VIDITHA WUDARU, JAYAVANI AKKIRAJU

CSE 587 DATA INTENSIVE COMPUTING

PROJECT PHASE 2

Introduction:

Using census data, we are trying to predict whether income exceeds $50k per year. An objective of this model is to identify the most critical factors that contribute to increasing an individual's income. Through such an analysis, important areas of income improvement can be identified.
This data in Kaggle is taken from the Census Bureau database in 1994. There are 14 features and 32561 records in the input dataset. This is primarily a binary classification problem. The main goal is to predict whether the income of the person will be <=50k or >50k based on the 14 features available.

Dataset Link:
https://archive.ics.uci.edu/ml/datasets/adult
https://www.kaggle.com/datasets/uciml/adult-census-income

Below is the list of features and output variables present in the dataset. (0-13 are features and column 14 is the target variable).

```
 #    Column           Non-Null Count   Dtype
---   ------           --------------   -----
 0    age              32561 non-null   int64
 1    workclass        32561 non-null   object
 2    fnlwgt           32561 non-null   int64
 3    education        32561 non-null   object
 4    education.num    32561 non-null   int64
 5    marital.status   32561 non-null   object
 6    occupation       32561 non-null   object
 7    relationship     32561 non-null   object
 8    race             32561 non-null   object
 9    sex              32561 non-null   object
 10   capital.gain     32561 non-null   int64
 11   capital.loss     32561 non-null   int64
 12   hours.per.week   32561 non-null   int64
 13   native.country   32561 non-null   object
 14   income           32561 non-null   object
```

After pre-processing the data, the output is:

```
] adultIncome.head()
```

	age	workclass	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	37	3	9	6	7	1	4	0	0	4356	40	38	0
1	37	3	9	6	3	1	4	0	0	4356	40	38	0
2	66	3	10	6	7	4	2	0	0	4356	40	38	0
3	54	3	10	0	6	4	4	0	0	3900	40	38	0
4	41	3	10	5	10	3	4	0	0	3900	40	38	0

```
] adultIncome.shape

(32561, 13)
```

Splitting the AdultIncome data into train and test :

The train data is 80% and the test data is 20%.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=10)
```

```
print(X_train.shape, y_train.shape,X_test.shape,y_test.shape)
```

```
(26048, 12) (26048,) (6513, 12) (6513,)
```

X_train represents the train data of variables;X_test represents the test data of variables
y_train represents the train data of the target;y_test represents the test data of the target

Printing the X_train information:

```
print(X_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26048 entries, 190 to 17673
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             26048 non-null  int64
 1   workclass       26048 non-null  int64
 2   education.num   26048 non-null  int64
 3   marital.status  26048 non-null  int64
 4   occupation      26048 non-null  int64
 5   relationship    26048 non-null  int64
 6   race            26048 non-null  int64
 7   sex             26048 non-null  int64
 8   capital.gain    26048 non-null  int64
 9   capital.loss    26048 non-null  int64
 10  hours.per.week  26048 non-null  int64
 11  native.country  26048 non-null  int64
dtypes: int64(12)
memory usage: 2.6 MB
None
```

Printing the y_train information:

```
print(y_train.info())
```

```
<class 'pandas.core.series.Series'>
Int64Index: 26048 entries, 190 to 17673
Series name: income
Non-Null Count  Dtype
--------------  -----
26048 non-null  int64
dtypes: int64(1)
memory usage: 407.0 KB
None
```

Machine Learning Algorithms:

1. LOGISTIC REGRESSION

Machine Learning commonly uses logistic regression to classify binary data. Based on a logistic function, it models the probability of a binary response variable (i.e., 0 or 1). The algorithm is capable of handling categorical as well as numerical data, and it can solve linear as well as

nonlinear classification problems. Since we are dealing with a binary classification problem, it is expected that logistic regression will be a suitable algorithm for this type of data. Therefore, we are applying the logistic regression model to the data to make predictions.

Applying Logistic Regression on the train and test data:

```
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression()
```

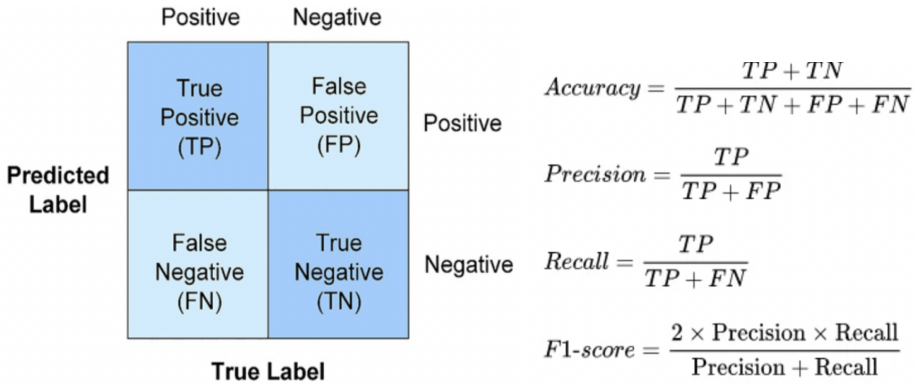
```
model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

The classification report of the Logistic Regression Model

```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.95      0.89      4943
           1       0.71      0.39      0.50      1570

    accuracy                           0.81      6513
   macro avg       0.77      0.67      0.69      6513
weighted avg       0.80      0.81      0.79      6513
```



$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

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

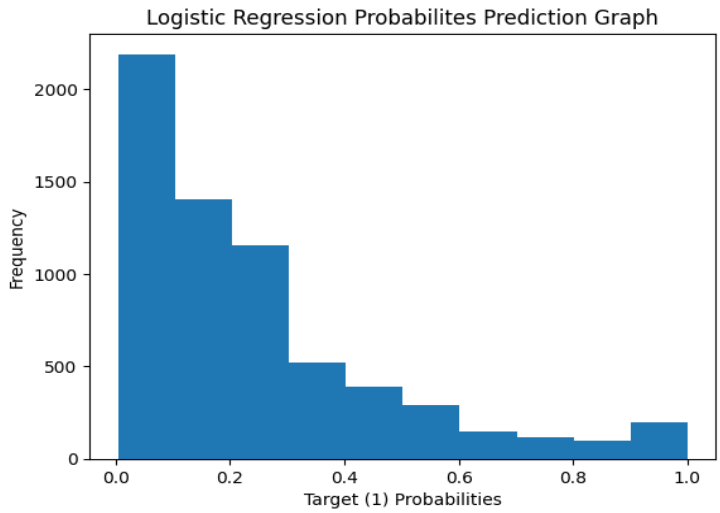
$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

We can see that the Precision rate for target variable ‘0’ is high at 83% and 95%. The f1-score is also high at 89% for the target variable ‘0’.
0: <=50k income

Using a confusion matrix, we can gain insight into how well our predictions held up to reality.

Predicted	0	1	All
Actual			
0	4695	248	4943
1	964	606	1570
All	5659	854	6513

The graph shows the probability of the target variable being '1' that is spread in the range (0,1) and the respective counts associated with it.



The metric score of roc_auc is printed as shown below

```
# calculate scores
auc1 = metrics.roc_auc_score(y_test, pr1)
print('Logistic: ROC AUC=%.3f' % (auc1))
```

```
Logistic: ROC AUC=0.809
```

AUC is a measure of how well a model distinguishes between positive and negative classes. The higher the AUC, the better the model performs. When AUC is 1, the classifier is able to distinguish between all points in the Positive and Negative classes, which means that the model's performance is perfect. As the ROC AUC score for the logistic model above is 0.8, it indicates that the model is working and can detect more True positives and True negatives.

Printing the Accuracy:
The training accuracy of the logistic regression model is: 82.5%

```
train_acc = accuracy_score(y_train, model.predict(X_train))
print(train_acc)
```

```
0.8252073095823096
```

Accuracy of the Model:

Accuracy = (True Positive + True Negative) / Total Predictions

```
[ ]  acc = metrics.accuracy_score(y_test, pred)
     print(acc)

     0.8139106402579457
```

The test accuracy of the logistic regression model is 81.39%. Higher the accuracy better the performance of the model.

Printing the Precision

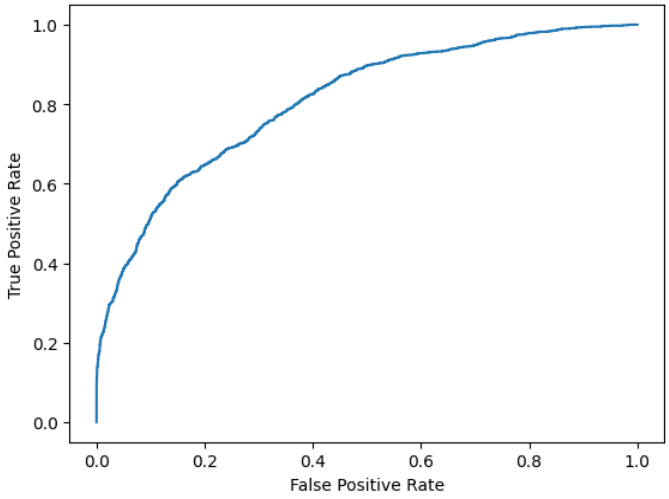
Precision:

Precision = True Positive / (True Positive + False Positive)

```
[ ]  prec= metrics.precision_score(y_test, pred)
     print(prec)

     0.7096018735362998
```

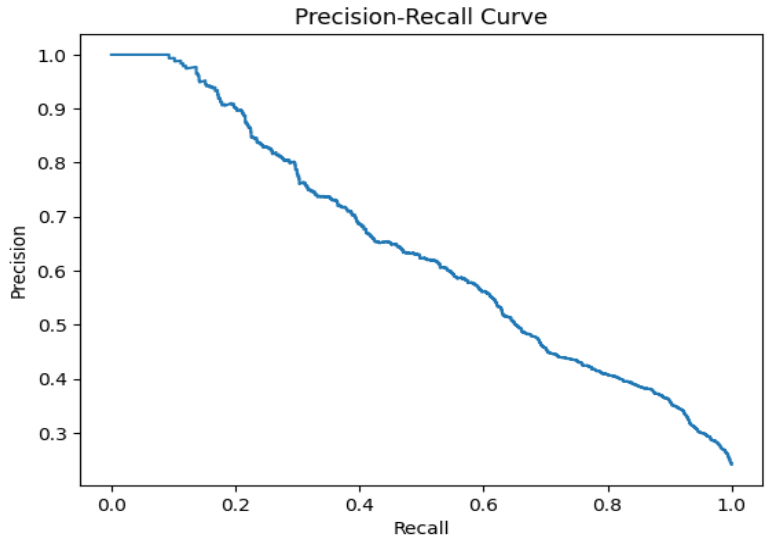
The precision of the logistic regression model is 70.9%. Higher the precision more the relevant results the model returns.

ROC Graph of Logistic Regression Model:



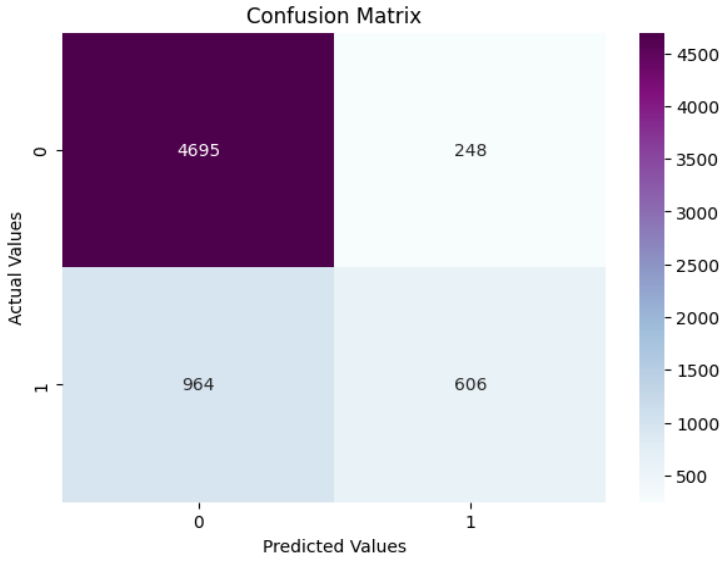
To create a ROC graph in Machine Learning, we plot the True Positive Rate (TPR) on the y-axis against the False Positive Rate (FPR) on the x-axis for varying classification thresholds. The curve represents the model's performance. The area under the ROC curve (AUC) is a popular metric used to assess the performance of binary classification models.

The precision-Recall graph for the logistic regression model is as follows:



We can see that precision is at its highest when the recall is at its lowest initially. Later, as the precision decreases the recall seems to increase.

The confusion matrix of the logistic regression model with actual and predicted values.



Feature Scaling

To optimize a logistic regression model, the cost function must be minimized using an optimization algorithm such as gradient descent. For this reason, it is often advisable to perform feature scaling on the input variables to ensure that they have similar scales or variances. Therefore, it is recommended to scale features for logistic regression models to achieve better results.

Using StandardScaler():

StandardScaler() scales each feature to have zero mean and unit variance.

