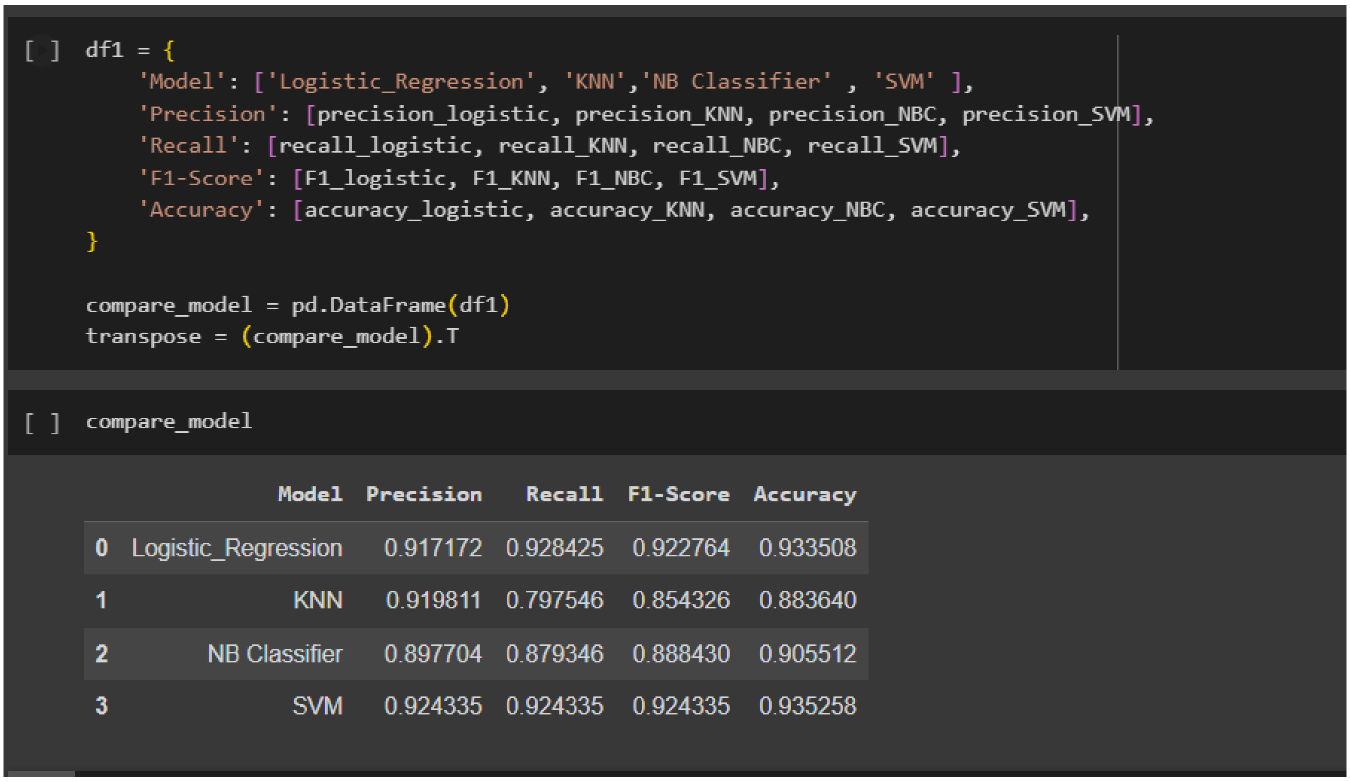
Receiver Operating Characteristics(ROC) curves used for Visual comparison of classification models. It shows the trade-off between the **true positive rate** and the **false positive rate**. The area under the ROC curve is a measure of the **accuracy** of the model. The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model. A model with perfect accuracy will have an area of 1.0.

