CAPSTONE PROJECT 2: Health Care Case Study

PROBLEM STATEMENT

NIDDK(National Institute of Diabetes and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly and consequential diseases. The dataset used in this project is originally from NIDDK. The objetive is to predict whether or not a patient has Diabetes, based on certain diagnostic measurements included in the dataset.

Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Import Data

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from itertools import cycle
```

```
train_df = pd.read_csv('health care diabetes.csv')
```

```
print('train_df Shape:', train_df.shape)
```

```
train_df Shape: (768, 9)
```

```
train_df.head()
```

```
   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

3            1       89             66             23       94  28.1

4            0      137             40             35      168  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
```

```
train_df.tail()
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
763           10      101             76             48      180  32.9

764            2      122             70             27        0  36.8

765            5      121             72             23      112  26.2

766            1      126             60              0        0  30.1

767            1       93             70             31        0  30.4


     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

Project Task: Week 1

1. Perform descriptive analysis.

Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

• Glucose

• BloodPressure

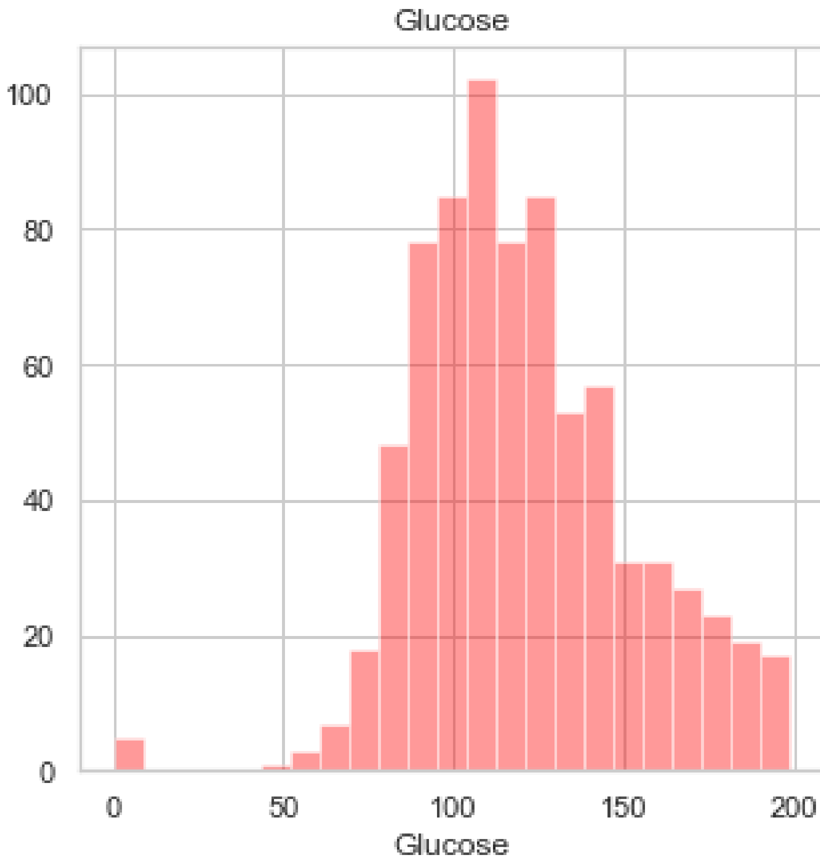
• SkinThickness

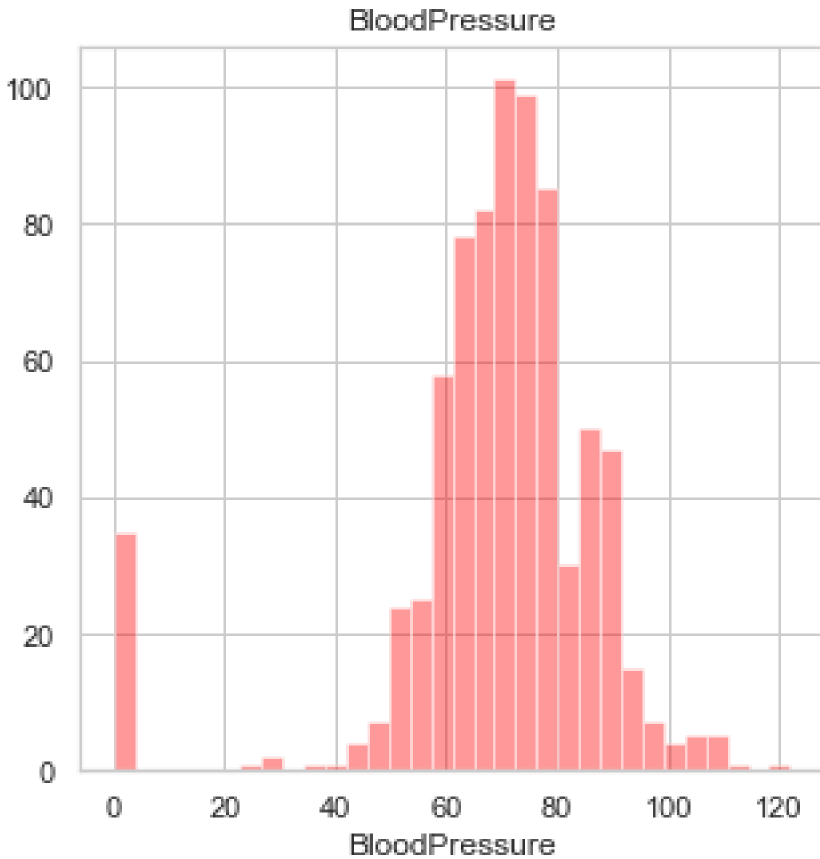
• Insulin

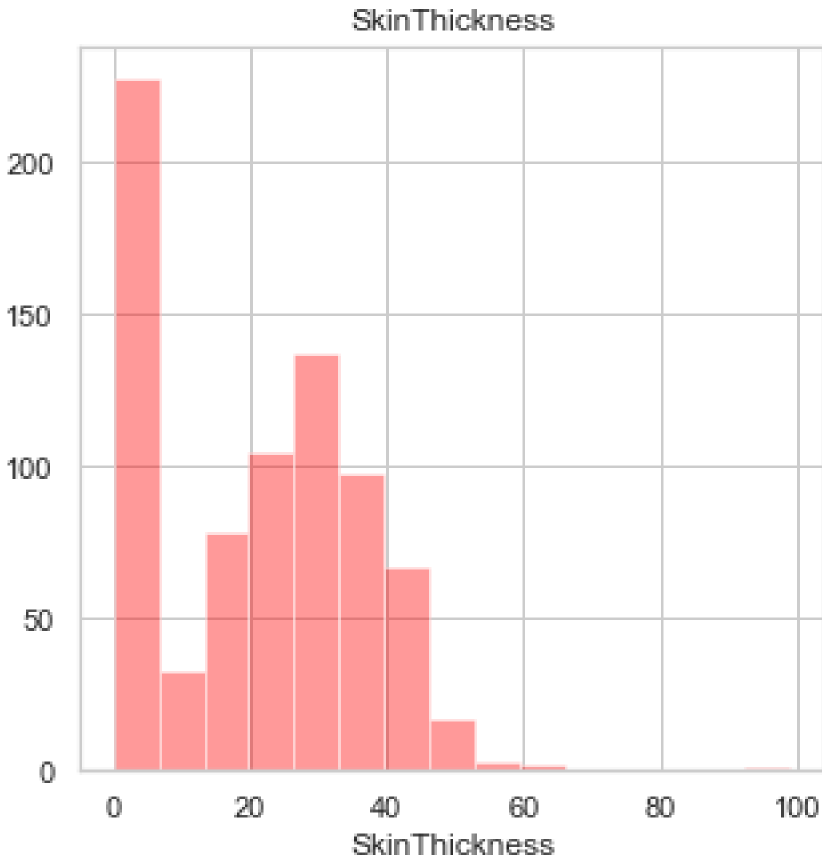
• BMI

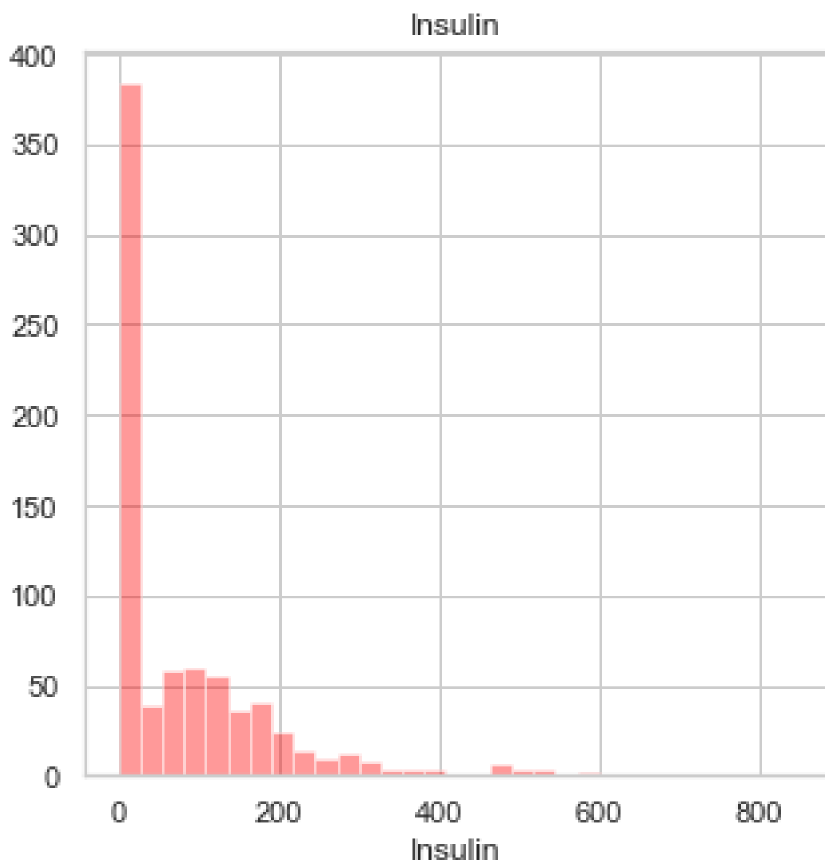
2. Visually explore these variables using histograms. Treat the missing values accordingly.

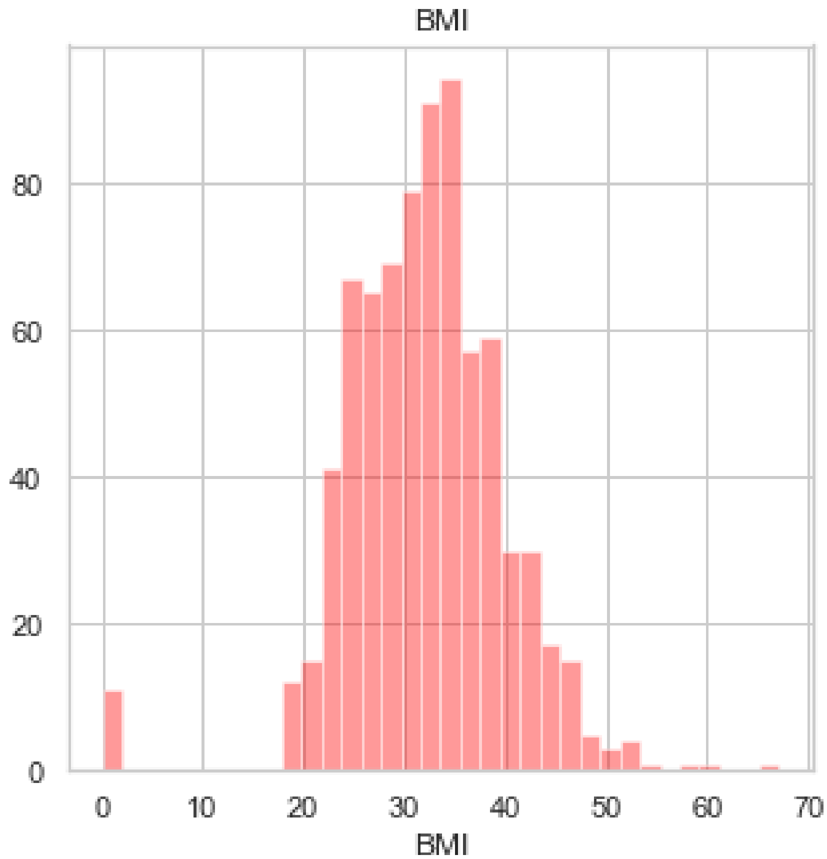
```
miss_cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

for col in miss_cols:
 plt.figure(figsize = (5, 5))
 plt.title(col)
 sns.distplot(train_df[col], kde = False, color = 'red')
```


Glucose
100
80
60
40
20
0
0
50
100
150
200
Glucose


BloodPressure
100
80
60
40
20
0
0
20
40
60
80
100
120
BloodPressure


SkinThickness
200
150
100
50
0
0
20
40
60
80
100
SkinThickness


Insulin
400
350
300
250
200
150
100
50
0
0
200
400
600
800
Insulin




```
train_df.isnull()

     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0          False    False          False          False    False
False
1          False    False          False          False    False
False
2          False    False          False          False    False
False
3          False    False          False          False    False
False
4          False    False          False          False    False
False
..           ...      ...            ...            ...      ...    ..
.
763        False    False          False          False    False
False
764        False    False          False          False    False
False
765        False    False          False          False    False
False
766        False    False          False          False    False
False
```

```
767        False    False          False          False    False
False

     DiabetesPedigreeFunction    Age  Outcome
0                       False  False    False
1                       False  False    False
2                       False  False    False
3                       False  False    False
4                       False  False    False
..                        ...    ...      ...
763                     False  False    False
764                     False  False    False
765                     False  False    False
766                     False  False    False
767                     False  False    False

[768 rows x 9 columns]
```

```
train_df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
#total null values in the dataset

train_df.isnull().sum().sum()
```

```
0
```

```
df1 = train_df.fillna(value=0)
df1
```

```
     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

3              1       89             66             23       94  28.1

4              0      137             40             35      168  43.1
```

```
..            ...      ...             ...            ...      ...   ...

763            10      101              76             48      180  32.9

764             2      122              70             27        0  36.8

765             5      121              72             23      112  26.2

766             1      126              60              0        0  30.1

767             1       93              70             31        0  30.4


     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
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