After scaling the data, the logistic regression model is fitted with a random_state value which ensures the shuffling of data before the model is fitted.

```
log = LogisticRegression(random_state = 10)
log.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(random_state=10)
```

The train and test Accuracy of the model:

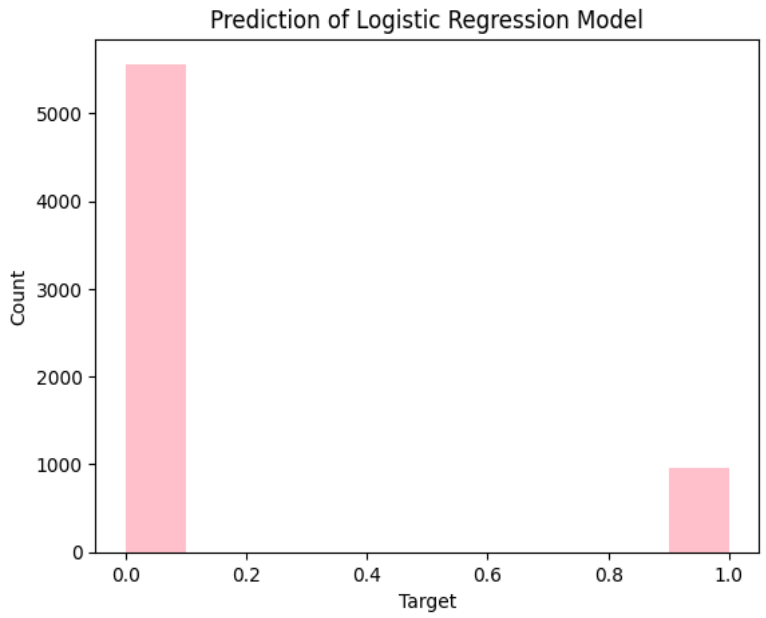
```
train_acc = accuracy_score(y_train, log.predict(X_train))
print("Train Accuracy:",train_acc*100)
```

```
Train Accuracy: 82.52073095823096
```

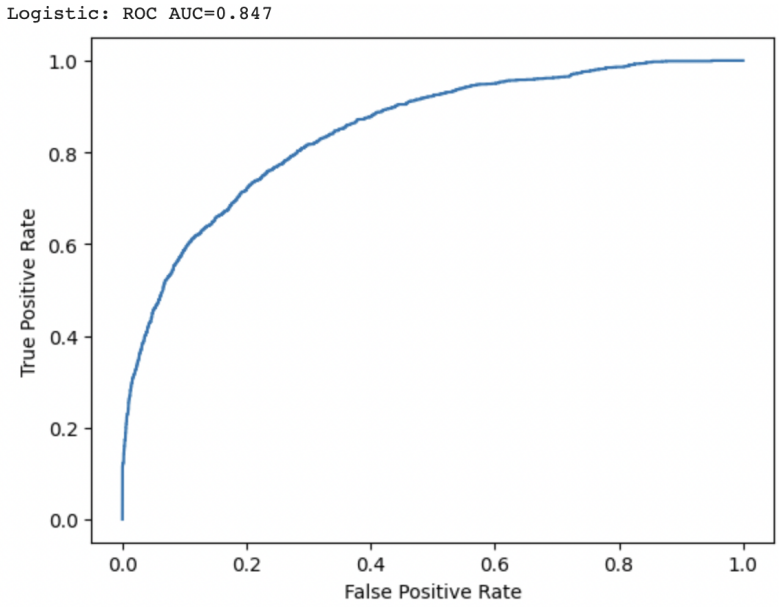
```
y_pred = log.predict(X_test)
test_acc = accuracy_score(y_test, y_pred)
print("Test Accuracy:",test_acc*100)
```

```
Test Accuracy: 83.04928604329803
```

The prediction of the Logistic Regression model with the target variable and the respective count.

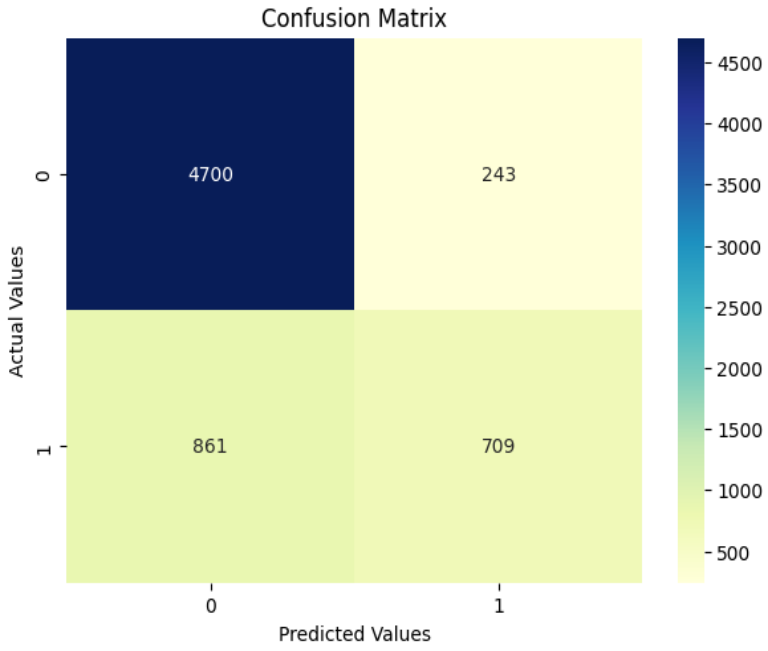


The ROC AUC metric score is 0.84 as shown below.



The ROC graph curve plots the False Positive rate against the True Positive rate. The ROC AUC metric score has significantly increased after the feature scaling for the logistic regression model.

The Confusion matrix displaying the actual vs predicted values for the model:



The Confusion Matrix for the model:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.95      0.89      4943
           1       0.74      0.45      0.56      1570

    accuracy                           0.83      6513
   macro avg       0.79      0.70      0.73      6513
weighted avg       0.82      0.83      0.81      6513
```

After feature scaling, the accuracy of the logistic regression model increased from 81% to 83%. Precision and recall for '0' as the target is high at 85% and 95% respectively. f1-score is also high for '0' as a target with 89%.

Using Solver in Logistic Regression:
In logistic regression, the term 'solver' refers to the optimization algorithm used to minimize the cost function. 'solver': lbfgs

```
logi = LogisticRegression(solver='lbfgs')
logi.fit(X_train, y_train)
pr = logi.predict_proba(X_test)
pr = pr[:, 1]
auc = metrics.roc_auc_score(y_test, pr)
```

The train and test Accuracies of the Logistic Regression model using 'solver':

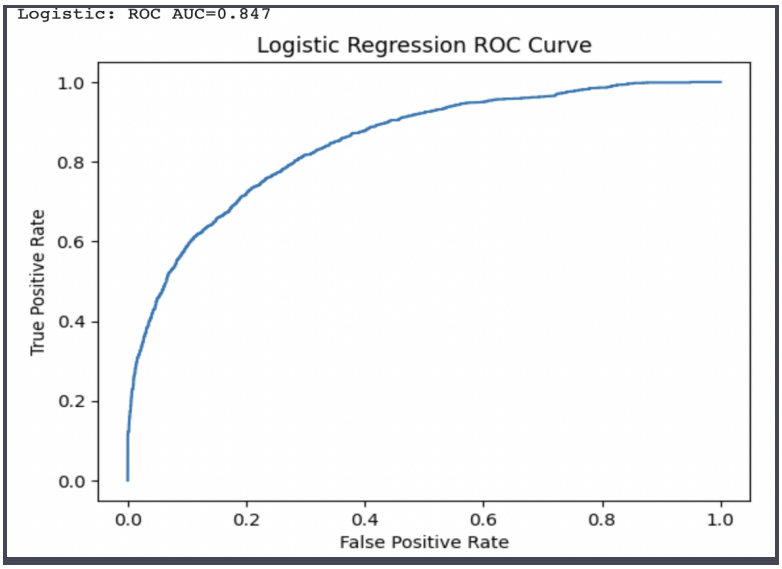
```
train_acc = accuracy_score(y_train, logi.predict(X_train))
print("Train Accuracy:",train_acc*100)
```

```
Train Accuracy: 82.52073095823096
```

```
y_pred = logi.predict(X_test)
test_acc = accuracy_score(y_test, y_pred)
print("Test Accuracy:",test_acc*100)
```

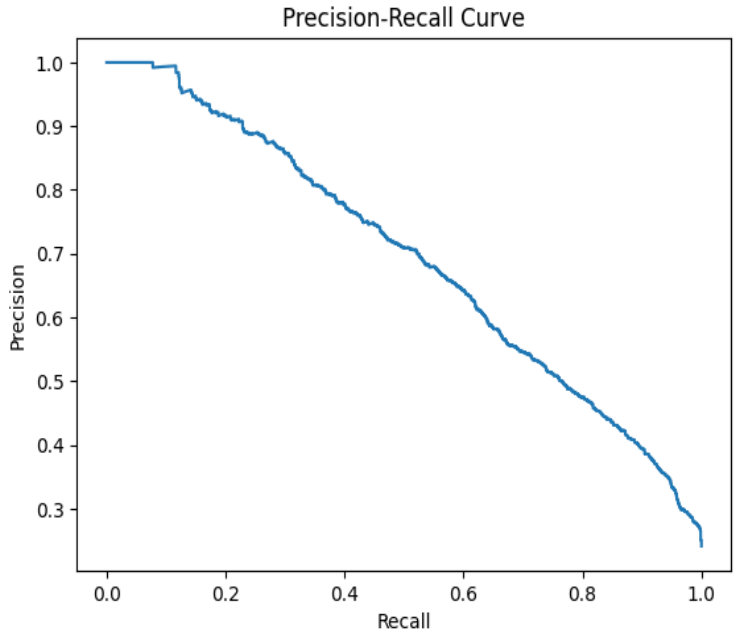
```
Test Accuracy: 83.04928604329803
```

ROC Curve for Logistic Regression:



The ROC AUC metric for logistic regression with the solver is 0.84. The True Positive rate increased very little with an increase in the False Positive rate initially. A true positive rate of 1.0 has a False Positive rate of 1.0.

The Precision vs Recall graph is shown below:



As the precision rate increases, recall decreases.

Precision-recall (PR) graphs visually illustrate the balance between precision and recall at different threshold values. These graphs typically start at the origin, where both precision and recall are zero, and end at the upper-right corner, where both precision and recall are one. A higher PR curve indicates better performance for the classifier. Using PR graphs can help to evaluate and optimize classification models.

2. Naive Bayes

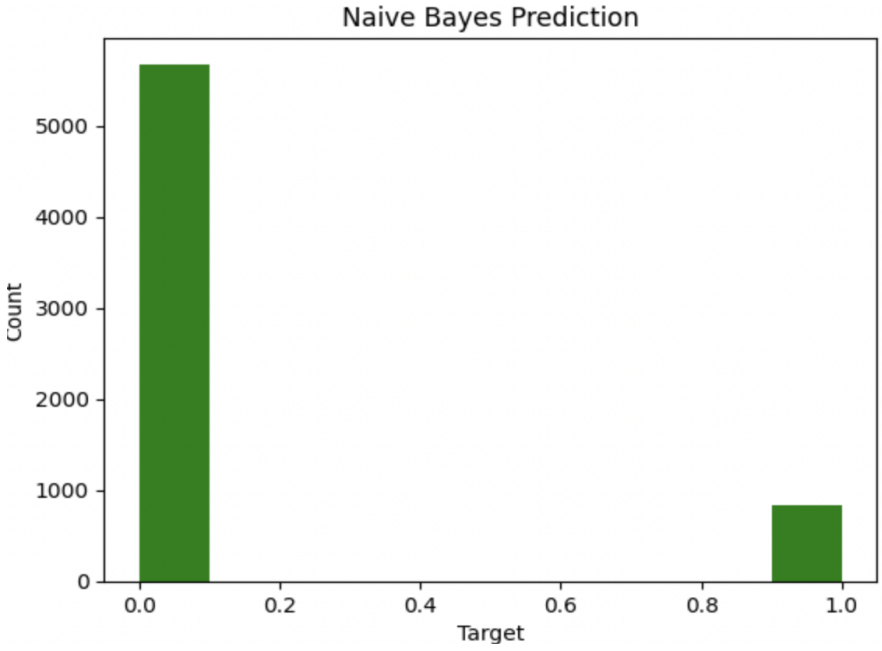
Often used for binary classification problems, Naive Bayes is a probabilistic algorithm. As there are only two possible classes in binary classification, the Naive Bayes algorithm calculates the probability of each class and predicts the class with the highest probability. The Naive Bayes algorithm is well-suited for handling datasets that contain many irrelevant or redundant features. Thus, we use Naive Bayes for our classification.

Initially, we fit the GaussianNB model onto the data.

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

Graph of Naive Bayes prediction:



The plot of the target (0/1) with respect to their counts based on the Naive Bayes prediction is as above. We can observe that most predictions ought to output 0 as the class label.

The Confusion Matrix of the model:

```
print(cm3)
```

```
[[4689  254]
 [ 987  583]]
```

The training accuracy of the model:

```
train_acc = accuracy_score(y_train, gnb.predict(X_train))
print("Train Accuracy:",train_acc*100)
```

```
Train Accuracy: 80.94671375921376
```

The test accuracy of the Model:

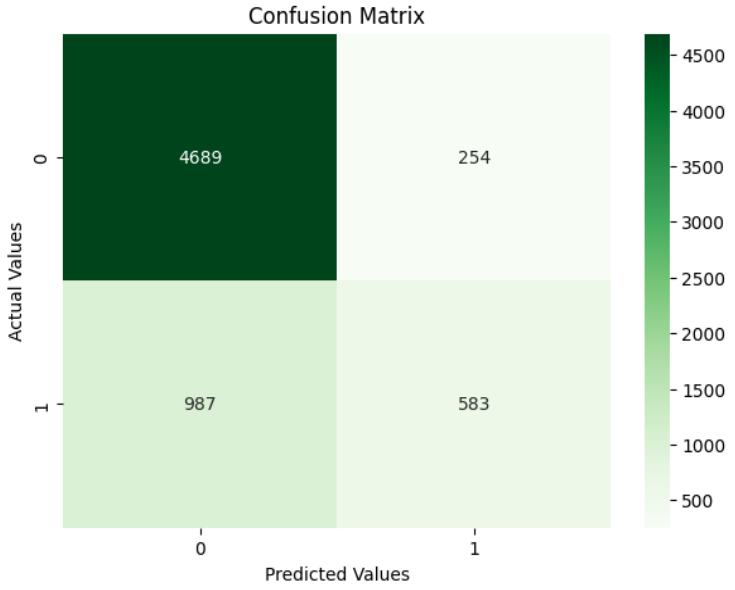
```
print(ac3)
```

```
0.8094580070627975
```

We can see that the accuracy of the model with NaiveBayes fit is 80.9% which is less than the Logistic Regression.

Plotting the confusion Matrix:

```
plt.figure(figsize=(7,5))
plt.title('Confusion Matrix')
sns.heatmap(cm3, annot=True,fmt='d', cmap='Greens')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
```



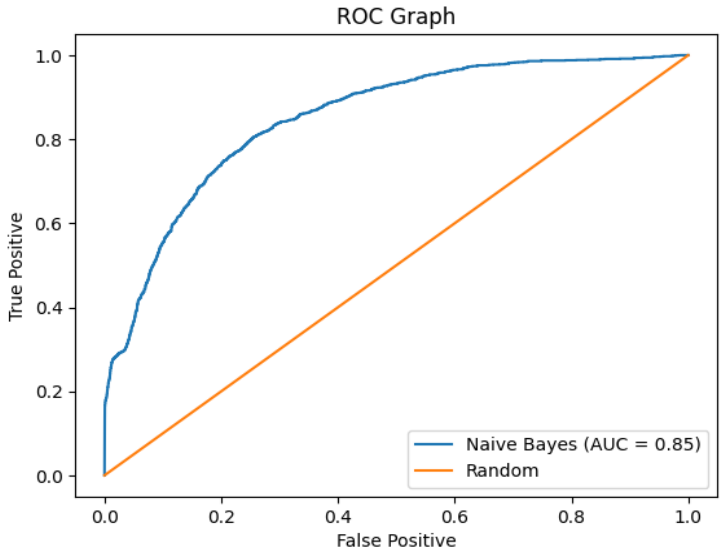
The classification report of Naive Bayes:

```
print(classification_report(y_test, y_pred))
```

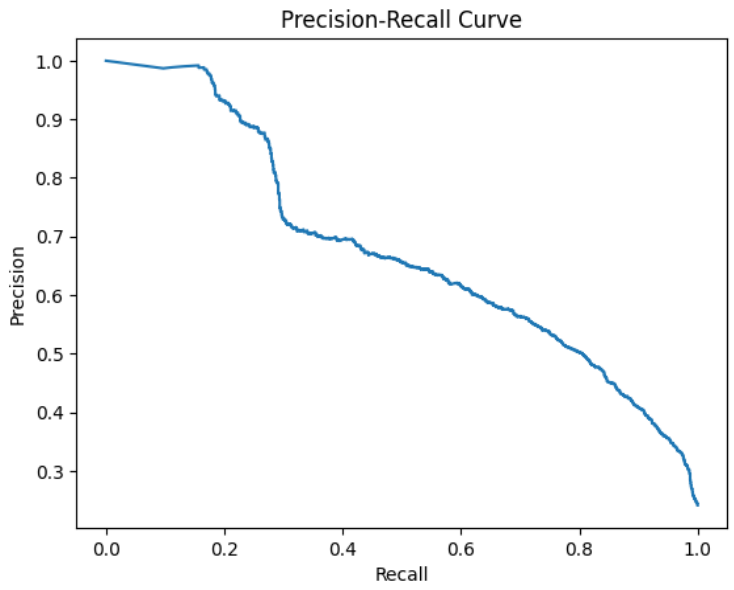
```
              precision    recall  f1-score   support

           0       0.83      0.95      0.88      4943
           1       0.70      0.37      0.48      1570

    accuracy                           0.81      6513
   macro avg       0.76      0.66      0.68      6513
weighted avg       0.79      0.81      0.79      6513
```

The scores of precision, recall, and f1-score are high in the case of the class label '0' with 83%, 95%, and, 88% respectively. The accuracy of the model is 81%.

The ROC AUC Graph is plotted as shown below:
The Area Under Curve(AUC) rate for the Naive Bayes model is 0.85. The random line represents the ROC curve of a random classifier that randomly predicts the positive or negative class with a probability of 0.5 each. It serves as a baseline to compare the performance of the KNN classifier.



The Precision-Recall(PR) curve is plotted below for Gaussian Naive Bayes:



We can see that the graph is not a sharp line but has curves and inclinations.

3. KNN

K-Nearest Neighbors (KNN) is an algorithm that classifies a test point by looking at the class labels of its nearest k training data points. Choosing the right value of k is important to avoid overfitting (k too small) or underfitting (k too large). KNN is effective for binary classification tasks with nonlinear decision boundaries.

i. Using an Arbitrary 'K' value (KNeighborsRegressor)

n_neighbors = '3'(Number of neighbors)

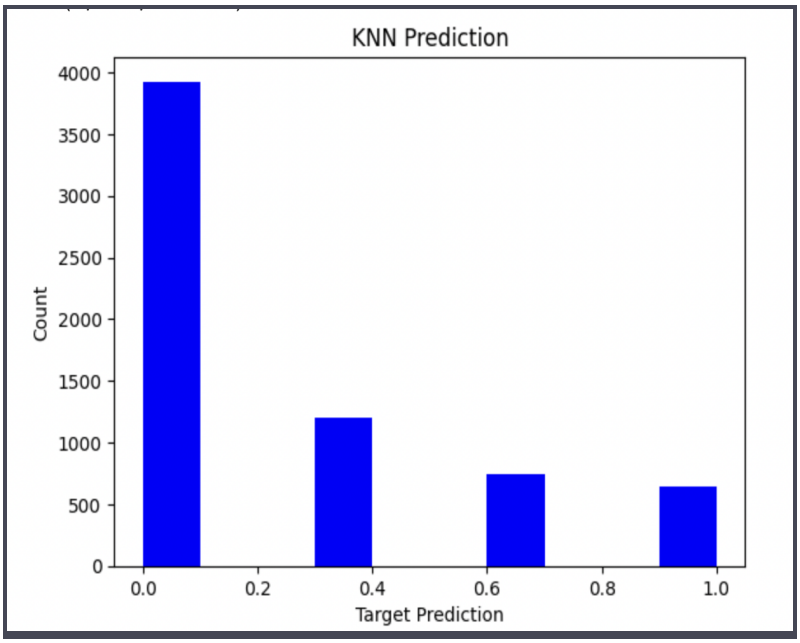
```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=3)
```

```
knn.fit(X_train, y_train)
```

```
KNeighborsRegressor
KNeighborsRegressor(n_neighbors=3)
```

The KNN prediction graph is as shown:

The target prediction probability is spread between 0 and 1. Their respective count is plotted.



Calculating Root Mean Squared Error(RMSE) for the KNN prediction:

```
from sklearn.metrics import mean_squared_error
from math import sqrt
trp = knn.predict(X_train)
rmse = sqrt(mean_squared_error(y_train,trp))
print(rmse)
```

```
0.26270442685386153
```

ii. KNN Using GridsearchCV for finding the best K value

GridSearchCV is a method by the Scikit-learn library that enables us to find the optimal hyperparameters for a machine-learning model by performing an exhaustive search over a specified parameter grid.

```
from sklearn.model_selection import GridSearchCV
p = {"n_neighbors": range(1, 50)}
gridsearch = GridSearchCV(KNeighborsRegressor(), p)
gridsearch.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: KNeighborsRegressor
    ▸ KNeighborsRegressor
```

Fitting GridSearchCV where the parameters are in the range of 1,50) to find the best k value.
By using the grid search best_params, the number of neighbors: 20 is the best k value.

```
gridsearch.best_params_
```

```
{'n_neighbors': 20}
```

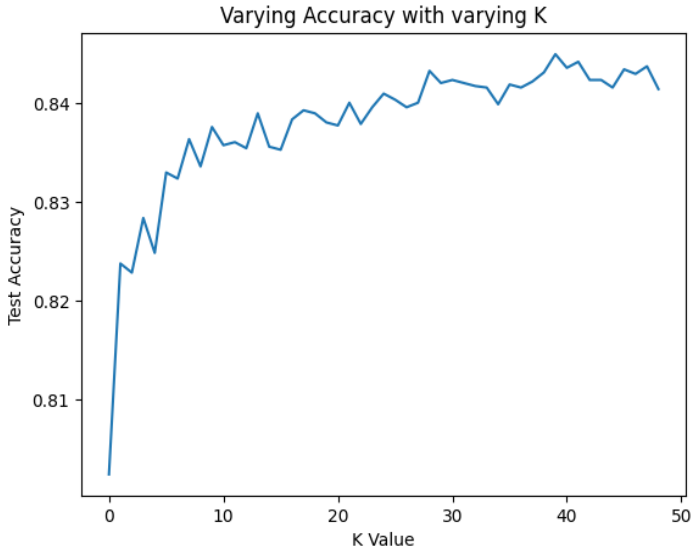
The train and test Root Mean Square Error(RMSE) for the KNN model using GridsearchCV:
The train RMSE for the KNN model: 0.319
The test RMSE for KNN Model: 0.335

iii. Using KNeighborsClassifier

KNeighborsClassifier predicts the class label for an input sample by finding its k-nearest neighbors in the training data. The distance between each input sample and all other samples in the training data is calculated, and the k-nearest neighbors are chosen based on this distance. The majority class label among these neighbors is then assigned to the input sample.
We are trying to find k(number of neighbors) in the range of (1,50) :

```
sc = []
for k in range(1,50):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    yp = knn.predict(X_test)
    sc.append(metrics.accuracy_score(y_test, yp))
```

Graph demonstrating the test accuracy varying with change in K value:



We can see that the test accuracy is high when the K value is nearly 39.

Printing the best K value:

```
print("Best K value:",sc.index(max(sc)))
```

```
Best K value: 39
```

The train and test Accuracies:

```
train_acc = accuracy_score(y_train, knnc.predict(X_train))
print("Train Accuracy:",train_acc*100)
```

```
Train Accuracy: 84.40187346437347
```

```
print("Test Accuracy:",metrics.accuracy_score(y_test, knnpred))
```

```
Test Accuracy: 0.8429295255642562
```

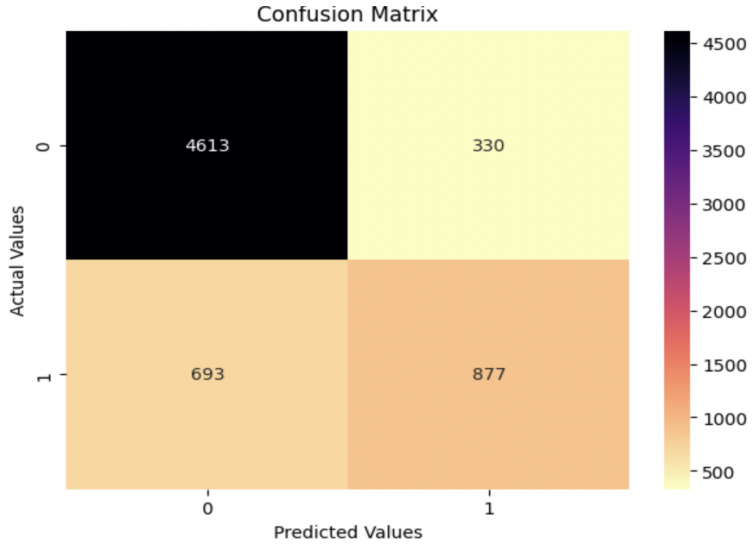
The classification report for KNNClassifier:

```
print(classification_report(y_test, knnpred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      4943
           1       0.73      0.56      0.63      1570

    accuracy                           0.84      6513
   macro avg       0.80      0.75      0.77      6513
weighted avg       0.83      0.84      0.84      6513
```

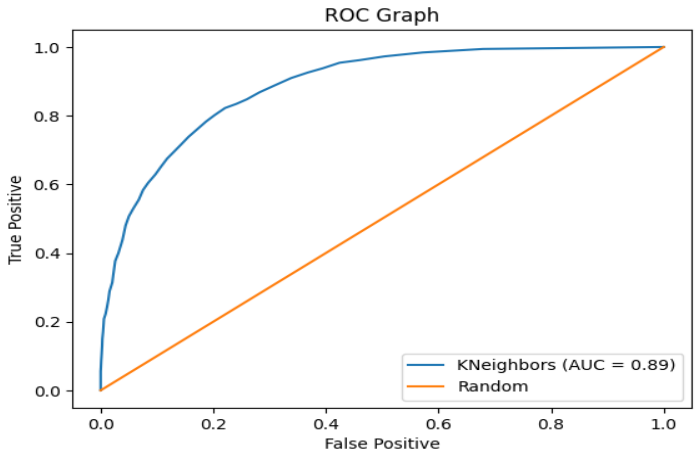
The precision, recall, and f1-score for the class label '0' is high with 87%, 93%, and 90%. The test Accuracy is 84%.

Confusion matrix of the KNN Model:

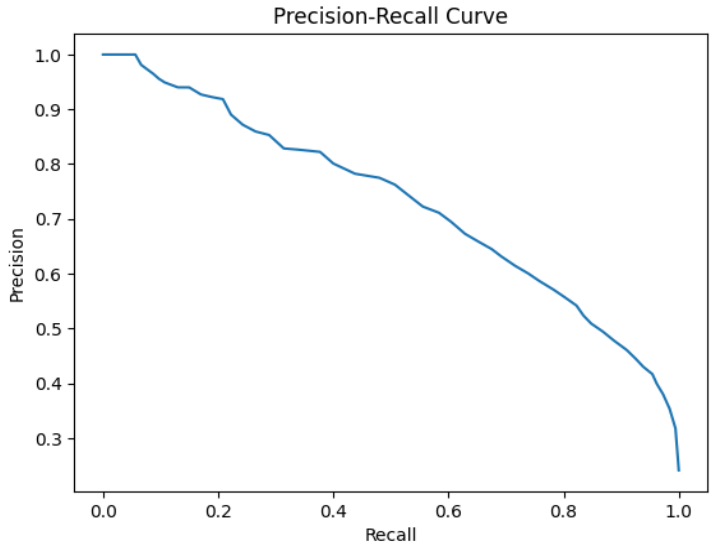


The confusion matrix above is plotted where the actual values and predicted values are listed.

ROC Graph



The blue curve represents the K Neighbors classifier ROC Curve. The random line represents the ROC curve of a random classifier that randomly predicts the positive or negative class with a probability of 0.5 each. It serves as a baseline to compare the performance of the KNN classifier.



The Precision-Recall Curve plot is as shown above with precision on the y-axis and recall on the x-axis.

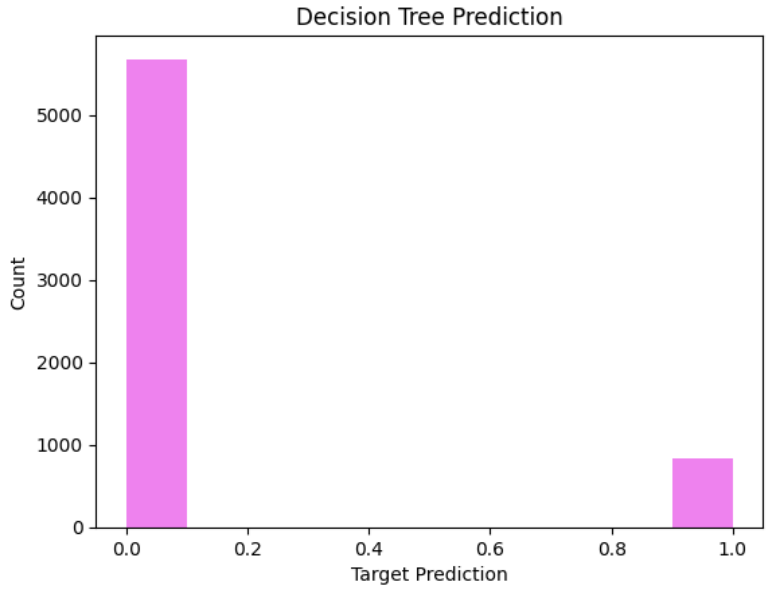
4. Decision Tree

A decision tree for binary classification creates a model that consists of a series of decisions, resulting in either of the two possible outcomes which is the suitable model for our binary classification problem. Further, the algorithm divides the input data space into smaller regions, based on the feature values, and forms a tree structure by recursively partitioning these regions.

The decision tree algorithm uses the training data to derive the most suitable set of rules for making decisions, which is then depicted as a tree structure.
Fitting Decision Tree classifier to the data.

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
dpred = clf.predict(X_test)
```

The Graph shows the class label (0/1) with respect to its count in the decision tree predicted.



