Project Title : Credit Default Prediction at BestCard

The below Code has been developed by Team-Data-02

Problem Statement:

BestCard, a credit card company, seeks to leverage data-driven insights to enhance their risk management strategies and minimize financial losses due to customer defaults. As a consultant, tasked with analyzing a dataset encompassing demographic and recent financial information for 30,000 account holders, our objective is to develop predictive models that can accurately identify customers at risk of defaulting on their credit card payments. The dataset includes individual credit account-level data, with each account uniquely represented by a single row. Additionally, each row indicates whether the account owner defaulted in the subsequent month following a six-month historical data review period. Defaulting, in this context, refers to the failure to meet the minimum payment requirement. Our analysis aims to uncover patterns, trends, and key factors associated with defaulting behavior, ultimately enabling BestCard to implement proactive measures, such as targeted interventions or personalized financial products, to mitigate default risks and optimize customer retention strategies.

Data Description

- LIMIT_BAL: it includes both the individual consumer credit and his/her family (supplementary) credit.
- Gender (1 = male; 2 = female).
- Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- Marital status (1 = married; 2 = single; 3 = others).
- Age (year).
- PAY_1 to PAY_6 - represent the payment status of credit card holders for the last six months:
- **-2**: Indicates no consumption; the credit card holder had no outstanding balance.
- **-1**: Indicates that the credit card holder paid the full balance.
- **0**: Indicates that the credit card holder paid the minimum payment but not the full balance.
- **1**: Indicates that the credit card holder made a payment delay for one month.
- **2**: Indicates that the credit card holder made a payment delay for two months.

- Amount of bill statement (BILL_AMT1 - BILL_AMT6): amount of bill statement for the last six months
- Amount of previous payment (PAY_AMT1 - PAY_AMT6): amount paid for the last six months
- Default Payment Next Month:
  - This column typically represents whether a credit card holder defaulted on their payment in the next month.
    - `1` indicates that the individual defaulted on their payment.
    - `0` indicates that the individual did not default on their payment.
- Education_cat:
  The "Education_cat" column likely represents the education level of the cardholders, encoded into categorical values such as graduate school, university, high school, or others.
- graduate school, high school, others, university:
  These columns seem to encode the education levels of cardholders into binary values, indicating whether they belong to specific education categories

Data Exploration

```
# Importing warning for ignore warnings
import warnings
warnings.filterwarnings("ignore")

# Importing pandas for data manipulation and sklearn for machine
learning utilities
import pandas as pd

#Reading the dataset from the CSV file named "BestCard_Data.csv" into
a Pandas DataFrame named "credit"
credit = pd.read_csv(r'BestCard_Data.csv')
#Printing the variable customers in a Jupyter Notebook cell will
display the contents of the DataFrame
credit

                  ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_1
PAY_2  \
0      798fc410-45c1      20000    2          2         1   24      2
2
1      8a8c8f3b-8eb4     120000    2          2         2   26     -1
2
2      85698822-43f5      90000    2          2         2   34      0
0
3      0737c11b-be42      50000    2          2         1   37      0
```

```
0
4      3b7f77cc-dbc0      50000    1          2          1   57      -1
0
...              ...        ...  ...        ...        ...  ...     ...
...
26659  ecff42d0-bdc6     220000    1          3          1   39       0
0
26660  99d1fa0e-222b     150000    1          3          2   43      -1
-1
26661  95cdd3e7-4f24      30000    1          2          2   37       4
3
26662  00d03f02-04cd      80000    1          3          1   41       1
-1
26663  15d69f9f-5ad3      50000    1          2          1   46       0
0

       PAY_3  PAY_4  ...  PAY_AMT4  PAY_AMT5  PAY_AMT6  \
0         -1     -1  ...         0         0         0
1          0      0  ...      1000         0      2000
2          0      0  ...      1000      1000      5000
3          0      0  ...      1100      1069      1000
4         -1      0  ...      9000       689       679
...      ...    ...  ...       ...       ...       ...
26659      0      0  ...      3047      5000      1000
26660     -1     -1  ...       129         0         0
26661      2     -1  ...      4200      2000      3100
26662      0      0  ...      1926     52964      1804
26663      0      0  ...      1000      1000      1000

       default payment next month  EDUCATION_CAT  graduate school  \
0                               1     university                0
1                               1     university                0
2                               0     university                0
3                               0     university                0
4                               0     university                0
...                           ...            ...              ...
26659                           0    high school                0
26660                           0    high school                0
26661                           1     university                0
26662                           1    high school                0
26663                           1     university                0

       high school  none  others  university
0                0     0       0           1
1                0     0       0           1
2                0     0       0           1
3                0     0       0           1
4                0     0       0           1
...            ...   ...     ...         ...
26659            1     0       0           0
```

```
26660             1     0       0           0
26661             0     0       0           1
26662             1     0       0           0
26663             0     0       0           1

[26664 rows x 31 columns]
```

```
# Displaying the shape of the 'credit' DataFrame to show the number of
rows and columns
credit.shape
```

```
(26664, 31)
```

```
credit.describe()
```

```
           LIMIT_BAL           SEX     EDUCATION      MARRIAGE
AGE  \
count   26664.000000  26664.000000  26664.000000  26664.000000
26664.000000
mean   167919.054905      1.603060      1.842334      1.556031
35.505213
std    129839.453081      0.489272      0.744661      0.521463
9.227442
min     10000.000000      1.000000      1.000000      1.000000
21.000000
25%     50000.000000      1.000000      1.000000      1.000000
28.000000
50%    140000.000000      2.000000      2.000000      2.000000
34.000000
75%    240000.000000      2.000000      2.000000      2.000000
41.000000
max    800000.000000      2.000000      4.000000      3.000000
79.000000

              PAY_1         PAY_2         PAY_3         PAY_4
PAY_5  \
count  26664.000000  26664.000000  26664.000000  26664.000000
26664.000000
mean      -0.017777     -0.133363     -0.167679     -0.225023     -
0.269764
std        1.126769      1.198640      1.199165      1.167897
1.131735
min       -2.000000     -2.000000     -2.000000     -2.000000     -
2.000000
25%       -1.000000     -1.000000     -1.000000     -1.000000     -
1.000000
50%        0.000000      0.000000      0.000000      0.000000
0.000000
75%        0.000000      0.000000      0.000000      0.000000
0.000000
```

```
max         8.000000      8.000000      8.000000      8.000000
8.000000

       ...       PAY_AMT3       PAY_AMT4       PAY_AMT5       PAY_AMT6
\
count  ...   26664.000000   26664.000000   26664.000000   26664.000000

mean   ...    5259.514964    4887.048717    4843.729973    5257.843047

std    ...   17265.439561   15956.349371   15311.721795   17635.468185

min    ...       0.000000       0.000000       0.000000       0.000000

25%    ...     390.000000     294.750000     242.750000     111.000000

50%    ...    1822.000000    1500.000000    1500.000000    1500.000000

75%    ...    4556.250000    4050.500000    4082.750000    4015.000000

max    ...  889043.000000  621000.000000  426529.000000  528666.000000


       default payment next month  graduate school   high school  \
count                26664.000000     26664.000000  26664.000000
mean                     0.221797         0.352985      0.164266
std                      0.415463         0.477907      0.370524
min                      0.000000         0.000000      0.000000
25%                      0.000000         0.000000      0.000000
50%                      0.000000         0.000000      0.000000
75%                      0.000000         1.000000      0.000000
max                      1.000000         1.000000      1.000000

               none        others    university
count  26664.000000  26664.000000  26664.000000
mean       0.011214      0.004313      0.467222
std        0.105301      0.065532      0.498934
min        0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000
75%        0.000000      0.000000      1.000000
max        1.000000      1.000000      1.000000

[8 rows x 29 columns]
```