```
df1.isnull().sum().sum()  #No null values
```

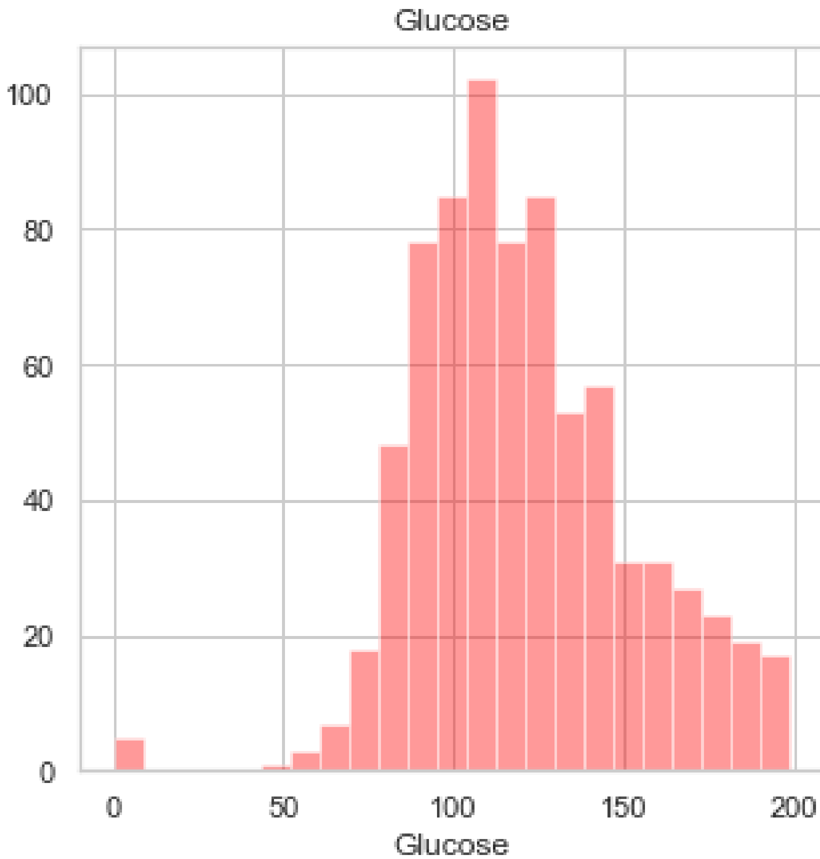
```
0
```

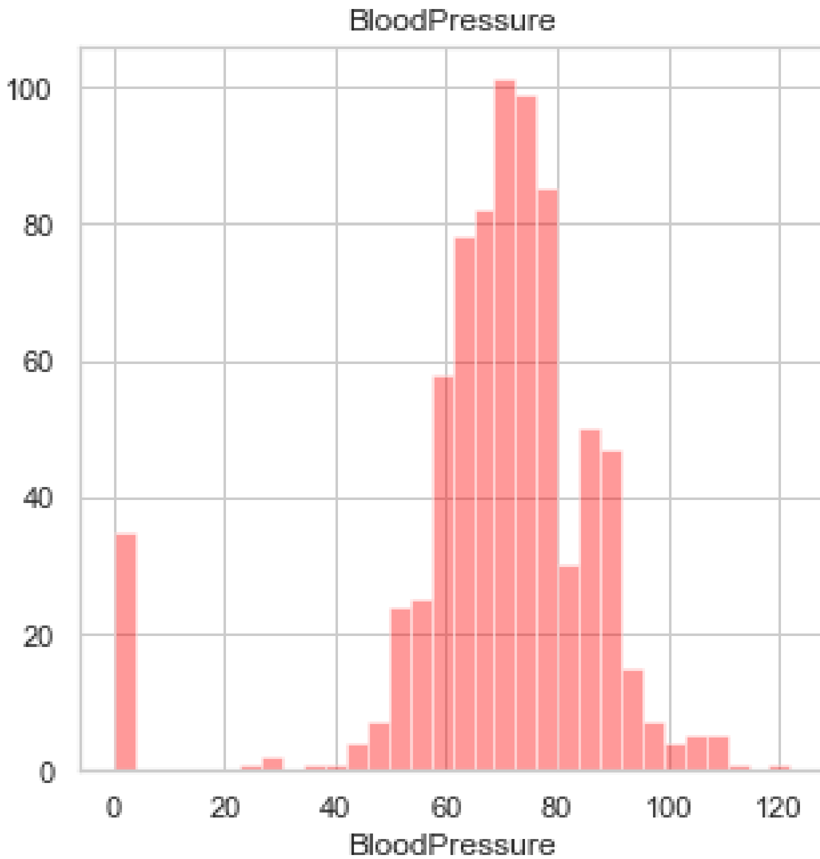
```
df1.columns
```

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

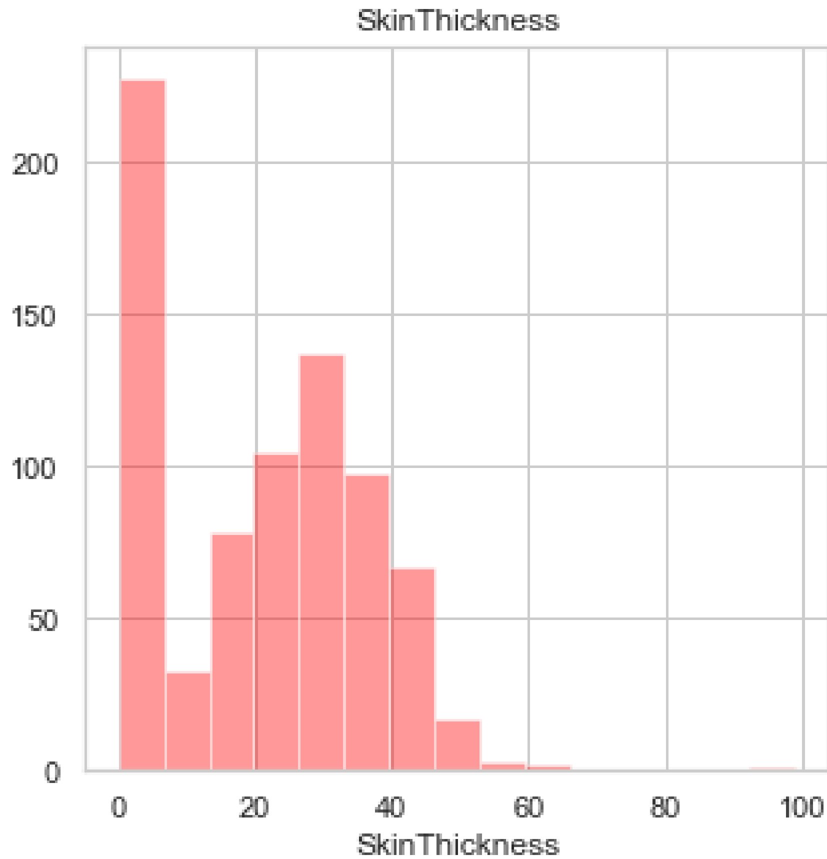
```
df1.columns = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin',
 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
```

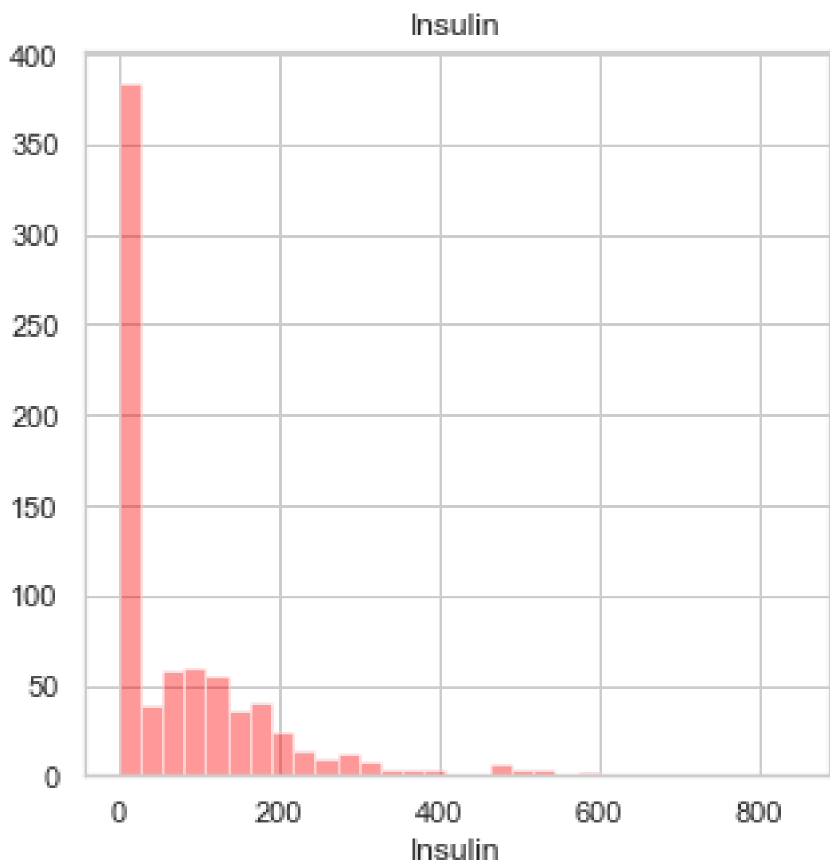
```
for col in miss_cols:
 plt.figure(figsize = (5, 5))
 plt.title(col)
 sns.distplot(train_df[col], kde = False, color = 'red')
```

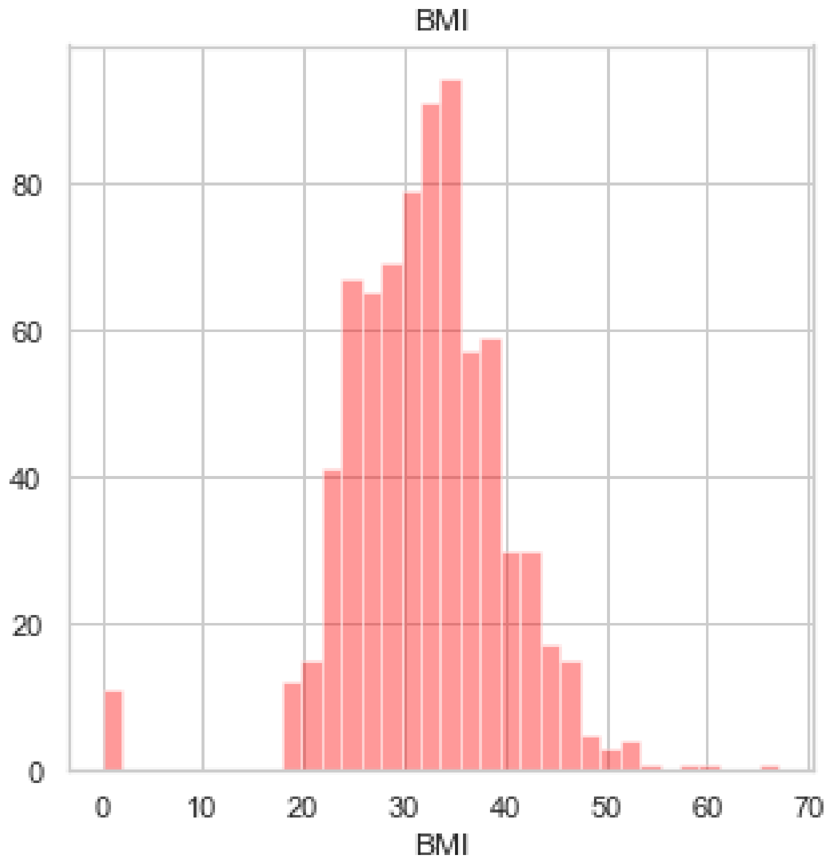

Glucose
100
80
60
40
20
0
0
50
100
150
200
Glucose



BloodPressure
100
80
60
40
20
0
0
20
40
60
80
100
120
BloodPressure


SkinThickness
200
150
100
50
0
0
20
40
60
80
100
SkinThickness


Insulin
400
350
300
250
200
150
100
50
0
0
200
400
600
800
Insulin




```
df1.dtypes
```

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

```
df2 = df1.astype(float) #converting all the columns into float datatype.
```

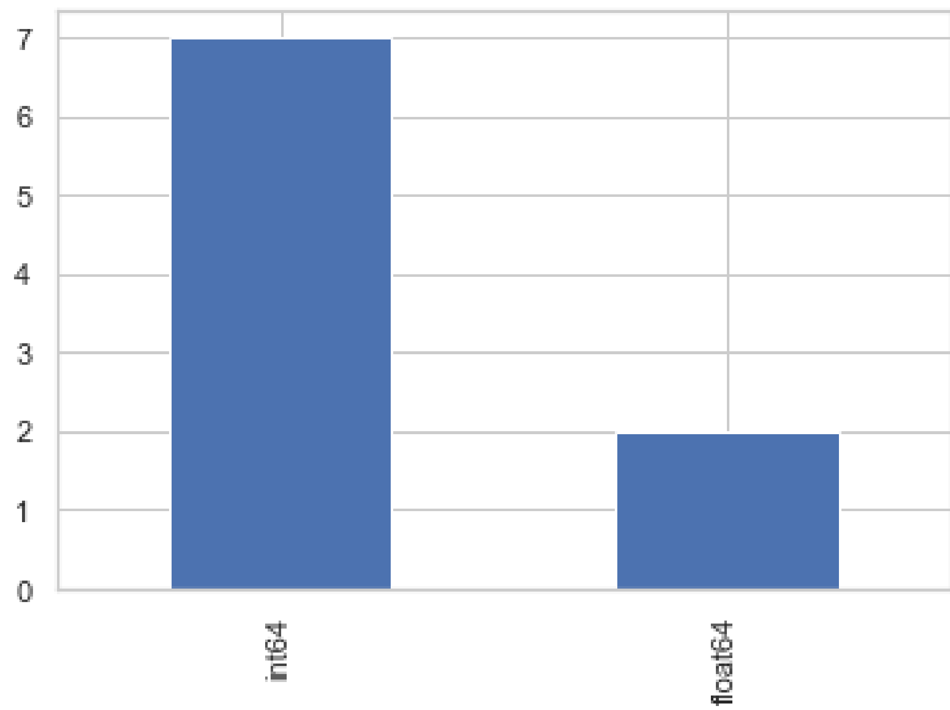
```
print(df2.dtypes)
```

```
Pregnancies                 float64
Glucose                     float64
BloodPressure               float64
SkinThickness               float64
Insulin                     float64
BMI                         float64
```

```
DiabetesPedigreeFunction    float64
Age                         float64
Outcome                     float64
dtype: object
```

3. 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.

```
 (train_df.dtypes).value_counts().plot(kind = 'bar')
plt.show()
```

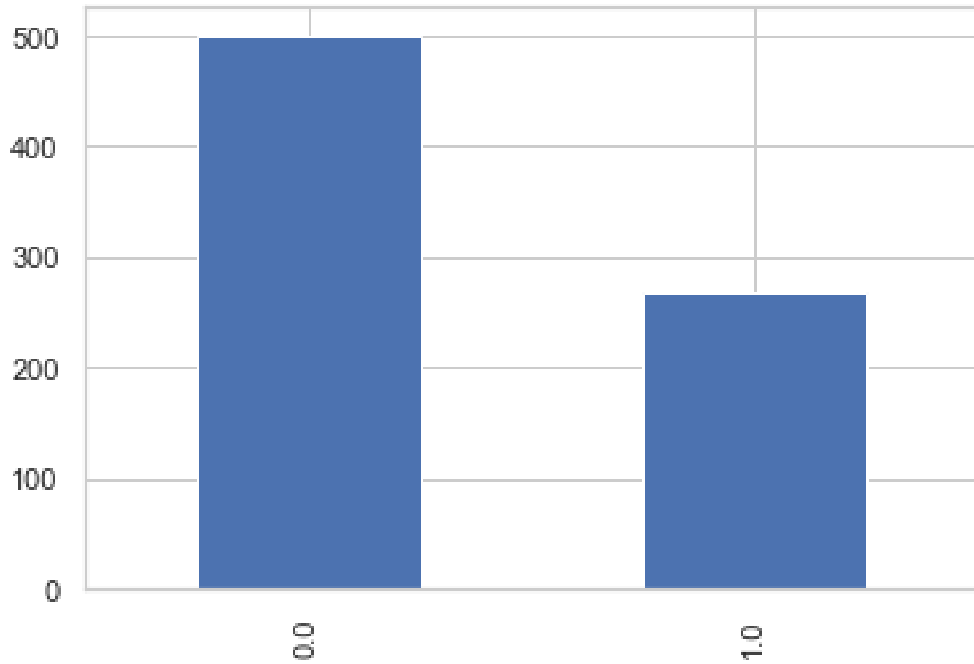


Project Task: Week 2

Data Exploration:

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.