It can be observed that the occurrence of the class label '0' is higher compared to the occurrence of the class label '1', which is much less frequent.

Train Accuracy of the Decision Tree classifier:

```
train_acc = accuracy_score(y_train, clf.predict(X_train))
print("Train Accuracy:",train_acc*100)
```

```
Train Accuracy: 97.50844594594594
```

Printing the tets Accuracy of the Decision Tree classifier:

```
print("Accuracy:",metrics.accuracy_score(y_test, dpred))
```

```
Accuracy: 0.8159066482419776
```

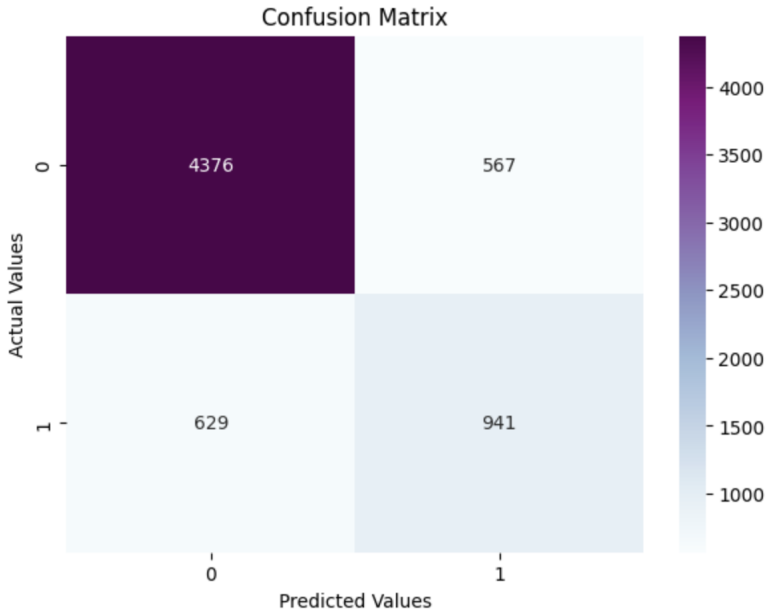
The classification report of the Decision Tree classifier:

```
print(classification_report(y_test, dpred))
```

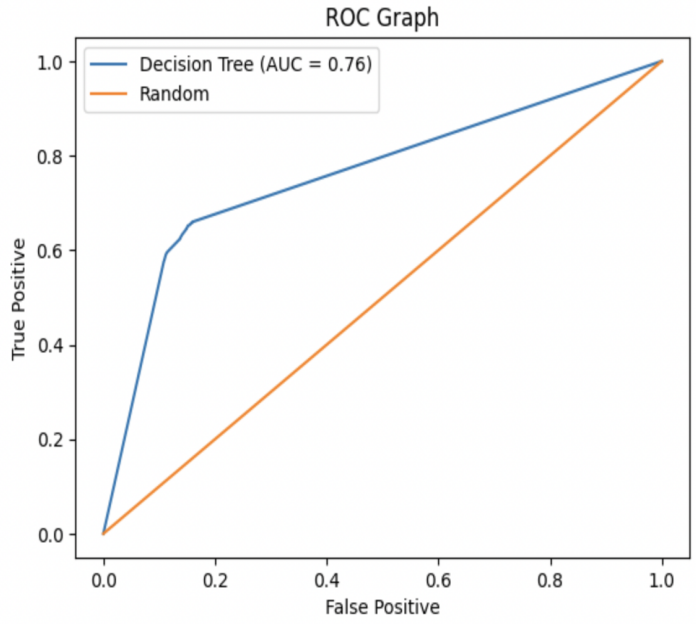
```
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      4943
           1       0.62      0.60      0.61      1570

    accuracy                           0.82      6513
   macro avg       0.75      0.74      0.75      6513
weighted avg       0.81      0.82      0.82      6513
```

The precision, recall, and, f1-score are high for the class label '0' as observed from the classification report with 87%, 89%, 88% respectively. The accuracy of the model from above is 82%.

The Confusion Matrix of the Decision Tree Classifier:

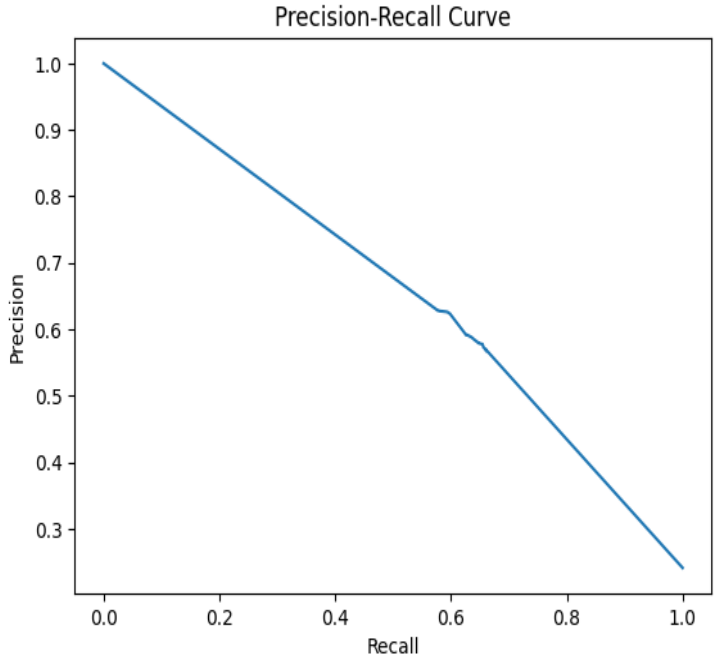


The ROC Graph for the Decision Tree Classifier is as shown. The AUC(Area under curve) score is 0.76. Higher the AUC, the better the model. The blue line represents the varying true positive rate with the False Positive rate of the decision tree. The random line represents the ROC curve of a random classifier that randomly predicts the positive or negative class with a probability of 0.5 each. It serves as a baseline to compare the performance of the KNN classifier.

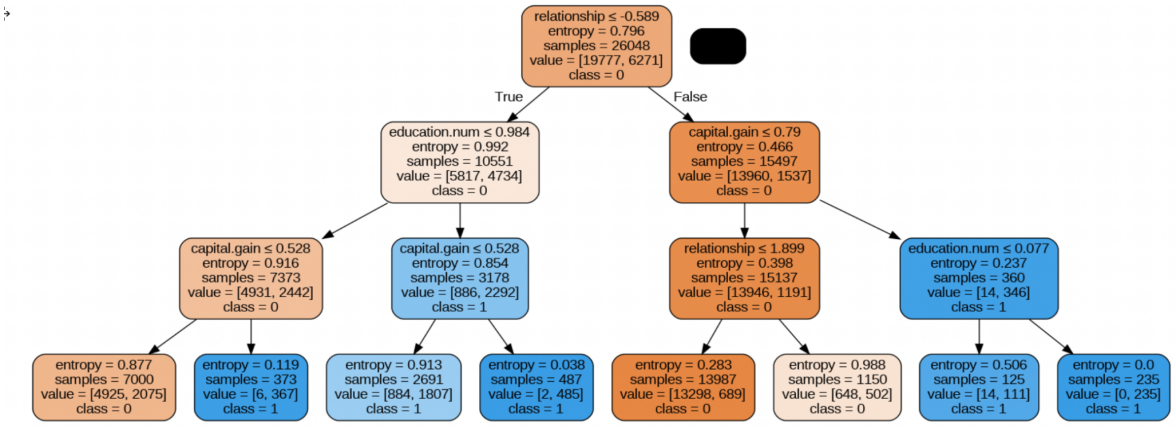


The precision-recall curve for the Decision tree classifier that plots Precision against recall rates of the model is as shown below:

We can see that the precision rate is decreasing as the recall rate is increasing.



Plotting Decision Tree of max_depth : 3



We can see that the relationship variable has an entropy of 0.79 where the total samples that are passed are 26048 out of which 19777 are classified as 0 and 6271 as 1. For, education.num entropy - is 0.23 and the samples passed are 360 out of which 14 are classified as 0 and 346 as '1'. Thus, class 1 is selected in this case. The same follows for each attribute and True or False is evaluated in each case and the decision is made. The figure is drawn based on the max_depth 3. As the depth increases the decision tree is more exploited and the results are considered.

The classification report of the Decision tree Classifier:

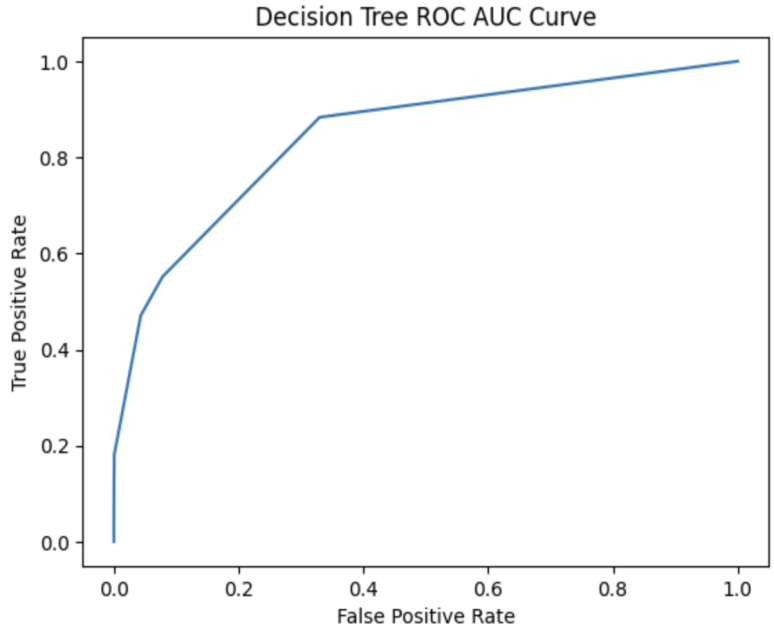
```
print(classification_report(y_test, dpred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      4943
           1       0.78      0.47      0.59      1570

    accuracy                           0.84      6513
   macro avg       0.81      0.71      0.74      6513
weighted avg       0.83      0.84      0.82      6513
```

The precision, recall, and f1-score of the class label '0' is high at 85%, 96%, and, 90%.
The accuracy of the Decision tree classifier model is 84%.
The decision Tree classifier ROC curve with max_depth 3 is as shown below: AUC score is 0.84 which is a pretty good score that indicates the better performance of the model.

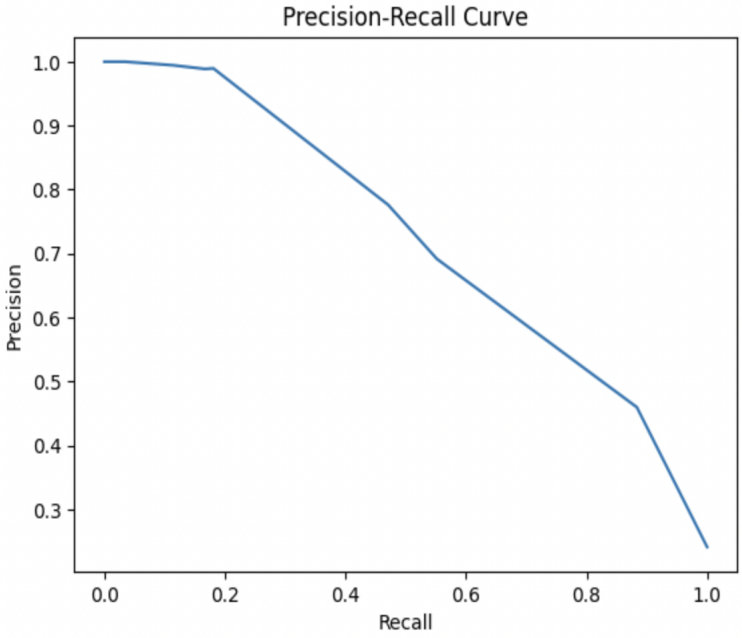
ROC Graph for Decision Tree classifier:

```
Decision Tree: ROC AUC=0.844
Text(0.5, 1.0, 'Decision Tree ROC AUC Curve')
```



As the false positive rate is increasing, the true positive rate seems to increase slightly.

The Precision-recall curve of the decision tree classifier with max_depth 3:



When the recall rate is at its low, the precision rate is at its high initially. Gradually, as the recall rate increased, the precision rate dropped.

5. Support Vector Classifier:

Kernels like linear, poly, and rbf can be applied, and custom kernels can also be used. Choosing a proper kernel affects the model's accuracy. For 'rbf' kernel hyperparameter tuning can be done by using different values of gamma and C and selecting the combination for which high accuracy is obtained. SVC uses a subset of training samples at the decision boundary, making it memory efficient. Taking this memory-efficient nature of SVC into account, we are implementing this model. Imported the required libraries and fitted the Support Vector Classifier model with 'rbf' kernel on training data and obtained 84.95% accuracy using test data.

