```
cd C:\Users\dhant\OneDrive\Desktop\simplilearn\Capstone\Project 2\
Healthcare - Diabetes
```

```
C:\Users\dhant\OneDrive\Desktop\simplilearn\Capstone\Project 2\
Healthcare - Diabetes
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import pylab as p
import missingno as msno
import warnings
warnings.filterwarnings('ignore')
```

```
data=pd.read_csv('health care diabetes.csv')
data.head(3)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
```

1. Understand the dataset:

a. Identify the shape of the dataset

```
data.shape
```

```
(768, 9)
```

b. Identify the size of the dataset

```
data.size
```

```
6912
```

c. Identify the columns of the dataset

```
data.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
```

```
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

d. Identify the data types of the dataset

```
data.dtypes
```

```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

e. Identify the information of the dataset

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

f. identifying the number of unique values of dataset

```
data.nunique()
```

```
Pregnancies                  17
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
BMI                         248
DiabetesPedigreeFunction    517
Age                          52
Outcome                       2
dtype: int64
```

Information of data:

a. Identifying the total profile report of dataset

```
import pandas_profiling as pp
from pandas_profiling import ProfileReport
pp.ProfileReport(data)
```

{"version_major":2,"version_minor":0,"model_id":"dda5798e932c463eb270fef44af83e9c"}

{"version_major":2,"version_minor":0,"model_id":"b261d0051a3646eb9130cfe4107addb9"}

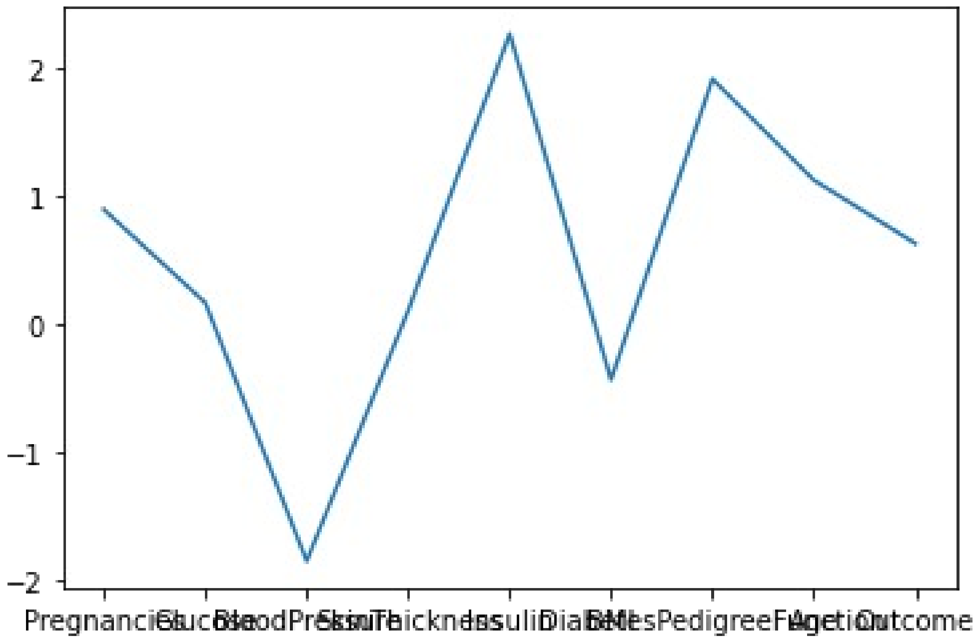
{"version_major":2,"version_minor":0,"model_id":"5b53fa6f70ba47e7bf970c7ee412ce1b"}

```
<IPython.core.display.HTML object>
```

skewness of data and its visualization

```
print(data.skew() )
p.plot(data.skew())
print( '\nSkewness for data : ')
```

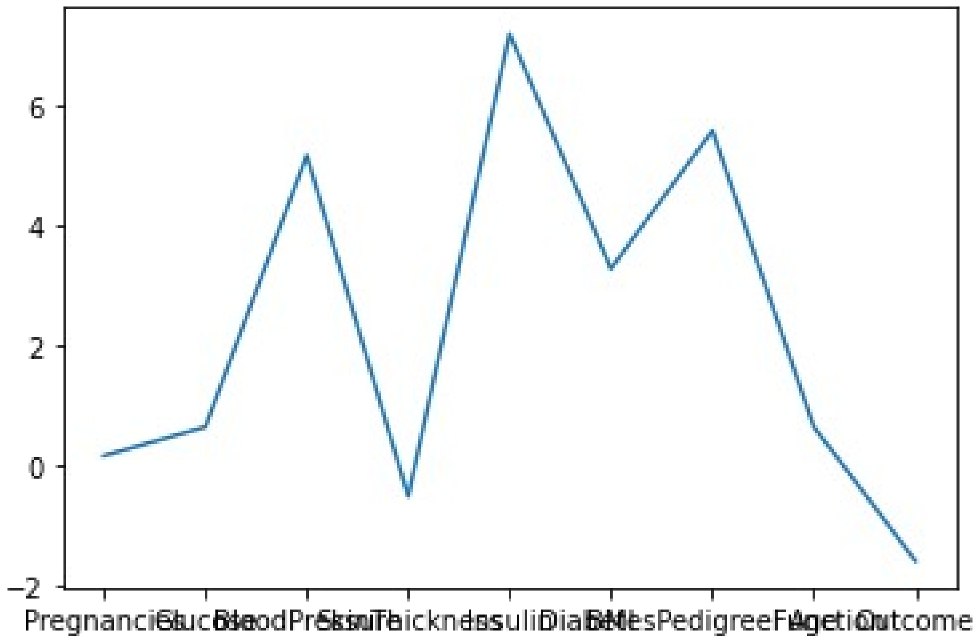
```
Pregnancies                  0.901674
Glucose                      0.173754
BloodPressure               -1.843608
SkinThickness                0.109372
Insulin                      2.272251
BMI                         -0.428982
DiabetesPedigreeFunction     1.919911
Age                          1.129597
Outcome                      0.635017
dtype: float64

Skewness for data :
```



kurtosis and its visualization

```
print(data.kurtosis())
p.plot(data.kurtosis())
print( '\nkurtosis of data : ')
```

```
Pregnancies                 0.159220
Glucose                     0.640780
BloodPressure               5.180157
SkinThickness              -0.520072
Insulin                     7.214260
BMI                         3.290443
DiabetesPedigreeFunction    5.594954
Age                         0.643159
Outcome                    -1.600930
dtype: float64

kurtosis of data :
```



Statistical summary

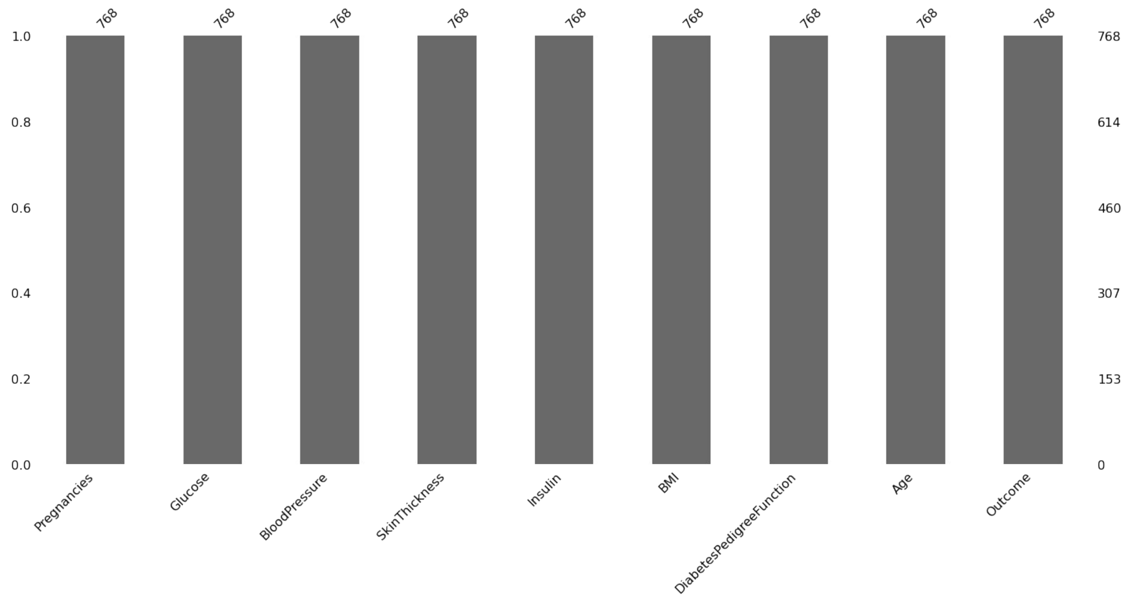
```
data.describe().style.background_gradient(axis=1,cmap=sns.light_palett
e('green', as_cmap=True))
```

```
<pandas.io.formats.style.Styler at 0x1e045ff5940>
```

checking missing values

```
print(data.isnull().sum())
print("Display the missing values : there is no missing values","\n")
msno.bar(data)
plt.show()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
Display the missing values : there is no missing values
```




```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Data Preprocessing

```
data.info()
```

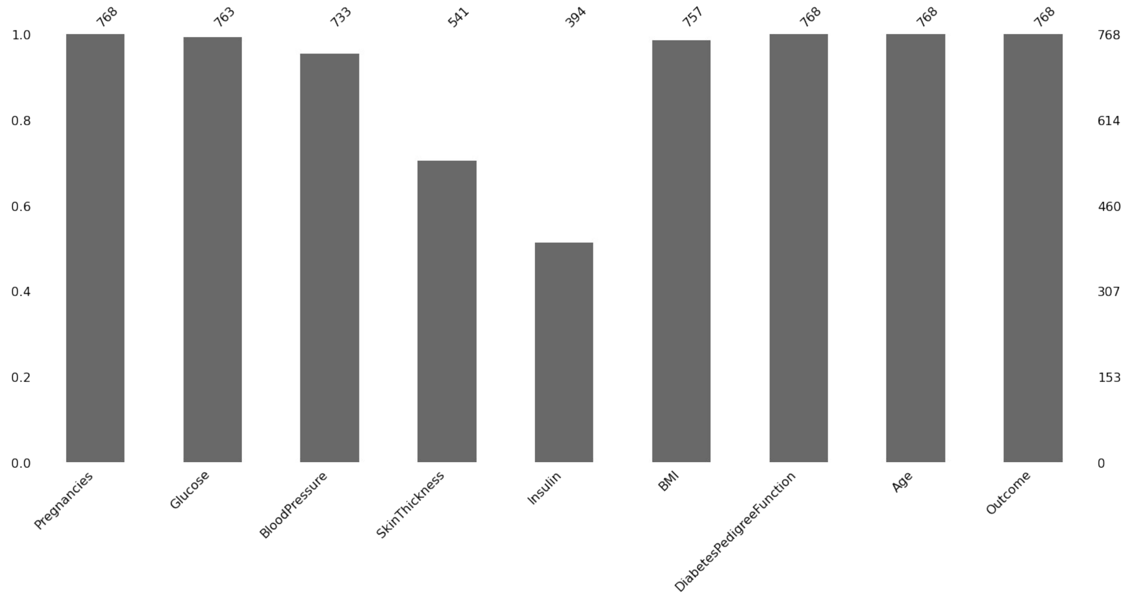
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
```

```
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

By observing the data file there are some varaible with zero(0) that means the values is missing values…so the values replace with nan and treat by techinque

```
data[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]]
= data[["Glucose", "BloodPressure", "SkinThickness", "Insulin",
"BMI"]].replace(0, np.NaN)

data.head()

   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6    148.0           72.0           35.0      NaN  33.6

1            1     85.0           66.0           29.0      NaN  26.6

2            8    183.0           64.0            NaN      NaN  23.3

3            1     89.0           66.0           23.0     94.0  28.1

4            0    137.0           40.0           35.0    168.0  43.1


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1

print(data.isnull().sum())
print("Display the missing values : there is some missing values","\
n")
msno.bar(data)
plt.show()

Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
```

```
Outcome                       0
dtype: int64
Display the missing values : there is some missing values
```



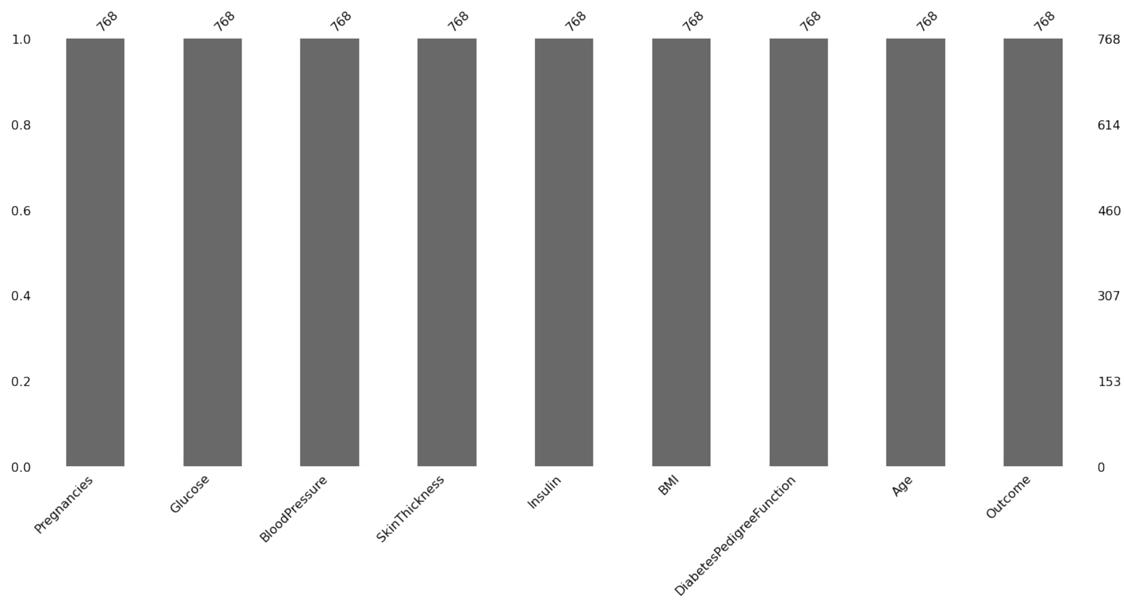
Observations

treating missing values with mean

```
# Replacing NaN with mean values
data["Glucose"].fillna(data["Glucose"].mean(), inplace = True)
data["BloodPressure"].fillna(data["BloodPressure"].mean(), inplace =
True)
data["SkinThickness"].fillna(data["SkinThickness"].mean(), inplace =
True)
data["Insulin"].fillna(data["Insulin"].mean(), inplace = True)
data["BMI"].fillna(data["BMI"].mean(), inplace = True)

print(data.isnull().sum())
print("After treating the  missing values : there is no missing
values","\n")
msno.bar(data)
plt.show()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
```