```
(df2.Outcome).value_counts().plot(kind = 'bar')
plt.show()
```




```
df2.Outcome.value_counts()
```

```
0.0    500
1.0    268
Name: Outcome, dtype: int64
```

```
round(df2.Outcome.value_counts(normalize = True)*100, 2)
```

```
0.0    65.1
1.0    34.9
Name: Outcome, dtype: float64
```

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

```
sns.set()
g = sns.pairplot(df2, hue = 'Outcome')
g.map_lower(sns.kdeplot)
g.map_upper(plt.scatter)
g.map_diag(sns.kdeplot)
plt.show()
```

This pairsplot shows the extent and direction of correlation between variables as well as the spread of the data for each pair, distinguishing them by the "Outcome"

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

```
round(df2.corr()['Outcome'][:], 3).sort_values(ascending = False)
```

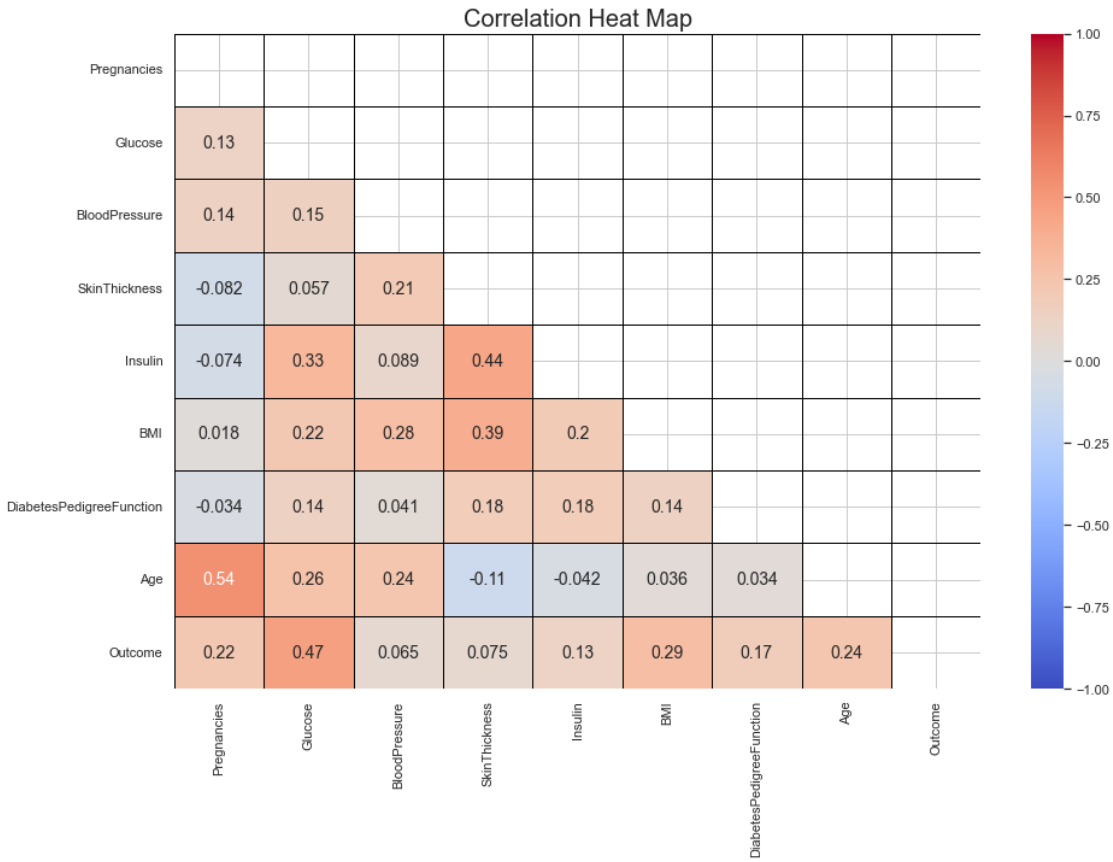
```
Outcome                     1.000
Glucose                     0.467
BMI                         0.293
Age                         0.238
Pregnancies                 0.222
DiabetesPedigreeFunction    0.174
Insulin                     0.131
SkinThickness               0.075
BloodPressure               0.065
Name: Outcome, dtype: float64
```

```
def color_negative_red(value):

#Colors elements in a dateframe
#green if positive and red if
#negative. Does not color NaN
#values.

    if value < -0.1:
        color = 'red'
    elif value > 0.1:
        color = 'green'
    else:
        color = 'white'
    return 'color: %s' % color
```

```
round(df2.corr(), 3).style.applymap(color_negative_red)
```

```
<pandas.io.formats.style.Styler at 0x1bdac040f40>
```

```
sns.set_style("whitegrid")
corr = train_df.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
#kot = corr[corr>=.6]
plt.figure(figsize=(15,10))
sns.heatmap(round(df2.corr(), 3), cmap="coolwarm", vmin=-1,
vmax=1, annot = True, mask = mask, linewidths=1, linecolor='black',
annot_kws={"fontsize":14}).set_title('Correlation Heat Map', fontsize
= 20)
plt.grid('on', )
plt.show()
```



From the HeatMap, we can see that Majority of the correlations are "Positive", but weak. Strongest correlated pairs are "BMI : Skin Thickness", "Age : Pregnancies", "Glucose : Outcome(Target Variable)", "Insulin : Glucose"

Data Modeling

Project Task: Week 3

1) From our EDA visualisations, we realized that: There is missing data, So we Impute the data

2) Since our variables are on different scale and not distributed in Gaussian form, I chose to scale the data using MinMaxScaler, for better performance of the model.

3) Started with Logistic Regression, without sampling the data, but got a poor recall score

4) Also, we know that Our classes are imbalanced, So I have used "SMOTE" sampling to balance the classes, to improve recall score.

5) Then, I chose to try different classifiers, to get better "Recall" score.

```
df2.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   
BMI  \
0          6.0    148.0           72.0           35.0      0.0  33.6   

1          1.0     85.0           66.0           29.0      0.0  26.6   

2          8.0    183.0           64.0            0.0      0.0  23.3   

3          1.0     89.0           66.0           23.0     94.0  28.1   

4          0.0    137.0           40.0           35.0    168.0  43.1   


   DiabetesPedigreeFunction   Age  Outcome  
0                     0.627  50.0      1.0  
1                     0.351  31.0      0.0  
2                     0.672  32.0      1.0  
3                     0.167  21.0      0.0  
4                     2.288  33.0      1.0  
```

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

df2_scaled = scaler.fit_transform(df2)

df2_scaled = pd.DataFrame(df2_scaled, columns=df2.columns)

df2_scaled.head()
```

```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin   
BMI  \
0     0.352941  0.743719       0.590164       0.353535  0.000000  
0.500745   
1     0.058824  0.427136       0.540984       0.292929  0.000000  
0.396423   
2     0.470588  0.919598       0.524590       0.000000  0.000000  
0.347243   
3     0.058824  0.447236       0.540984       0.232323  0.111111  
0.418778   
4     0.000000  0.688442       0.327869       0.353535  0.198582  
0.642325   

   DiabetesPedigreeFunction       Age  Outcome  
0                  0.234415  0.483333      1.0  
1                  0.116567  0.166667      0.0  
2                  0.253629  0.183333      1.0  
3                  0.038002  0.000000      0.0  
4                  0.943638  0.200000      1.0  
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

```python
df2_scaled.head()
```

```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin
BMI  \
0     0.352941  0.743719       0.590164       0.353535  0.000000
0.500745
1     0.058824  0.427136       0.540984       0.292929  0.000000
0.396423
2     0.470588  0.919598       0.524590       0.000000  0.000000
0.347243
3     0.058824  0.447236       0.540984       0.232323  0.111111
0.418778
4     0.000000  0.688442       0.327869       0.353535  0.198582
0.642325

   DiabetesPedigreeFunction       Age  Outcome
0                  0.234415  0.483333      1.0
1                  0.116567  0.166667      0.0
2                  0.253629  0.183333      1.0
3                  0.038002  0.000000      0.0
4                  0.943638  0.200000      1.0
```

```python
y = df2_scaled['Outcome']
x = df2_scaled.drop('Outcome', axis = 1)
```

```python
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size
= 0.2, stratify = y)
```

```python
x_train.head()
```

```
     Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin
BMI  \
174     0.117647  0.376884       0.524590       0.242424  0.065012
0.442623
468     0.470588  0.603015       0.000000       0.000000  0.000000
0.447094
745     0.705882  0.502513       0.688525       0.333333  0.124113
0.447094
700     0.117647  0.613065       0.622951       0.272727  0.236407
0.535022
453     0.117647  0.597990       0.000000       0.000000  0.000000
0.292101

     DiabetesPedigreeFunction       Age
174                  0.124680  0.200000
468                  0.044833  0.283333
745                  0.175064  0.416667
```

```
700                  0.172929  0.083333
453                  0.321947  0.850000
```

```
y_train.head()
```

```
174    0.0
468    1.0
745    0.0
700    0.0
453    0.0
Name: Outcome, dtype: float64
```

```
lr = LogisticRegression()
```

```
lr.fit(x_train, y_train)
```

```
LogisticRegression()
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
 multi_class='auto', n_jobs=None, penalty='l2',
 random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
 warm_start=False)
```

```
LogisticRegression()
```

```
pred = lr.predict(x_test)
```

```
 cnf_matrix = confusion_matrix(y_test, pred)
```

```
 cnf_matrix
```

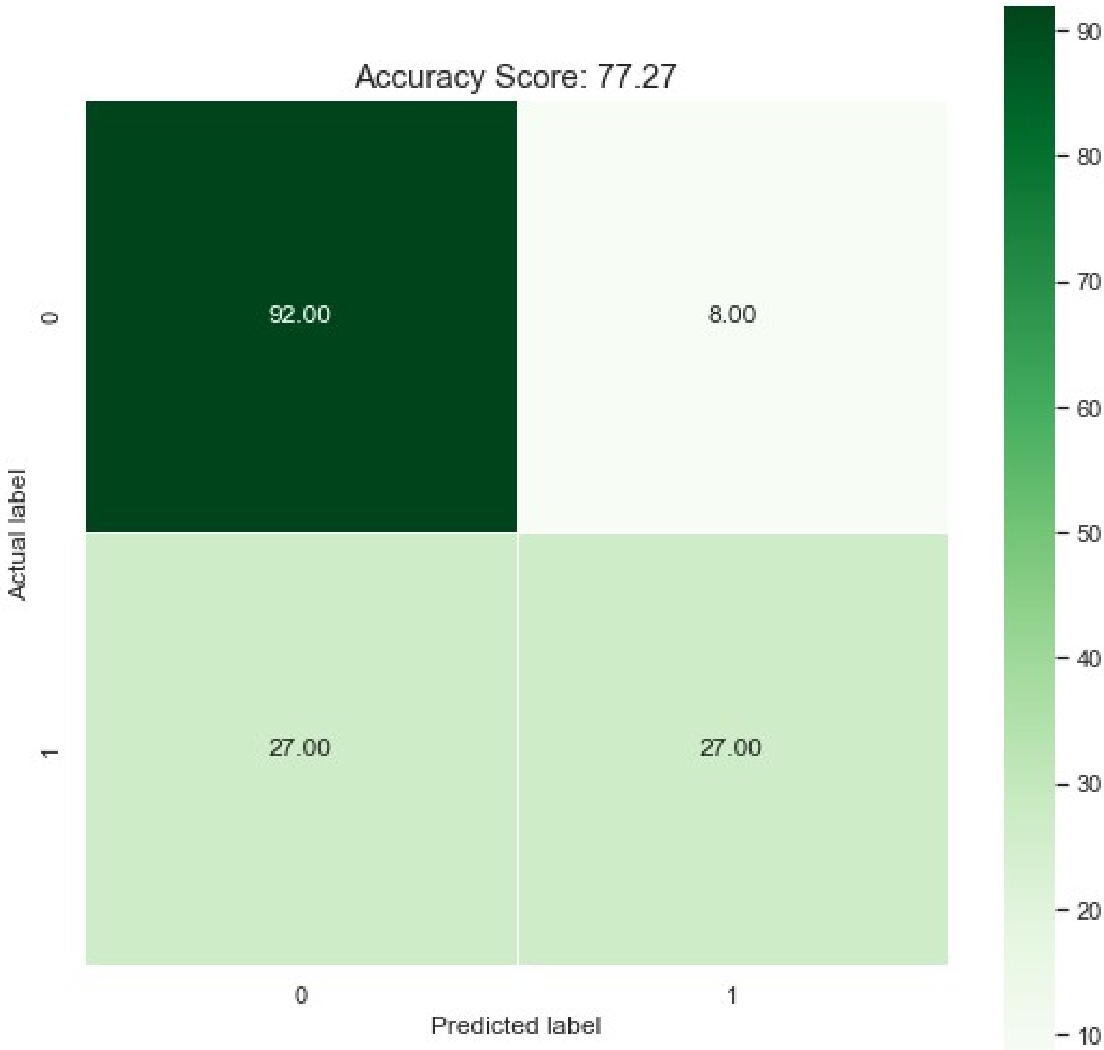
```
array([[92,  8],
       [27, 27]], dtype=int64)
```

```
def sens_spec(cnf_matrix):
 total_cm = sum(sum(cnf_matrix))
 accuracy_clf = (cnf_matrix[0,0] + cnf_matrix[1,1]) / total_cm
 sensitivity_clf = cnf_matrix[0,0] / (cnf_matrix[0, 0] + cnf_matrix[0,
1])
 specificity_clf = cnf_matrix[1,1] / (cnf_matrix[1, 0] + cnf_matrix[1,
1])

 #print('accuracy of {} is {}'.format(accurac)
 return('Accuracy: {}'.format(round(accuracy_clf, 2)), 'Sensitivity:
{}'.format(round(sensitivity_clf, 2)), 'Specificity:
{}'.format(round(specificity_clf, 2)))
```

```
# Use score method to get accuracy of model
score = lr.score(x_test, y_test)
print(score)
```

```
0.7727272727272727
```

```
 plt.figure(figsize=(9,9))
sns.heatmap(cnf_matrix, annot=True, fmt=".2f", linewidths=.5, square =
True,  cmap = "Greens");
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(round(score*100, 2))
plt.title(all_sample_title, size = 15);
```




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

```
('Accuracy: 0.77', 'Sensitivity: 0.92', 'Specificity: 0.5')

              precision    recall  f1-score   support
```

```
         0.0       0.77      0.92      0.84       100
         1.0       0.77      0.50      0.61        54