Using kernel='RBF'

```
[ ]  # Imports SVC from sklearn
     from sklearn.svm import SVC

[ ]  # Uses 'rbf' kernel
     clf_SVC_rbf = SVC(kernel='rbf')

     # Fits the SVC model
     clf_SVC_rbf.fit(X_train, y_train)
```

SVC
SVC()

```
[ ]  # Prints the accuracy of model
     clf_SVC_rbf.score(X_test, y_test)
```

```
0.849531705819131
```

Obtained accuracy of 80.23% on training data.

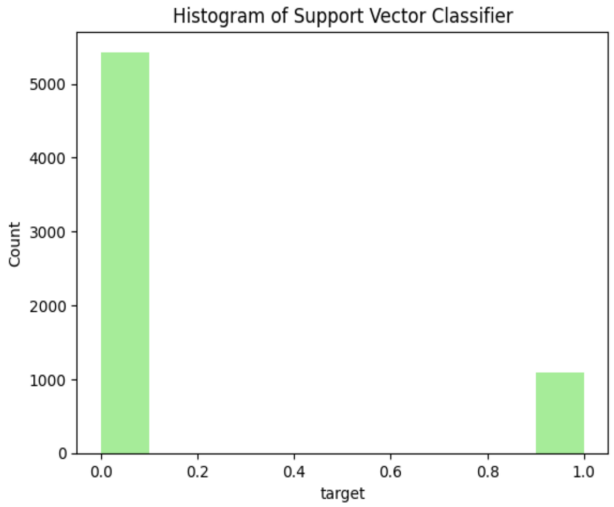
```
# Prints accuracy on training data
from sklearn.metrics import accuracy_score
train_acc_SVC_rbf = accuracy_score(y_train, clf_SVC_rbf.predict(X_train))
print(train_acc_SVC_rbf)
```

```
0.8023264742014742
```

Using predict(), predicted the values on test data and plotted the counts. From the below graph, we can observe that there are more than 5000 samples whose income is <=50K, and there are around 1000+ samples whose income is >50K.

```
# Calculates the predicted value of test data and plots Histogram
y_pred = clf_SVC_rbf.predict(X_test)
plt.hist(y_pred, color = 'lightgreen')
plt.title('Histogram of Support Vector Classifier')
plt.xlabel('target')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



Printed Classification Report. From the report, we can see that precision and recall are high for target variable '0' with 86 and 95 percentages respectively when compared to target variable '1'. F1-score for target variable '0' is 91% while for '1' it is 63%.

```
# Imports and prints classification_report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.86	0.95	0.91	4943
1	0.77	0.53	0.63	1570
accuracy			0.85	6513
macro avg	0.82	0.74	0.77	6513
weighted avg	0.84	0.85	0.84	6513

Printed Confusion Matrix. From the below matrix, we can observe that 4696 of the samples with income <=50K are predicted correctly and 837 samples with income >50K are predicted correctly. 247 samples with income <=50K are predicted as income>50K and 733 samples with income>=50K are predicted as incorrectly under the <=50K category.

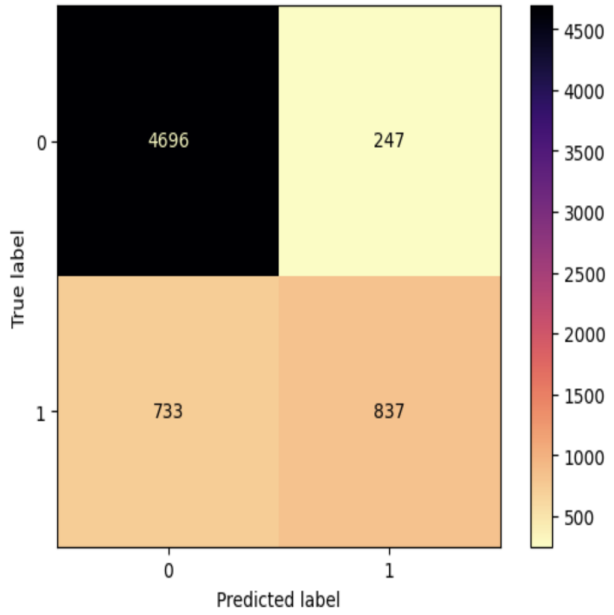
```
# Imports and prints confusion_matrix
from sklearn.metrics import confusion_matrix
cfm_clf_SVC_rbf = confusion_matrix(y_test, y_pred)
pd.crosstab(y_test, y_pred, rownames = ['Actual'], colnames =['Predicted'], margins = True)
```

Predicted	0	1	All
Actual			
0	4696	247	4943
1	733	837	1570
All	5429	1084	6513

Plotted Confusion Matrix

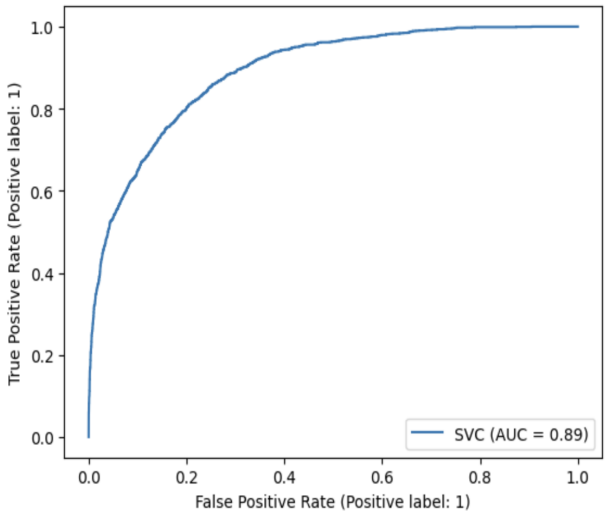
```
# Plots Confusion Matrix
from sklearn.metrics import ConfusionMatrixDisplay
display_cm_clf_SVC_rbf = ConfusionMatrixDisplay(confusion_matrix=cfm_clf_SVC_rbf)
display_cm_clf_SVC_rbf.plot(cmap="magma_r")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff1dbc23580>
```



Printed ROC Curve. The AUC(Area Under Curve) for Support Vector Classifier is 0.89.

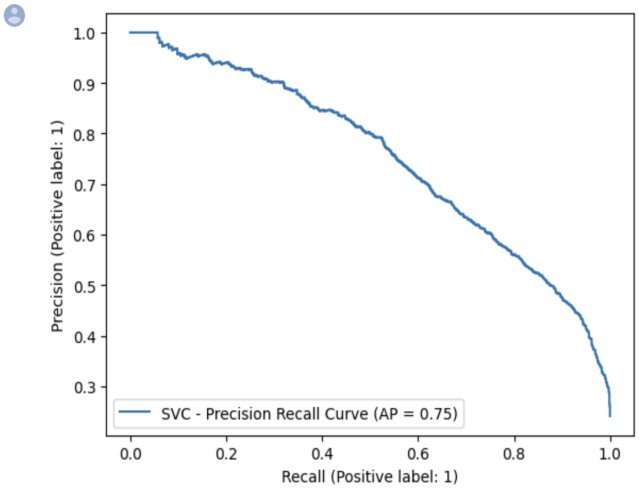
```
# Imports RocCurveDisplay from sklearn and displays ROC curve
from sklearn.metrics import RocCurveDisplay
clf_SVC_rbf_display = RocCurveDisplay.from_estimator(clf_SVC_rbf, X_test, y_test)
```



Printed Precision Recall display.

```
[ ] # Imports PrecisionRecallDisplay from sklearn
    from sklearn.metrics import PrecisionRecallDisplay
```

```
# Displays Precision Recall Curve
display_clf_SVC_rbf = PrecisionRecallDisplay.from_estimator(
    clf_SVC_rbf, X_test, y_test, name="SVC - Precision Recall Curve"
)
```



Printed the number of support vectors for each class, support vectors, and indices of support vectors.

From below we can see that the support vectors for 2 classes are 4811 and 4628.

```
[ ] # Prints the number of Support Vectors for every class
    clf_SVC_rbf.n_support_
```

```
array([4811, 4628], dtype=int32)
```

```
[ ] # Prints the Support Vectors
    clf_SVC_rbf.support_vectors_
```

```
array([[ 0.34726925, -1.89741549, -0.15017697, ..., -0.21579488,
        -0.41673451,  0.25911384],
       [ 0.12145494, -0.08472624,  1.21092593, ..., -0.21579488,
        -0.41673451,  0.25911384],
       [-0.17963079, -0.08472624,  1.21092593, ..., -0.21579488,
         0.43860396,  0.42294212],
       ...,
       [ 1.32579789, -0.08472624, -0.60387794, ..., -0.21579488,
        -0.41673451,  0.25911384],
       [ 0.87416929, -0.08472624, -0.60387794, ..., -0.21579488,
        -0.41673451,  0.25911384],
       [ 0.64835498, -0.08472624, -0.60387794, ..., -0.21579488,
        -0.41673451,  0.25911384]])
```

```
# Prints the indices of Support Vectors
clf_SVC_rbf.support_
```

```
array([    1,     3,    14, ..., 26034, 26035, 26046], dtype=int32)
```

6. Random Forest Classifier:

Random Forest Classifier is a Supervised Machine Learning algorithm used for both Classification and Regression problems. Random Forest creates multiple decision trees. We can control the number of decision trees a Random Forest uses by using the n_estimators parameter. The higher the n_estimators value the greater the number of decision trees. We have chosen this model because it outperforms the Decision Tree. Calculated the accuracies and saw that the

accuracy of Random Forest is more than that of Decision Tree. Accuracies from Random Forest are 84.63% and 84.96% which is greater than the accuracy obtained from the decision tree which is 81% on test data.

a. 170 n_estimators, gini criterion:

Imported required libraries, used 170 n_estimators, gini criterion, and fitted the model using training data. Obtained an accuracy of 84.63%.

```
[ ] # Imports RandomForestClassifier from sklearn
    from sklearn.ensemble import RandomForestClassifier
```

Random Forest Classifier with n_estimators: 170 and 'gini' criterion. n_extimators is the number of decision trees in the Random Forest

```
[ ] clf_RandomForest = RandomForestClassifier(n_estimators=170, criterion='gini')
```

```
[ ] # Fits the RandomForest model
    clf_RandomForest.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=170)
```

```
# Prints the accuracy of RandomForest model
clf_RandomForest.score(X_test, y_test)
```

```
0.846307385229541
```

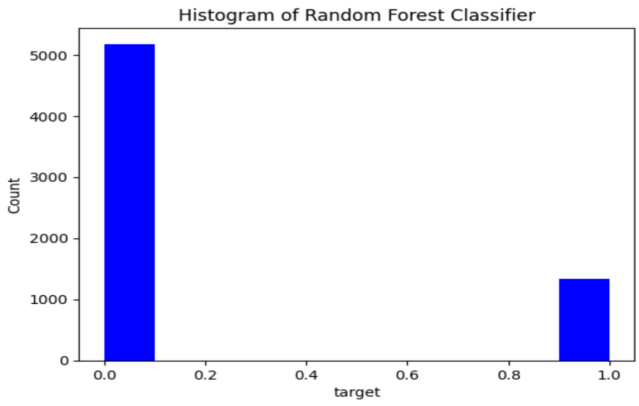
Obtained an accuracy of 97.5% on training data.

```
[134] from sklearn.metrics import accuracy_score
      train_acc_RandomForest = accuracy_score(y_train, clf_RandomForest.predict(X_train))
      print(train_acc_RandomForest)

      0.9750844594594594
```

Used test data and predicted the output values using test data. Plotted the histogram of predicted value counts. From the below graph, we can see that there are around 5000 values under the target variable ‘0’ category and there are nearly 1500 values under the target variable ‘1’ category.

```
# Predicts the RandomForest on test dataset and plots histogram
y_pred = clf_RandomForest.predict(X_test)
plt.hist(y_pred, color = 'blue')
plt.title('Histogram of Random Forest Classifier')
plt.xlabel('target')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



Printed Classification Report. From the below figure, we can see that the precision, recall, and f1-score for the target '0' variable are 88%, 92%, and 90% respectively. For target '1' variable they are 71%, 61%, and 66% respectively. These 3 values are higher for the target '0' variable when compared with the target '1' variable.

```
[ ] # Imports classification_report from sklearn.metrics and prints classification_report
    from sklearn.metrics import classification_report
    print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      4943
           1       0.71      0.61      0.66      1570

    accuracy                           0.85      6513
   macro avg       0.80      0.76      0.78      6513
weighted avg       0.84      0.85      0.84      6513
```

Printed Confusion Matrix. From the below matrix, we can see that 4560 target '0' values are predicted correctly as '0' and 952 targets '1' variables are predicted correctly as '1'. On the other hand, 383 target '0' values are predicted incorrectly as '1', and 618 target '1' variables are predicted incorrectly as '0'.

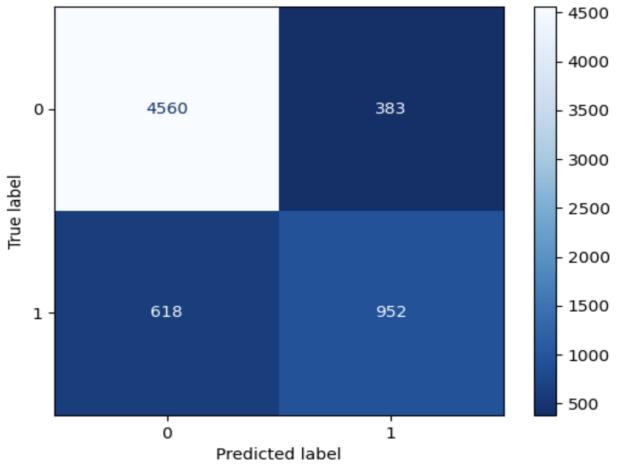
Predicted	0	1	All
Actual			
0	4560	383	4943
1	618	952	1570
All	5178	1335	6513

Plotted Confusion Matrix