```
dtype: int64
After treating the  missing values : there is no missing values
```

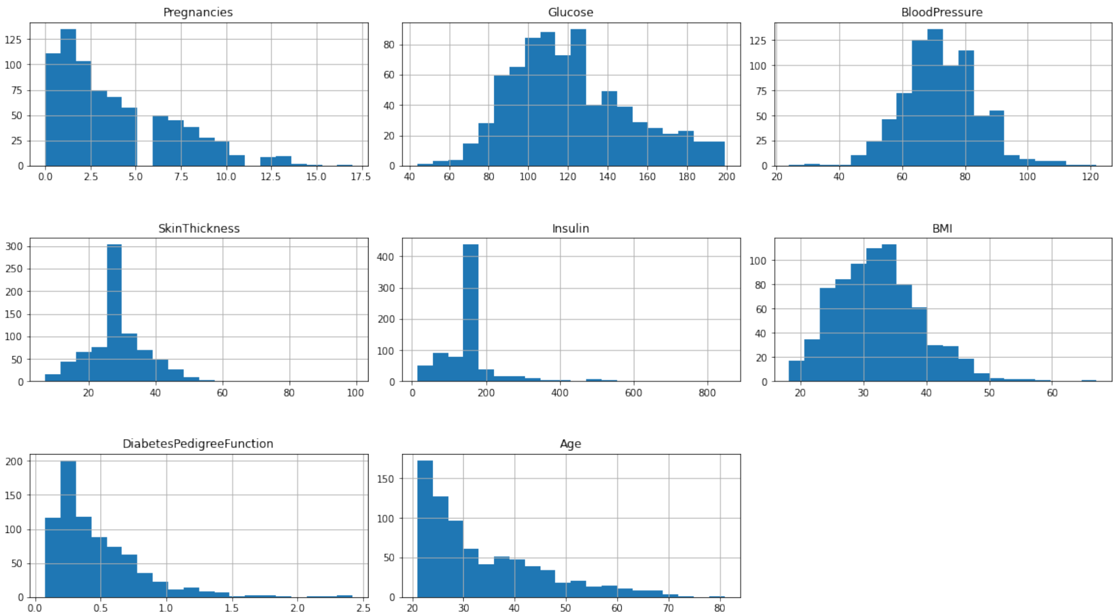



```
data.describe().style.background_gradient(axis=1,cmap=sns.light_palett
e('green', as_cmap=True))
```

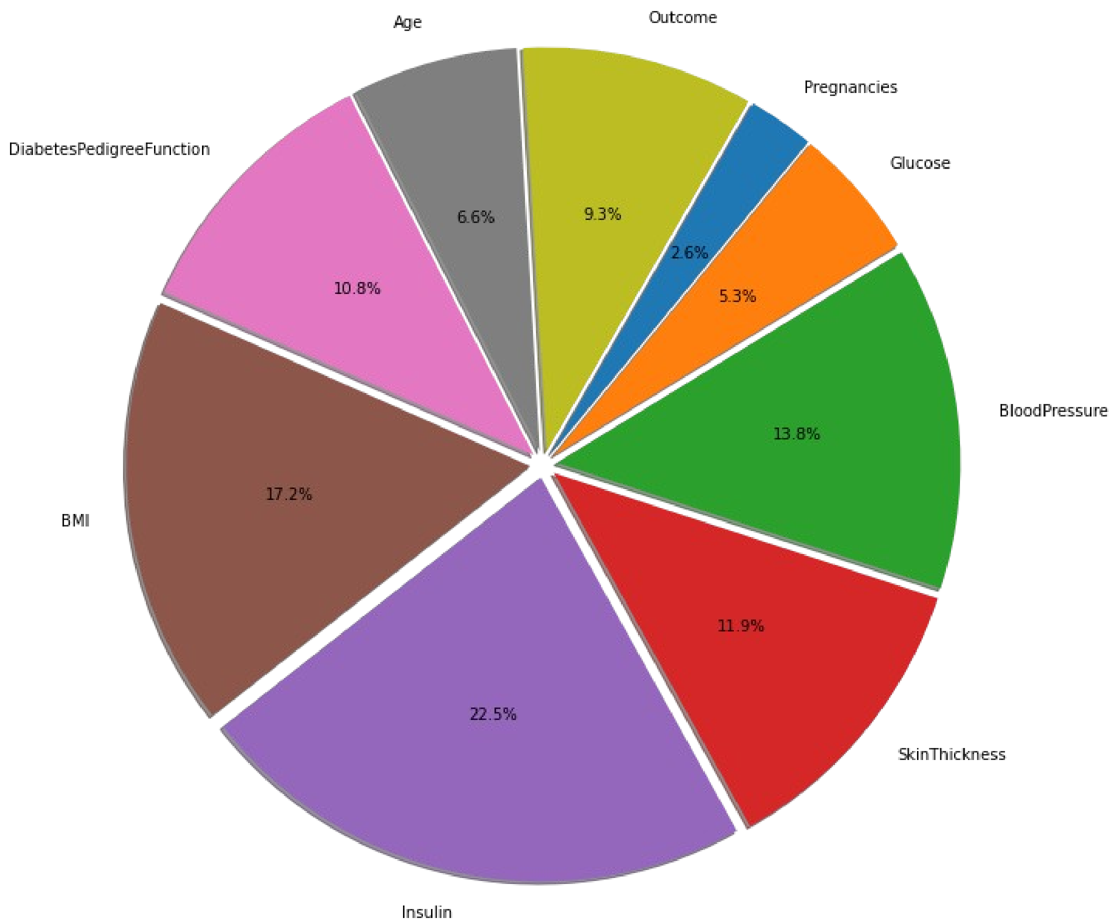
```
<pandas.io.formats.style.Styler at 0x2e0e5ccd340>
```

There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

```
import itertools

col = data.columns[:8]
plt.subplots(figsize = (20, 15))
length = len(col)

for i, j in itertools.zip_longest(col, range(length)):
    plt.subplot((length/2), 3, j + 1)
    plt.subplots_adjust(wspace = 0.1,hspace = 0.5)
    data[i].hist(bins = 20)
    plt.title(i)
plt.show()
```




```
size=[10,20,52,45,85,65,41,25,35]
plt.axis("equal")
plt.pie(size,labels=data.columns,autopct="%1.1f%
%",radius=3,shadow=True,explode=[0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1],
startangle=60,counterclock=False)
plt.show()
```



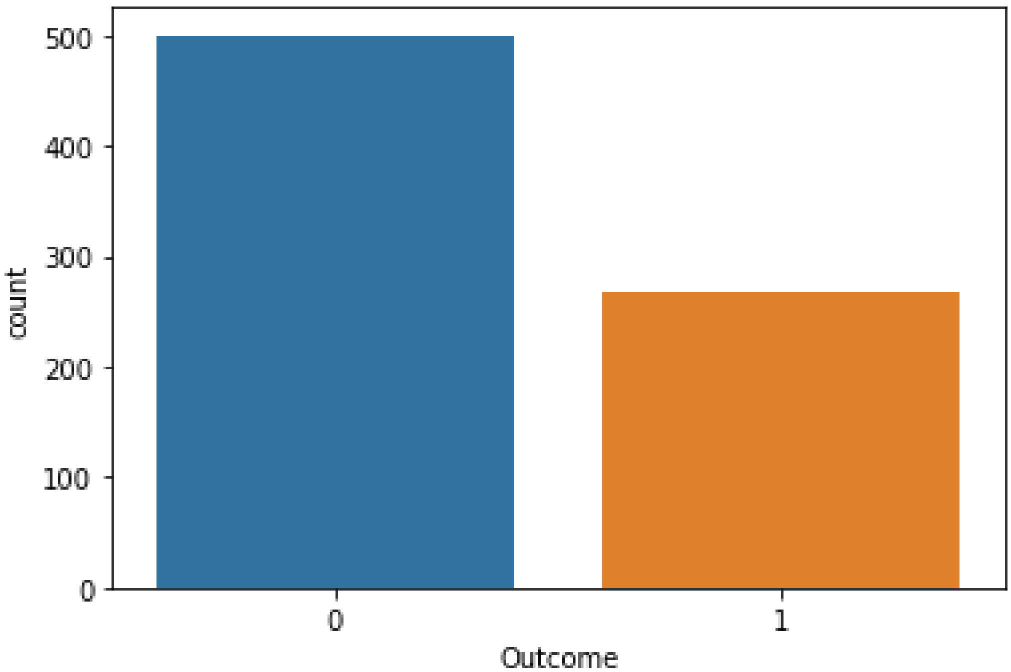
Project Task: Week 2

1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

```
print(data['Outcome'].value_counts())
print('The total number of outcomes of 0  are 500 and the number of outcomes of 1 are 268')
sns.countplot('Outcome',data=data)
```

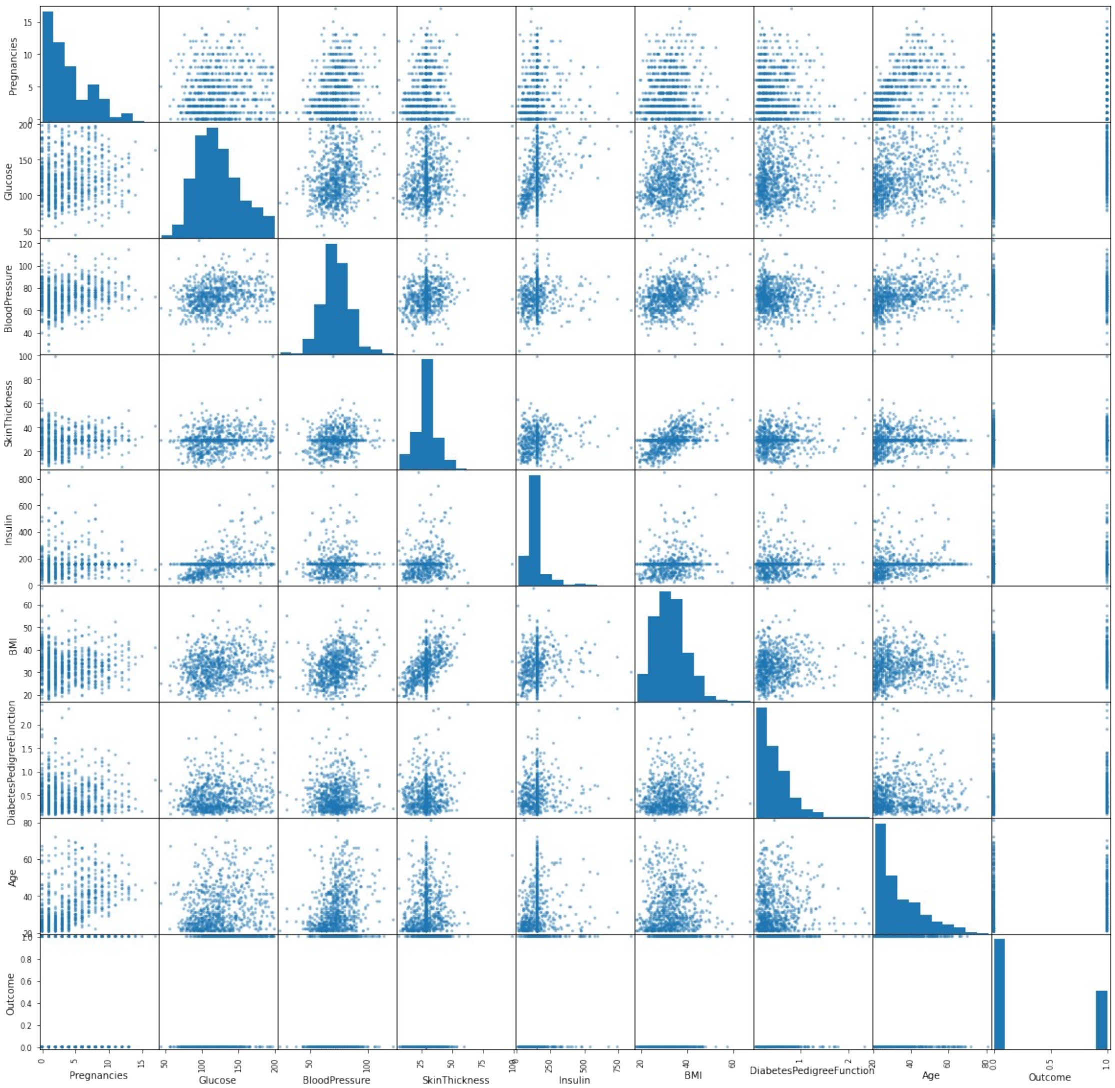
```
0    500
1    268
Name: Outcome, dtype: int64
The total number of outcomes of 0  are 500 and the number of outcomes of 1 are 268
```

```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```



2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

```
# Scatter plot matrix
from pandas.plotting import scatter_matrix
scatter_matrix(data, figsize = (20, 20));
```



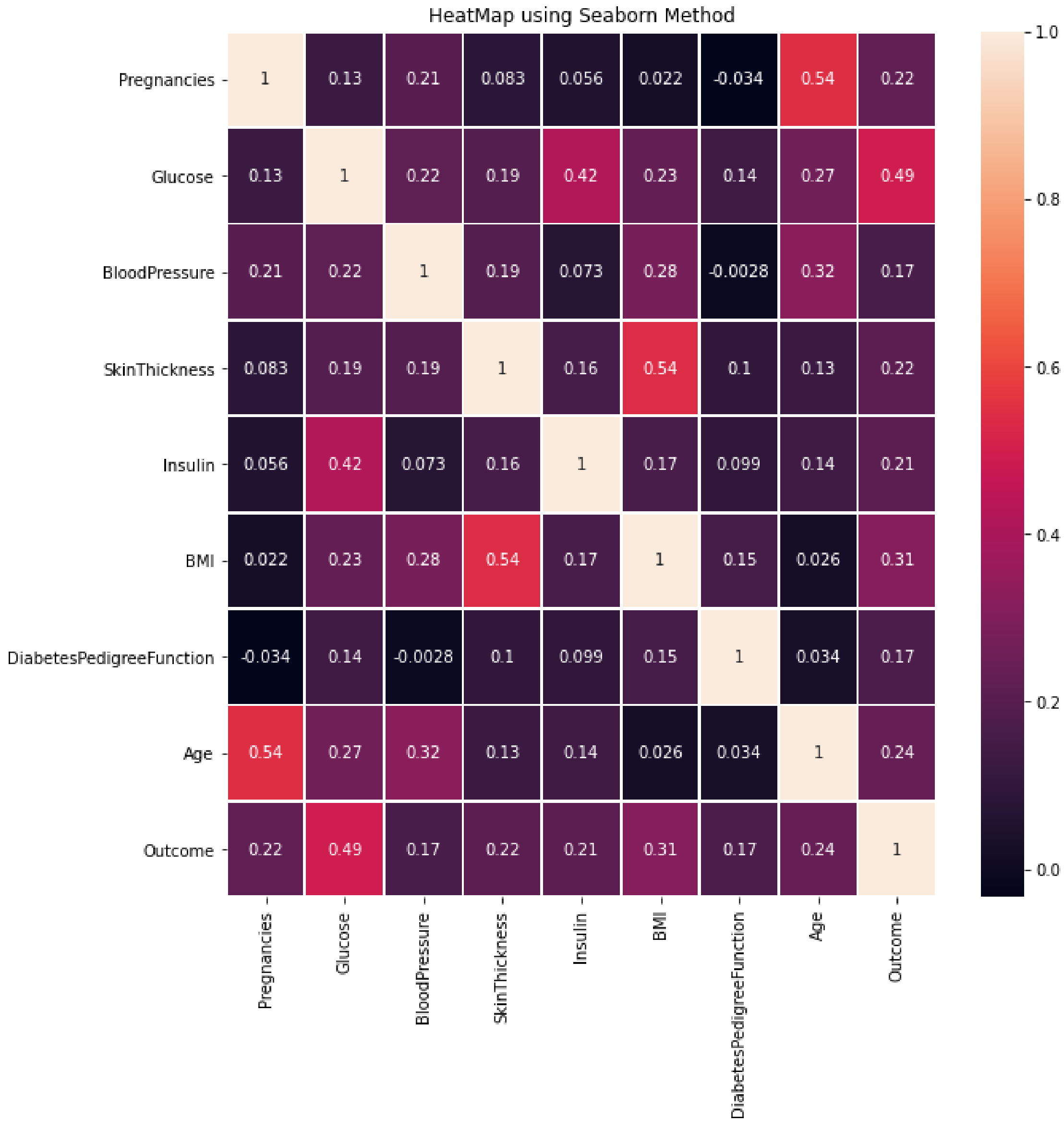
3. Perform correlation analysis. Visually explore it using a heat map.

```
print('correlation of each coulmns along with that values')
data.corr()
```