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

install imbalanced-learn and then run

from imblearn.over_sampling import SMOTE

os = SMOTE(random_state=0)

columns = x_train.columns

os_data_X,os_data_y=os.fit_sample(x, y) os_data_X = pd.DataFrame(data=os_data_X,columns=columns) os_data_y= pd.DataFrame(data=os_data_y,columns=['Outcome'])

we can Check the numbers of our data

print("length of oversampled data is ",len(os_data_X))

print("Number of NEGATIVE in oversampled data",len(os_data_y[os_data_y['Outcome']==0]))

print("Number of POSITIVE",len(os_data_y[os_data_y['Outcome']==1]))

print("Proportion of NEGATIVE data in oversampled data is ",len(os_data_y[os_data_y['Outcome']==0])/len(os_data_X))

print("Proportion of POSITIVE data in oversampled data is ",len(os_data_y[os_data_y['Outcome']==1])/len(os_data_X))

length of oversampled data is 1000

Number of NEGATIVE in oversampled data 500

Number of POSITIVE 500

Proportion of NEGATIVE data in oversampled data is 0.5

Proportion of POSITIVE data in oversampled data is 0.5

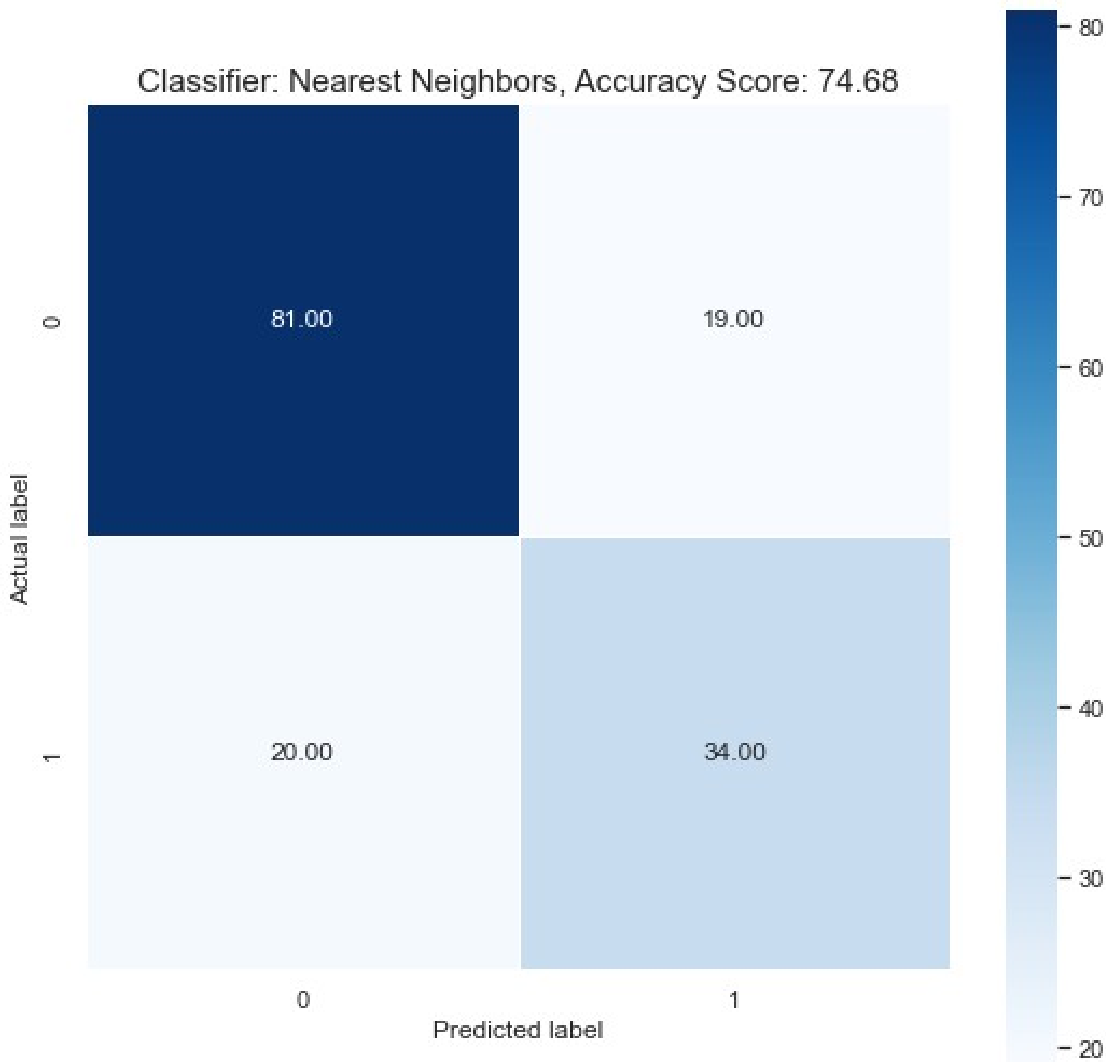
Project Task: Week 4

Data Modeling:

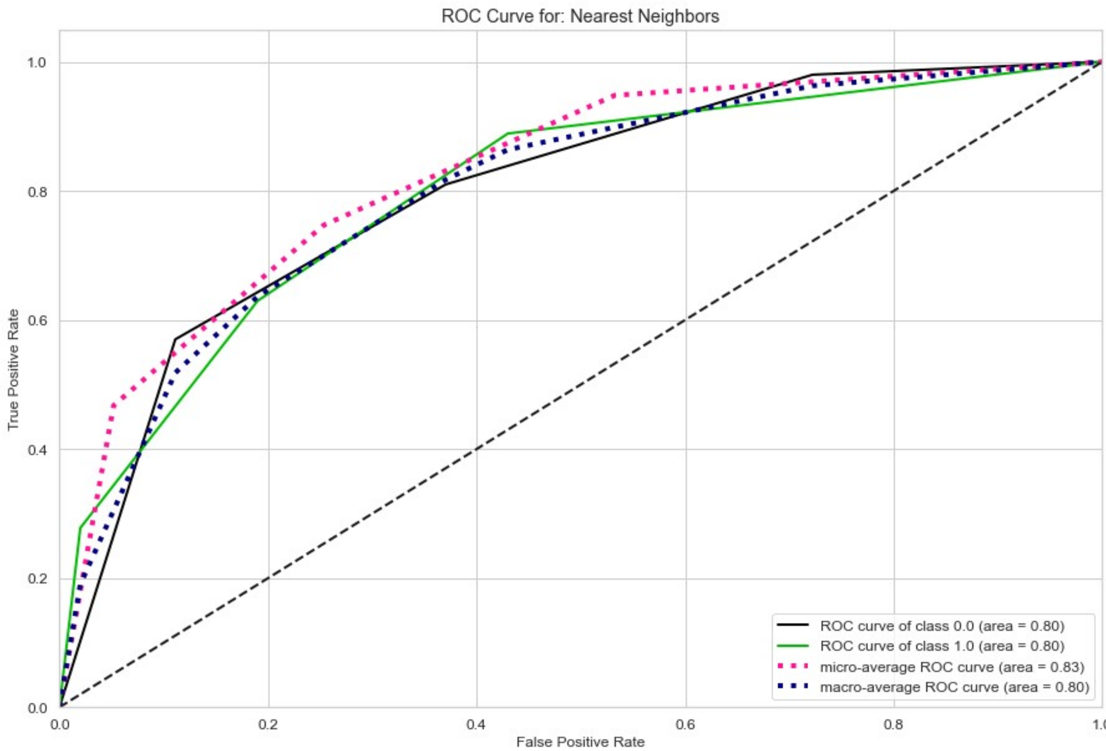
```
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import
QuadraticDiscriminantAnalysis

import scikitplot as skplt

 names = ["Nearest Neighbors", "Logistic Regression", "Linear SVM",
"RBF SVM", "Gaussian Process",
 "Decision Tree", "Random Forest", "Neural Net", "AdaBoost", "Naive
Bayes", "QDA"]

classifiers = [ KNeighborsClassifier(3), LogisticRegression(),
SVC(kernel="linear", C=0.025, probability=True),
 SVC(gamma=2, C=1, probability=True),
 GaussianProcessClassifier(1.0 * RBF(1.0)),
 DecisionTreeClassifier(max_depth=5),
 RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
 MLPClassifier(alpha=1, max_iter=1000),
 AdaBoostClassifier(),
 GaussianNB(),
 QuadraticDiscriminantAnalysis()]

# iterate over classifiers
for name, clf in zip(names, classifiers):

 #ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    print('classifier:', name)
    clf.fit(x_train, y_train)
    score = clf.score(x_test, y_test)
    pred = clf.predict(x_test)
    prob = clf.predict_proba(x_test)
    print(round(score*100, 2))
    print(sens_spec(cnf_matrix))
    print()

    cnf_matrix = confusion_matrix(y_test, pred)
    plt.figure(figsize=(9,9))
    sns.heatmap(cnf_matrix, annot=True, fmt=".2f", linewidths=.5,
square = True, cmap = "Blues");
    plt.ylabel('Actual label');
    plt.xlabel('Predicted label');
 #all_sample_title = 'Accuracy Score: {0}'.format(round(score*100, 2))
    all_sample_title ='Classifier: {}, Accuracy Score:
```

```
{}'.format(name, round(score*100, 2))
    plt.title(all_sample_title, size = 15);
 #print()
    print(classification_report(y_test, pred))
    print()
    skplt.metrics.plot_roc_curve(y_test, prob, figsize = (15, 10),
title = 'ROC Curve for: {}'.format(name))
    plt.show()

print('_______________________________________________________________
______________________')
    print()
```

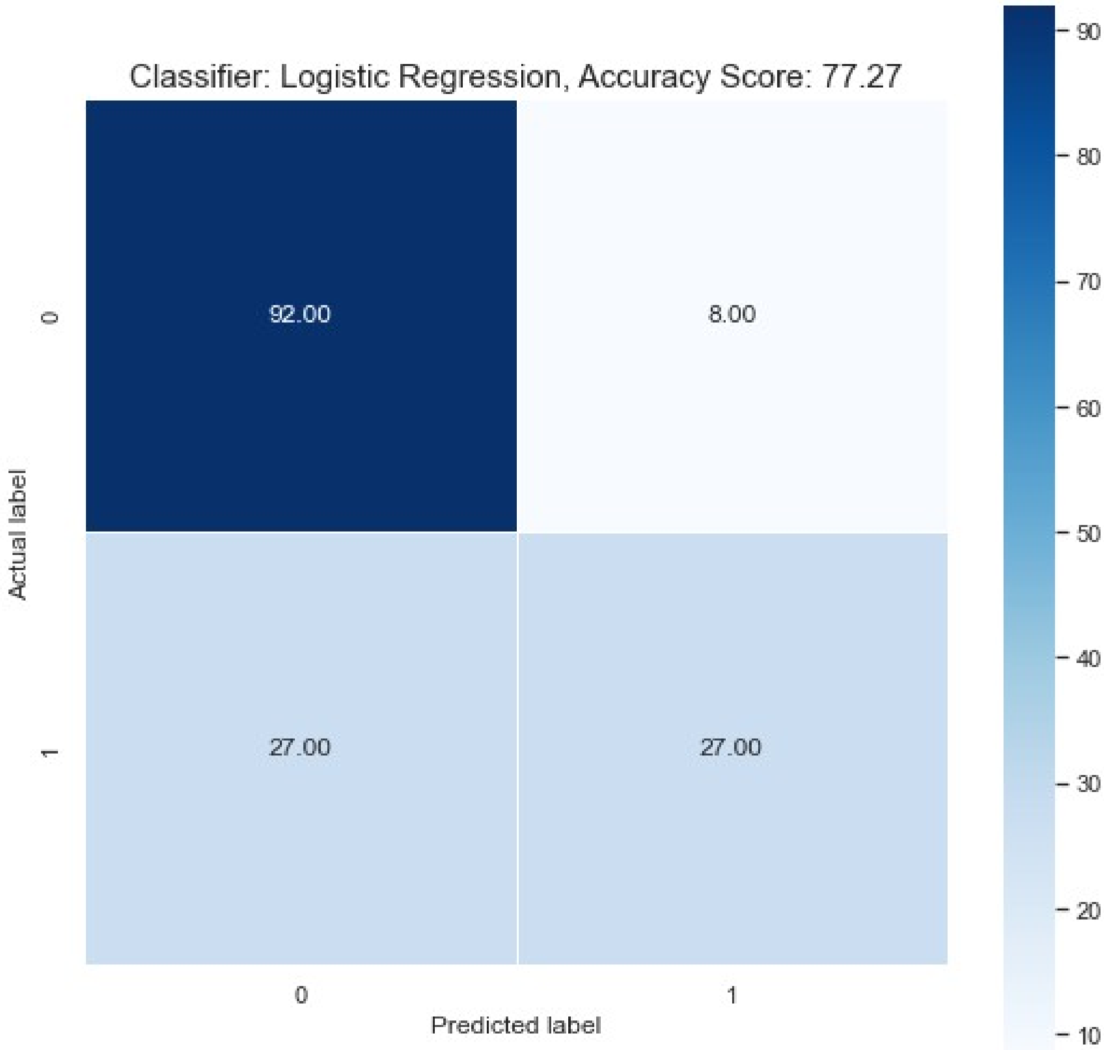
```
classifier: Nearest Neighbors
74.68
('Accuracy: 0.75', 'Sensitivity: 0.81', 'Specificity: 0.63')

              precision    recall  f1-score   support

         0.0       0.80      0.81      0.81       100
         1.0       0.64      0.63      0.64        54

    accuracy                           0.75       154
   macro avg       0.72      0.72      0.72       154
weighted avg       0.75      0.75      0.75       154
```

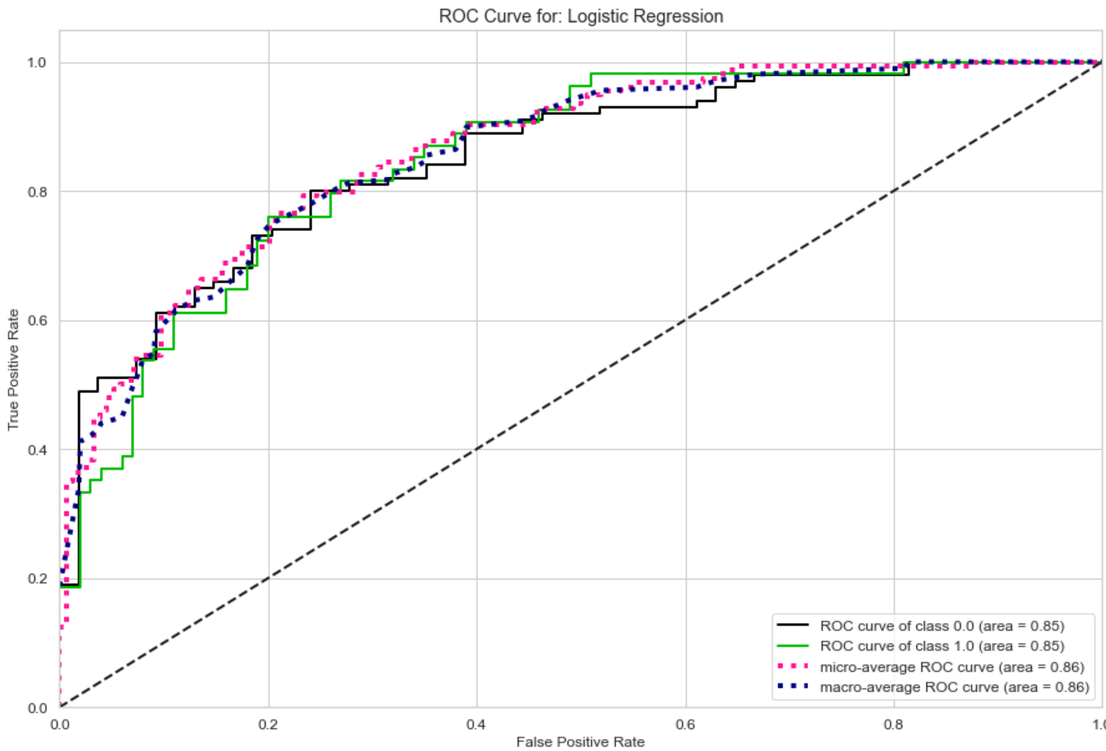


Classifier: Nearest Neighbors, Accuracy Score: 74.68
81.00
19.00
20.00
34.00
Actual label
0
1
Predicted label
0
1
80
70
60
50
40
30
20