```
[ ] display_cm_clf_RandomForest = ConfusionMatrixDisplay(confusion_matrix=cfm_clf_RandomForest)
```

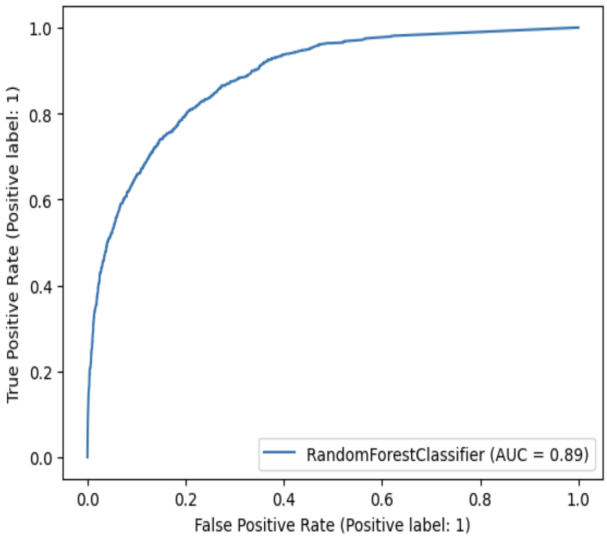
```
# Plots the Confusion Matrix
display_cm_clf_RandomForest.plot(cmap="Blues_r")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff1db7b3d60>
```



Plotted ROC Curve. From the below graph, we can see that the AUC(Area Under Curve) is 0.89 for Random Forest Classifier.

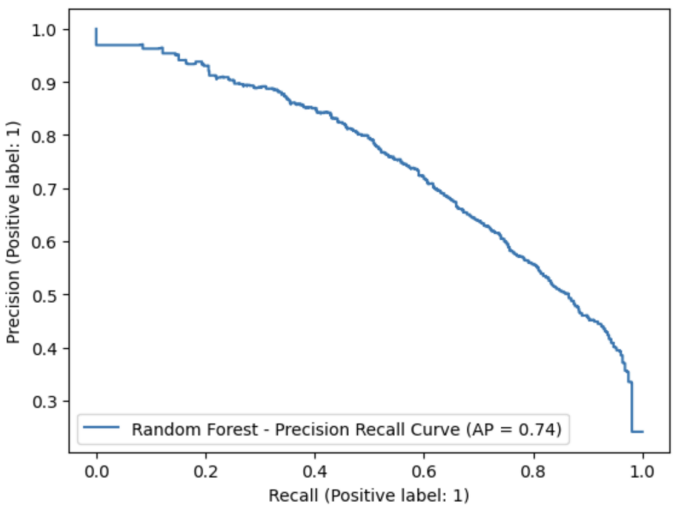
```
# from sklearn.metrics import RocCurveDisplay and displays the ROC Curve
from sklearn.metrics import RocCurveDisplay
clf_RandomForest_display = RocCurveDisplay.from_estimator(clf_RandomForest, X_test, y_test)
```



Plotted Precision Recall display. From the below graph, we can see that the precision and accuracy are inversely proportional.

```
[ ] # Imports PrecisionRecallDisplay from sklearn.metrics
    from sklearn.metrics import PrecisionRecallDisplay
```

```
# Displays Precision Recall Curve
display_clf_RandomForest = PrecisionRecallDisplay.from_estimator(
    clf_RandomForest, X_test, y_test, name="Random Forest - Precision Recall Curve"
)
```



b. 590 n_estimators, entropy criterion:

Imported required libraries, used 590 n_estimators, entropy criterion, and fitted the model using training data. Obtained an accuracy of 84.96%.

```
[ ] # Imports RandomForestClassifier from sklearn.ensemble
    from sklearn.ensemble import RandomForestClassifier
```

```
[ ] clf_RandomForest = RandomForestClassifier(n_estimators=590, criterion='entropy')
```

```
[ ] # Fits the RandomForest Model
    clf_RandomForest.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=590)
```

```
[ ] # prints accuracy of RandomForest model
    clf_RandomForest.score(X_test, y_test)
```

```
0.8496852448948258
```

Got an accuracy of 97.5% on training data.

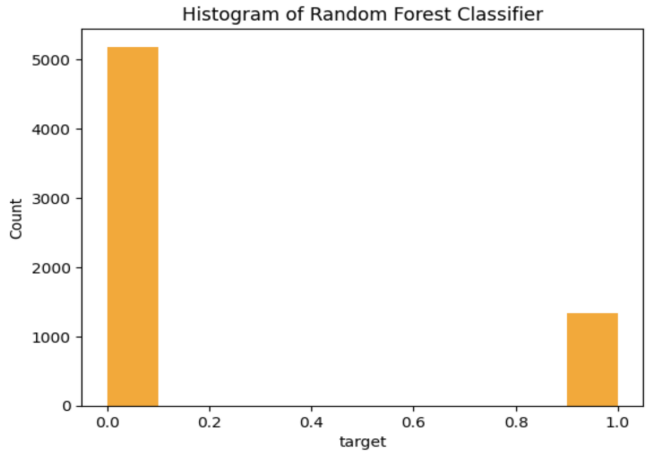
```
[138] from sklearn.metrics import accuracy_score
      train_acc_clf_RandomForest = accuracy_score(y_train, clf_RandomForest.predict(X_train))
      print(train_acc_clf_RandomForest)
```

```
0.9750844594594594
```

Used test data and predicted the output values using test data. Plotted the histogram of predicted value counts. From the below graph, we can see that around 5000 values fall under category ‘0’ of the target variable and 1000+ values fall under category ‘1’ of the target variable.

```
# Calculates predicted output for test input and plots histogram of counts
y_pred = clf_RandomForest.predict(X_test)
plt.hist(y_pred, color = 'orange')
plt.title('Histogram of Random Forest Classifier')
plt.xlabel('target')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



Printed Classification Report. From the below figure, we can see that precision, recall, and f1-score for target variable '0' are 88%, 93%, and 90% respectively. On the other hand precision, recall, and f1-score for target variable '1' are 72%, 61%, and 66% respectively.

```
[ ] # Imports classification_report from sklearn.metrics and prints classification_report
    from sklearn.metrics import classification_report
    print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      4943
           1       0.72      0.61      0.66      1570

    accuracy                           0.85      6513
   macro avg       0.80      0.77      0.78      6513
weighted avg       0.84      0.85      0.85      6513
```

Printed Confusion Matrix. From the below matrix, we can see that 4573 of the target variable with a '0' value is predicted accurately and 961 values with a target '1' are predicted correctly. On the other hand, 609 target variables of the '1' value are predicted incorrectly as '0', and 370 values with '0' are predicted as '1'.

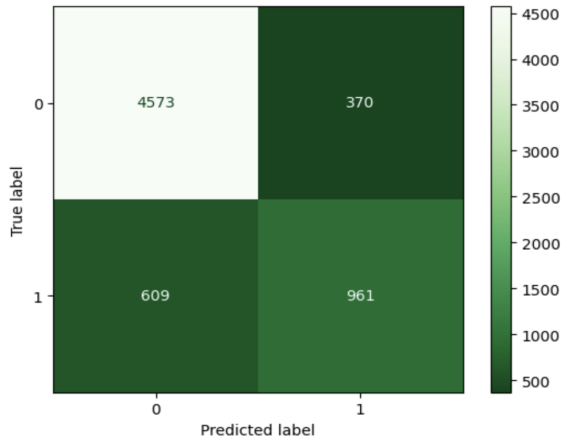
```
[ ] # Imports confusion_matrix from sklearn.metrics and prints Confusion Matrix
    from sklearn.metrics import confusion_matrix
    cfm = confusion_matrix(y_test, y_pred)
    pd.crosstab(y_test, y_pred, rownames = ['Actual'], colnames =['Predicted'], margins = True)
```

Predicted	0	1	All
Actual			
0	4573	370	4943
1	609	961	1570
All	5182	1331	6513

Plotted Confusion Matrix

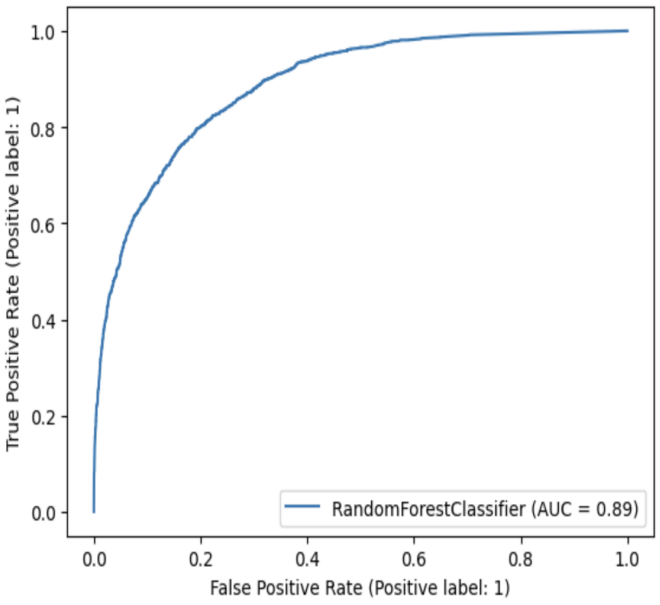
```
# Imports ConfusionMatrixDisplay from sklearn.metrics and plots Confusion Matrix
from sklearn.metrics import ConfusionMatrixDisplay
display_cm_clf_RandomForest = ConfusionMatrixDisplay(confusion_matrix=cfm)
display_cm_clf_RandomForest.plot(cmap="Greens_r")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff1db5ac070>
```



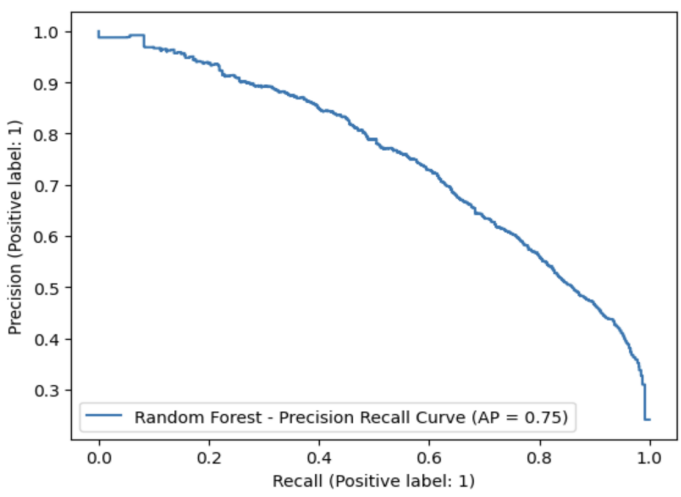
Plotted ROC Curve. From the below graph, we can see that AUC(Area Under Curve) for Random Forest is 0.89.

```
# Imports RocCurveDisplay from sklearn.metrics and displays ROC Curve
from sklearn.metrics import RocCurveDisplay
clf_RandomForest_display = RocCurveDisplay.from_estimator(clf_RandomForest, X_test, y_test)
```



Precision-Recall curve. From the below graph, we can see that precision and recall are inversely proportional.

```
# Precision Recall Curve
from sklearn.metrics import PrecisionRecallDisplay
display = PrecisionRecallDisplay.from_estimator(
    clf_RandomForest, X_test, y_test, name="Random Forest - Precision Recall Curve"
)
```



7. Stochastic Gradient Descent:

We have chosen Stochastic Gradient Descent because there are a lot of options for parameter tuning. We have used “hinge” loss in one model and “log_loss” loss and elasticnet in another model. There are different loss functions available like hinge, log_loss. Penalties like L2, L1, and elasticnet can be used for hyperparameter tuning.

a. Used “hinge” loss

Imported required libraries and fitted the model using training data and computed the accuracy of the model. Got an accuracy of 81.08% using hinge loss.

```
[ ] # Imports SGDClassifier from sklearn.linear_model
    from sklearn.linear_model import SGDClassifier
```

Stochastic Gradient Descent with "hinge" loss

```
[ ] clf_SGD = SGDClassifier(loss="hinge")
```

```
[ ] # Fits the Stochastic Gradient Descent model
    clf_SGD.fit(X_train, y_train)
```

```
▾ SGDClassifier
SGDClassifier()
```

```
# Prints the accuracy of Stochastic Gradient Descent model
clf_SGD.score(X_test, y_test)
```

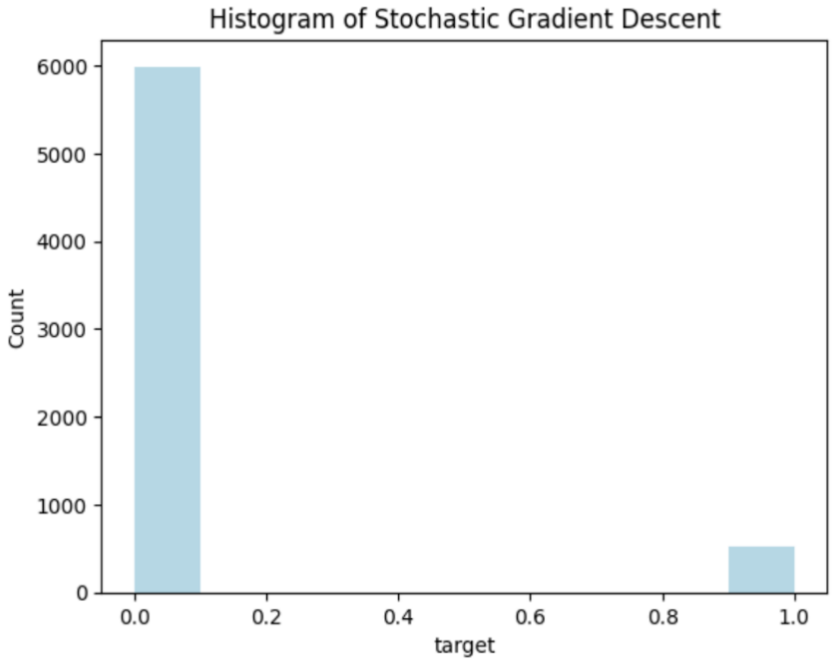
```
0.8108398587440504
```

Got an accuracy of 78.01% on training data.

```
# Prints accuracy on training data
from sklearn.metrics import accuracy_score
train_acc_clf_SGD = accuracy_score(y_train, clf_SGD.predict(X_train))
print(train_acc_clf_SGD)
```

```
0.7801750614250614
```

Used test data and predicted the output values using test data. Plotted the histogram of predicted value counts. From the below graph, we can see that around 6000 values fall under the category of target variable ‘0’ and <1000 values fall under the target variable category ‘1’.