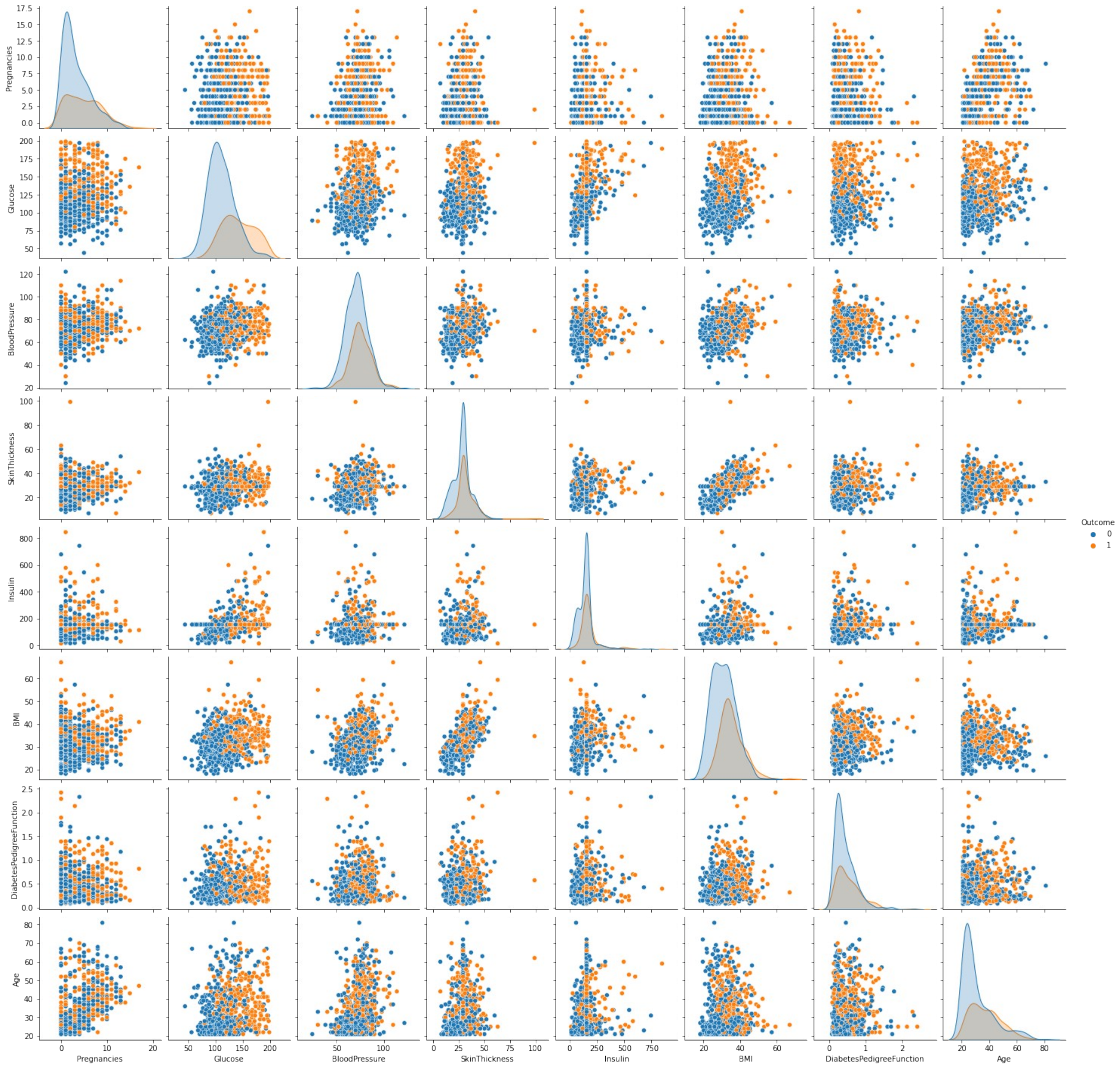
```
correlation of each coulmns along with that values

                          Pregnancies   Glucose  BloodPressure
SkinThickness  \
Pregnancies                  1.000000  0.127911       0.208522
0.082989
Glucose                      0.127911  1.000000       0.218367
0.192991
BloodPressure                0.208522  0.218367       1.000000
0.192816
SkinThickness                0.082989  0.192991       0.192816
1.000000
Insulin                      0.056027  0.420157       0.072517
0.158139
```

```
BMI                             0.021565  0.230941       0.281268
0.542398
DiabetesPedigreeFunction      -0.033523  0.137060      -0.002763
0.100966
Age                             0.544341  0.266534       0.324595
0.127872
Outcome                         0.221898  0.492928       0.166074
0.215299

                           Insulin       BMI  DiabetesPedigreeFunction
\
Pregnancies               0.056027  0.021565                 -0.033523

Glucose                   0.420157  0.230941                  0.137060

BloodPressure             0.072517  0.281268                 -0.002763

SkinThickness             0.158139  0.542398                  0.100966

Insulin                   1.000000  0.166586                  0.098634

BMI                       0.166586  1.000000                  0.153400

DiabetesPedigreeFunction  0.098634  0.153400                  1.000000

Age                       0.136734  0.025519                  0.033561

Outcome                   0.214411  0.311924                  0.173844


                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.266534  0.492928
BloodPressure             0.324595  0.166074
SkinThickness             0.127872  0.215299
Insulin                   0.136734  0.214411
BMI                       0.025519  0.311924
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

```python
# 3. Plot the heatmap
plt.figure(figsize=(10,10))
heat_map = sns.heatmap( data.corr(), linewidth = 1 , annot = True)
plt.title( "HeatMap using Seaborn Method" )
plt.show()
```




```
# Pairplot
sns.pairplot(data = data, hue = 'Outcome')
plt.show()
```



Observations:

Project Task: Week 3

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

```
data.head(3)

   Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin
BMI  \
0            6    148.0           72.0       35.00000  155.548223
33.6
1            1     85.0           66.0       29.00000  155.548223
26.6
2            8    183.0           64.0       29.15342  155.548223
23.3
```

```
   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
```

```
#Feature scaling using MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
dataset_scaled = sc.fit_transform(data)

dataset_scaled = pd.DataFrame(dataset_scaled)
dataset_scaled
```

```
            0         1         2         3         4         5
6  \
0    0.352941  0.670968  0.489796  0.304348  0.170130  0.314928
0.234415
1    0.058824  0.264516  0.428571  0.239130  0.170130  0.171779
0.116567
2    0.470588  0.896774  0.408163  0.240798  0.170130  0.104294
0.253629
3    0.058824  0.290323  0.428571  0.173913  0.096154  0.202454
0.038002
4    0.000000  0.600000  0.163265  0.304348  0.185096  0.509202
0.943638
..        ...       ...       ...       ...       ...       ...
...
763  0.588235  0.367742  0.530612  0.445652  0.199519  0.300613
0.039710
764  0.117647  0.503226  0.469388  0.217391  0.170130  0.380368
0.111870
765  0.294118  0.496774  0.489796  0.173913  0.117788  0.163599
0.071307
766  0.058824  0.529032  0.367347  0.240798  0.170130  0.243354
0.115713
767  0.058824  0.316129  0.469388  0.260870  0.170130  0.249489
0.101196


            7    8
0    0.483333  1.0
1    0.166667  0.0
2    0.183333  1.0
3    0.000000  0.0
4    0.200000  1.0
..        ...  ...
763  0.700000  0.0
764  0.100000  0.0
765  0.150000  0.0
766  0.433333  1.0
767  0.033333  0.0
```

```
[768 rows x 9 columns]
```

```
x=dataset_scaled.drop([8],axis=1)
y=dataset_scaled[[8]]

# Splitting X and Y
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
0.20, random_state = 42, stratify = data['Outcome'] )

print('The size of our train of "x" is ',x_train.shape)
print('The size of our test of "x" is ',x_test.shape)
print('The size of our train of "y" is ',y_train.shape)
print('The size of our test of "y" is ',y_test.shape)
```

```
The size of our train of "x" is  (614, 8)
The size of our test of "x" is  (154, 8)
The size of our train of "y" is  (614, 1)
The size of our test of "y" is  (154, 1)
```

```
x_train.head(3)
```

```
            0         1         2         3         4         5
6  \
353  0.058824  0.296774  0.387755  0.054348  0.034856  0.184049
0.214347
711  0.294118  0.529032  0.551020  0.217391  0.009615  0.233129
0.154142
373  0.117647  0.393548  0.346939  0.358696  0.096154  0.341513
0.062767

            7
353  0.050000
711  0.316667
373  0.066667
```

```
y_train.head(3)
```

```
       8
353  0.0
711  0.0
373  0.0
```

```
x_test.head(3)
```

```
            0         1         2         3         4         5
6  \
44   0.411765  0.741935  0.408163  0.240798  0.170130  0.188139
0.092229
672  0.588235  0.154839  0.836735  0.173913  0.042067  0.353783
0.088386
```

```
700  0.117647  0.503226  0.530612  0.217391  0.223558  0.361963
0.172929

            7
44   0.316667
672  0.433333
700  0.083333
```

```
y_test.head(3)
```

```
       8
44   0.0
672  0.0
700  0.0
```

Data Modelling

1. Logistic Regression Algorithm

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state = 42)
lr.fit(x_train, y_train)
```

```
LogisticRegression(random_state=42)
```

```
print(lr.coef_)
print(lr.intercept_)
y_train_pred=pd.DataFrame(lr.predict(x_train))
y_train_pred
```

```
[[1.33192165 4.39310948 0.29586671 0.78907784 0.50191049 2.86163994
  1.15755148 0.78847976]]
[-4.90126158]

       0
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
609  0.0
610  0.0
611  0.0
612  0.0
613  0.0

[614 rows x 1 columns]
```

```
y_test_pred=pd.DataFrame(lr.predict(x_test))
y_test_pred
```

```
       0
0    1.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
149  0.0
150  0.0
151  0.0
152  1.0
153  0.0

[154 rows x 1 columns]
```

Accuracy score

```
from sklearn.metrics import accuracy_score,classification_report,
confusion_matrix,plot_confusion_matrix
print('train accuracy:',accuracy_score(y_train_pred,y_train))
print('test accuracy:',accuracy_score(y_test_pred,y_test))
```

```
train accuracy: 0.7866449511400652
test accuracy: 0.7077922077922078
```

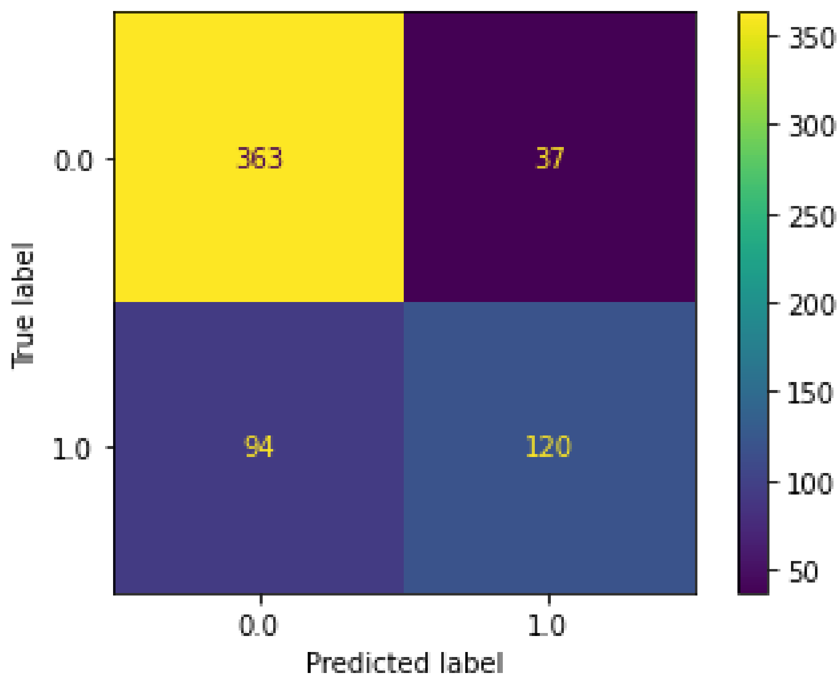
Confusion matrix

```
print("Train confusion_matrix \n \n ",confusion_matrix(y_train,
y_train_pred))
print("Test confusion_matrix \n \n ",confusion_matrix(y_test,
y_test_pred))
```

```
Train confusion_matrix

  [[363  37]
 [ 94 120]]
Test confusion_matrix

  [[83 17]
 [28 26]]
```

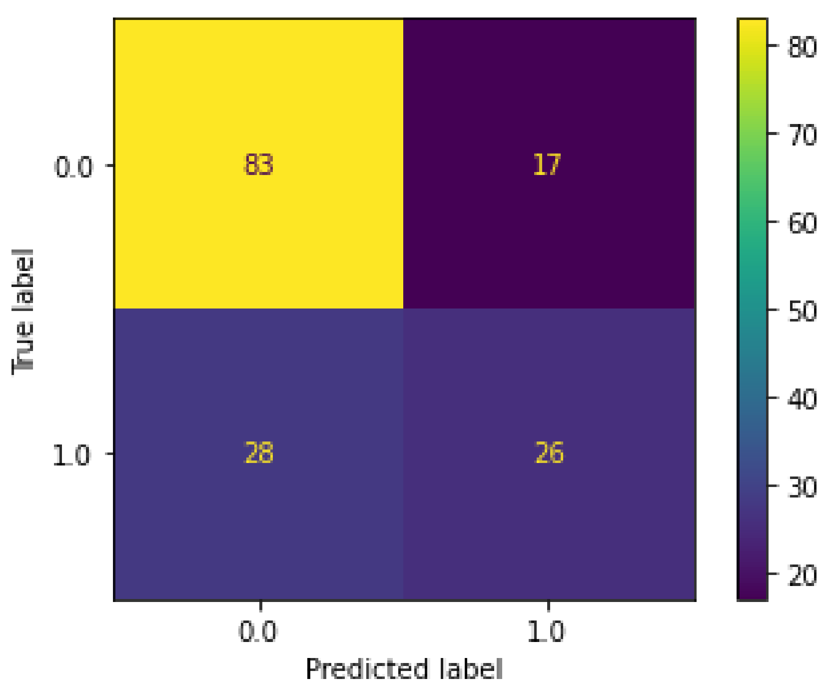
```
plot_confusion_matrix(lr,x_train,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0e15fb520>
```




```
plot_confusion_matrix(lr,x_test,y_test)
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2e0e4653fa0>



Classification report