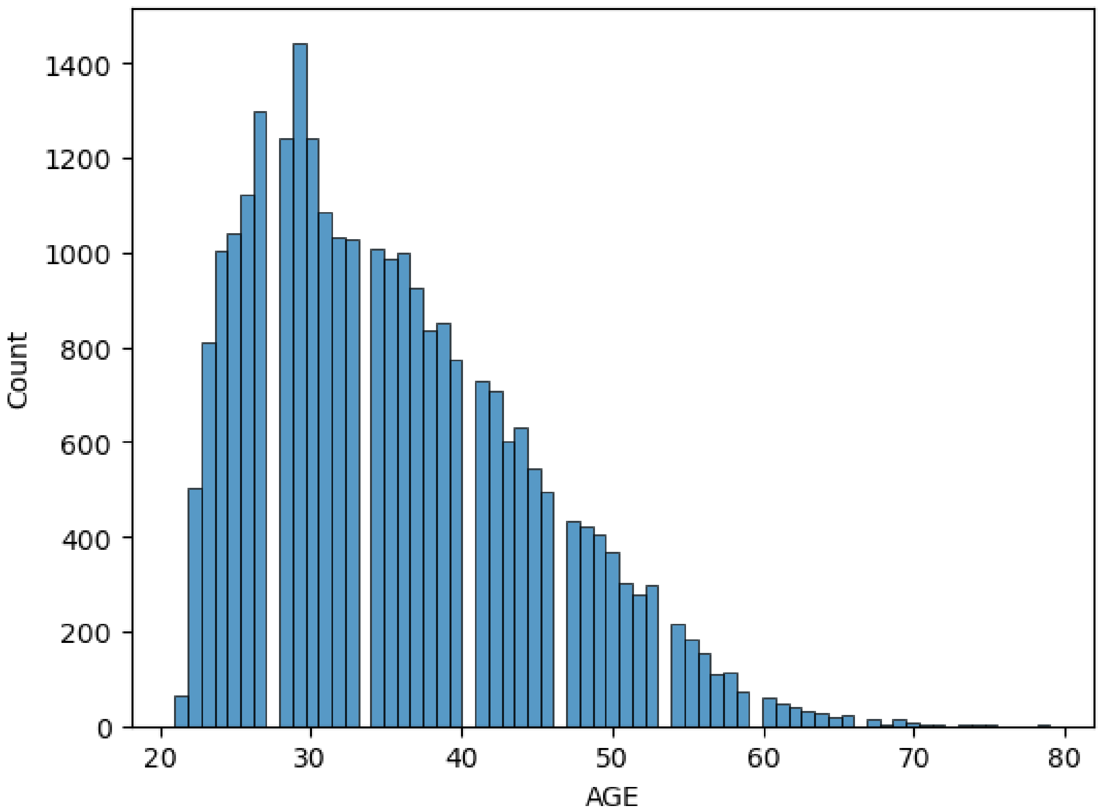
```
# Classify the types of features included in the dataset
credit.dtypes
```

```
ID                               object
LIMIT_BAL                         int64
SEX                               int64
EDUCATION                         int64
```

```
MARRIAGE                          int64
AGE                               int64
PAY_1                             int64
PAY_2                             int64
PAY_3                             int64
PAY_4                             int64
PAY_5                             int64
PAY_6                             int64
BILL_AMT1                         int64
BILL_AMT2                         int64
BILL_AMT3                         int64
BILL_AMT4                         int64
BILL_AMT5                         int64
BILL_AMT6                         int64
PAY_AMT1                          int64
PAY_AMT2                          int64
PAY_AMT3                          int64
PAY_AMT4                          int64
PAY_AMT5                          int64
PAY_AMT6                          int64
default payment next month        int64
EDUCATION_CAT                    object
graduate school                   int64
high school                       int64
none                              int64
others                            int64
university                        int64
dtype: object
```

```python
# visualizing the Age column for better understanding with histplot
import seaborn as sns
import matplotlib.pyplot as plt
sns.histplot(credit['AGE'])
```

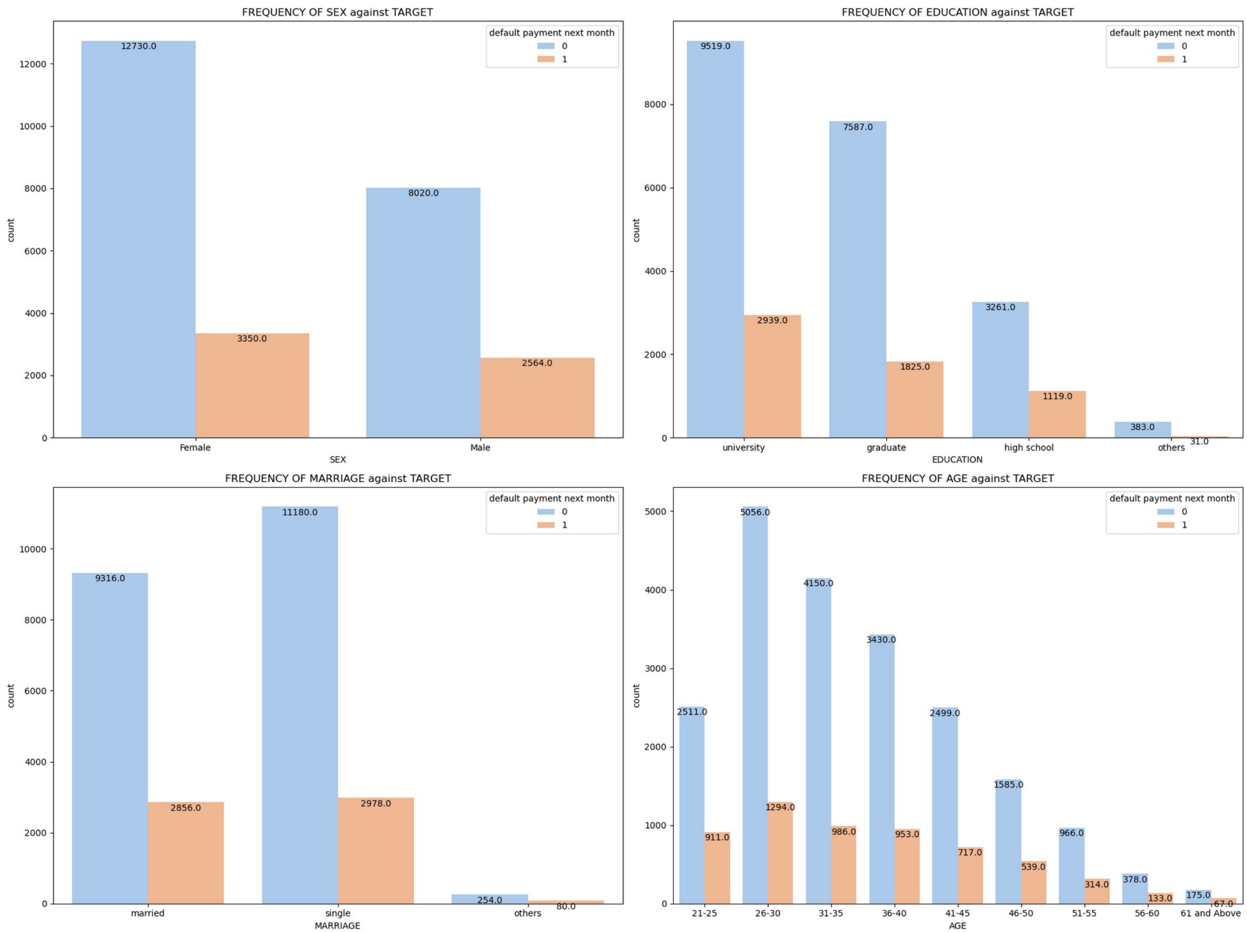
```
<Axes: xlabel='AGE', ylabel='Count'>
```