Printed Classification Report. From the report, we can see that the precision for target variable '0' is less than that of target variable '1' which is in contrast to other models. Whereas recall and f1-score for target variable '0' are way more than those values for target variable '1'. Precision, recall, and f1-score for target variable '0' are 81%, 98%, and 89% respectively. These values for target variable '1' are 82%, 27%, and 41% respectively.

```
[ ]  # Imports and prints classification_report
     from sklearn.metrics import classification_report
     print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.81      0.98      0.89      4943
               1       0.82      0.27      0.41      1570

        accuracy                           0.81      6513
       macro avg       0.82      0.63      0.65      6513
    weighted avg       0.81      0.81      0.77      6513
```

Printed Confusion Matrix. From the matrix, we can observe that 4943 values fall under the target variable '0' and 1570 values fall under the target variable '1' category. The target '0' variables are far more than the target '1' variable values. 4850 values of target variable '0' are predicted correctly as '0' and 431 values of the target under '1' are predicted correctly. 93 values of the target under the '0' category are predicted incorrectly as '1' and 1139 values of the target variable under the '1' category are predicted incorrectly as '0'.

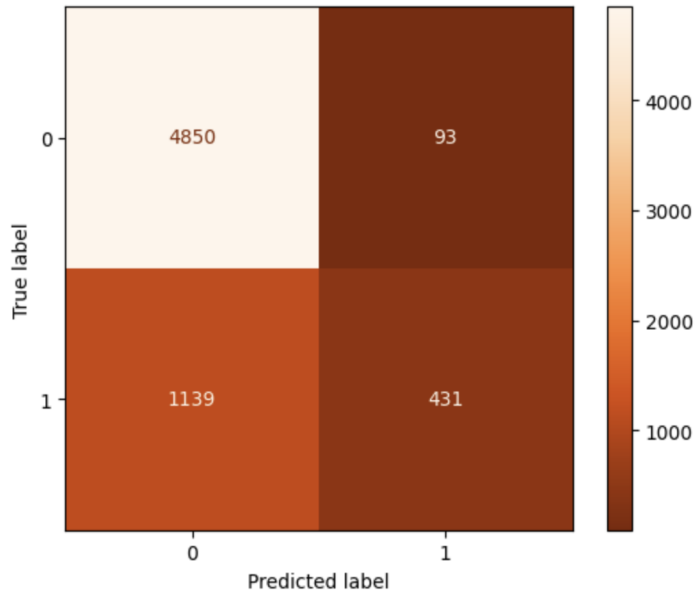
```
[ ] # Imports and prints Confusion Matrix
    from sklearn.metrics import confusion_matrix
    cfm_SGD = confusion_matrix(y_test, y_pred)
    pd.crosstab(y_test, y_pred, rownames = ['Actual'], colnames =['Predicted'], margins = True)
```

Predicted	0	1	All
Actual			
0	4850	93	4943
1	1139	431	1570
All	5989	524	6513

Below is the plot of Confusion Matrix.

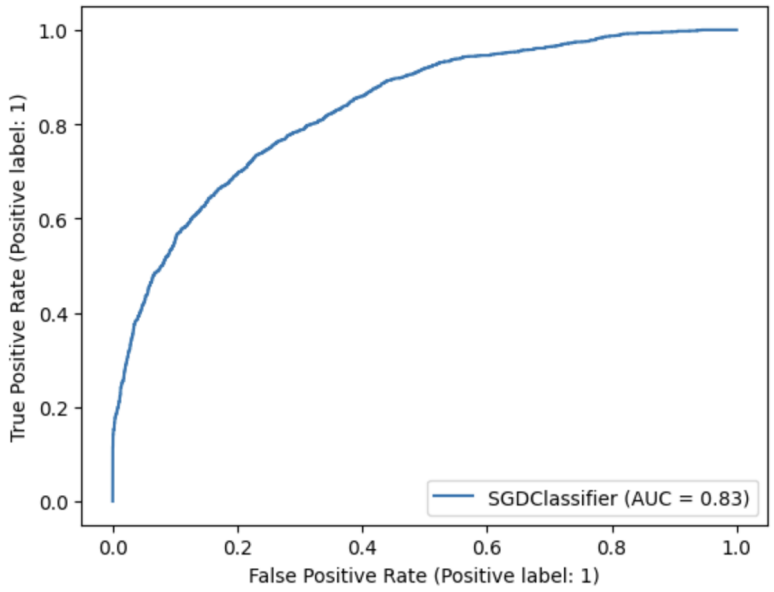
```
# Imports ConfusionMatrixDisplay and plots Confusion Matrix
from sklearn.metrics import ConfusionMatrixDisplay
display_cm_clf_StochasticGradientDescent = ConfusionMatrixDisplay(confusion_matrix=cfm_SGD)
display_cm_clf_StochasticGradientDescent.plot(cmap="Oranges_r")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff1db4a3f10>
```



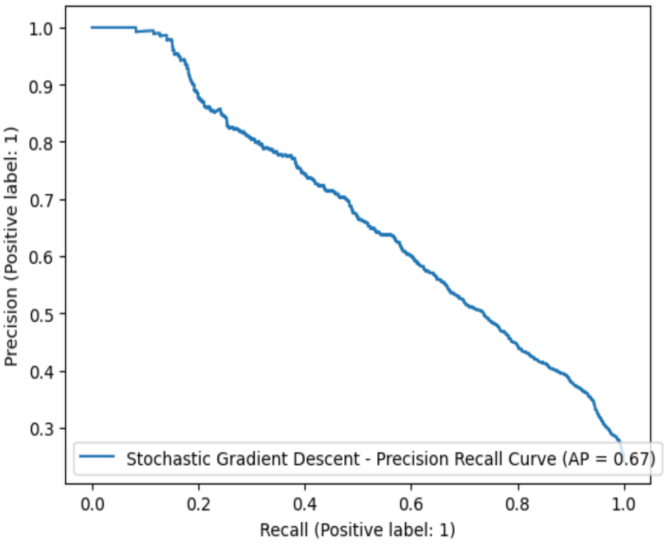
Plotted ROC Curve. From the below graph, we can see that the AUC(Area Under Curve) for Stochastic Gradient Descent is 0.83.

```
# Imports RocCurveDisplay and prints ROC Curve
from sklearn.metrics import RocCurveDisplay
clf_StochasticGradientDescent_display = RocCurveDisplay.from_estimator(clf_SGD, X_test, y_test)
```



Plotted Precision Recall display

```
# Precision Recall Curve
from sklearn.metrics import PrecisionRecallDisplay
display = PrecisionRecallDisplay.from_estimator(
    clf_SGD, X_test, y_test, name="Stochastic Gradient Descent - Precision Recall Curve"
)
```



b. Used “log_loss” loss and “elasticnet” penalty

Imported required libraries and fitted the model using training data and computed the accuracy of the model. With log_loss and elastcinet, obtained an accuracy of 83.12% which is higher than the hinge loss for which the accuracy was 81.08%

Stochastic Gradient Descent with "log_loss" loss and "elasticnet" penalty

```
[ ] clf_SGD_logloss_elasticnet = SGDClassifier(loss="log_loss", penalty="elasticnet")
```

```
[ ] # fits the model
    clf_SGD_logloss_elasticnet.fit(X_train, y_train)
```

```
SGDClassifier
SGDClassifier(loss='log_loss', penalty='elasticnet')
```

```
# Prints accuracy of model
clf_SGD_logloss_elasticnet.score(X_test, y_test)
```

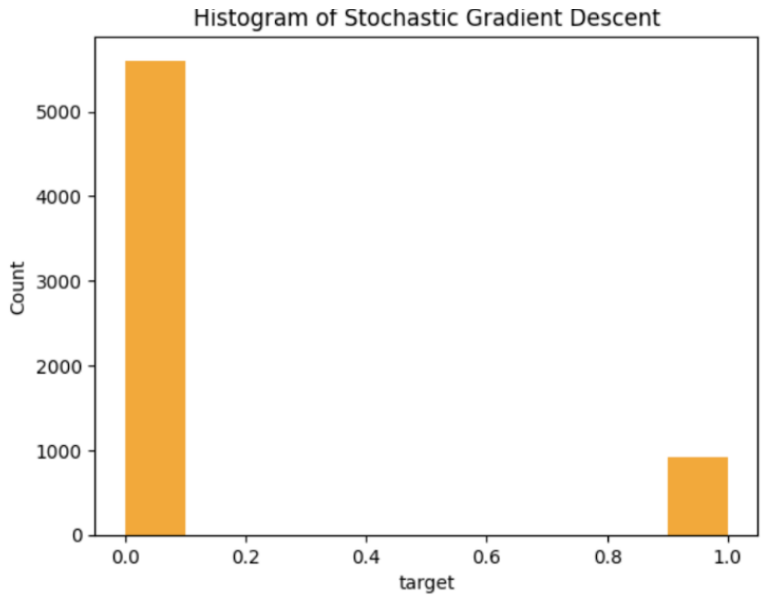
```
0.831260555811454
```

Printed the accuracy of training data which is 79.10%

```
# Prints acuracy on training data
from sklearn.metrics import accuracy_score
train_acc_clf_SGD_logloss_elasticnet = accuracy_score(y_train, clf_SGD_logloss_elasticnet.predict(X_train))
print(train_acc_clf_SGD_logloss_elasticnet)
```

```
0.7910780098280098
```

Used test data and predicted the output values using test data. Plotted the histogram of predicted value counts. From the below graph, we can see that 5000+ values are predicted under the target variable ‘0’ category, and 1000+ values are predicted under the target variable ‘1’ category.



Printed Classification Report. From the below report, we can see that the precision, recall, and f1-score for the target ‘0’ variable is higher than the target ‘1’ variable. For the target ‘0’ variable the precision, recall, and f1-score values are 84%, 96%, and 90% respectively. On the other hand, these values for target variable ‘1’ are 76%, 44%, and 56% respectively.

```
[ ]  # Imports and prints Classification Report
     from sklearn.metrics import classification_report
     print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.84      0.96      0.90      4943
               1       0.76      0.44      0.56      1570

        accuracy                           0.83      6513
       macro avg       0.80      0.70      0.73      6513
    weighted avg       0.82      0.83      0.81      6513
```

Printed Confusion Matrix. From the below matrix, we can see that there are a total of 6513 samples. Out of which 4723 of target ‘0’ values are predicted correctly as ‘0’ and 691 targets ‘1’ variables are predicted correctly as ‘1’. On the other hand, 220 target ‘0’ values are predicted incorrectly as ‘1’, and 879 target ‘1’ values are predicted incorrectly as ‘0’.

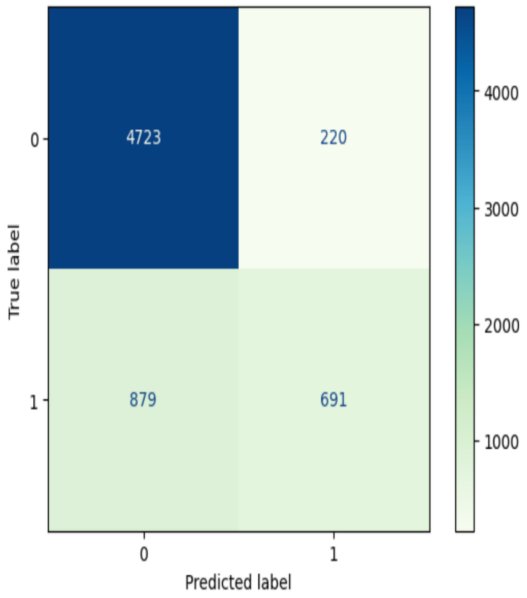
```
[ ]  # Imports and prints Confusion Matrix
     from sklearn.metrics import confusion_matrix
     cfm_clf_SGD_logloss_elasticnet = confusion_matrix(y_test, y_pred)
     pd.crosstab(y_test, y_pred, rownames = ['Actual'], colnames =['Predicted'], margins = True)
```

Predicted	0	1	All
Actual			
0	4723	220	4943
1	879	691	1570
All	5602	911	6513

Below is the plot of Confusion Matrix.

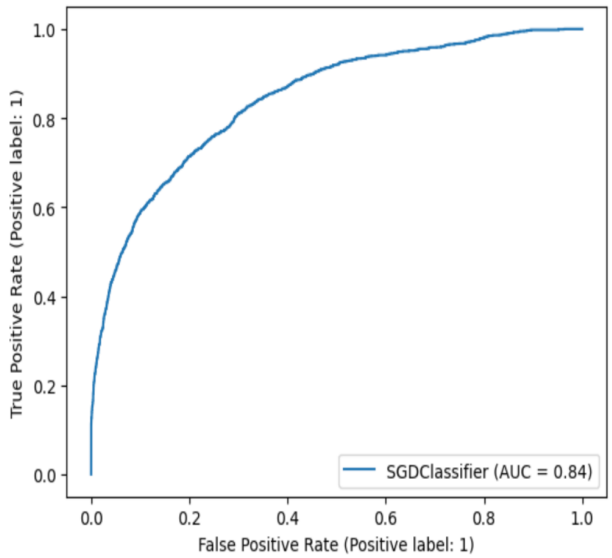
```
# Imports ConfusionMatrixDisplay and plots Confusion Matrix
from sklearn.metrics import ConfusionMatrixDisplay
display_cfm_clf_SGD_logloss_elasticnet = ConfusionMatrixDisplay(confusion_matrix=cfm_clf_SGD_logloss_elasticnet)
display_cfm_clf_SGD_logloss_elasticnet.plot(cmap="GnBu")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff1db28c8e0>
```



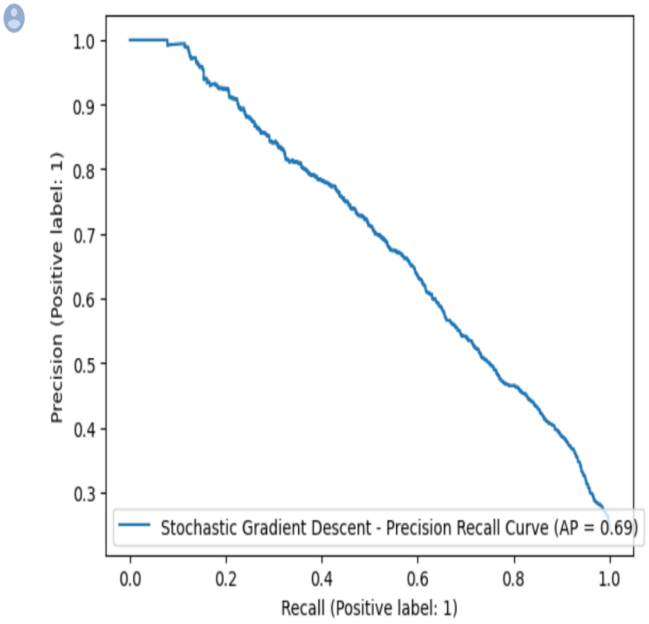
Plotted ROC Curve. From the below graph, we can see that the AUC(Area Under Curve) for the Stochastic Gradient Descent model is 0.84.

```
[ ] # Imports RocCurveDisplay and displays ROC Curve
    from sklearn.metrics import RocCurveDisplay
    clf_SGD_logloss_elasticnet_display_ROC = RocCurveDisplay.from_estimator(clf_SGD_logloss_elasticnet, X_test, y_test)
```



Plotted Precision Recall display

```
# Precision Recall Curve
from sklearn.metrics import PrecisionRecallDisplay
display_clf_SGD_logloss_elasticnet = PrecisionRecallDisplay.from_estimator(
    clf_SGD_logloss_elasticnet, X_test, y_test, name="Stochastic Gradient Descent - Precision Recall Curve"
)
```



As the recall rate is increasing, the precision rate is decreasing. The Precision-recall graph is plotted based on considering only positive outcomes of the model prediction.