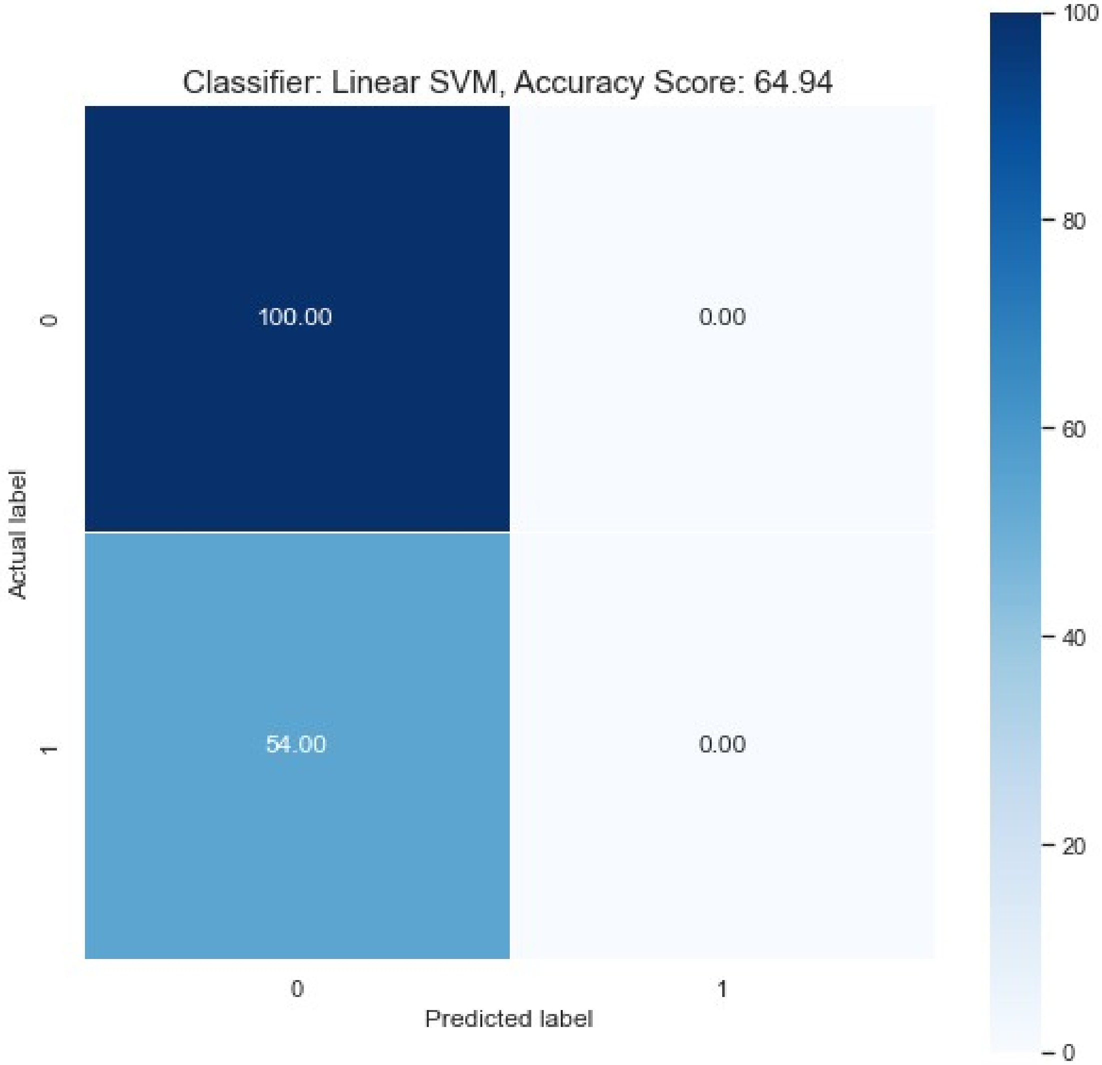


```
classifier: Logistic Regression
77.27
('Accuracy: 0.75', 'Sensitivity: 0.81', 'Specificity: 0.63')

              precision    recall  f1-score   support

         0.0       0.77      0.92      0.84       100
         1.0       0.77      0.50      0.61        54

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

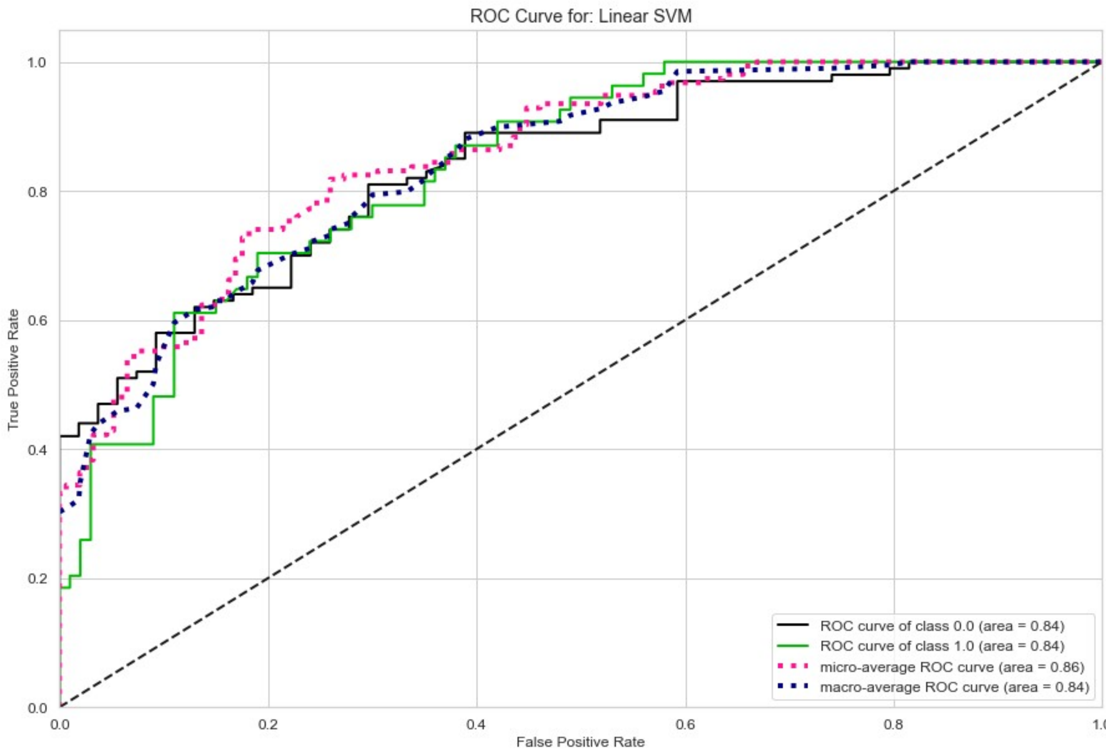

Classifier: Logistic Regression, Accuracy Score: 77.27
92.00
8.00
27.00
27.00
0
1
Actual label
0
1
Predicted label
90
80
70
60
50
40
30
20
10



```
classifier: Linear SVM
64.94
('Accuracy: 0.77', 'Sensitivity: 0.92', 'Specificity: 0.5')

              precision    recall  f1-score   support

         0.0       0.65      1.00      0.79       100
         1.0       0.00      0.00      0.00        54

    accuracy                           0.65       154
   macro avg       0.32      0.50      0.39       154
weighted avg       0.42      0.65      0.51       154
```

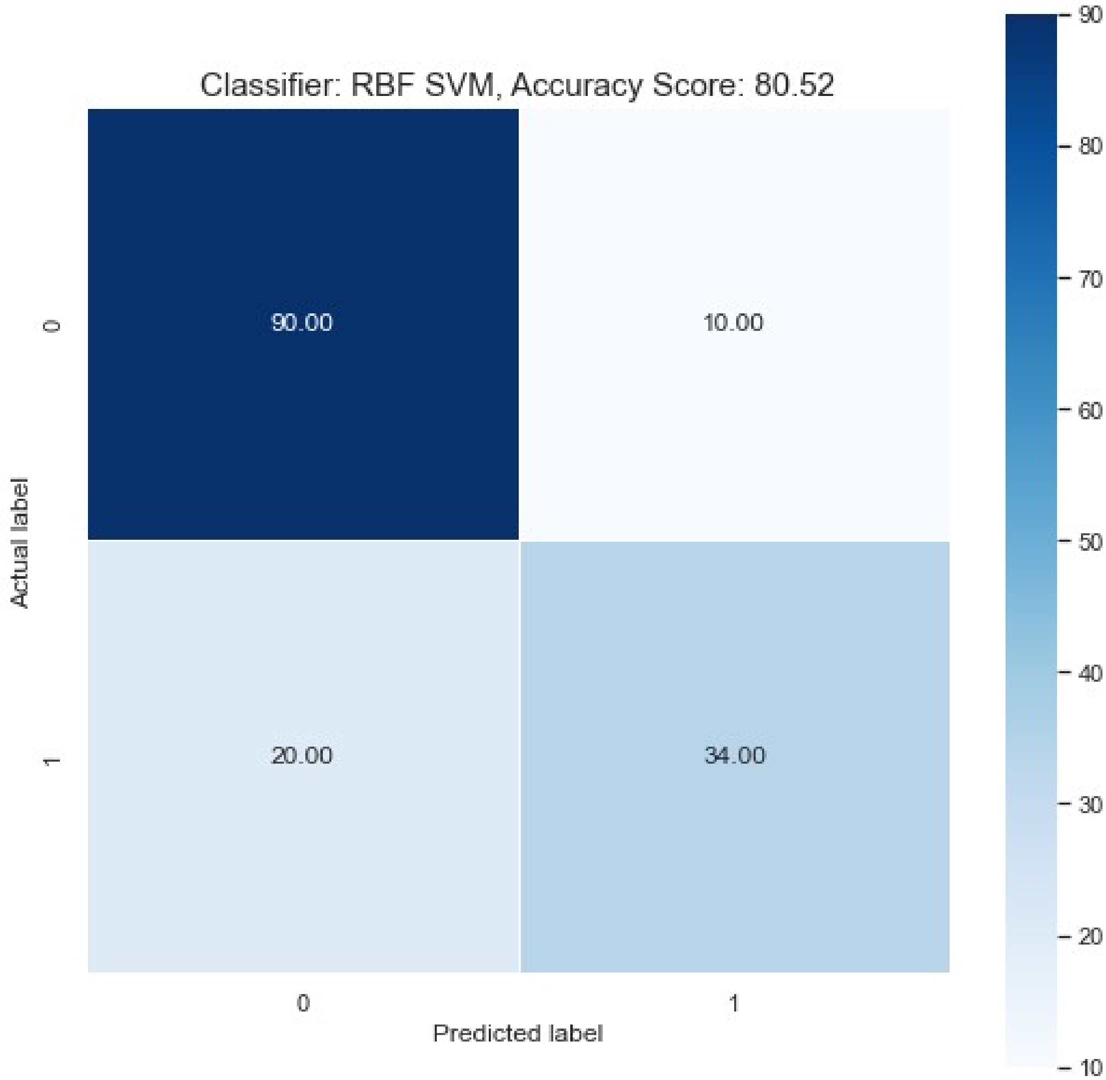


Classifier: Linear SVM, Accuracy Score: 64.94
100.00
0.00
54.00
0.00
0
1
Actual label
0
1
Predicted label
100
80
60
40
20
0

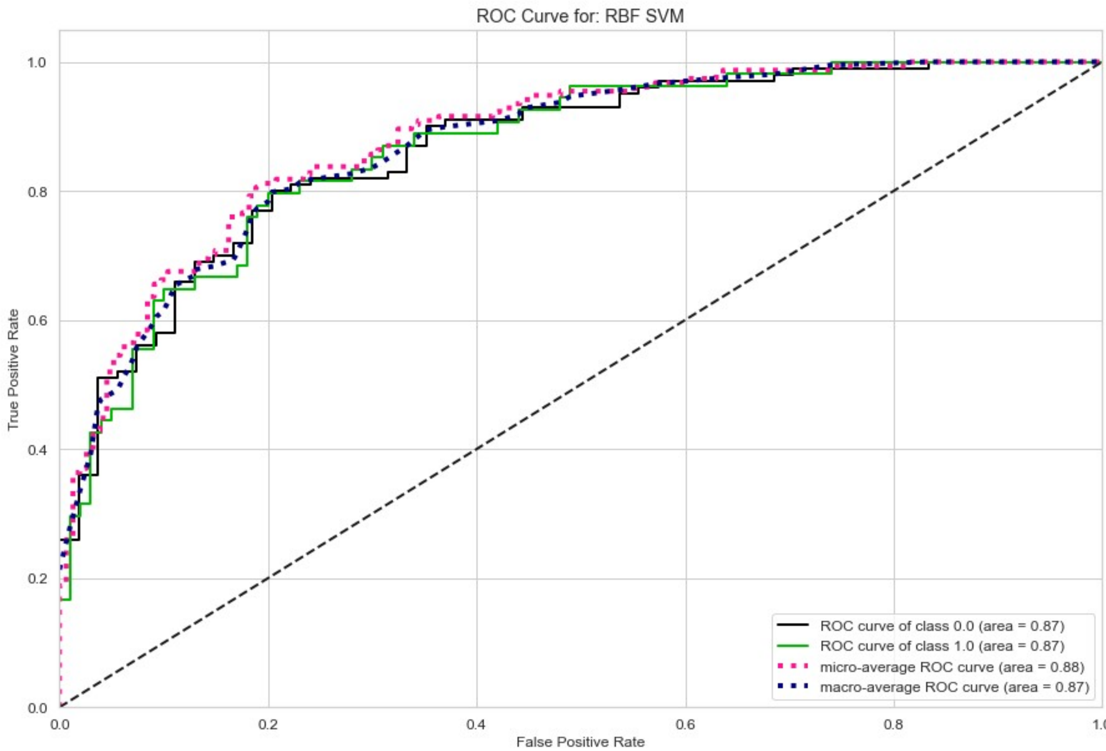