```
print("Train classification_report \n \n
",classification_report(y_train, y_train_pred))
```

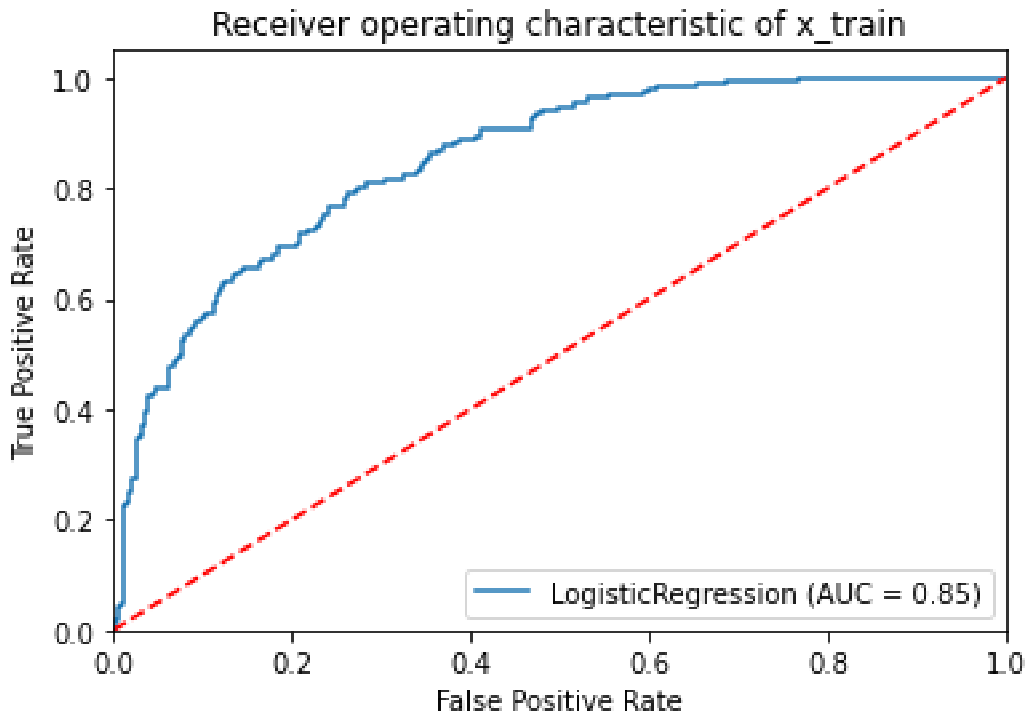
```
print("Test classification_report \n \n
",classification_report(y_test, y_test_pred))
```

```
Train classification_report

               precision    recall  f1-score   support

         0.0       0.79      0.91      0.85       400
         1.0       0.76      0.56      0.65       214

    accuracy                           0.79       614
   macro avg       0.78      0.73      0.75       614
weighted avg       0.78      0.79      0.78       614

Test classification_report

               precision    recall  f1-score   support

         0.0       0.75      0.83      0.79       100
         1.0       0.60      0.48      0.54        54

    accuracy                           0.71       154
   macro avg       0.68      0.66      0.66       154
weighted avg       0.70      0.71      0.70       154
```

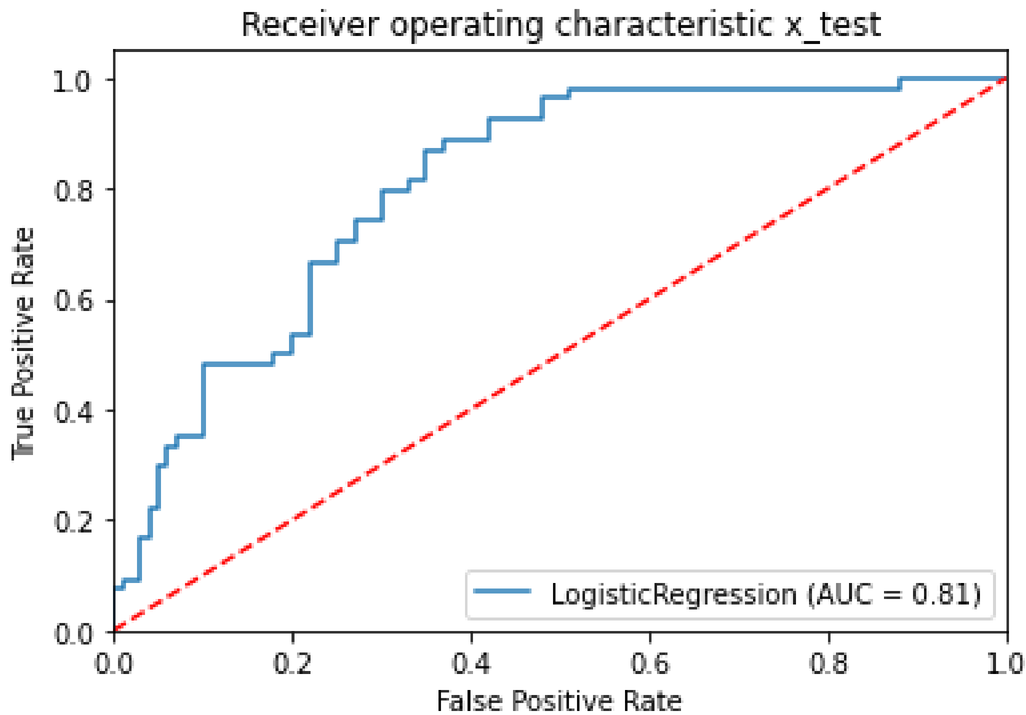
ROC

Roc curve is another common tool used with binary classifiers. The dotted line represents the ROC curve of a purely random classifier; a good classifier stays as far away from that line as possible

```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(lr,x_train,y_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_train')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```

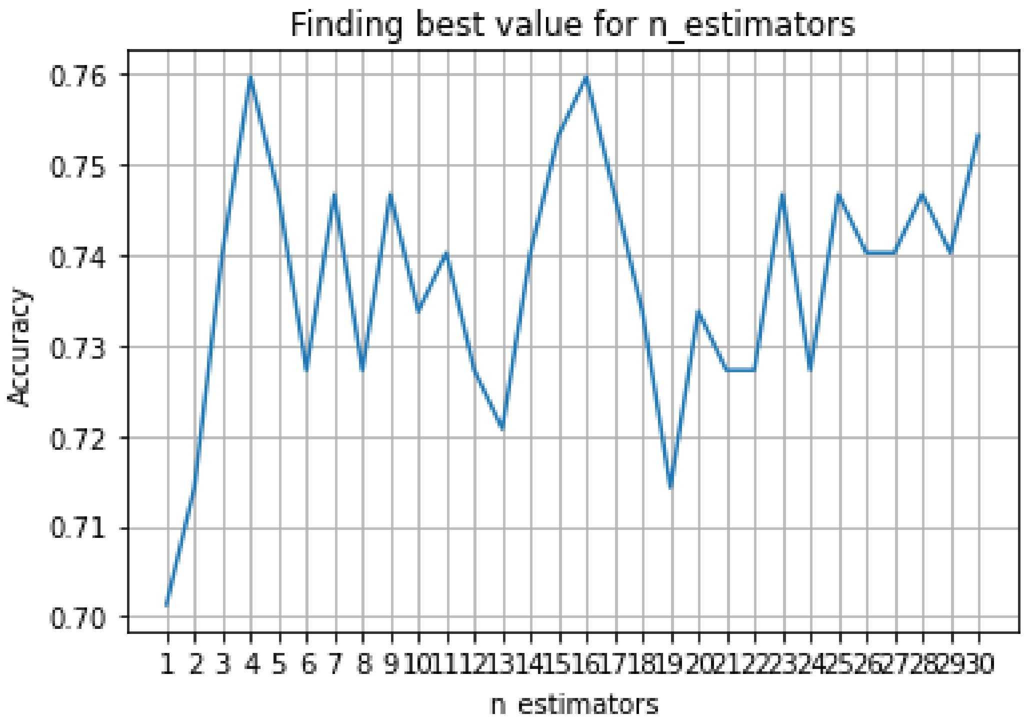



```
plot_roc_curve(lr,x_test,y_test)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



KNN

```
# Plotting a graph for n_neighbors
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier

x_axis = list(range(1, 31))
acc = pd.Series()
x = range(1,31)

for i in list(range(1, 31)):
    knn_model = KNeighborsClassifier(n_neighbors = i)
    knn_model.fit(x_train, y_train)
    prediction = knn_model.predict(x_test)
    acc = acc.append(pd.Series(metrics.accuracy_score(prediction,
y_test)))
plt.plot(x_axis, acc)
plt.xticks(x)
plt.title("Finding best value for n_estimators")
plt.xlabel("n_estimators")
plt.ylabel("Accuracy")
plt.grid()
plt.show()
print('Highest value: ',acc.values.max())
```




```
Highest value:  0.7597402597402597

# K nearest neighbors Algorithm
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 24, metric = 'minkowski', p =
2)
knn.fit(x_train, y_train)

KNeighborsClassifier(n_neighbors=24)

Knn_y_pred_train =pd.DataFrame(knn.predict(x_train))
Knn_y_pred_train

       0
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
609  0.0
610  0.0
611  0.0
612  0.0
613  0.0

[614 rows x 1 columns]
```

```
Knn_y_pred_test =pd.DataFrame(knn.predict(x_test))
Knn_y_pred_test
```

```
       0
0    1.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
149  0.0
150  0.0
151  0.0
152  1.0
153  0.0

[154 rows x 1 columns]
```

```
print('train accuracy:',accuracy_score(Knn_y_pred_train,y_train))
print('test accuracy:',accuracy_score(Knn_y_pred_test,y_test))
```

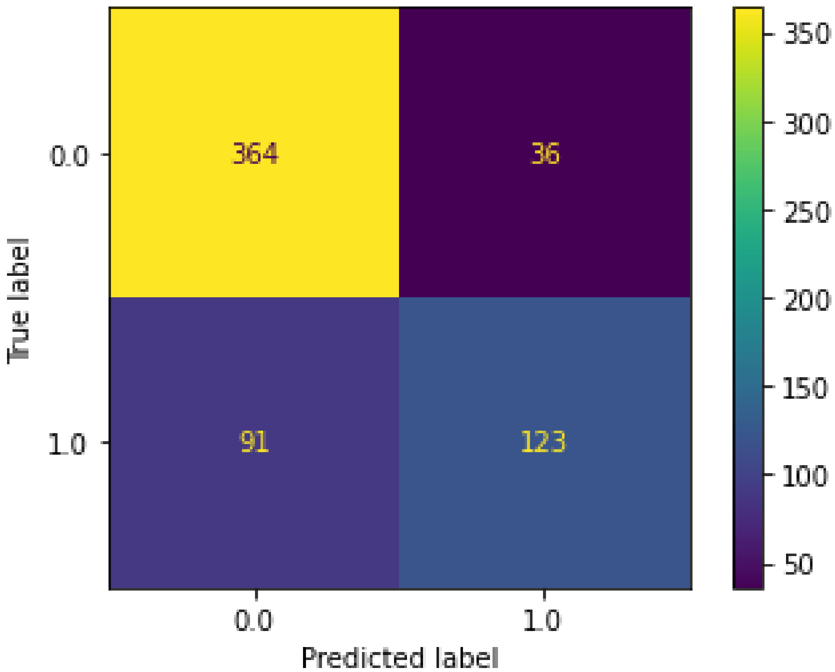
```
train accuracy: 0.7931596091205212
test accuracy: 0.7272727272727273
```

```
print("Train confusion_matrix \n \n
",confusion_matrix(y_train,Knn_y_pred_train ))
print("Test confusion_matrix \n \n
",confusion_matrix(y_test,Knn_y_pred_test ))
```

```
Train confusion_matrix

  [[364  36]
 [ 91 123]]
Test confusion_matrix

  [[87 13]
 [29 25]]
```

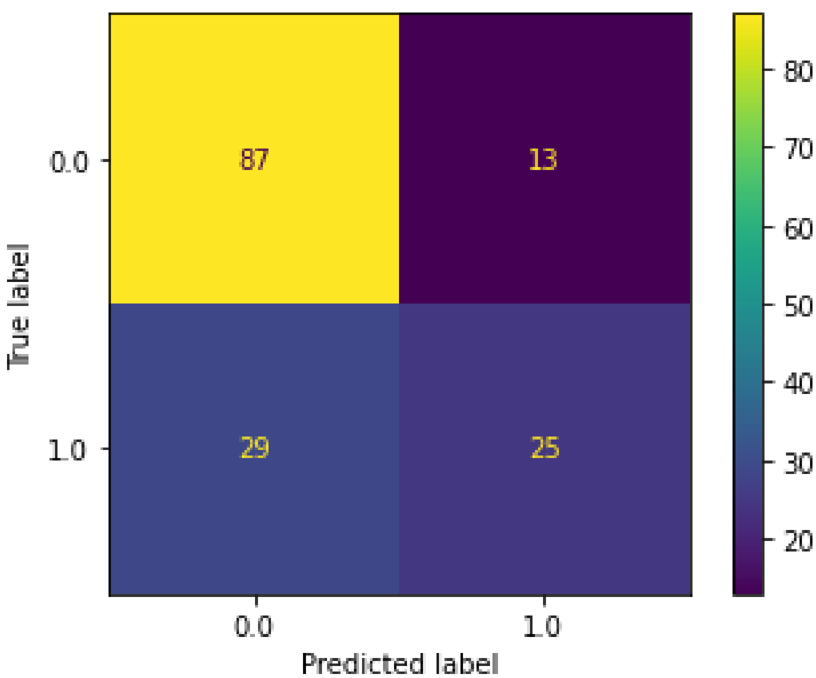
```
plot_confusion_matrix(knn,x_train,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0dddb7b20>
```