From this hist plot we can divide age into 9 bins

```
# Selecting specific categorical features
cat_df = credit[['SEX', 'EDUCATION', 'MARRIAGE', 'AGE']]

# we are using Bining in Age column
cat_df['AGE'] = pd.cut(credit.AGE,
bins=[20,25,30,35,40,45,50,55,60,80],labels=['21-25','26-30','31-
35','36-40','41-45','46-50','51-55','56-60','61 and Above'])
# In data given values are  1 = graduate school; 2 = university; 3 =
high school; 4 = others so all other are not given in data we include
them on others
cat_df['EDUCATION']=cat_df['EDUCATION'].apply(lambda x :'graduate' if
x==1 else ('university' if x==2 else ('high school' if x==3  else
'others')))

# data given values are 1 = married; 2 = single; 3 = others so 0 is
not given in data, we include them on others
cat_df['MARRIAGE']=cat_df['MARRIAGE'].apply(lambda x : "married" if
x==1 else ("single" if x==2 else 'others'))

# converting Gender 1 & 2 value in Male & Female for better
understanding
cat_df['SEX'] = cat_df['SEX'].apply(lambda x:'Male'  if x == 1 else
'Female')
```

```
# Plotting FREQUENCY OF categorical feature against TARGET variable
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'cat_df' contains categorical variables and 'credit'
contains the target variable
n = 1
plt.figure(figsize=(20, 15))
for i in cat_df.columns:
    plt.subplot(2, 2, n)
    ax = sns.countplot(x=cat_df[i], hue=credit["default payment next
month"], palette='pastel')
    plt.title(f'FREQUENCY OF {i} against TARGET')
    plt.tight_layout()
    n += 1

    # Add annotations
    for p in ax.patches:
        ax.annotate(f'\n{p.get_height()}', (p.get_x() + p.get_width()
/ 2., p.get_height()), ha='center', va='top', color='black',
xytext=(0, 10), textcoords='offset points')
```



Data Cleaning

```
# Checking for duplicates
duplicates = credit.duplicated().sum()
print("Number of duplicate rows:", duplicates)
```

```
Number of duplicate rows: 0
```

```
# Checking if there are any missing values in each column of the
'credit' DataFrame
credit.isnull().any()
```

```
ID                              False
LIMIT_BAL                       False
SEX                             False
EDUCATION                       False
MARRIAGE                        False
AGE                             False
PAY_1                           False
PAY_2                           False
PAY_3                           False
PAY_4                           False
```

```
PAY_5                             False
PAY_6                             False
BILL_AMT1                         False
BILL_AMT2                         False
BILL_AMT3                         False
BILL_AMT4                         False
BILL_AMT5                         False
BILL_AMT6                         False
PAY_AMT1                          False
PAY_AMT2                          False
PAY_AMT3                          False
PAY_AMT4                          False
PAY_AMT5                          False
PAY_AMT6                          False
default payment next month        False
EDUCATION_CAT                     False
graduate school                   False
high school                       False
none                              False
others                            False
university                        False
dtype: bool
```

```
# Checking the outliers
# Binning the limit balance columns in 10 bins
limit_bal_group = pd.cut(credit['LIMIT_BAL'],bins=10)
limit_bal_group.value_counts()
```

```
LIMIT_BAL
(9210.0, 89000.0]       9572
(89000.0, 168000.0]     5554
(168000.0, 247000.0]    5233
(247000.0, 326000.0]    2753
(326000.0, 405000.0]    1943
(484000.0, 563000.0]     778
(405000.0, 484000.0]     725
(563000.0, 642000.0]      65
(642000.0, 721000.0]      28
(721000.0, 800000.0]      13
Name: count, dtype: int64
```

```
# Checking the dependent variable values on the outlier points
credit[credit['LIMIT_BAL'] >= 703000]['default payment next
month'].value_counts()
```

```
default payment next month
0    18
1     3
Name: count, dtype: int64
```

```
# Removing the outliers
credit=credit[credit['LIMIT_BAL']<=703000].reset_index(drop=True)
credit
```

```
                   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_1
PAY_2  \
0      798fc410-45c1      20000    2          2         1   24      2
2
1      8a8c8f3b-8eb4     120000    2          2         2   26     -1
2
2      85698822-43f5      90000    2          2         2   34      0
0
3      0737c11b-be42      50000    2          2         1   37      0
0
4      3b7f77cc-dbc0      50000    1          2         1   57     -1
0
...              ...        ...  ...        ...       ...  ...    ...
...
26638  ecff42d0-bdc6     220000    1          3         1   39      0
0
26639  99d1fa0e-222b     150000    1          3         2   43     -1
-1
26640  95cdd3e7-4f24      30000    1          2         2   37      4
3
26641  00d03f02-04cd      80000    1          3         1   41      1
-1
26642  15d69f9f-5ad3      50000    1          2         1   46      0
0

       PAY_3  PAY_4  ...  PAY_AMT4  PAY_AMT5  PAY_AMT6  \
0         -1     -1  ...         0         0         0
1          0      0  ...      1000         0      2000
2          0      0  ...      1000      1000      5000
3          0      0  ...      1100      1069      1000
4         -1      0  ...      9000       689       679
...      ...    ...  ...       ...       ...       ...
26638      0      0  ...      3047      5000      1000
26639     -1     -1  ...       129         0         0
26640      2     -1  ...      4200      2000      3100
26641      0      0  ...      1926     52964      1804
26642      0      0  ...      1000      1000      1000

       default payment next month  EDUCATION_CAT  graduate school  \
0                               1     university                0
1                               1     university                0
2                               0     university                0
3                               0     university                0
4                               0     university                0
...                           ...            ...              ...
26638                           0    high school                0
```

```
26639                             0    high school                0
26640                             1     university                0
26641                             1    high school                0
26642                             1     university                0

       high school  none  others  university
0                0     0       0           1
1                0     0       0           1
2                0     0       0           1
3                0     0       0           1
4                0     0       0           1
...            ...   ...     ...         ...
26638            1     0       0           0
26639            1     0       0           0
26640            0     0       0           1
26641            1     0       0           0
26642            0     0       0           1

[26643 rows x 31 columns]
```

```
# Counting the occurrences of each unique value in the target variable
'default payment next month' column
credit['default payment next month'].value_counts()
```

```
default payment next month
0    20732
1     5911
Name: count, dtype: int64
```

```
# Iterating over a list of column names and converting each column to
categorical data type
for col in
['SEX','EDUCATION','MARRIAGE','PAY_1','PAY_2','PAY_3','PAY_4','PAY_5',
'PAY_6']:
    credit[col] = credit[col].astype('category')

# Creating a new DataFrame 'features' by dropping irrelevent columns
from 'credit'
# The dropped columns are: 'ID', 'default payment next month',
'EDUCATION_CAT', 'graduate school', 'high school', 'none', 'others',
'university'
features=credit.drop(['ID','default payment next
month','EDUCATION_CAT','graduate school','high
school','none','others','university'], axis=1)

# Displaying the shape of the 'features' DataFrame to show the number
of rows and columns
features.shape
```

```
(26643, 23)
```

```
# Applying one-hot encoding using get_dummies to the DataFrame
'features' without the unnecessary columns
x= pd.get_dummies(features)

# Displaying the resulting DataFrame after Transformation
x

       LIMIT_BAL  AGE  BILL_AMT1  BILL_AMT2  BILL_AMT3  BILL_AMT4
BILL_AMT5  \
0          20000   24       3913       3102        689          0
0
1         120000   26       2682       1725       2682       3272
3455
2          90000   34      29239      14027      13559      14331
14948
3          50000   37      46990      48233      49291      28314
28959
4          50000   57       8617       5670      35835      20940
19146
...          ...  ...        ...        ...        ...        ...
...
26638     220000   39     188948     192815     208365      88004
31237
26639     150000   43       1683       1828       3502       8979
5190
26640      30000   37       3565       3356       2758      20878
20582
26641      80000   41      -1645      78379      76304      52774
11855
26642      50000   46      47929      48905      49764      36535
32428

       BILL_AMT6  PAY_AMT1  PAY_AMT2  ...  PAY_6_-2  PAY_6_-1  PAY_6_0
\
0              0         0       689  ...      True     False    False

1           3261         0      1000  ...     False     False    False

2          15549      1518      1500  ...     False     False     True

3          29547      2000      2019  ...     False     False     True

4          19131      2000     36681  ...     False     False     True

...          ...       ...       ...  ...       ...       ...      ...

26638      15980      8500     20000  ...     False     False     True

26639          0      1837      3526  ...     False     False     True
```

```
26640      19357        0        0  ...      False      False      True
26641      48944    85900     3409  ...      False       True     False
26642      15313     2078     1800  ...      False      False      True

       PAY_6_2  PAY_6_3  PAY_6_4  PAY_6_5  PAY_6_6  PAY_6_7  PAY_6_8
0        False    False    False    False    False    False    False
1         True    False    False    False    False    False    False
2        False    False    False    False    False    False    False
3        False    False    False    False    False    False    False
4        False    False    False    False    False    False    False
...        ...      ...      ...      ...      ...      ...      ...
26638    False    False    False    False    False    False    False
26639    False    False    False    False    False    False    False
26640    False    False    False    False    False    False    False
26641    False    False    False    False    False    False    False
26642    False    False    False    False    False    False    False

[26643 rows x 87 columns]
```

```python
# Importing LabelEncoder from sklearn.preprocessing
from sklearn.preprocessing import LabelEncoder

# Initializing LabelEncoder
labelencoder_data = LabelEncoder()

# Encoding the target variable 'default payment next month' using
LabelEncoder
y = labelencoder_data.fit_transform(credit['default payment next
month'].values)
```