```
classifier: RBF SVM
80.52
('Accuracy: 0.65', 'Sensitivity: 1.0', 'Specificity: 0.0')
```

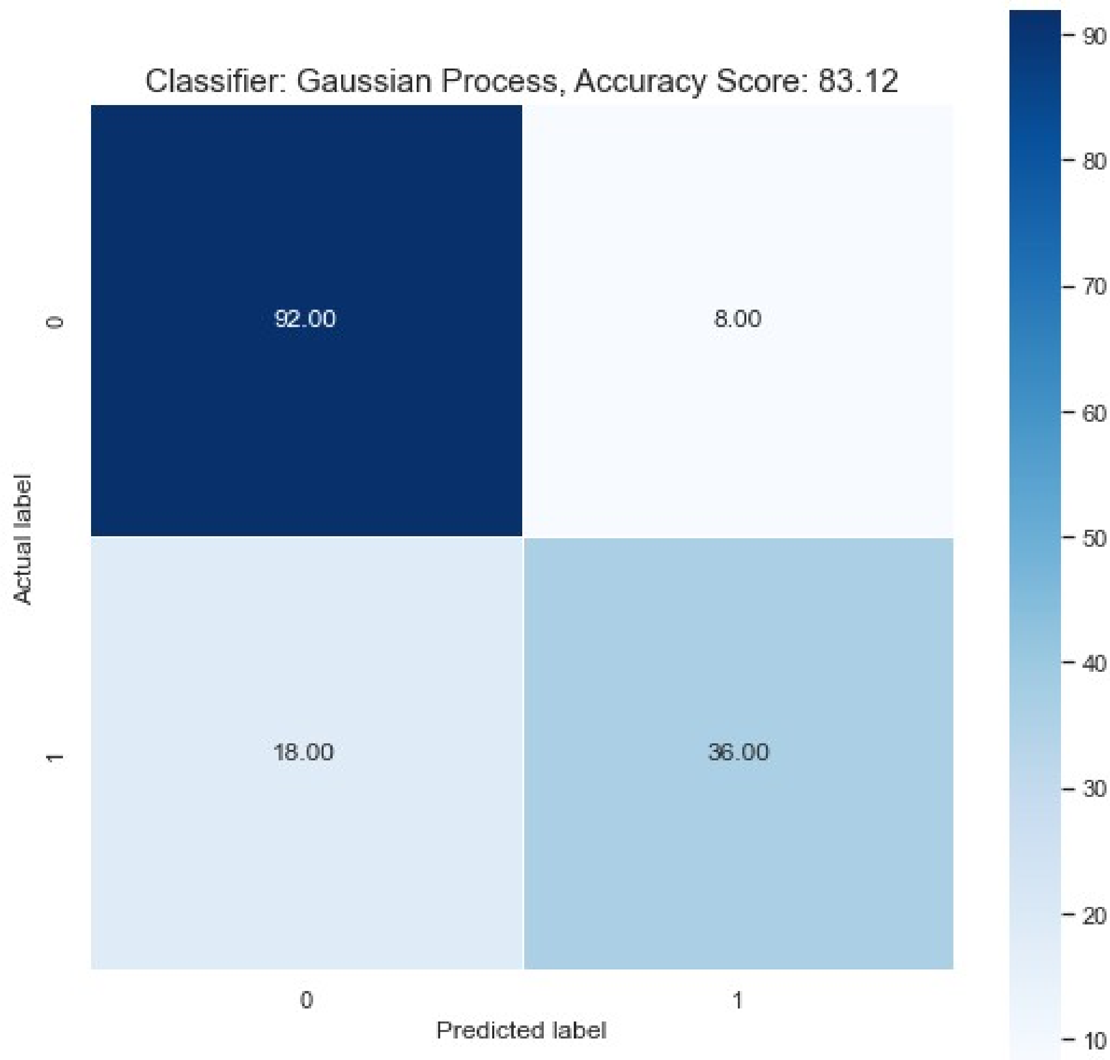
	precision	recall	f1-score	support
0.0	0.82	0.90	0.86	100
1.0	0.77	0.63	0.69	54
accuracy			0.81	154
macro avg	0.80	0.76	0.78	154
weighted avg	0.80	0.81	0.80	154

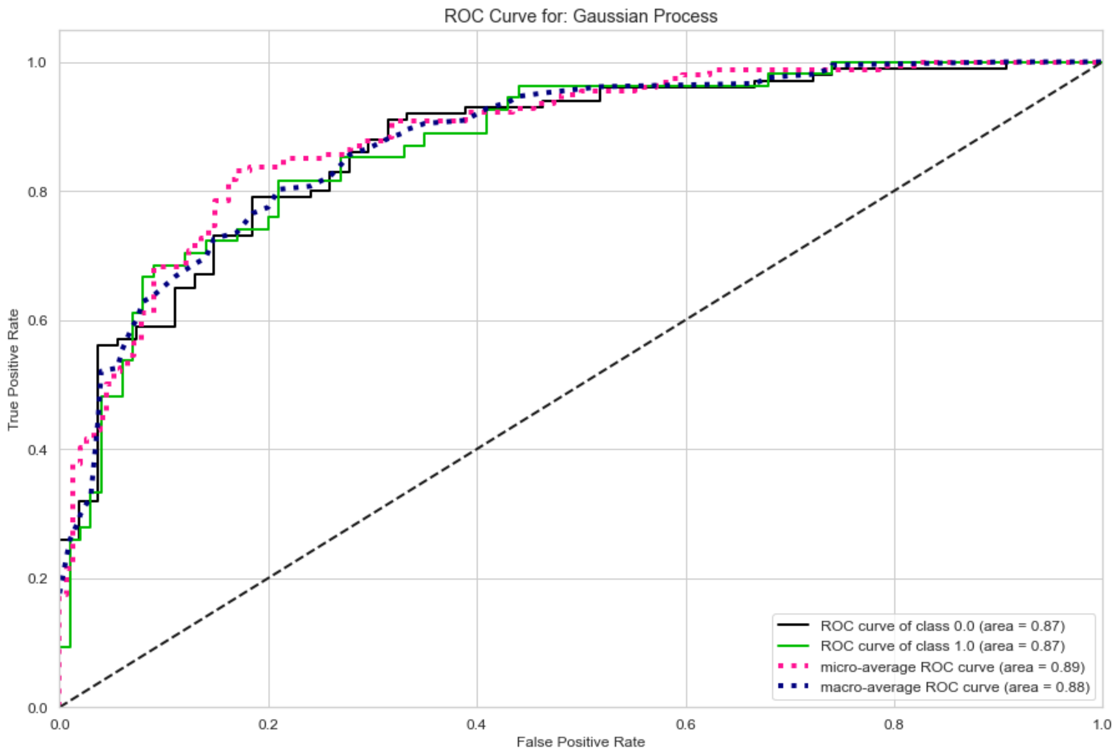


Classifier: RBF SVM, Accuracy Score: 80.52
90.00
10.00
20.00
34.00
Actual label
0
1
Predicted label
0
1
90
80
70
60
50
40
30
20
10

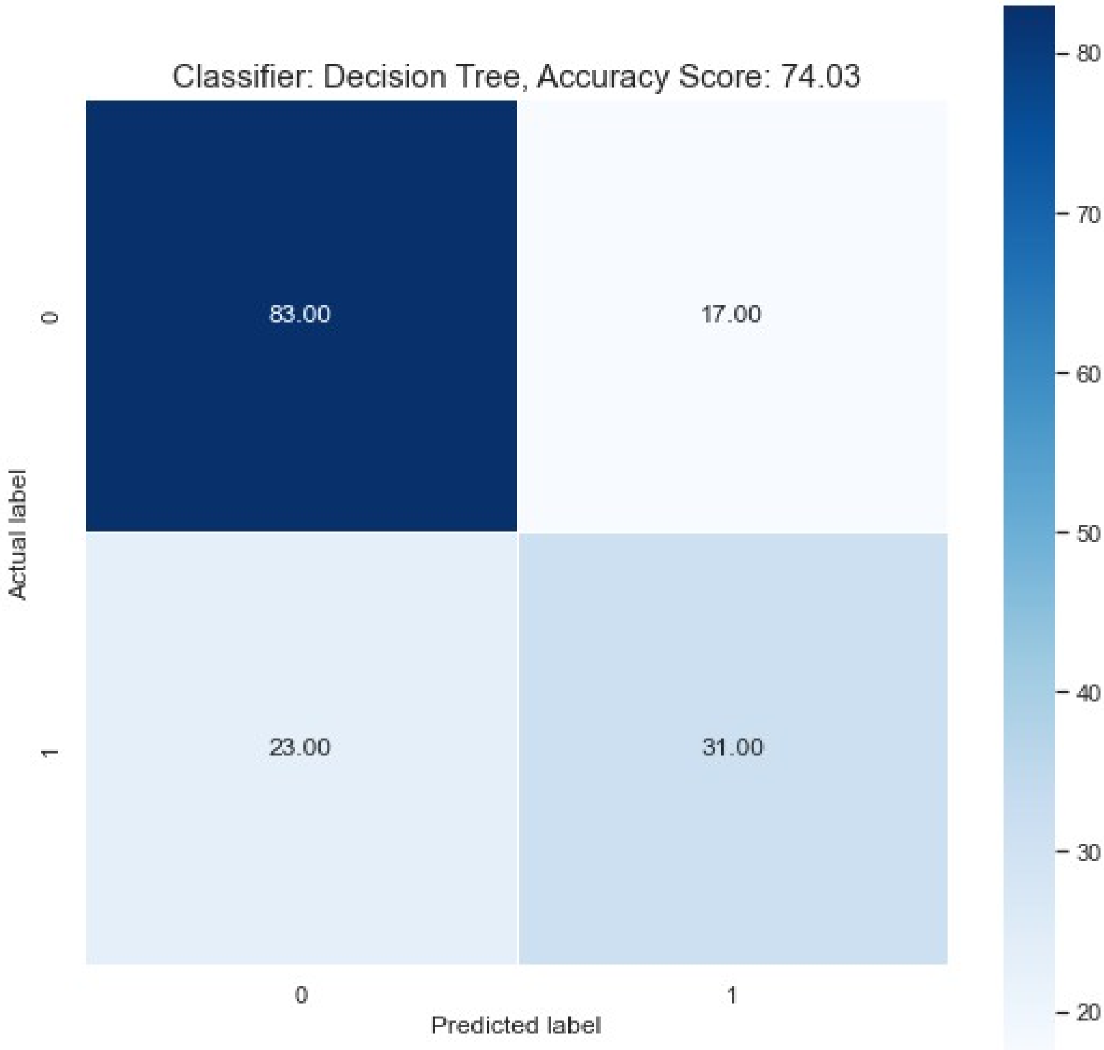


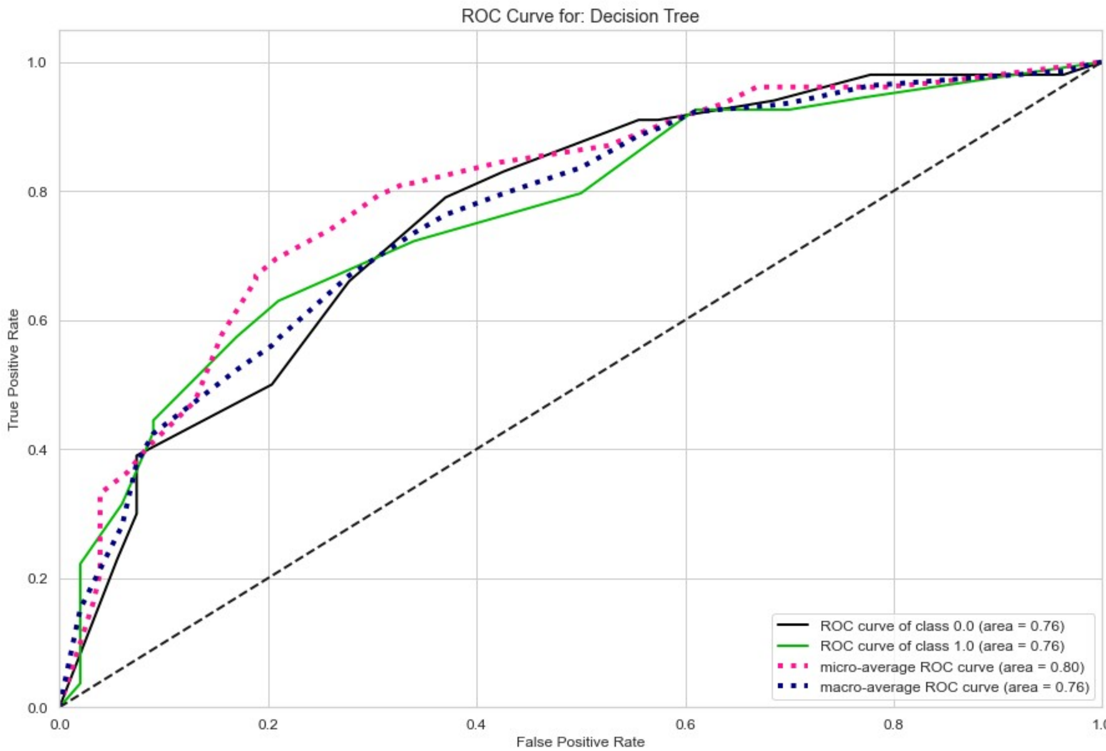
```
classifier: Gaussian Process
83.12
('Accuracy: 0.81', 'Sensitivity: 0.9', 'Specificity: 0.63')

              precision    recall  f1-score   support

         0.0       0.84      0.92      0.88       100
         1.0       0.82      0.67      0.73        54

    accuracy                           0.83       154
   macro avg       0.83      0.79      0.81       154
weighted avg       0.83      0.83      0.83       154
```


Classifier: Gaussian Process, Accuracy Score: 83.12
92.00
8.00
18.00
36.00
Actual label
0
1
Predicted label
0
1
90
80
70
60
50
40
30
20
10

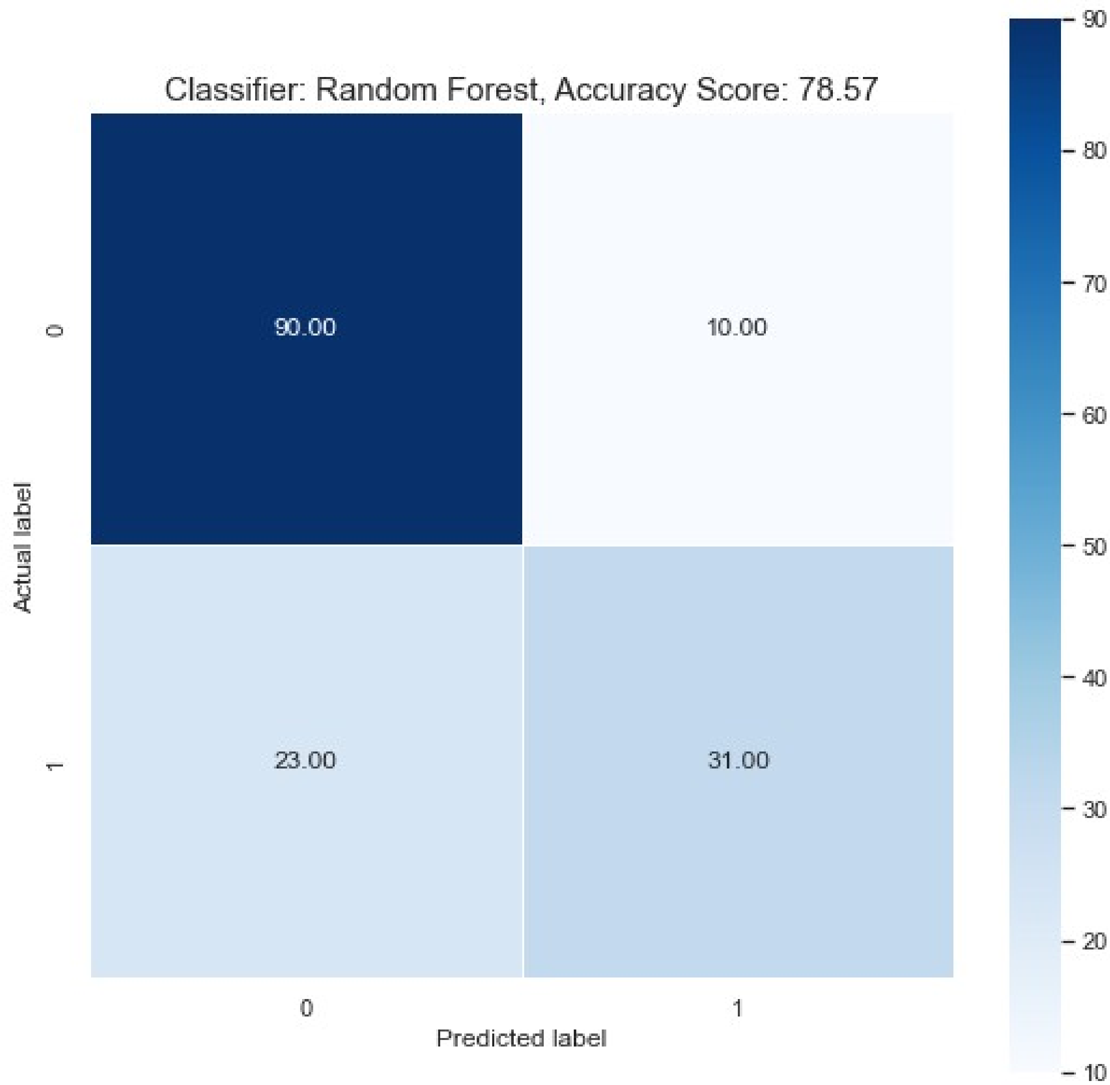


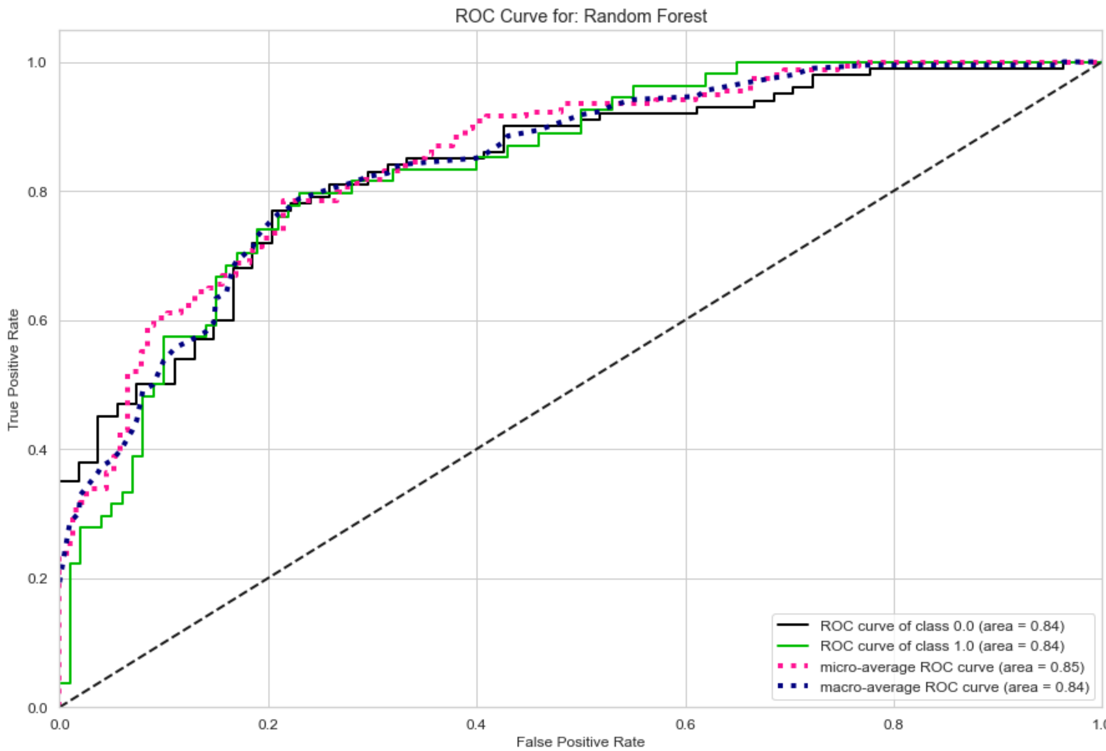
```
classifier: Decision Tree
74.03
('Accuracy: 0.83', 'Sensitivity: 0.92', 'Specificity: 0.67')