```
plot_confusion_matrix(knn,x_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0dc8ccdf0>
```



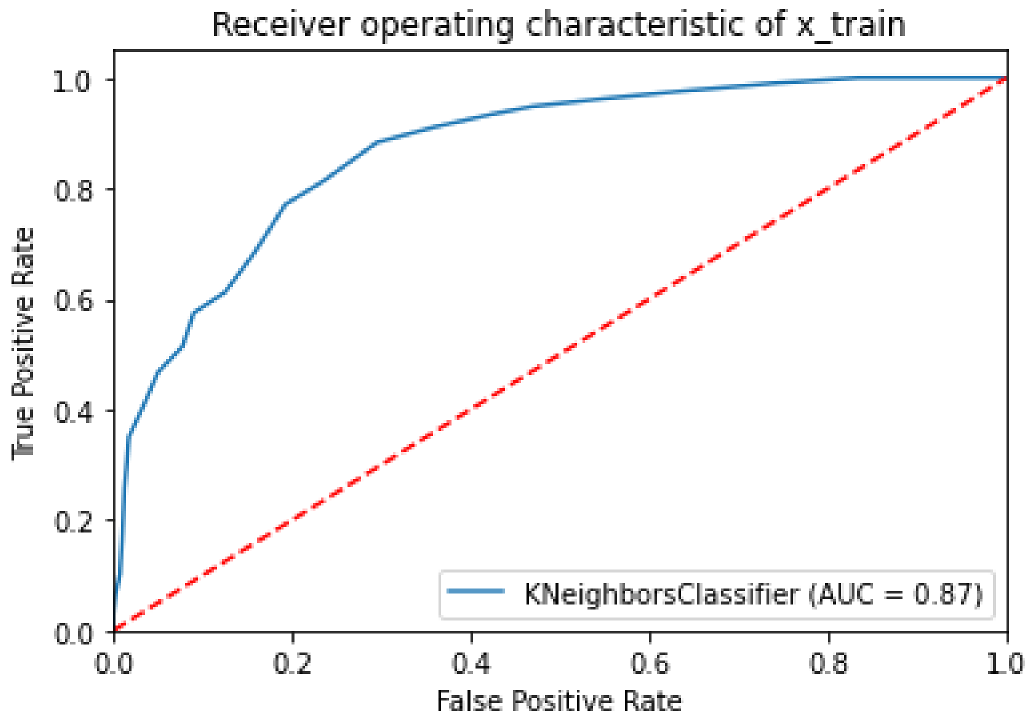

```
print("Train classification_report \n \n
",classification_report(y_train, Knn_y_pred_train))
```

```
Train classification_report

               precision    recall  f1-score   support

         0.0       0.80      0.91      0.85       400
         1.0       0.77      0.57      0.66       214

    accuracy                           0.79       614
   macro avg       0.79      0.74      0.76       614
weighted avg       0.79      0.79      0.78       614
```

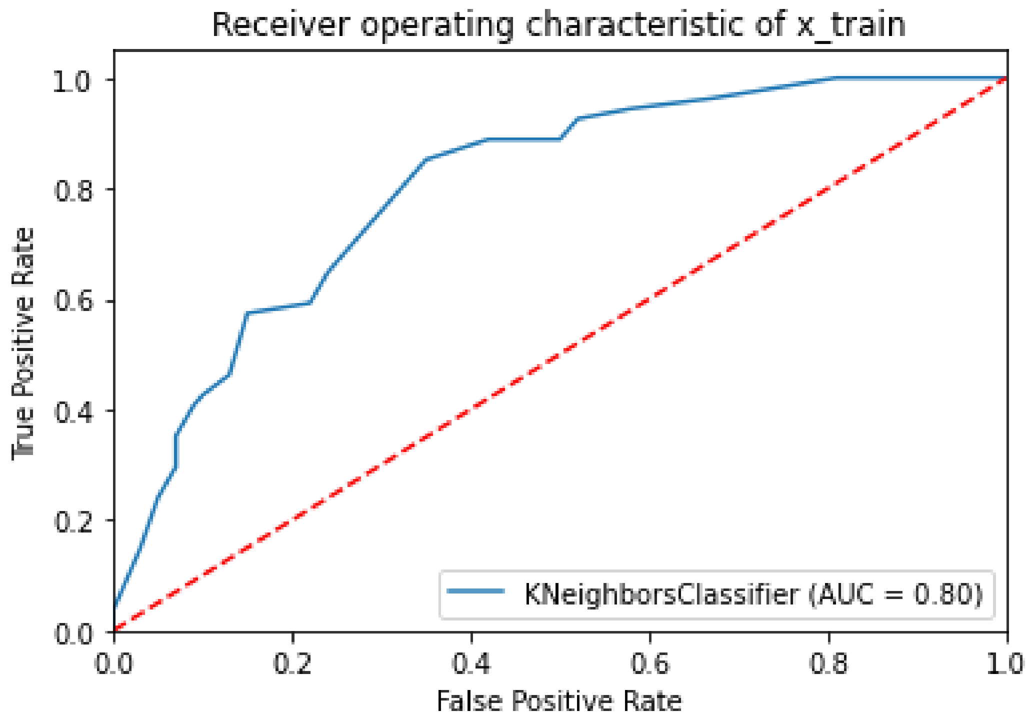
```
print("Test classification_report \n \n
",classification_report(y_test, Knn_y_pred_test))
```

```
Test classification_report

               precision    recall  f1-score   support

         0.0       0.75      0.87      0.81       100
         1.0       0.66      0.46      0.54        54

    accuracy                           0.73       154
   macro avg       0.70      0.67      0.67       154
weighted avg       0.72      0.73      0.71       154
```

```
plot_roc_curve(knn,x_train,y_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_train')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```




```
plot_roc_curve(knn,x_test,y_test)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_train')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



Decesion tree classifier

```
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn import tree

dtc=DecisionTreeClassifier(criterion="entropy",max_depth=4,min_samples
_split= 4)
dtc.fit(x_train,y_train)

DecisionTreeClassifier(criterion='entropy', max_depth=4,
min_samples_split=4)

dtc_y_train_pred=pd.DataFrame(dtc.predict(x_train))
dtc_y_train_pred

       0
0    0.0
1    1.0
2    0.0
3    1.0
4    0.0
..   ...
609  0.0
610  0.0
611  0.0
612  1.0
613  1.0
```

```
[614 rows x 1 columns]
```

```
dtc_y_test_pred=pd.DataFrame(dtc.predict(x_test))
dtc_y_test_pred
```

```
       0
0    1.0
1    0.0
2    0.0
3    1.0
4    0.0
..   ...
149  0.0
150  0.0
151  0.0
152  1.0
153  0.0

[154 rows x 1 columns]
```

```
print('train accuracy:',accuracy_score(dtc_y_train_pred,y_train))
print('test accuracy:',accuracy_score(dtc_y_test_pred,y_test))
```

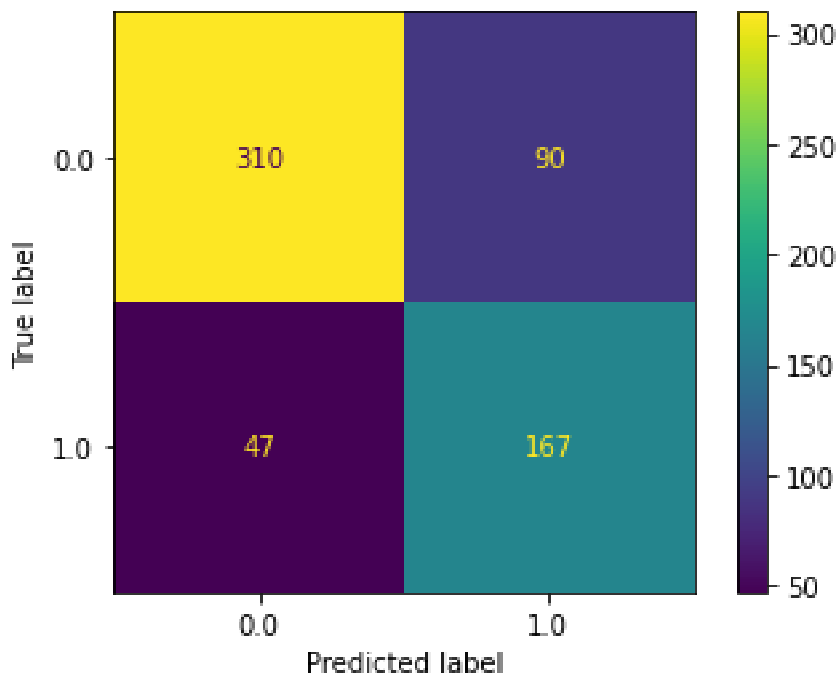
```
train accuracy: 0.7768729641693811
test accuracy: 0.7922077922077922
```

```
print("Train confusion_matrix \n \n ",confusion_matrix(y_train,
dtc_y_train_pred))
print("Test confusion_matrix \n \n ",confusion_matrix(y_test,
dtc_y_test_pred))
```

```
Train confusion_matrix 
 
  [[310  90]
 [ 47 167]]
Test confusion_matrix 
 
  [[80 20]
 [12 42]]
```

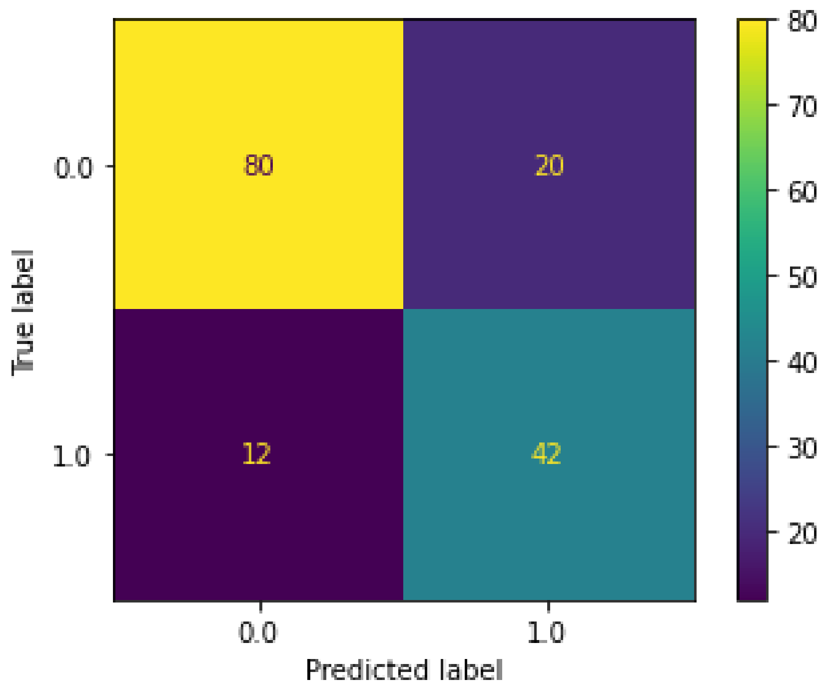
```
plot_confusion_matrix(dtc,x_train,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0ddae9460>
```




```
plot_confusion_matrix(dtc,x_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0ddadff40>
```

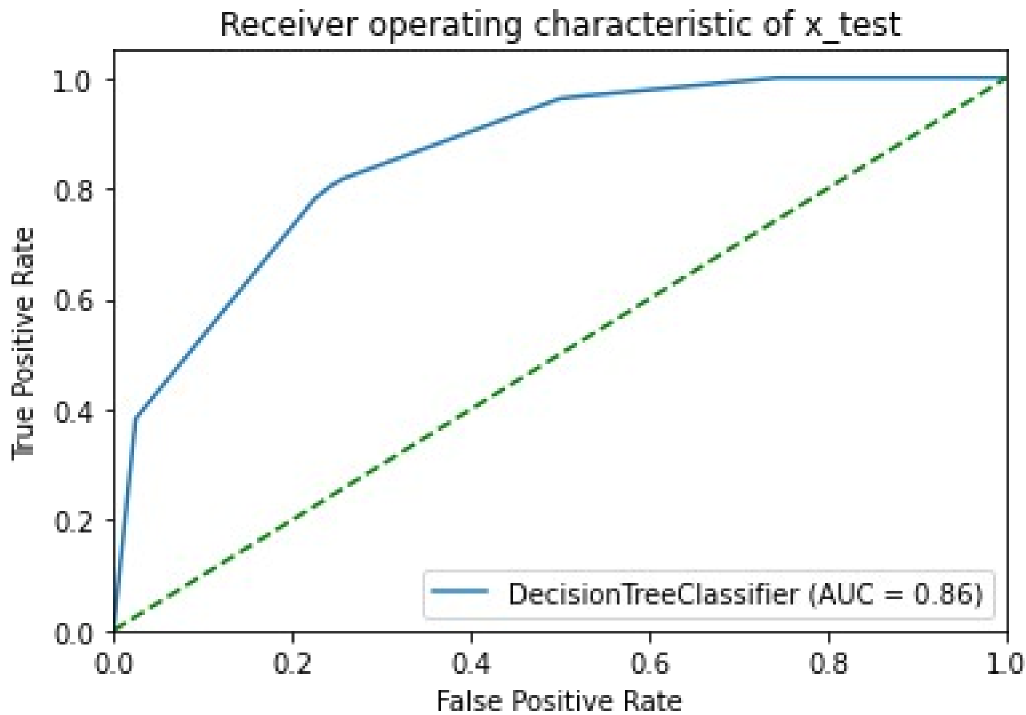



```
print("Train classification_report \n \n
",classification_report(y_train, dtc_y_train_pred))
```

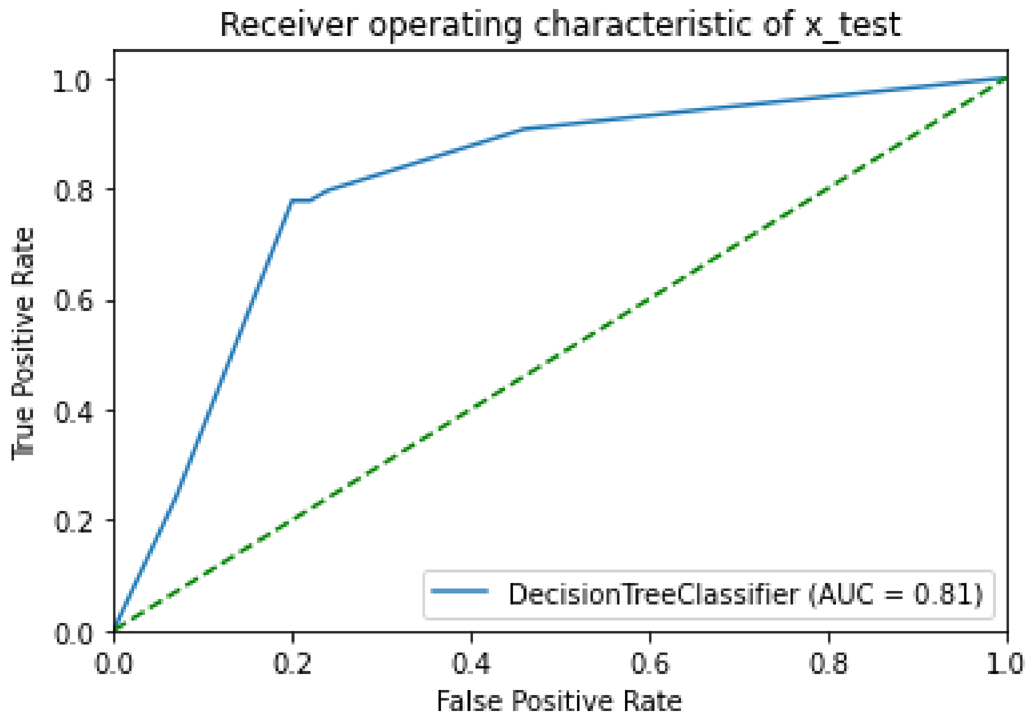
```
print("Test classification_report \n \n
",classification_report(y_test, dtc_y_test_pred))
```