Model Building:

Splitting the Data:

```
Split the dataset into training, validation, and testing sets to
evaluate the performance of the models properly.
```

```python
# Importing train_test_split from sklearn.model_selection
from sklearn.model_selection import train_test_split

# Splitting the dataset into training and testing sets with a test
size of 20% and a random state of 0
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.20,random_state=0)
```

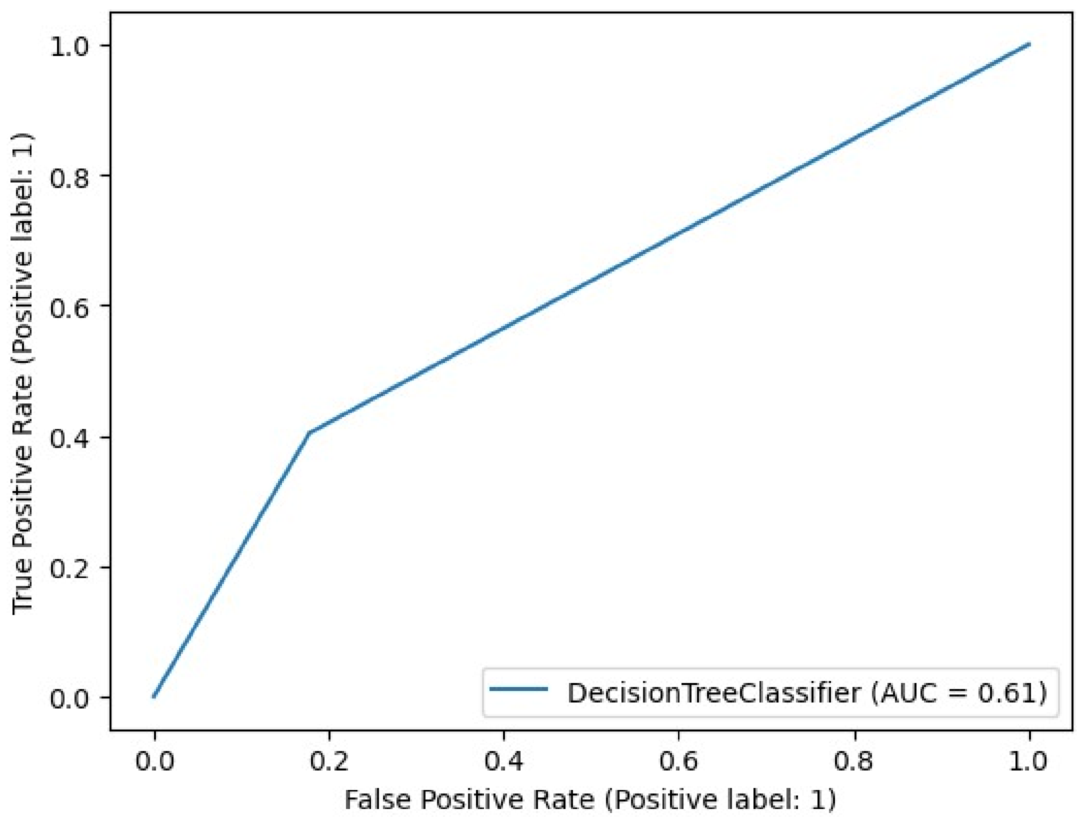
Setting random_state=0 in the train_test_split function ensures reproducibility by fixing the randomness of data splitting, facilitating consistent and comparable machine learning experiments.

```
#decision tree classifier
# Importing DecisionTreeClassifier from sklearn.tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score


# Initializing DecisionTreeClassifier with a random state
tree = DecisionTreeClassifier(random_state=0)

# Fitting the decision tree classifier on the training data
tree.fit(X_train, y_train)

# Making predictions on the test data
y_pred = tree.predict(X_test)

# Printing the accuracy of the model on the training set
print("Accuracy on training set: {:.3f}".format(tree.score(X_train,
y_train)))

# Printing the accuracy of the model on the test set using
accuracy_score from sklearn.metrics
print("Accuracy on test set: {:.3f}".format(accuracy_score(y_pred,
y_test)))
```