              precision    recall  f1-score   support

         0.0       0.78      0.83      0.81       100
         1.0       0.65      0.57      0.61        54

    accuracy                           0.74       154
   macro avg       0.71      0.70      0.71       154
weighted avg       0.73      0.74      0.74       154
```


Classifier: Decision Tree, Accuracy Score: 74.03
83.00
17.00
23.00
31.00
Actual label
0
1
Predicted label
0
1
80
70
60
50
40
30
20

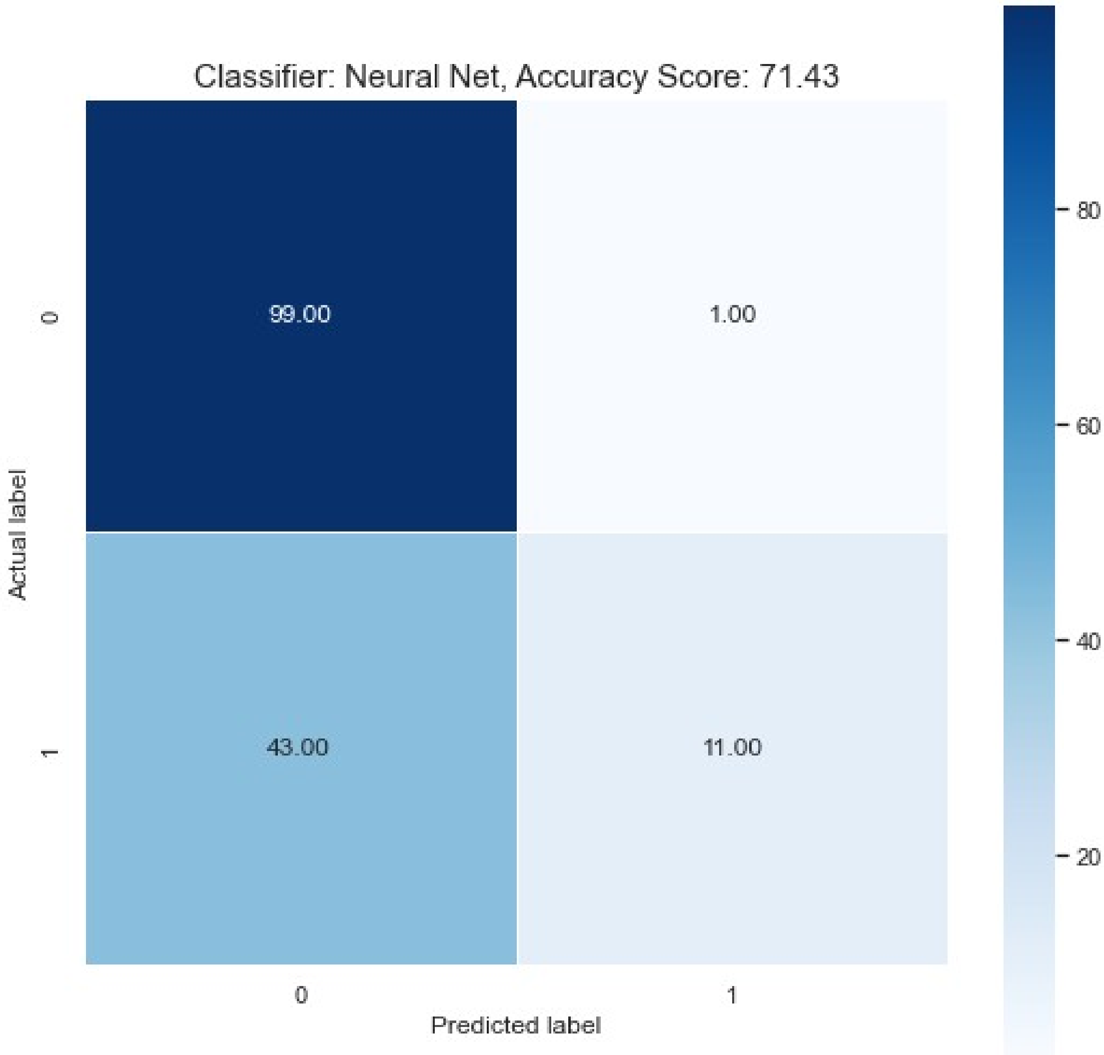


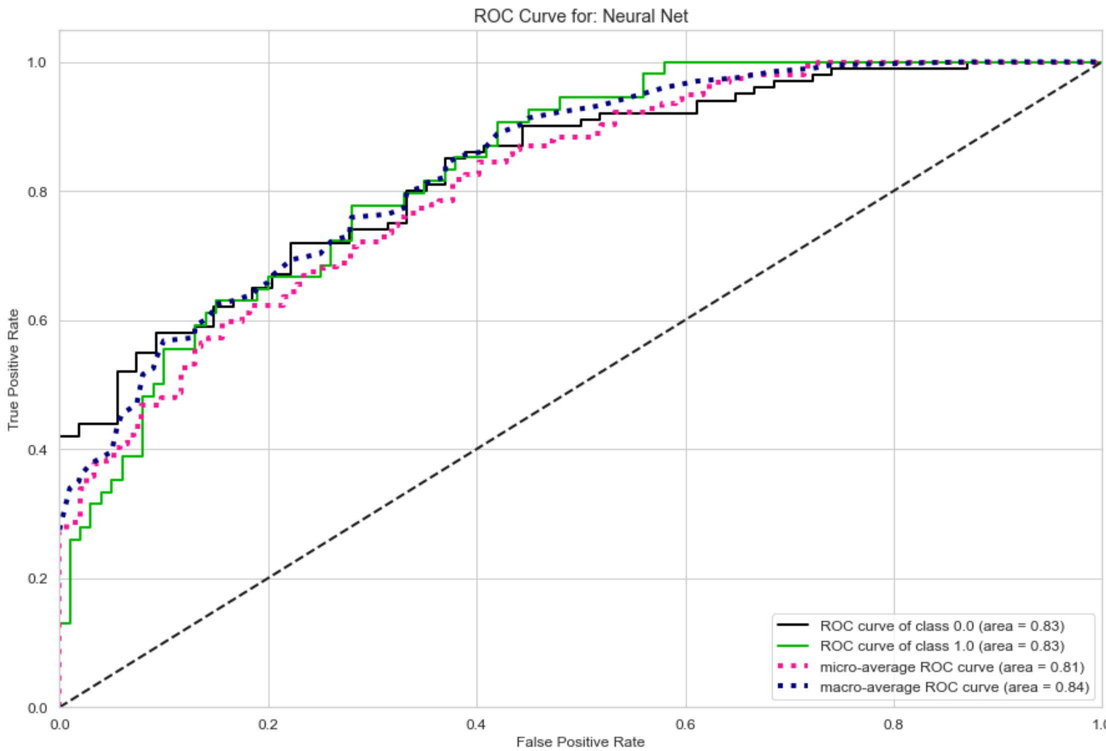
```
classifier: Random Forest
78.57
('Accuracy: 0.74', 'Sensitivity: 0.83', 'Specificity: 0.57')

              precision    recall  f1-score   support

         0.0       0.80      0.90      0.85       100
         1.0       0.76      0.57      0.65        54

    accuracy                           0.79       154
   macro avg       0.78      0.74      0.75       154
weighted avg       0.78      0.79      0.78       154
```


Classifier: Random Forest, Accuracy Score: 78.57
90.00
10.00
23.00
31.00
0
1
Actual label
0
1
Predicted label
90
80
70
60
50
40
30
20
10

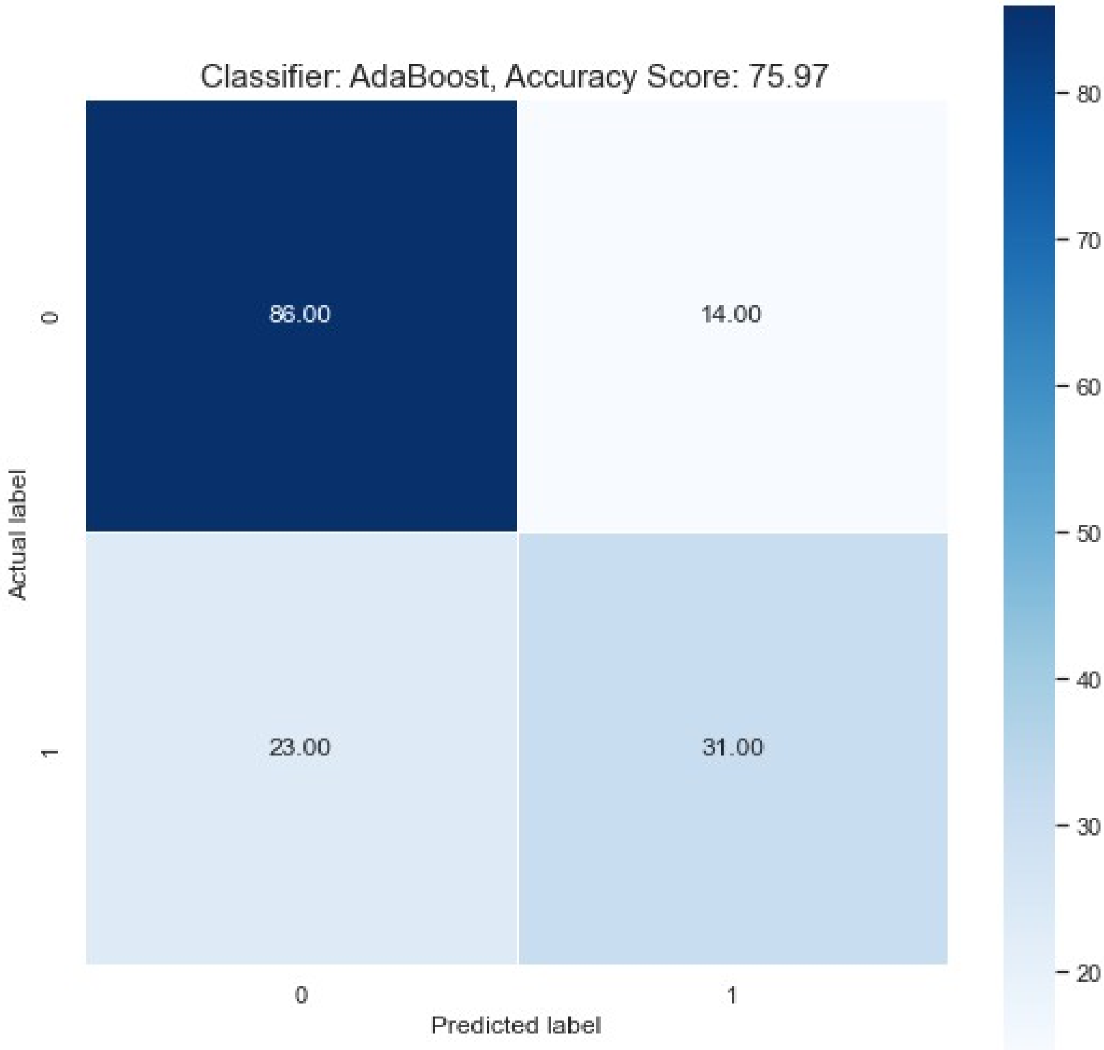


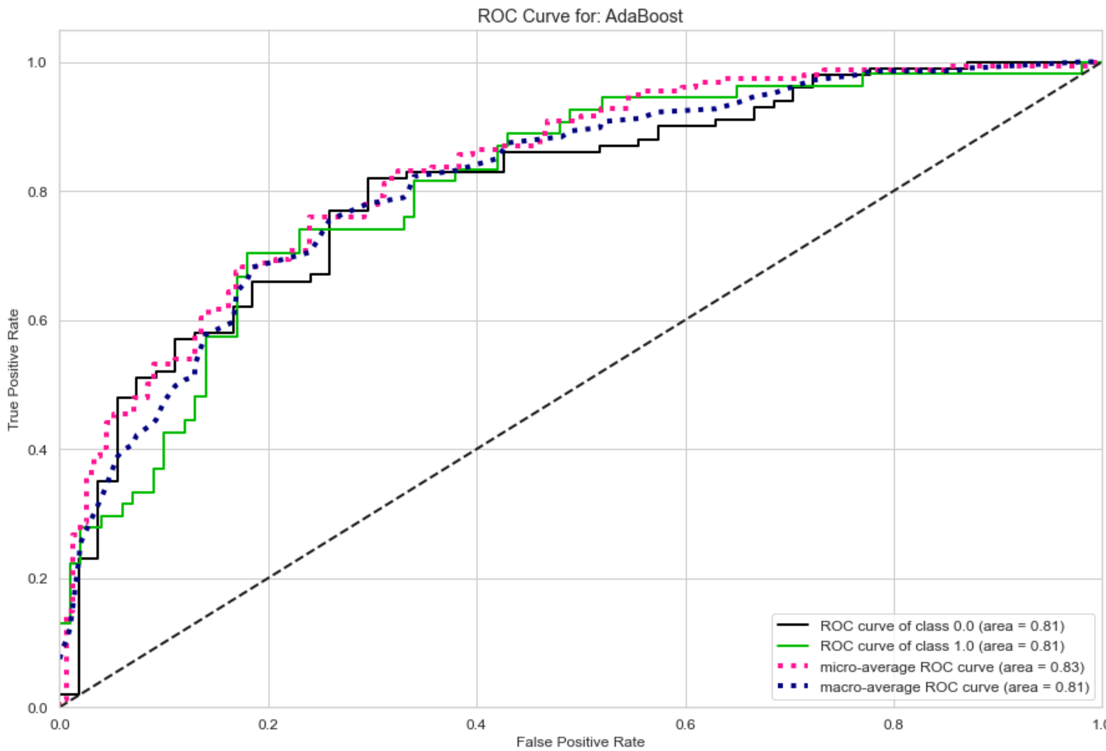
```
classifier: Neural Net
71.43
('Accuracy: 0.79', 'Sensitivity: 0.9', 'Specificity: 0.57')