```
Train classification_report

                precision    recall  f1-score   support

         0.0       0.87      0.78      0.82       400
         1.0       0.65      0.78      0.71       214

    accuracy                           0.78       614
   macro avg       0.76      0.78      0.76       614
weighted avg       0.79      0.78      0.78       614

Test classification_report

                precision    recall  f1-score   support

         0.0       0.87      0.80      0.83       100
         1.0       0.68      0.78      0.72        54

    accuracy                           0.79       154
   macro avg       0.77      0.79      0.78       154
weighted avg       0.80      0.79      0.80       154
```

```
plot_roc_curve(dtc,x_train,y_train)
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```




```
plot_roc_curve(dtc,x_test,y_test)
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
RR=RandomForestClassifier(criterion='entropy',max_depth=6,n_estimators
=25)
RR.fit(x_train,y_train)
```

```
RandomForestClassifier(criterion='entropy', max_depth=6,
n_estimators=25)
```

```
RR_y_train_pred=pd.DataFrame(RR.predict(x_train))
RR_y_train_pred
```

```
       0
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
609  0.0
610  0.0
611  0.0
612  0.0
613  0.0

[614 rows x 1 columns]
```

```
RR_y_test_pred=pd.DataFrame(RR.predict(x_test))
RR_y_test_pred
```

```
       0
0    1.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
149  0.0
150  0.0
151  0.0
152  1.0
153  0.0

[154 rows x 1 columns]
```

```
print('train accuracy:',accuracy_score(RR_y_train_pred,y_train))
print('test accuracy:',accuracy_score(RR_y_test_pred,y_test))
```

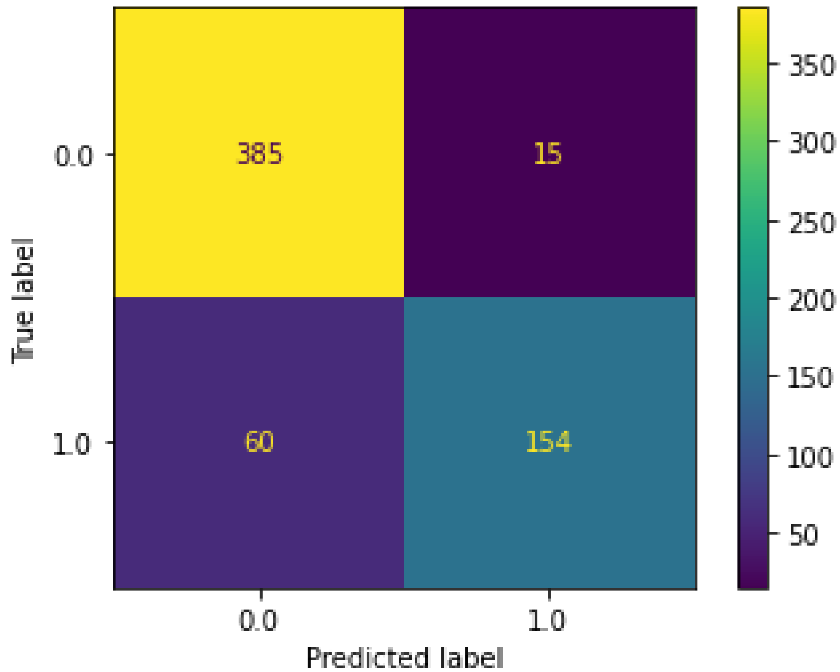
```
train accuracy: 0.8778501628664495
test accuracy: 0.7662337662337663
```

```
print("Train confusion_matrix \n \n ",confusion_matrix(y_train,
RR_y_train_pred))
print("TEST confusion_matrix \n \n ",confusion_matrix(y_test,
RR_y_test_pred))
```

```
Train confusion_matrix

  [[385  15]
 [ 60 154]]
TEST confusion_matrix

  [[89 11]
 [25 29]]
```

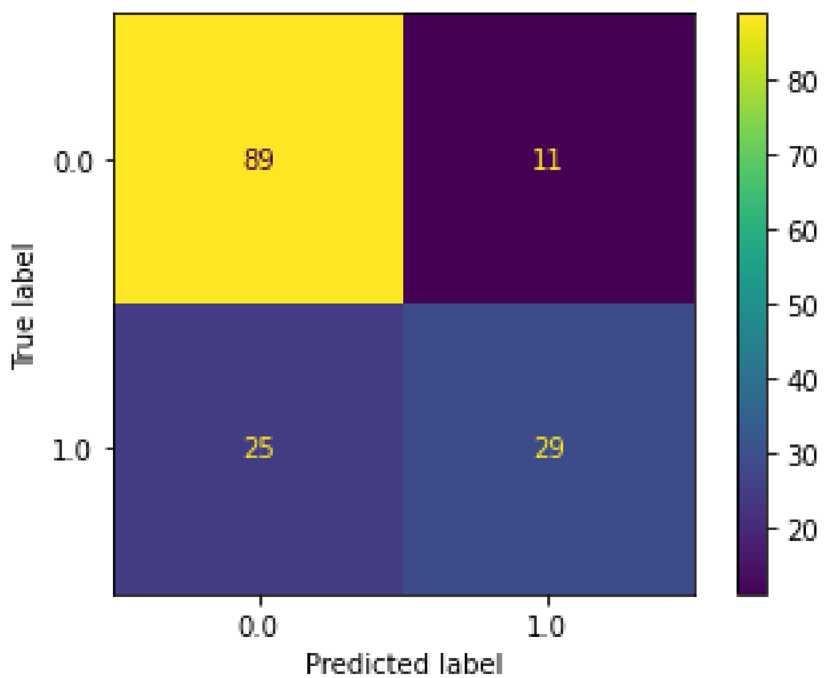
```
plot_confusion_matrix(RR,x_train,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0defe39d0>
```




```
plot_confusion_matrix(RR,x_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0def10e20>
```

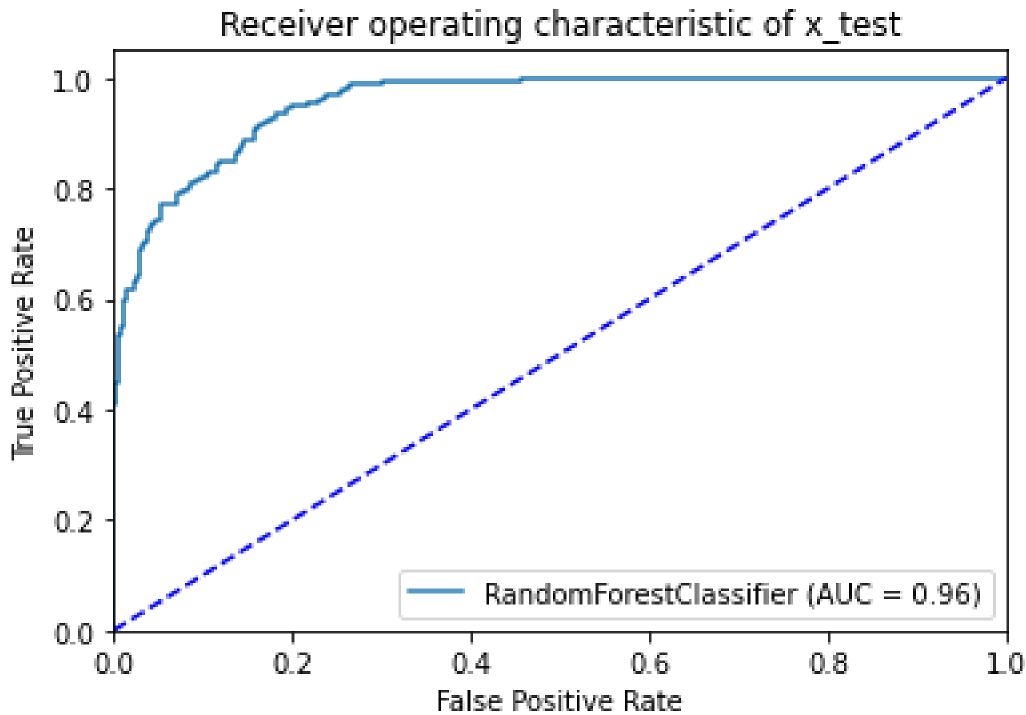



```
print("Train classification_report \n \n
",classification_report(y_train, RR_y_train_pred))
```

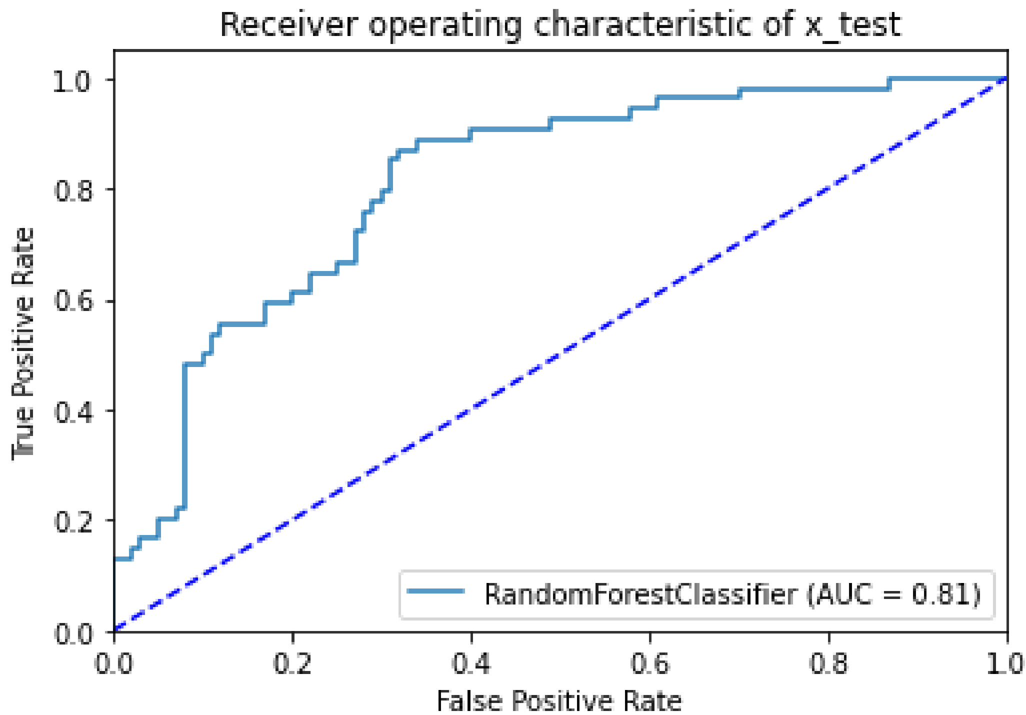
```
print("TEST classification_report \n \n
",classification_report(y_test, RR_y_test_pred))
```

```
Train classification_report

              precision    recall  f1-score   support

         0.0       0.87      0.96      0.91       400
         1.0       0.91      0.72      0.80       214

    accuracy                           0.88       614
   macro avg       0.89      0.84      0.86       614
weighted avg       0.88      0.88      0.87       614

TEST classification_report

              precision    recall  f1-score   support

         0.0       0.78      0.89      0.83       100
         1.0       0.72      0.54      0.62        54

    accuracy                           0.77       154
   macro avg       0.75      0.71      0.72       154
weighted avg       0.76      0.77      0.76       154
```

```
plot_roc_curve(RR,x_train,y_train)
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```




```
plot_roc_curve(RR,x_test,y_test)
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



AdaBoostClassifier

```
from sklearn.ensemble import AdaBoostClassifier
abc=AdaBoostClassifier(n_estimators=16)
abc.fit(x_train,y_train)
```

```
AdaBoostClassifier(n_estimators=16)
```

```
ad_pred_y_train=pd.DataFrame(abc.predict(x_train))
ad_pred_y_train
```

```
       0
0    0.0
1    1.0
2    0.0
3    0.0
4    0.0
..   ...
609  0.0
610  0.0
611  0.0
612  1.0
613  0.0

[614 rows x 1 columns]
```

```
ad_pred_y_test=pd.DataFrame(abc.predict(x_test))
ad_pred_y_test
```

```
       0
0    1.0
1    0.0
2    0.0
3    1.0
4    0.0
..   ...
149  0.0
150  0.0
151  0.0
152  1.0
153  0.0

[154 rows x 1 columns]
```

```
print('train accuracy:',accuracy_score(ad_pred_y_train,y_train))
print('test accuracy:',accuracy_score(ad_pred_y_test,y_test))
```

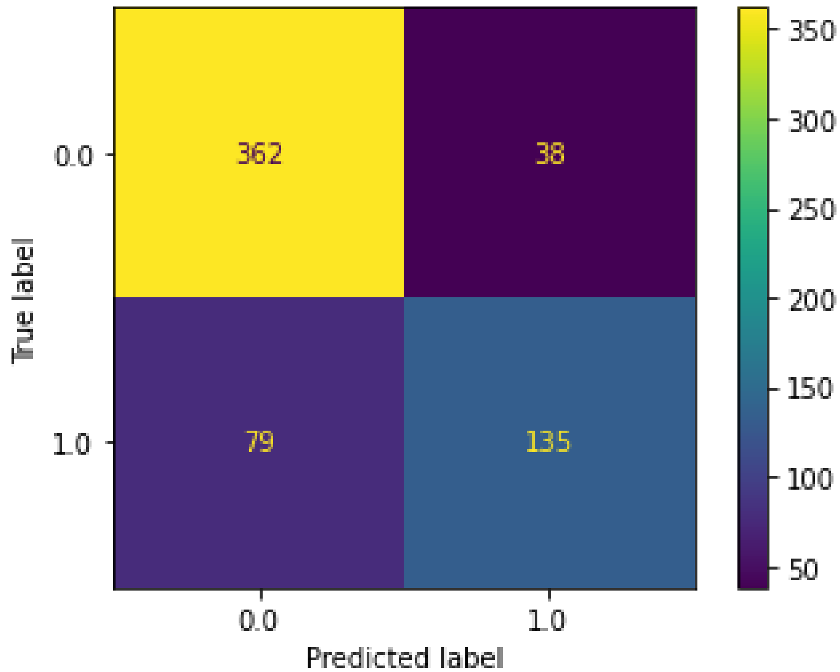
```
train accuracy: 0.8094462540716613
test accuracy: 0.7662337662337663
```

```
print("Train confusion_matrix \n \n ",confusion_matrix(y_train,
ad_pred_y_train))
print("Test confusion_matrix \n \n ",confusion_matrix(y_test,
ad_pred_y_test))
```

```
Train confusion_matrix 

  [[362  38]
 [ 79 135]]
Test confusion_matrix 

  [[86 14]
 [22 32]]
```