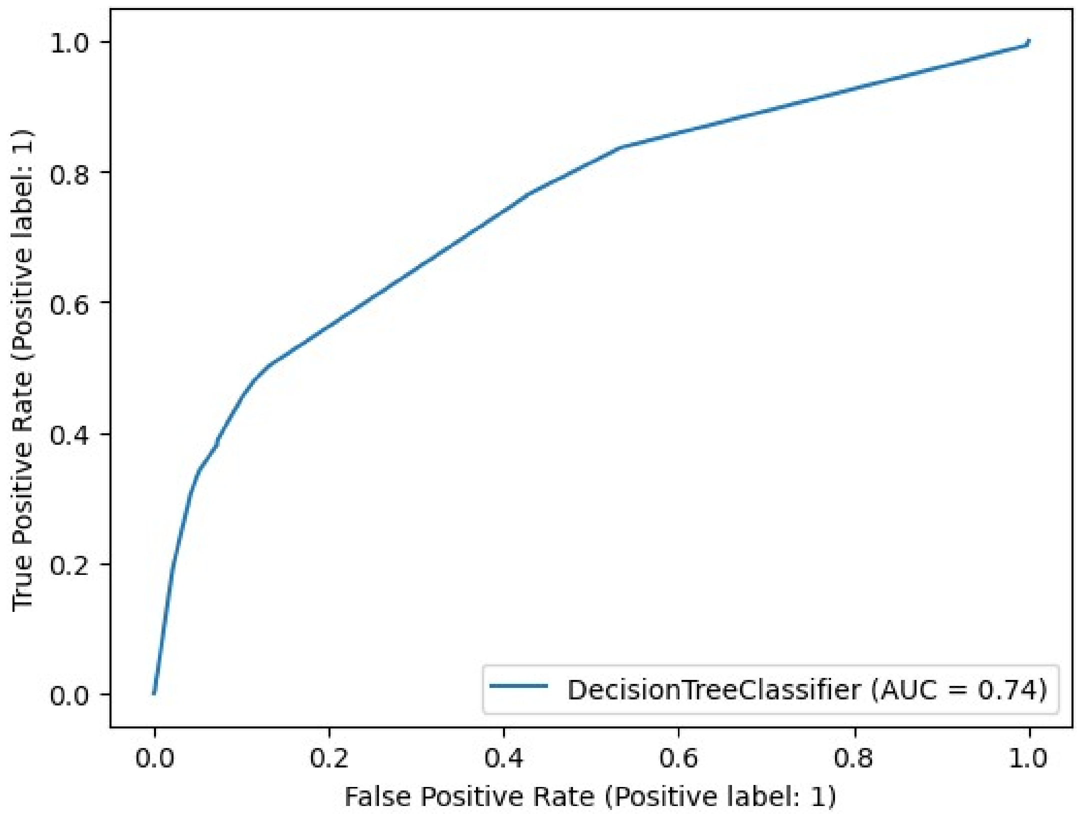
```
Accuracy on training set: 0.999
Accuracy on test set: 0.730
```

```
#decision tree pruning
# Importing DecisionTreeClassifier from sklearn.tree
from sklearn.tree import DecisionTreeClassifier

# Initializing DecisionTreeClassifier with a maximum depth of 5 and a
random state
tree_pruned = DecisionTreeClassifier(max_depth=5, random_state=0)

# Fitting the pruned decision tree classifier on the training data
tree_pruned.fit(X_train, y_train)

# Making predictions on the test data
y_pruned_pred = tree_pruned.predict(X_test)

# Printing the accuracy of the pruned model on the training set
print("Accuracy on training set:
{:.3f}".format(tree_pruned.score(X_train, y_train)))

# Printing the accuracy of the pruned model on the test set using
accuracy_score from sklearn.metrics
```

```
print("Accuracy on test set:
{:.3f}".format(accuracy_score(y_pruned_pred, y_test)))

Accuracy on training set: 0.825
Accuracy on test set: 0.813

# Importing plot_roc_curve from sklearn.metrics
from sklearn.metrics import RocCurveDisplay

# Plotting the ROC curve for the decision tree classifier 'tree' using
the test data
RocCurveDisplay.from_estimator(tree, X_test, y_test)

<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x17e0d97df10>
```




```
# Plotting the ROC curve for the pruned decision tree classifier
'tree_pruned' using the test data
RocCurveDisplay.from_estimator(tree_pruned, X_test, y_test)

<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x17e0da07690>
```

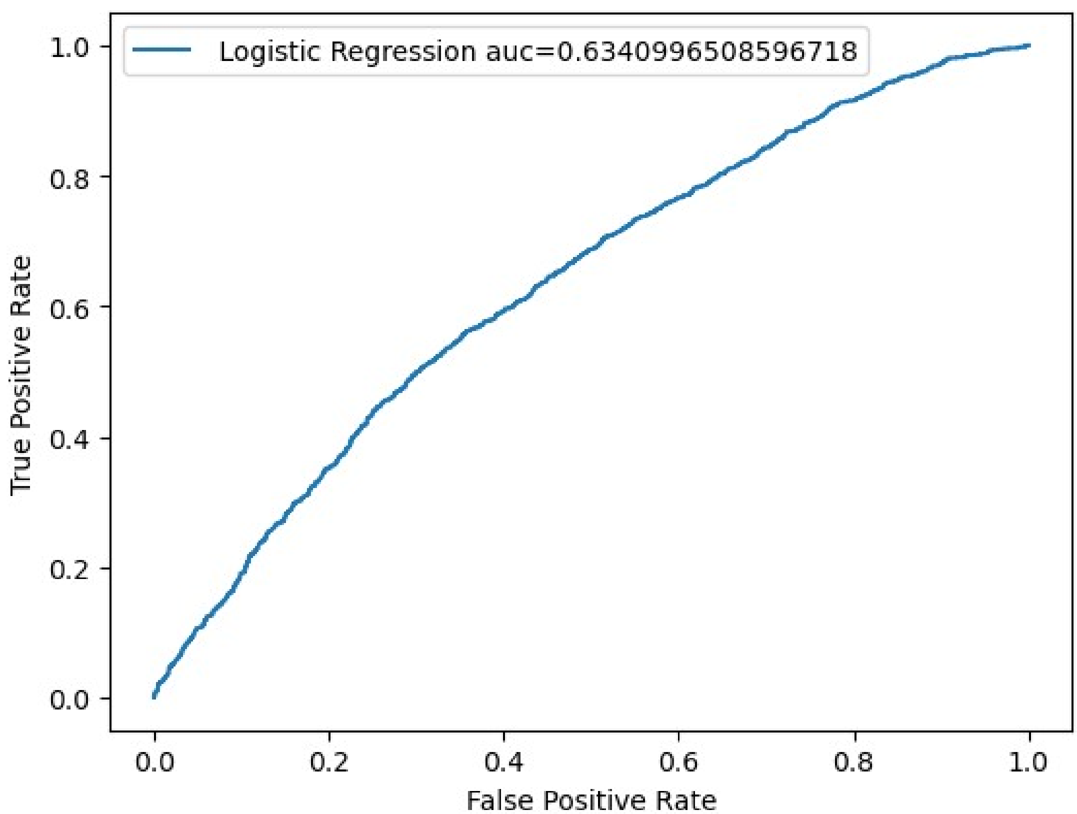



```
#Cross Validation
# Importing cross_val_score from sklearn.model_selection
from sklearn.model_selection import cross_val_score

# Computing cross-validation scores for the decision tree classifier
'tree' using 10-fold cross-validation
scores = cross_val_score(tree, x, y, cv= 10)

# Printing accuracy scores of each fold
print("Accuracy scores of each fold: {}".format(scores))

#A common way to summarize the cross-validation accuracy is to compute
the mean:
print("Average cross-validation score: {:.2f}".format(scores.mean()))
```

```
Accuracy scores of each fold: [0.7127859  0.72703412 0.70903637
0.7304087  0.72955739 0.71530383
 0.74943736 0.73705926 0.74381095 0.72955739]
Average cross-validation score: 0.73
```

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
# instantiate the model (using the default parameters)
logreg = LogisticRegression(max_iter = 200)
# fit the model with data
```