Inference drawn from ROC plots is that among all classifiers **support vector machine and logistic regression** have highest Area under the curve (**AUC =0.98**) and **KNN** has least among them (**AUC=0.93**) implying that these models are best suited for our Dataset.



```python
# Get predicted probabilities for each classifier

# Compute ROC curve and AUC for each classifier
fpr1, tpr1, _ = roc_curve(y_test, probs1)
roc_auc1 = auc(fpr1, tpr1)

fpr2, tpr2, _ = roc_curve(y_test, probs2)
roc_auc2 = auc(fpr2, tpr2)

fpr3, tpr3, _ = roc_curve(y_test, probs3)
roc_auc3 = auc(fpr3, tpr3)

fpr4, tpr4, _ = roc_curve(y_test, probs4)
roc_auc4 = auc(fpr4, tpr4)

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr1, tpr1, label=f'Logistic Regression (AUC = {roc_auc1:.2f})')
plt.plot(fpr2, tpr2, label=f'SVM (AUC = {roc_auc2:.2f})')
plt.plot(fpr3, tpr3, label=f'KNN (AUC = {roc_auc3:.2f})')
plt.plot(fpr4, tpr4, label=f'Naive Bayes (AUC = {roc_auc4:.2f})')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')

# Customize the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

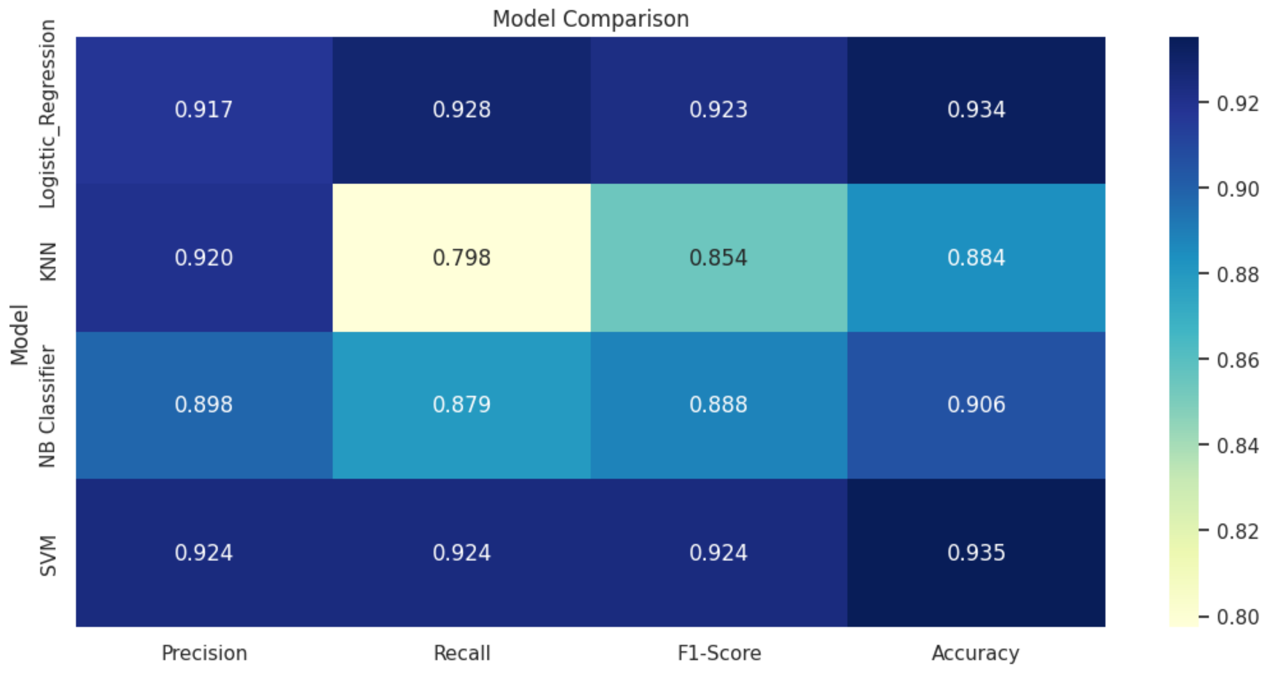




```
[ ]  df1 = {
         'Model': ['Logistic_Regression', 'KNN','NB Classifier' , 'SVM' ],
         'Precision': [precision_logistic, precision_KNN, precision_NBC, precision_SVM],
         'Recall': [recall_logistic, recall_KNN, recall_NBC, recall_SVM],
         'F1-Score': [F1_logistic, F1_KNN, F1_NBC, F1_SVM],
         'Accuracy': [accuracy_logistic, accuracy_KNN, accuracy_NBC, accuracy_SVM],
     }

     compare_model = pd.DataFrame(df1)
     transpose = (compare_model).T
```

```
[ ]  compare_model
```

	Model	Precision	Recall	F1-Score	Accuracy
0	Logistic_Regression	0.917172	0.928425	0.922764	0.933508
1	KNN	0.919811	0.797546	0.854326	0.883640
2	NB Classifier	0.897704	0.879346	0.888430	0.905512
3	SVM	0.924335	0.924335	0.924335	0.935258



Above plot depicts the Precision, Recall, F1-Score and Accuracy of all classification models. It is observed that Support Vector Machine and Logistic Regression both have similar and highest performance metrics among all.

Link to Project Folder - Project Folder

Direct Link to the code - Code

The above Project folder contains the following:
1. Dataset used for the project(dataset.csv)

2. A copy of this Report (Report-Group42.pdf)

3. Code file in py extension (code.py)

4. Code file in ipynb extension (code.ipynb)

REFERENCES

1.Introduction to Machine Learning with Python - A guide for Data Scientists, Andreas C. Muller, Sarah Guido.

2. Official Documentation of Numpy, Pandas, Seaborn, Matplotlib, and Sckitlearn.

3.An Article-Kernel Density Estimation, Matthew Conlen

4.Data Classification – Algorithms and Applications, Charu C Aggarwal

5. UCI Machine Learning Repository