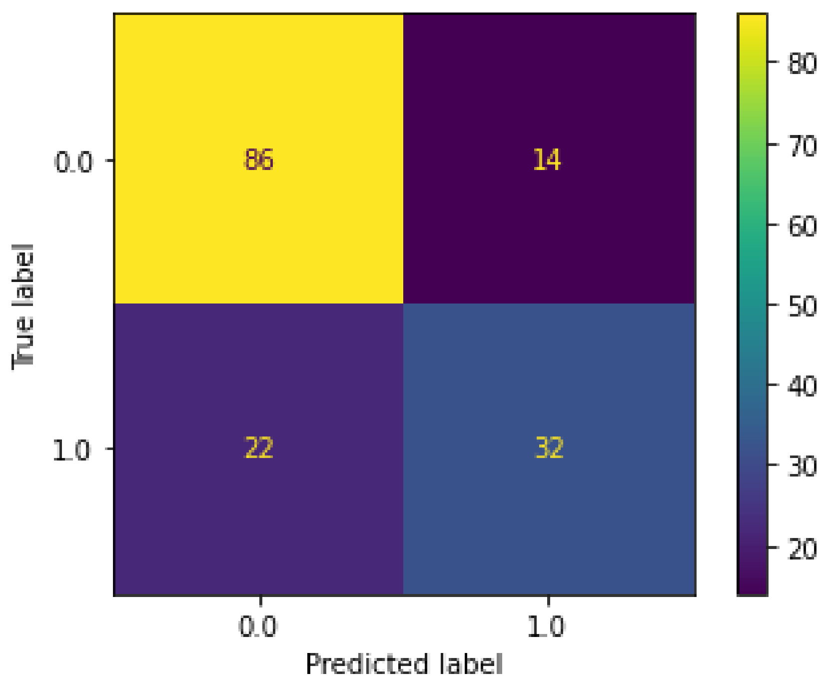
```
plot_confusion_matrix(abc,x_train,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0e2ebe220>
```




```
plot_confusion_matrix(abc,x_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0df0a5f10>
```

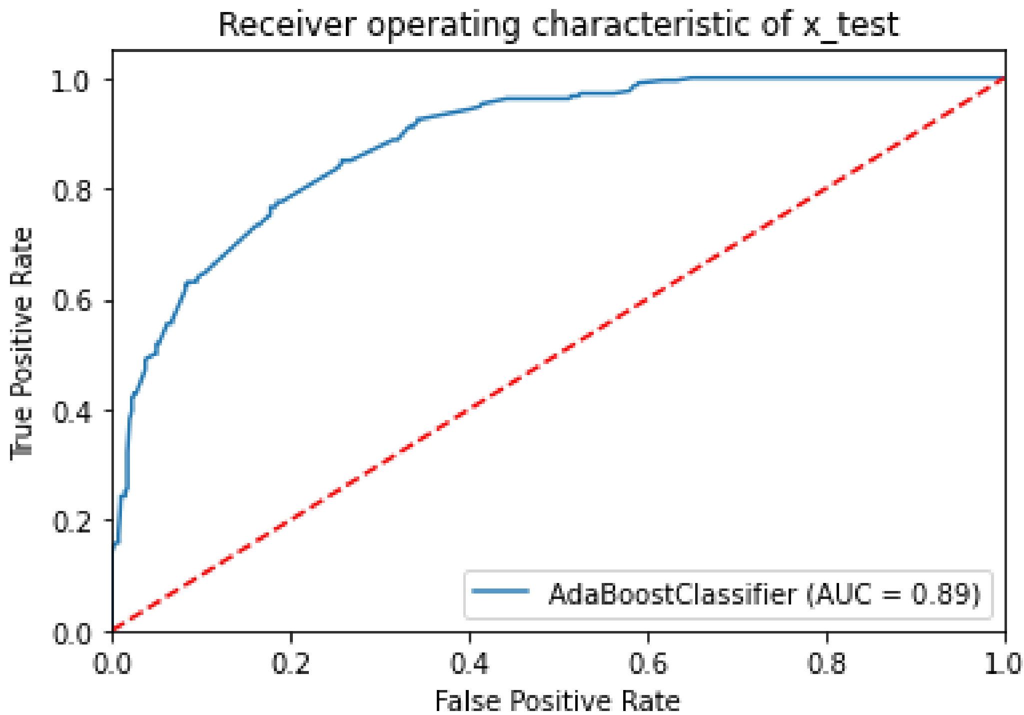



```
print("Train classification_report \n \n
",classification_report(y_train,ad_pred_y_train))
```

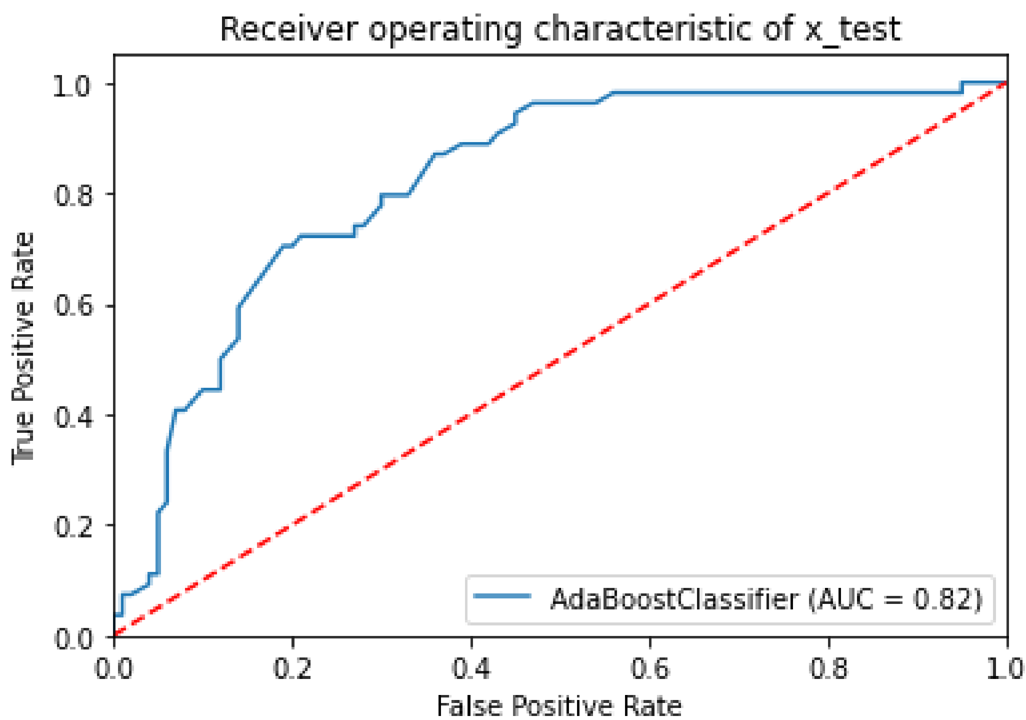
```
print("Test classification_report \n \n
",classification_report(y_test,ad_pred_y_test))
```

```
Train classification_report

                precision    recall  f1-score   support

         0.0       0.82      0.91      0.86       400
         1.0       0.78      0.63      0.70       214

    accuracy                           0.81       614
   macro avg       0.80      0.77      0.78       614
weighted avg       0.81      0.81      0.80       614

Test classification_report

                precision    recall  f1-score   support

         0.0       0.80      0.86      0.83       100
         1.0       0.70      0.59      0.64        54

    accuracy                           0.77       154
   macro avg       0.75      0.73      0.73       154
weighted avg       0.76      0.77      0.76       154
```

```
plot_roc_curve(abc,x_train,y_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```




```
plot_roc_curve(abc,x_test,y_test)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



GradientBoostingClassifier

```
from sklearn.ensemble import GradientBoostingClassifier
gbc=GradientBoostingClassifier(max_depth=2,n_estimators= 28)
gbc.fit(x_train,y_train)

GradientBoostingClassifier(max_depth=2, n_estimators=28)

gbc_y_train_pred=pd.DataFrame(gbc.predict(x_train))
gbc_y_train_pred

       0
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
609  0.0
610  0.0
611  0.0
612  0.0
613  0.0

[614 rows x 1 columns]

gbc_y_test_pred=pd.DataFrame(gbc.predict(x_test))
gbc_y_test_pred
```

```
       0
0    1.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
149  0.0
150  0.0
151  0.0
152  1.0
153  0.0

[154 rows x 1 columns]
```

```
print('train accuracy:',accuracy_score(gbc_y_train_pred,y_train))
print('test accuracy:',accuracy_score(gbc_y_test_pred,y_test))
```

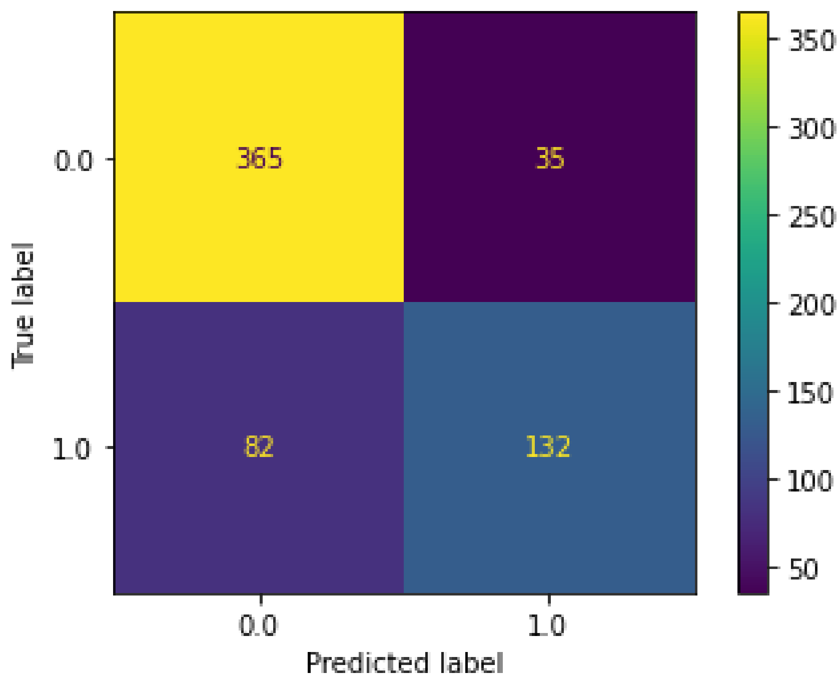
```
train accuracy: 0.8094462540716613
test accuracy: 0.7077922077922078
```

```
print("Train confusion_matrix \n \n ",confusion_matrix(y_train,
gbc_y_train_pred))
print("Test confusion_matrix \n \n ",confusion_matrix(y_test,
gbc_y_test_pred))
```

```
Train confusion_matrix

  [[365  35]
 [ 82 132]]
Test confusion_matrix

  [[83 17]
 [28 26]]
```

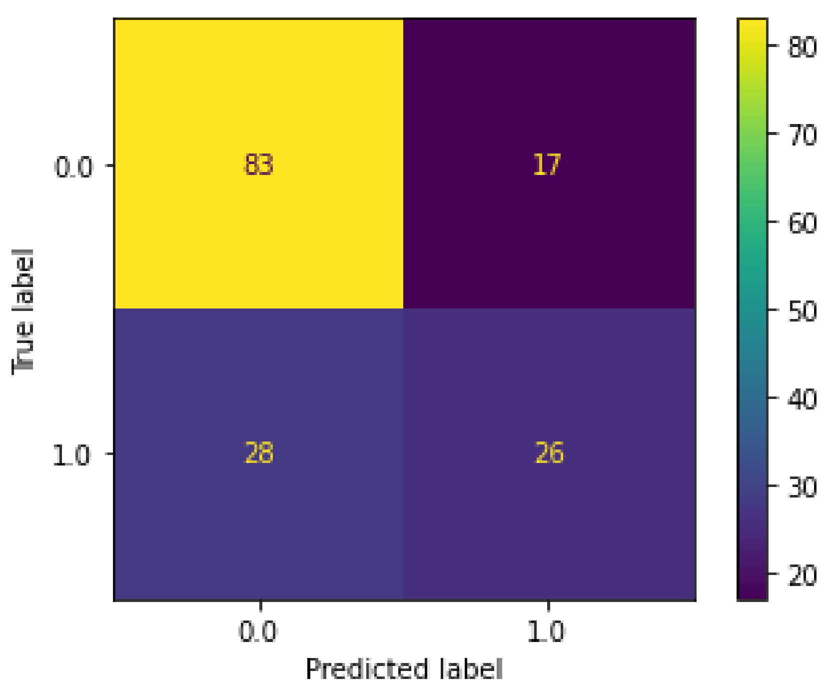
```
plot_confusion_matrix(gbc,x_train,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0e335cb80>
```




```
plot_confusion_matrix(gbc,x_test,y_test)
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2e0ddb85e50>

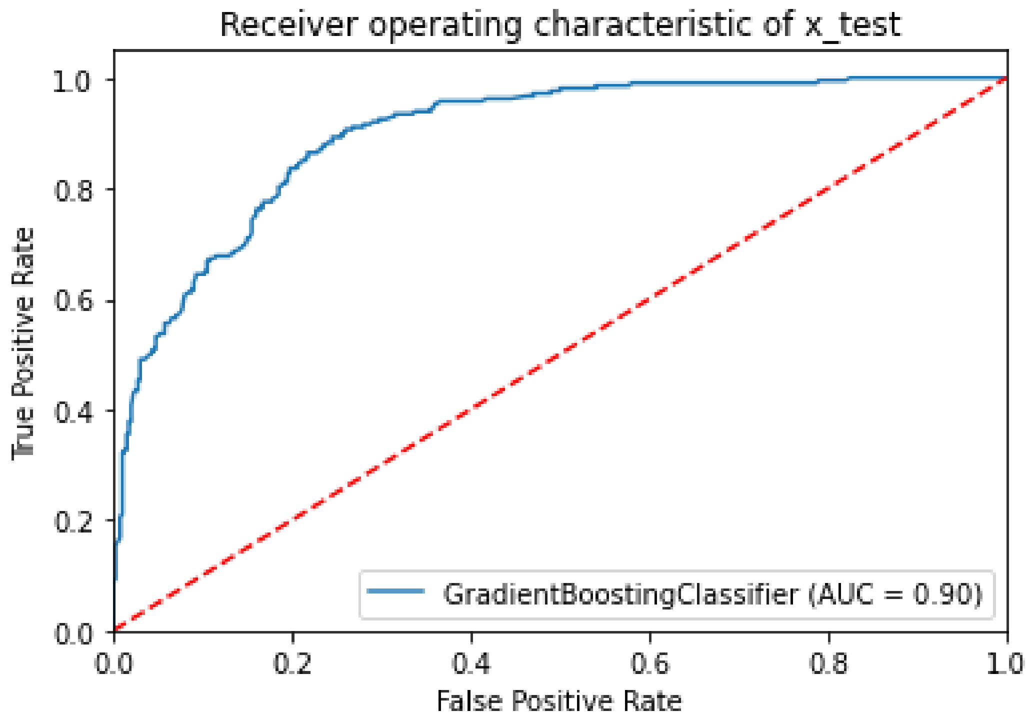