```
logreg.fit(X_train,y_train)
import sklearn.metrics as metrics
from matplotlib import pyplot
#keep probabilities for the positive outcome only
y_pred_proba = logreg.predict_proba(X_test)[:,1]
fpr, tpr, _= metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
pyplot.plot(fpr,tpr,label=" Logistic Regression auc="+str(auc))
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
#plt.legend(loc=4)
pyplot.legend()
pyplot.show()
y_pred=logreg.predict(X_test)
y_pred
print("Accuracy on training set: {:.3f}".format(logreg.score(X_train,
y_train)))
print("Accuracy on test set::
{:.3f}".format(metrics.accuracy_score(y_pred, y_test)))
training_accuracy_LR = logreg.score(X_train, y_train)
testing_accuracy_LR = metrics.accuracy_score(y_pred, y_test)
```



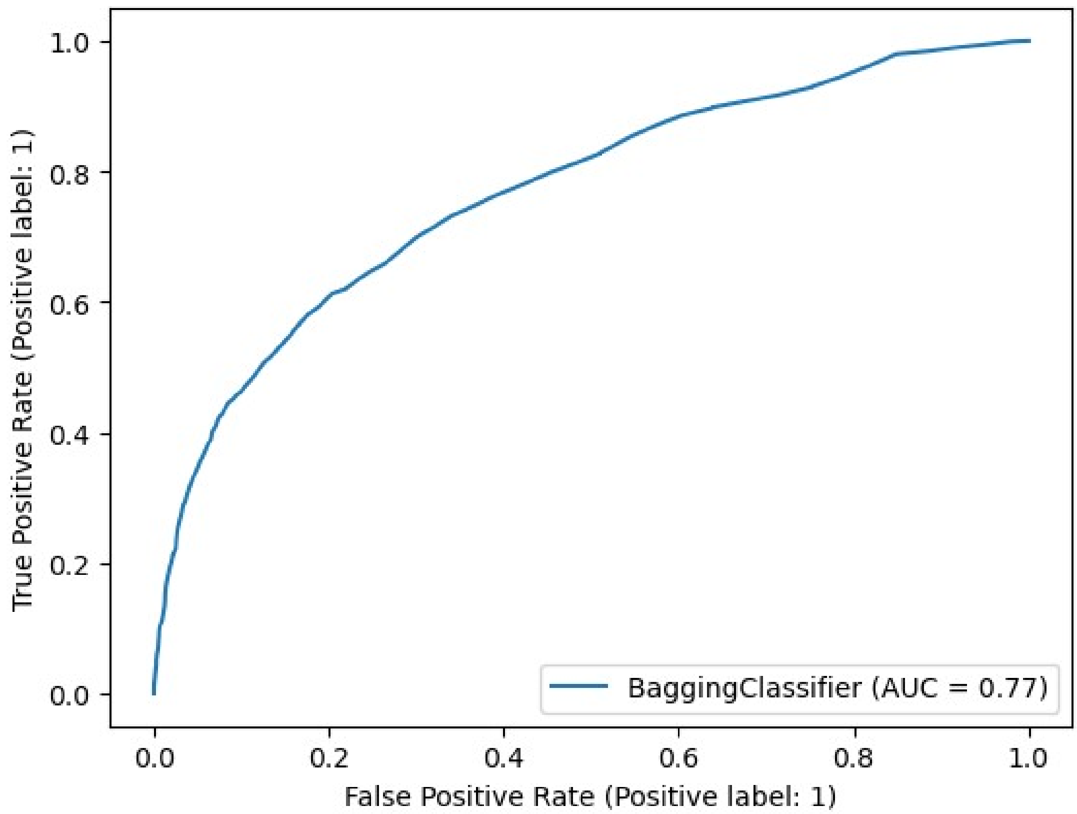
```
Accuracy on training set: 0.778
Accuracy on test set::0.778
```

```
#bagging model
# Importing BaggingClassifier from sklearn.ensemble
from sklearn.ensemble import BaggingClassifier

# Initializing BaggingClassifier with 100 base estimators and a random state
bagging = BaggingClassifier(n_estimators=100, random_state=0)

# Fitting the BaggingClassifier on the training data
bagging.fit(X_train, y_train)

# Making predictions on the test data
y_bagging_pred = bagging.predict(X_test)

# Printing the accuracy of the bagging model on the test set using accuracy_score from sklearn.metrics
print("Bagging Model Accuracy on test set: {:.3f}".format(accuracy_score(y_test,y_bagging_pred)))
```

```
Bagging Model Accuracy on test set: 0.819
```

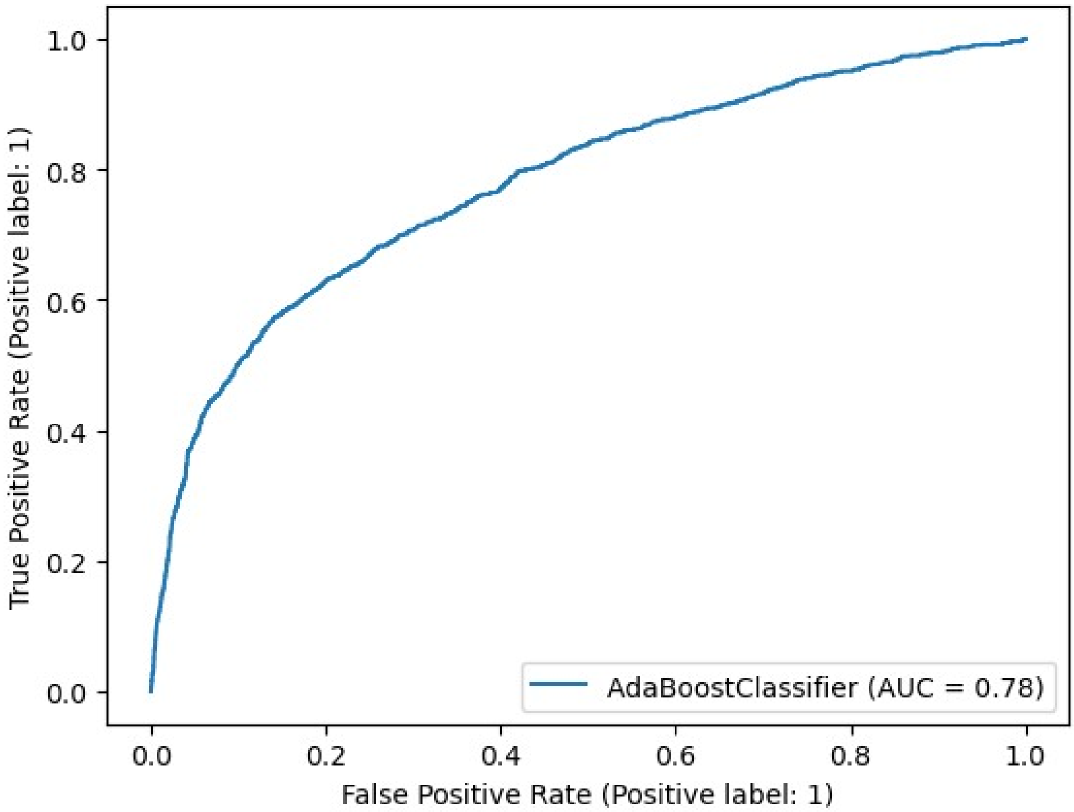
```
# Plotting the ROC curve for the BaggingClassifier 'bagging' using the test data
RocCurveDisplay.from_estimator(bagging, X_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fc52016190>
```




```
#AdaBoost classifier

# Importing AdaBoostClassifier from sklearn.ensemble
from sklearn.ensemble import AdaBoostClassifier

# Initializing AdaBoostClassifier with 100 base estimators and a
random state
boost = AdaBoostClassifier(n_estimators = 100, random_state=0)