8. AdaBoostClassifier:

We have chosen AdaBoostClassifier because it concentrates on fitting weak learners. Using n_estimators weak learners can be controlled. The default value of this is 50.

Imported required libraries and fitted the model using training data and computed the accuracy of the model. Printed the importance of various features using feature_importances. Achieved accuracy of 86.01% on test data.

```
[270] # Imports AdaBoostClassifier from sklearn
      from sklearn.ensemble import AdaBoostClassifier

[271] clf_AdaBoostClassifier = AdaBoostClassifier()

      # Fits AdaBoostClassifier Model
      clf_AdaBoostClassifier.fit(X_train, y_train)

      ▾ AdaBoostClassifier
      AdaBoostClassifier()

[ ]   # Prints the importance of various features
      clf_AdaBoostClassifier.feature_importances_

      array([0.12, 0.04, 0.12, 0.04, 0.16, 0.12, 0.  , 0.04, 0.18, 0.16, 0.02,
             0.  ])

[ ]   # Prints the accuracy of AdaBoostClassifier model
      clf_AdaBoostClassifier.score(X_test, y_test)

      0.8601259020420697
```

Obtained accuracy of 86.25% on training data.

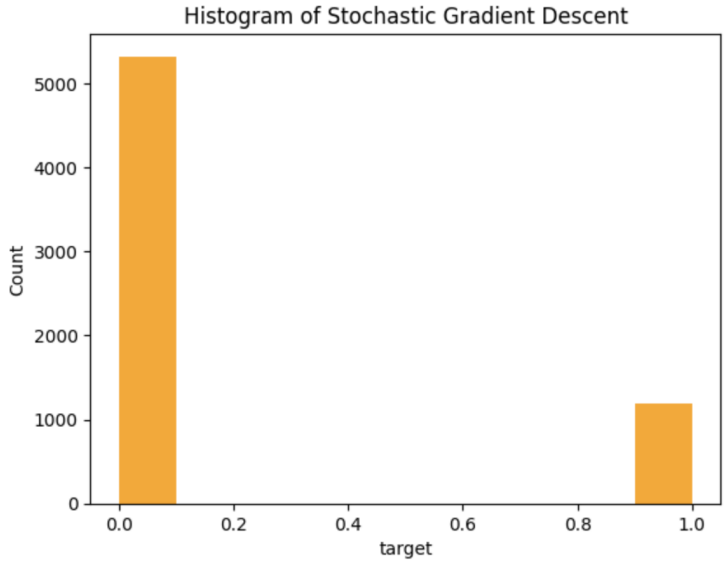
```
# Prints accuracy on training data
train_clf_AdaBoostClassifier_acc = accuracy_score(y_train, clf_AdaBoostClassifier.predict(X_train))
print(train_clf_AdaBoostClassifier_acc)
```

```
0.8625998157248157
```

Used test data and predicted the output values using test data. Plotted the histogram of predicted value counts. We can see that there are around 5000+ values under category ‘0’ of the target variable and 1000+ values under category ‘1’ of the target variable. More people are under the <=50K category when compared to the>50K category.

```
# Predicts the output of test data and plots histogram plot of output
y_pred_clf_AdaBoostClassifier = clf_AdaBoostClassifier.predict(X_test)
plt.hist(y_pred_clf_AdaBoostClassifier, color = 'orange')
plt.title('Histogram of Stochastic Gradient Descent')
plt.xlabel('target')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



Printed Classification Report. From the report, we can see that precision, recall, and f1-score for target variable ‘0’ are higher when compared to target variable ‘1’. Precision, recall, and f1-score for target variable ‘0’ are 88%, 95%, and 91% respectively. On the other hand, these values for target variable ‘1’ are 78%, 59%, and 67% respectively.

```
[ ] # Imports and prints classification_report
    from sklearn.metrics import classification_report
    print(classification_report(y_test, y_pred_clf_AdaBoostClassifier))
```

```
                  precision    recall  f1-score   support

               0       0.88      0.95      0.91      4943
               1       0.78      0.59      0.67      1570

        accuracy                           0.86      6513
       macro avg       0.83      0.77      0.79      6513
    weighted avg       0.85      0.86      0.85      6513
```

Printed Confusion Matrix. From the matrix we can see that 4676 target variables '0' are predicted correctly as '0' and 926 target variables with '1' are printed correctly as '1'. 267 target variables with '0' are predicted incorrectly as '1'. 644 values with target variable '1' are predicted incorrectly as '0'.

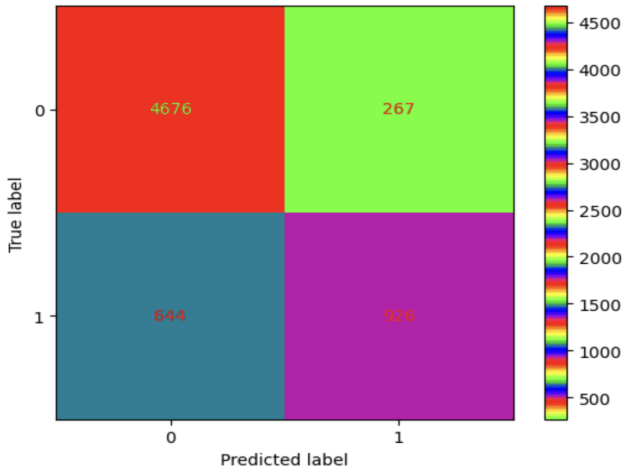
```
[ ] # Imports and prints Confusion Matrix
    from sklearn.metrics import confusion_matrix
    cfm_clf_AdaBoostClassifier = confusion_matrix(y_test, y_pred_clf_AdaBoostClassifier)
    pd.crosstab(y_test, y_pred_clf_AdaBoostClassifier, rownames = ['Actual'], colnames =['Predicted'], margins = True)
```

Predicted	0	1	All
Actual			
0	4676	267	4943
1	644	926	1570
All	5320	1193	6513

Plotted Confusion Matrix

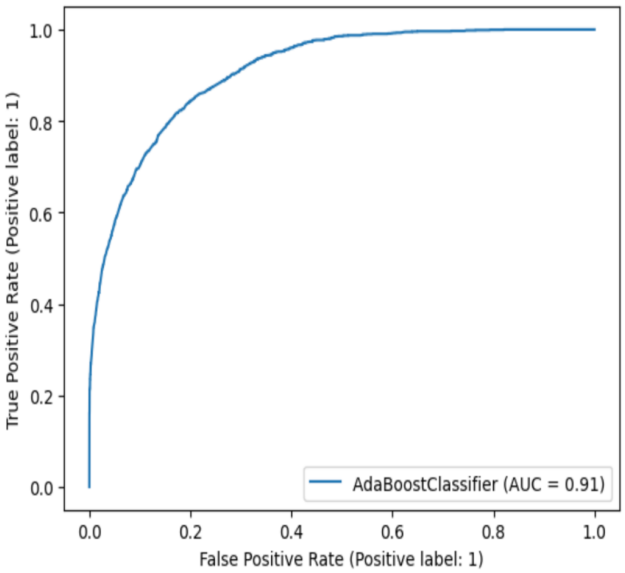
```
# Imports ConfusionMatrixDisplay and plots Confusion Matrix
from sklearn.metrics import ConfusionMatrixDisplay
display_cfm_clf_AdaBoostClassifier = ConfusionMatrixDisplay(confusion_matrix=cfm_clf_AdaBoostClassifier)
display_cfm_clf_AdaBoostClassifier.plot(cmap="prism_r")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff1d9cd8310>
```



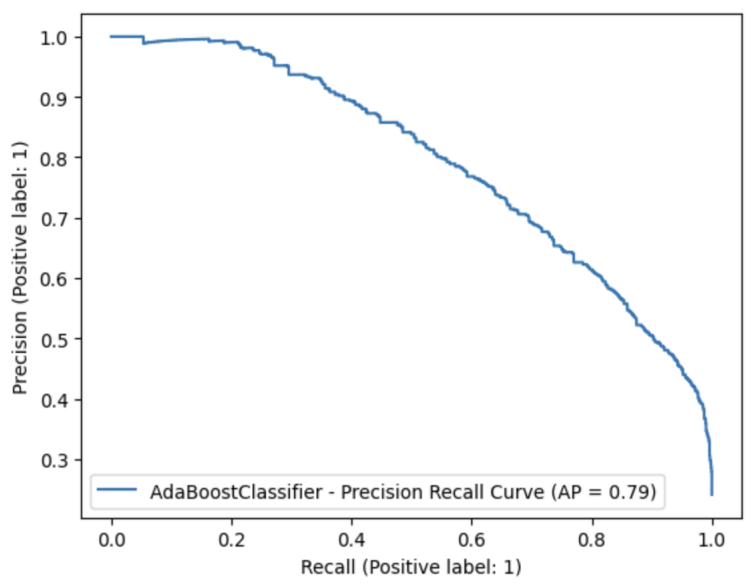
Plotted ROC Curve. From the below graph, we can see that AUC which is Area Under Curve for AdaBoost Classifier is 0.91.

```
# Imports RocCurveDisplay and displays ROC Curve
from sklearn.metrics import RocCurveDisplay
clf_AdaBoostClassifier_display_ROC = RocCurveDisplay.from_estimator(clf_AdaBoostClassifier, X_test, y_test)
```



Plotted Precision Recall display. From the below curve, we can see that precision and recall are inversely proportional. When the recall is 0 precision is '1' and as recall increased precision value decreased.

```
# Imports PrecisionRecallDisplay and displays Precision Recall Curve
from sklearn.metrics import PrecisionRecallDisplay
display_clf_AdaBoostClassifier = PrecisionRecallDisplay.from_estimator(
    clf_AdaBoostClassifier, X_test, y_test, name="AdaBoostClassifier - Precision Recall Curve"
)
```



Overview of Model's Accuracy:

Model	Metric	Train Accuracy	Test Accuracy
Logistic Regression	solver -'lbfgs'	82.52	83.04
Naive Bayes	model-GaussianNB()	80.94	80.94
Decision Tree	max_depth - 3	97.50	81.75
KNN	n_neighbors - 39	84.4	84.2
SVC	Kernel - 'rbf'	80.2	84.9
Random Forest Classifier	Criterion - 'entropy' n_estimators = 590	97.5	84.9
Stochastic Gradient Descent	loss - 'log_loss' penalty - 'elasticnet'	79.1	82.6
AdaBoostClassifier	model-AdaBoostClassifier()	86.25	86.01

Out of all the models executed, obtained highest test accuracy of 86.01% for AdaBoostClassifier whereas training accuracy is highest with 97.5% for Decision Tree and Random Forest Classifiers.

Name	UBIT Name	UB Email
Viditha Wudaru	vidithaw	vidithaw@buffalo.edu
Jayavani Akkiraju	jayavani	jayavani@buffalo.edu

References:

1. https://archive.ics.uci.edu/ml/datasets/adult
2. https://www.kaggle.com/datasets/uciml/adult-census-income
3. https://www.ibm.com/topics/logistic-regression - Logistic Regression
4. https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning - AUC
5. https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning
6. https://www.nickmccullum.com/python-machine-learning/logistic-regression-python/
7. https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn
8. https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/
9. https://www.datacamp.com/tutorial/naive-bayes-scikit-learn
10. https://machinelearningknowledge.ai/knn-classifier-in-sklearn-using-gridsearchcv-with-example/
11. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
12. https://www.tutorialexample.com/an-introduction-to-accuracy-precision-recall-f1-score-in-machine-learning-machine-learning-tutorial/ - Accuracy and scores
13. https://www.researchgate.net/figure/Confusion-matrix-The-accuracy-precision-recall-F1-score-and-AUC-mainly-rely-on-the_fig2_355985914 - Confusion Matrix
14. https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning -Precision-recall Graphs
15. https://scikit-learn.org/stable/modules/tree.html_ -Decision Tree
16. https://www.datacamp.com/tutorial/decision-tree-classification-python -Decision Tree
17. https://scikit-learn.org/stable/modules/svm.html#classification -SVM
18. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html#sklearn.metrics.RocCurveDisplay.from_estimator
19. https://medium.com/towards-data-science/a-practical-guide-to-seven-essential-performance-metrics-for-classification-using-scikit-learn-2de0e0a8a040 - ROC Curve
20. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_curve.html#
21. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#sphx-glr-auto-examples-model-selection-plot-precision-recall-py - Precision Recall Display
22. https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769
23. https://www.mikulskibartosz.name/how-to-visualise-prediction-errors/ - Prediction Errors Histogram

24. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html - Confusion Matrix Display
25. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
26. https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6
27. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier
28. https://scikit-learn.org/stable/modules/sgd.html#classification
29. https://stackoverflow.com/questions/57043260/how-change-the-color-of-boxes-in-confusion-matrix-using-sklearn - map colors
30. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
31. https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248
32. https://towardsdatascience.com/understanding-random-forest-58381e0602d2
33. https://www.datacamp.com/tutorial/decision-tree-classification-python
34.