```
print("Train classification_report \n \n
",classification_report(y_train,gbc_y_train_pred))
```

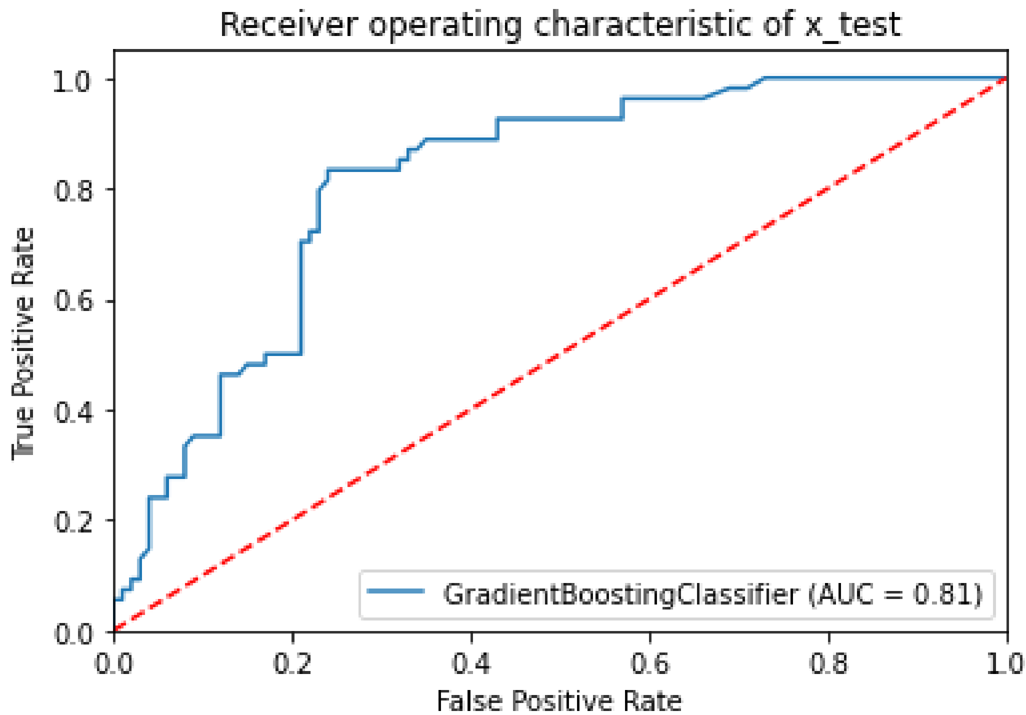
```
print("Test classification_report \n \n
",classification_report(y_test,gbc_y_test_pred))
```

```
Train classification_report

                precision    recall  f1-score   support

         0.0       0.82      0.91      0.86       400
         1.0       0.79      0.62      0.69       214

    accuracy                           0.81       614
   macro avg       0.80      0.76      0.78       614
weighted avg       0.81      0.81      0.80       614

Test classification_report

                precision    recall  f1-score   support

         0.0       0.75      0.83      0.79       100
         1.0       0.60      0.48      0.54        54

    accuracy                           0.71       154
   macro avg       0.68      0.66      0.66       154
weighted avg       0.70      0.71      0.70       154
```

```
plot_roc_curve(gbc,x_train,y_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```




```
plot_roc_curve(gbc,x_test,y_test)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



SVC

```
# Support Vector Classifier Algorithm
from sklearn.svm import SVC
svc = SVC(kernel = 'linear', random_state = 42)
svc.fit(x_train, y_train)
```

```
SVC(kernel='linear', random_state=42)
```

```
svc_y_train_pred=pd.DataFrame(svc.predict(x_train))
svc_y_train_pred
```

```
       0
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
609  0.0
610  0.0
611  0.0
612  0.0
613  0.0

[614 rows x 1 columns]
```

```python
svc_y_test_pred=pd.DataFrame(svc.predict(x_test))
svc_y_test_pred
```

```
       0
0    1.0
1    0.0
2    0.0
3    0.0
4    0.0
..   ...
149  0.0
150  0.0
151  0.0
152  1.0
153  0.0

[154 rows x 1 columns]
```

```python
print('train accuracy:',accuracy_score(svc_y_train_pred,y_train))
print('test accuracy:',accuracy_score(svc_y_test_pred,y_test))
```

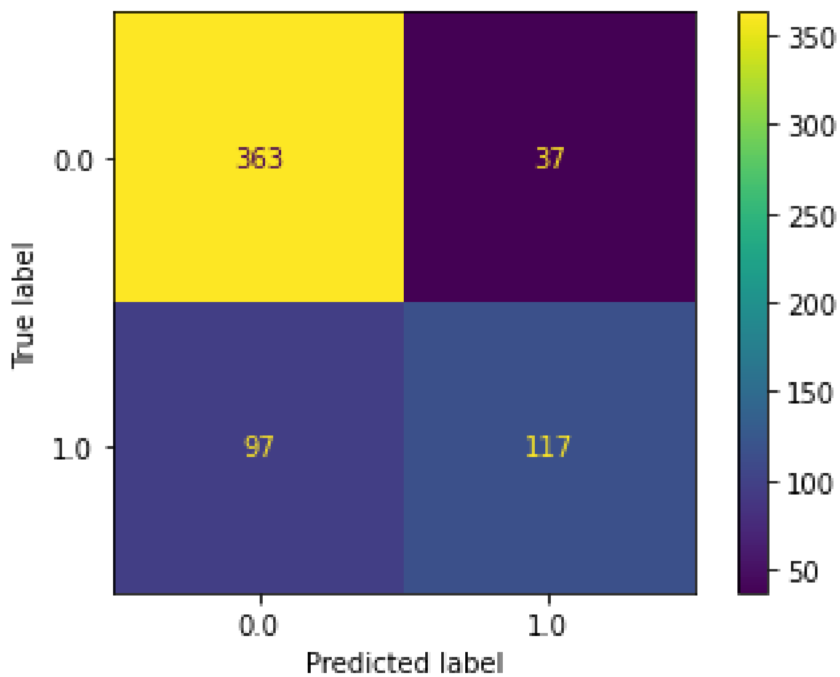
```
train accuracy: 0.7817589576547231
test accuracy: 0.7012987012987013
```

```python
print("Train confusion_matrix \n \n ",confusion_matrix(y_train,
svc_y_train_pred))
print("Test confusion_matrix \n \n ",confusion_matrix(y_test,
svc_y_test_pred))
```

```
Train confusion_matrix

  [[363  37]
 [ 97 117]]
Test confusion_matrix

  [[83 17]
 [29 25]]
```

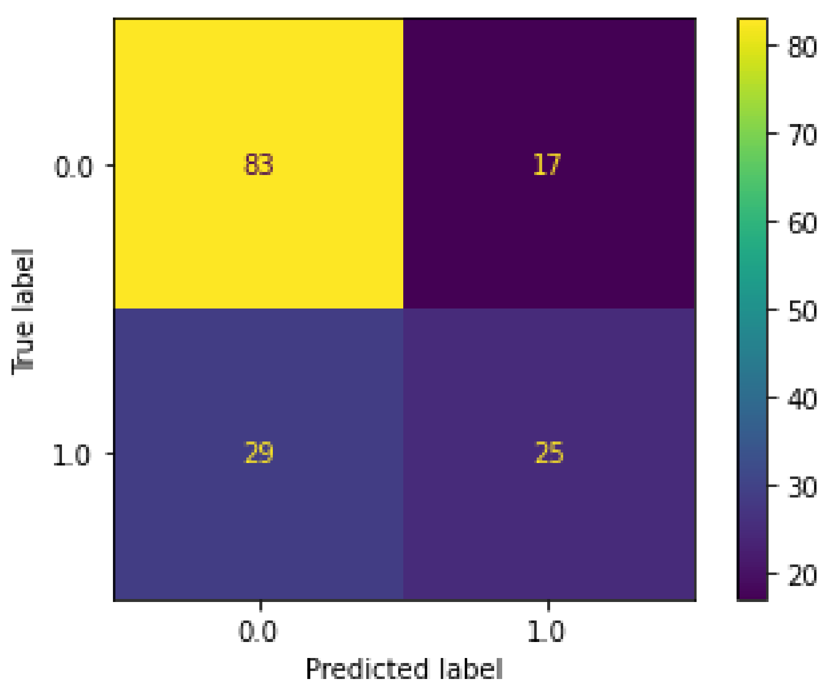
```python
plot_confusion_matrix(svc,x_train,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x2e0ddd6bbb0>
```




```
plot_confusion_matrix(svc,x_test,y_test)
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2e0e335ce80>

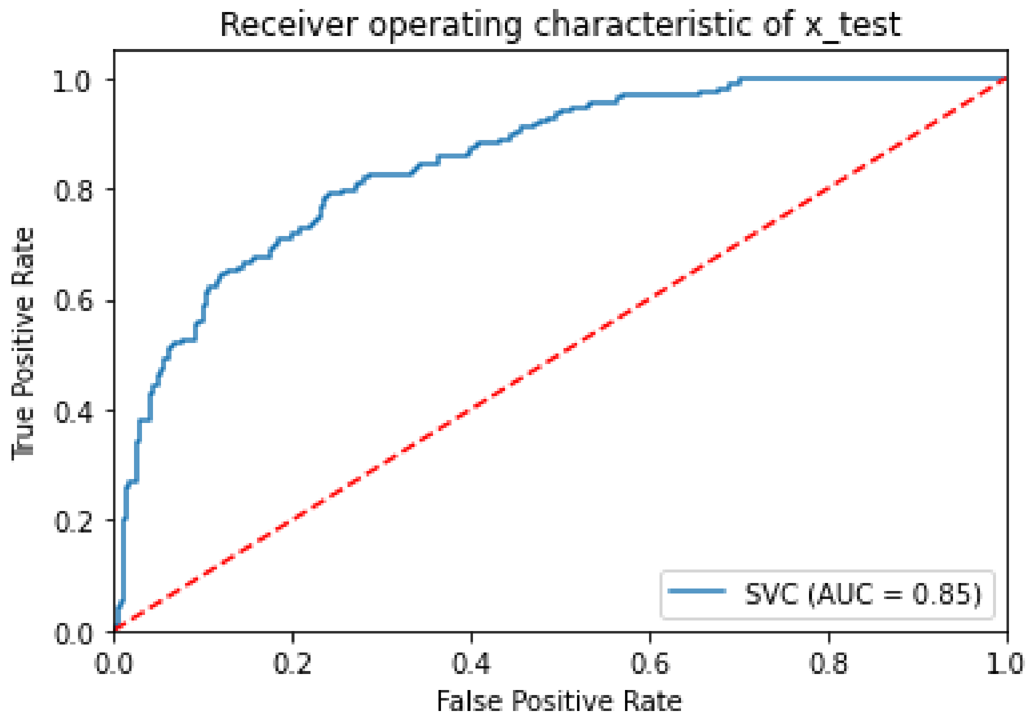



```
print("Train classification_report \n \n
",classification_report(y_train,svc_y_train_pred))
```

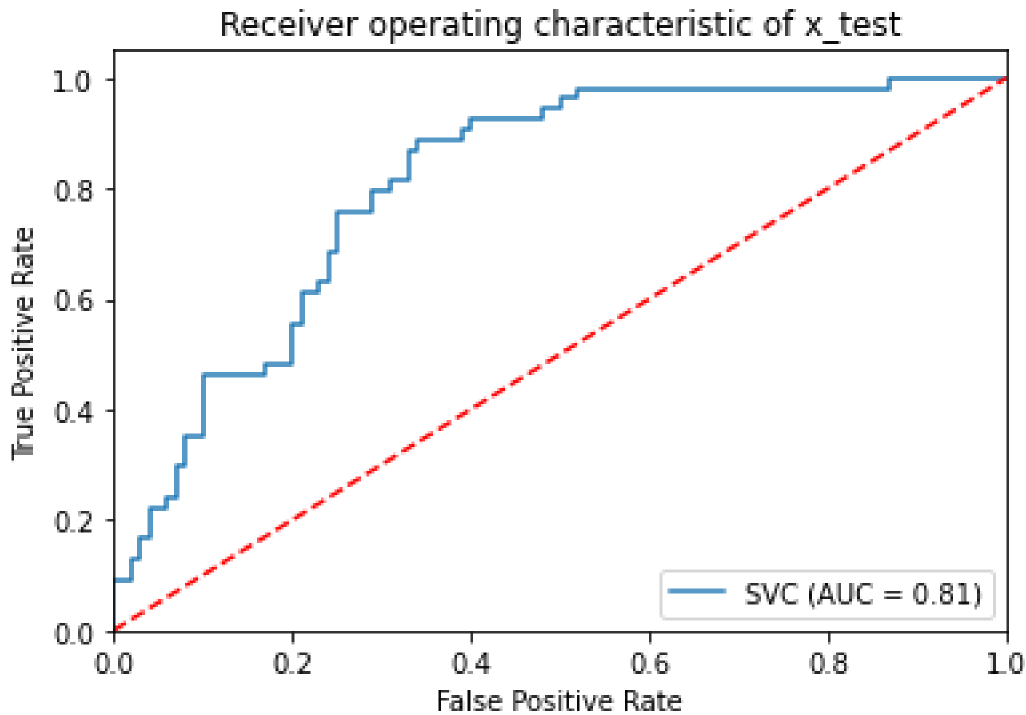
```
print("Test classification_report \n \n
",classification_report(y_test,svc_y_test_pred))
```

```
Train classification_report

               precision    recall  f1-score   support

         0.0       0.79      0.91      0.84       400
         1.0       0.76      0.55      0.64       214

    accuracy                           0.78       614
   macro avg       0.77      0.73      0.74       614
weighted avg       0.78      0.78      0.77       614

Test classification_report

               precision    recall  f1-score   support

         0.0       0.74      0.83      0.78       100
         1.0       0.60      0.46      0.52        54

    accuracy                           0.70       154
   macro avg       0.67      0.65      0.65       154
weighted avg       0.69      0.70      0.69       154
```

```
plot_roc_curve(svc,x_train,y_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```




```
plot_roc_curve(svc,x_test,y_test)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of x_test')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



Model Evaluation

```
data.columns

Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')

models = []
models.append(('Logistic Regression', lr))
models.append(('KNN', knn))
models.append(('SVC', svc))
models.append(('Decision tree', dtc))
models.append(('Random Forest', RR))
models.append(('Adboost', abc))
models.append(('Gboost', gbc))

from sklearn import model_selection
model_names=['Logistic Regression','KNN','SVC','Decision tree','Random
Forest','Adboost','Gboost']
results = []
reports = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age']
scoring = 'accuracy'
for report, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=7)
    cv_results = model_selection.cross_val_score(model, x_train,
```

```
y_train , cv=kfold, scoring=scoring)
    results.append(cv_results)
    reports.append(report)
    msg = "%s: %f (%f)" % (report,cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
plt.suptitle('Algorithm Comparison')
plt.boxplot(results)
plt.xlabel('models')
plt.show()
```

```
Logistic Regression: 0.781756 (0.038181)
KNN: 0.770307 (0.044869)
SVC: 0.776811 (0.043028)
Decision tree: 0.744209 (0.060441)
Random Forest: 0.752353 (0.034431)
Adboost: 0.745796 (0.054192)
Gboost: 0.773533 (0.048587)
```