# Fitting the AdaBoostClassifier on the training data
boost.fit(X_train, y_train)

# Making predictions on the test data
y_boost_pred = boost.predict(X_test)

# Printing the accuracy of the AdaBoost model on the test set using
accuracy_score from sklearn.metrics
print("Accuracy on test set:
{:.3f}".format(accuracy_score(y_boost_pred,y_test)))

Accuracy on test set: 0.833

# Plotting the ROC curve for the AdaBoostClassifier 'boost' using the
test data
RocCurveDisplay.from_estimator(boost, X_test, y_test)
```

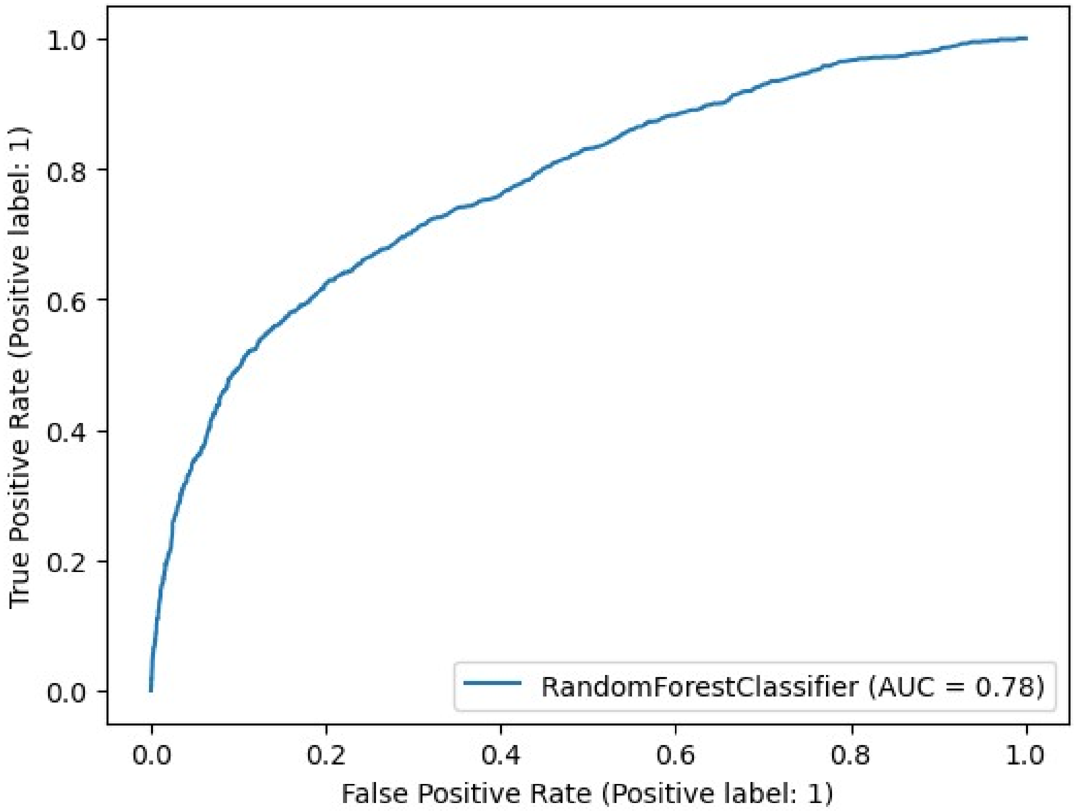
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fc4e3c4c90>
```




```
#random forest classifier

# Importing RandomForestClassifier from sklearn.ensemble
from sklearn.ensemble import RandomForestClassifier

# Initializing RandomForestClassifier with 1000 trees and a random
state
forest = RandomForestClassifier(n_estimators=1000, random_state=0)

# Fitting the RandomForestClassifier on the training data
forest.fit(X_train, y_train)

# Making predictions on the test data
y_rf_pred = forest.predict(X_test)

# Printing the accuracy of the random forest model on the test set
using accuracy_score from sklearn.metrics
print("Random Forest Accuracy on test set:
{:.3f}".format(accuracy_score(y_test, y_rf_pred)))

Random Forest Accuracy on test set: 0.821
```

```python
# Plotting the ROC curve for the RandomForestClassifier 'forest' using the test data
RocCurveDisplay.from_estimator(forest, X_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fc5565b390>
```




```python
# Calculating feature importances from the trained Random Forest Classifier model
importances = forest.feature_importances_

# Creating a DataFrame to store feature names and their corresponding importances
# X_train.columns: Names of the features used in training the model
# importances: Feature importances calculated by the Random Forest Classifier
df = pd.DataFrame({'feature': X_train.columns, 'importance': importances})

# Sorting the DataFrame by feature importance in descending order
df = df.sort_values('importance', ascending=False)

# Mapping feature names and importances into a new variable
Feature_importance = dict(zip(df['feature'], df['importance']))
```

```python
# Printing the DataFrame to display feature names and their
importances
print(df)
```

```
      feature    importance
1         AGE  6.412230e-02
0   LIMIT_BAL  6.029316e-02
2   BILL_AMT1  5.675321e-02
27    PAY_1_2  5.525731e-02
3   BILL_AMT2  5.258234e-02
..        ...           ...
59    PAY_4_1  8.939611e-06
55    PAY_3_8  7.530045e-06
66    PAY_4_8  5.569642e-06
44    PAY_2_8  4.180986e-06
76    PAY_5_8  3.040073e-07

[87 rows x 2 columns]
```

```python
# Initializing a DecisionTreeClassifier with a random state
tree = DecisionTreeClassifier(random_state=0)

# Getting the parameters of the DecisionTreeClassifier
tree.get_params()
```

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': 0,
 'splitter': 'best'}
```

```python
# Importing numpy for numerical operations and GridSearchCV from
sklearn.model_selection
import numpy as np
from sklearn.model_selection import GridSearchCV

# Defining a dictionary 'params' containing hyperparameters to be
tuned
params = {'criterion':['gini','entropy'],'max_leaf_nodes':
list(range(2, 50)),'max_depth': np.arange(3, 15)}

# Initializing GridSearchCV with the DecisionTreeClassifier 'tree',
parameter grid 'params', and 5-fold cross-validation
tree_grid = GridSearchCV(tree, params, cv=5)
```

```
# Performing grid search to find the best combination of
hyperparameters
tree_grid.fit(X_train, y_train)

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': array([ 3,  4,  5,  6,  7,  8,
9, 10, 11, 12, 13, 14]),
                         'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12,
                                            13, 14, 15, 16, 17, 18,
19, 20, 21,
                                            22, 23, 24, 25, 26, 27,
28, 29, 30,
                                            31, ...]})

# Retrieving the best estimator found during grid search
tree_grid.best_estimator_

----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[18], line 2
      1 # Retrieving the best estimator found during grid search
----> 2 tree_grid.best_estimator_

NameError: name 'tree_grid' is not defined

# Making predictions on the test data using the best estimator found
during grid search
y_pred_grid = tree_grid.predict(X_test)

# Printing the accuracy of the grid-search model on the test set using
accuracy_score from sklearn.metrics
print("Grid-search Model Accuracy on test set:
{:.3f}".format(accuracy_score(y_test, y_pred_grid)))

Grid-search Model Accuracy on test set: 0.828

# Plotting the ROC curve for the best estimator found during grid
search using the test data
RocCurveDisplay.from_estimator(tree_grid, X_test, y_test)

----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[17], line 2
      1 # Plotting the ROC curve for the best estimator found during
```

```
grid search using the test data
----> 2 RocCurveDisplay.from_estimator(tree_grid, X_test, y_test)

NameError: name 'tree_grid' is not defined
```

Model Comparison:

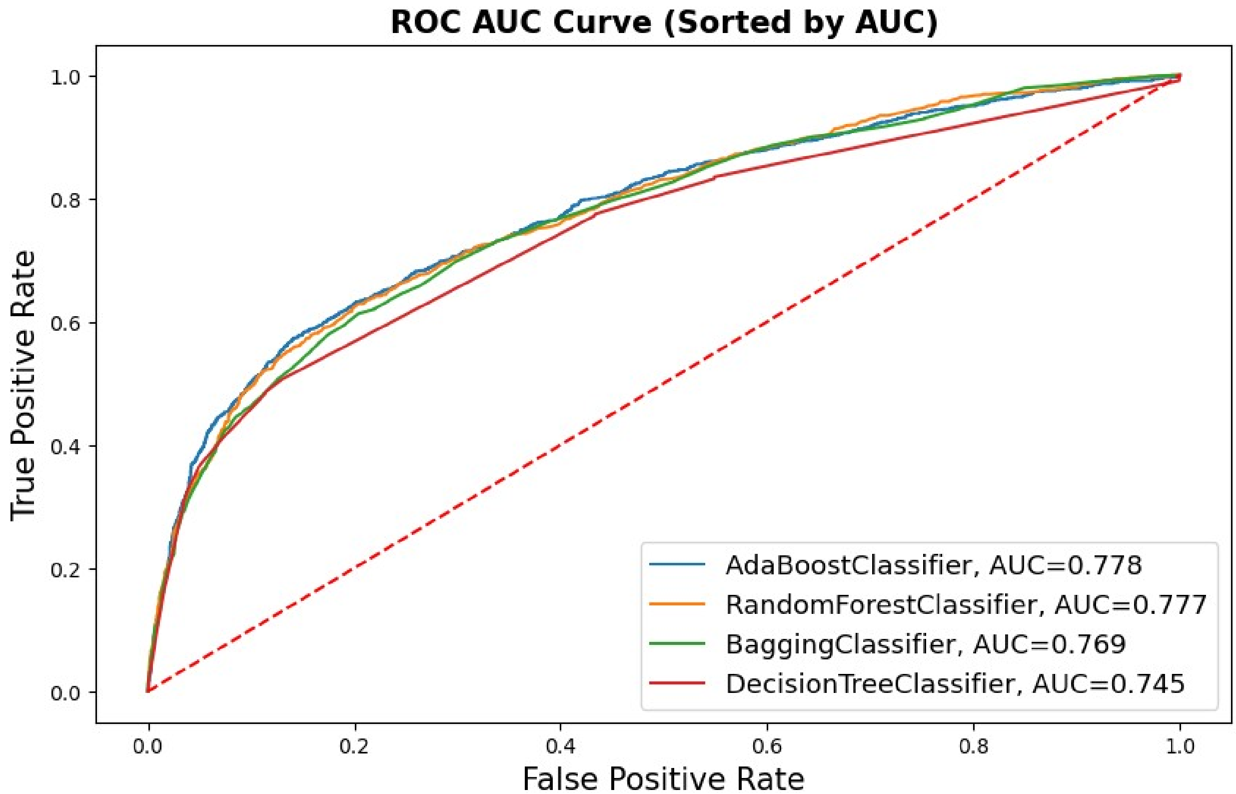
```
tree
tree_pruned
bagging
boost
forest

RandomForestClassifier(n_estimators=1000, random_state=0)

# y_pred_proba = tree.predict_proba(X_test)[:,1]
y__pruned_pred_proba = tree_pruned.predict_proba(X_test)[:,1]
y_bagging_pred = bagging.predict_proba(X_test)[:,1]
y_boost_pred = boost.predict_proba(X_test)[:,1]
y_rf_pred = forest.predict_proba(X_test)[:,1]

# Importing necessary libraries
from sklearn.metrics import accuracy_score, precision_score,
recall_score

# Initialize lists to store metrics
models = ['Decision Tree', 'Pruned Decision Tree', 'Bagging',
'AdaBoost', 'Random Forest']
accuracies = []
precisions = []
recalls = []

# Define a threshold for converting probabilities to binary
predictions
threshold = 0.5

# Calculate metrics for each model
for model, y_pred_proba in zip(models, [y_pred, y__pruned_pred_proba,
y_bagging_pred, y_boost_pred, y_rf_pred]):
    # Convert probabilities to binary predictions based on the
threshold
    y_pred = (y_pred_proba >= threshold).astype(int)

    # Calculate accuracy, precision, and recall
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)

    # Append metrics to respective lists
    accuracies.append(accuracy)
```

```python
    precisions.append(precision)
    recalls.append(recall)

# Create a DataFrame to display metrics
metrics_df = pd.DataFrame({
    'Model': models,
    'Accuracy': accuracies,
    'Precision': precisions,
    'Recall': recalls
})

# Sort the classifiers by accuracy in descending order
metrics_df_sorted_accuracy = metrics_df.sort_values(by='Accuracy',
ascending=False)

# Display the sorted DataFrame
print("Classifiers sorted by accuracy:")
print(metrics_df_sorted_accuracy)
```

```
Classifiers sorted by accuracy:
                  Model  Accuracy  Precision    Recall
3              AdaBoost  0.832927   0.700337  0.368468
1  Pruned Decision Tree  0.826552   0.665049  0.364039
0         Decision Tree  0.820364   0.627803  0.372011
2               Bagging  0.820364   0.616644  0.400354
4         Random Forest  0.820364   0.627803  0.372011
```

```python
from sklearn import metrics

#creating a list of tuples of all models and its probability
classifiers_proba = [(tree_pruned, y__pruned_pred_proba),
                    (bagging, y_bagging_pred),
                    (boost, y_boost_pred),
                    (forest, y_rf_pred),

                    ]

# Define a result table as a DataFrame
result_table = pd.DataFrame(columns=['classifiers',
'fpr','tpr','auc'])

# Train the models and record the results
for pair in classifiers_proba:

    fpr, tpr, _ = metrics.roc_curve(y_test,  pair[1])
    auc = metrics.roc_auc_score(y_test, pair[1])

    result_table =
result_table._append({'classifiers':pair[0].__class__.__name__,
                                        'fpr':fpr,
```

```
                                                'tpr':tpr,
                                                'auc':auc}, ignore_index=True)

# Set name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)

import matplotlib.pyplot as plt

# Sorting the result_table DataFrame by AUC in descending order
result_table_sorted = result_table.sort_values(by='auc',
ascending=False)

# Plotting the ROC AUC curve
fig = plt.figure(figsize=(10, 6))

for i in result_table_sorted.index:
    plt.plot(result_table_sorted.loc[i]['fpr'],
             result_table_sorted.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(i,
result_table_sorted.loc[i]['auc']))

plt.plot([0, 1], [0, 1], 'r--')

plt.xlabel("False Positive Rate", fontsize=15)
plt.ylabel("True Positive Rate", fontsize=15)
plt.title('ROC AUC Curve (Sorted by AUC)', fontweight='bold',
fontsize=15)
plt.legend(prop={'size': 13}, loc='lower right')

plt.show()
```


ROC AUC Curve (Sorted by AUC)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AdaBoostClassifier, AUC=0.778
RandomForestClassifier, AUC=0.777
BaggingClassifier, AUC=0.769
DecisionTreeClassifier, AUC=0.745