              precision    recall  f1-score   support

         0.0       0.70      0.99      0.82       100
         1.0       0.92      0.20      0.33        54

    accuracy                           0.71       154
   macro avg       0.81      0.60      0.58       154
weighted avg       0.77      0.71      0.65       154
```


Classifier: Neural Net, Accuracy Score: 71.43
99.00
1.00
43.00
11.00
0
1
Actual label
0
1
Predicted label
80
60
40
20

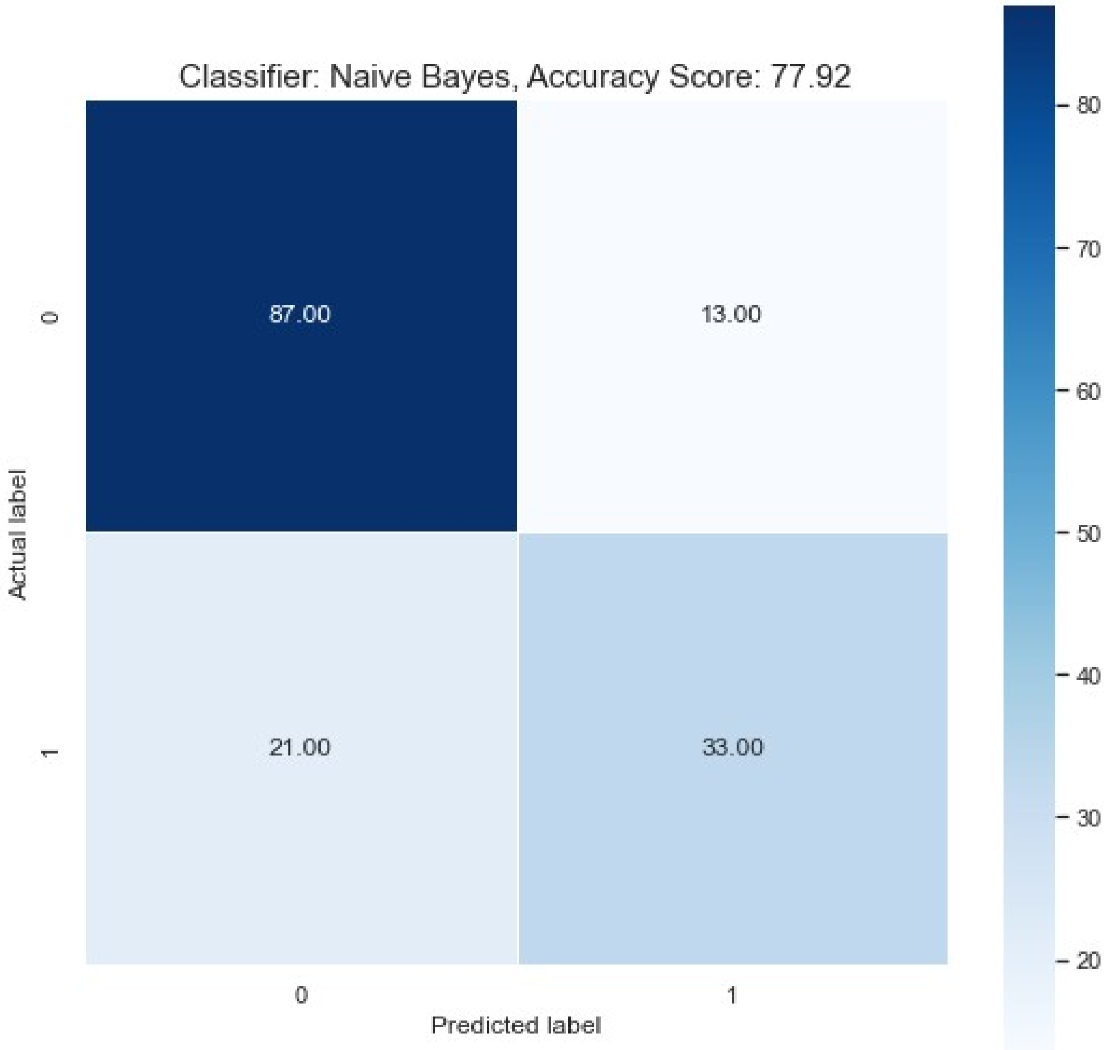


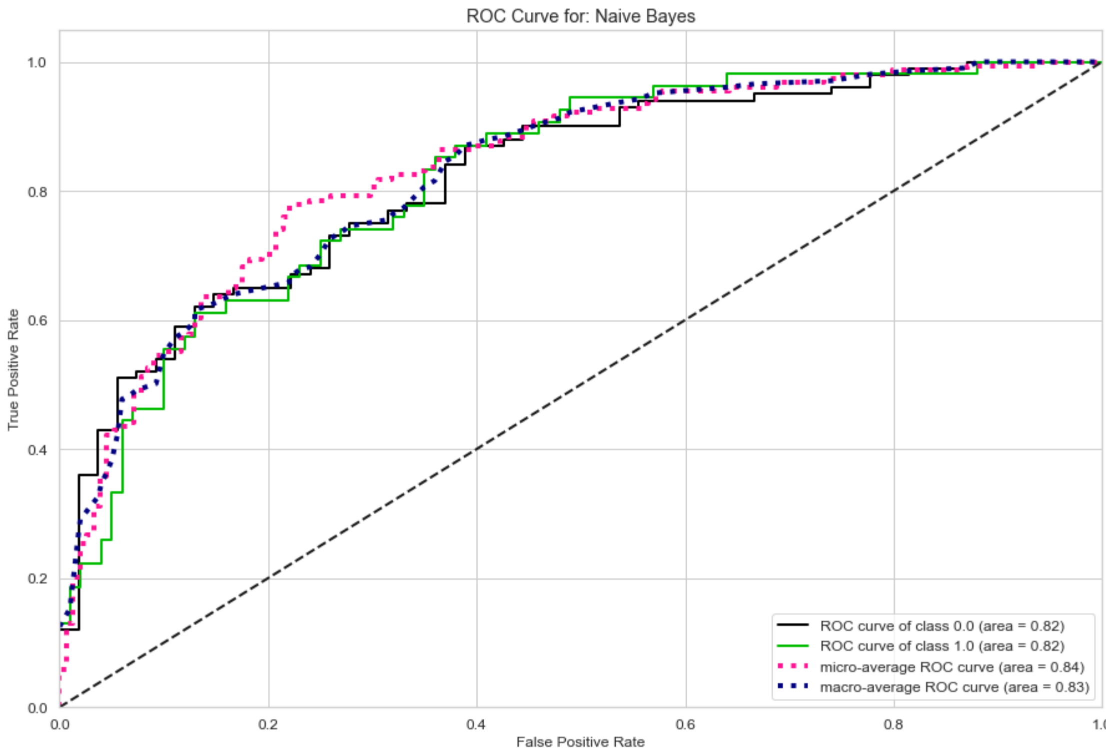
```
classifier: AdaBoost
75.97
('Accuracy: 0.71', 'Sensitivity: 0.99', 'Specificity: 0.2')

              precision    recall  f1-score   support

         0.0       0.79      0.86      0.82       100
         1.0       0.69      0.57      0.63        54

    accuracy                           0.76       154
   macro avg       0.74      0.72      0.72       154
weighted avg       0.75      0.76      0.75       154
```


Classifier: AdaBoost, Accuracy Score: 75.97
86.00
14.00
23.00
31.00
Actual label
0
1
Predicted label
0
1
80
70
60
50
40
30
20



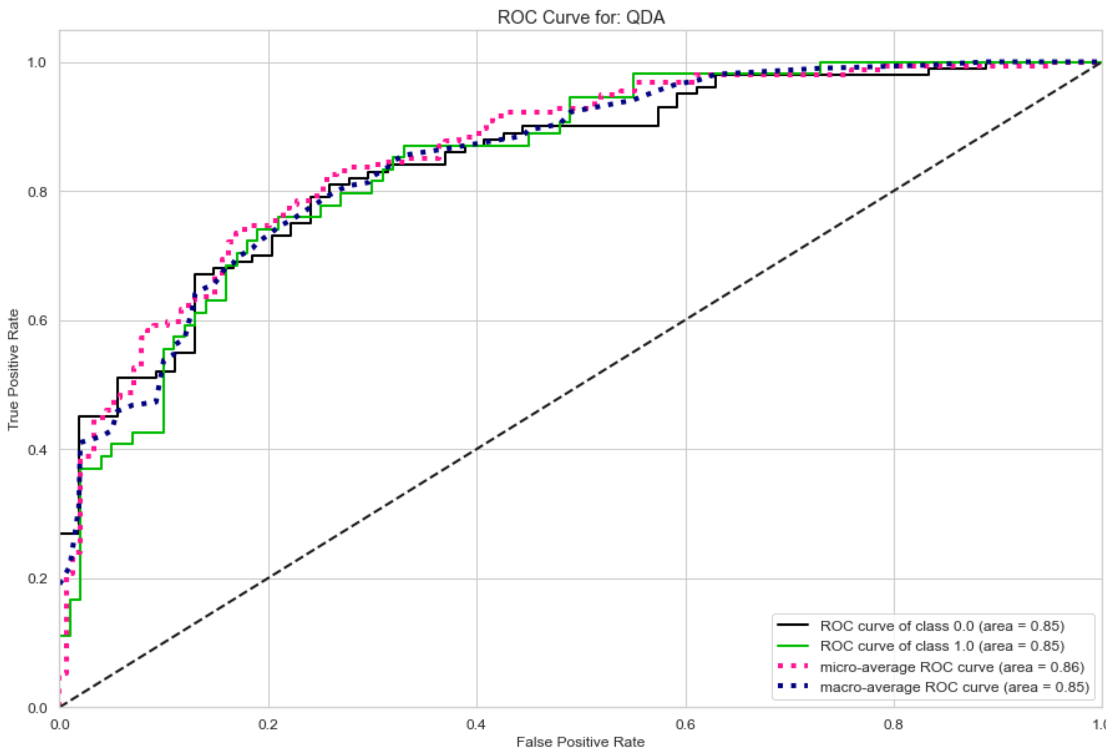
```
classifier: Naive Bayes
77.92
('Accuracy: 0.76', 'Sensitivity: 0.86', 'Specificity: 0.57')

              precision    recall  f1-score   support

         0.0       0.81      0.87      0.84       100
         1.0       0.72      0.61      0.66        54

    accuracy                           0.78       154
   macro avg       0.76      0.74      0.75       154
weighted avg       0.77      0.78      0.77       154
```


Classifier: Naive Bayes, Accuracy Score: 77.92
87.00
13.00
21.00
33.00
Actual label
0
1
Predicted label
0
1
80
70
60
50
40
30
20



```
classifier: QDA
77.92
('Accuracy: 0.78', 'Sensitivity: 0.87', 'Specificity: 0.61')

              precision    recall  f1-score   support

         0.0       0.81      0.87      0.84       100
         1.0       0.72      0.61      0.66        54

    accuracy                           0.78       154
   macro avg       0.76      0.74      0.75       154
weighted avg       0.77      0.78      0.77       154
```

Classifier: QDA, Accuracy Score: 77.92
87.00
13.00
21.00
33.00
0
1
Actual label
0
1
Predicted label
80
70
60
50
40
30
20



Let's Check how close our algorithm is predicting, by passing the inputs from our test set and compare them to the target values.

```
#import random
np.random.seed(1000)
randomlist = []
for i in range(0,10):
    n = np.random.randint(1,len(x_test))
    randomlist.append(n)
print(randomlist)
```

```
[88, 72, 95, 93, 2, 129, 90, 46, 41, 106]
```

```
list(x_test.iloc[88])
```

```
[0.29411764705882354,
 0.6080402010050251,
 0.5901639344262295,
 0.23232323232323235,
 0.13238770685579196,
 0.3904619970193741,
 0.0713065755764304,
 0.15000000000000002]
```

```
 pre_out = []
out = []
for i in randomlist:
    data_in = [list(x_test.iloc[i])]
    data_in = np.around(data_in, 2)
    pre_data_out = lr.predict(data_in)
    #data_out = y_test.iloc[i]['Outcome']

    mylist = [i, data_in, pre_data_out] # data_out
    print(*mylist,sep='\n')
    print('------------------------')
    pre_out.append(pre_data_out)
#out.append(data_out)
```

```
88
[[0.29 0.61 0.59 0.23 0.13 0.39 0.07 0.15]]
[0.]
------------------------
72
[[0.12 0.44 0.61 0.19 0.06 0.43 0.06 0.02]]
[0.]
------------------------
95
[[0.59 0.54 0.54 0.   0.   0.48 0.08 0.35]]
[0.]
------------------------
93
[[0.06 0.82 0.67 0.43 0.08 0.49 0.11 0.48]]
[1.]
------------------------
2
[[0.   0.59 0.52 0.23 0.11 0.   0.71 0.  ]]
[0.]
------------------------
129
[[0.12 0.72 0.48 0.33 0.16 0.47 0.15 0.07]]
[0.]
------------------------
90
[[0.71 0.46 0.51 0.07 0.3  0.41 0.36 0.38]]
[0.]
------------------------
46
[[0.06 0.72 0.67 0.46 0.21 0.69 0.11 0.42]]
[1.]
------------------------
41
[[0.59 0.58 0.   0.   0.   0.53 0.02 0.13]]
[0.]
```

```
------------------------
106
[[0.06 0.53 0.48 0.   0.   0.36 0.05 0.  ]]
[0.]
------------------------
```

```python
 svc = SVC(gamma=2, C=1, probability=True)
svc.fit(x_train, y_train)
```

```
SVC(C=1, gamma=2, probability=True)
```

```python
pre_out = []
out = []
for i in randomlist:
    data_in = [list(x_test.iloc[i])]
    data_in = np.around(data_in, 2)
    pre_data_out = svc.predict(data_in)
    #data_out = y_test.iloc[i]['Outcome']

    mylist = [i, data_in, pre_data_out] #data_out]
    print(*mylist,sep='\n')
    print('------------------------')

    pre_out.append(pre_data_out)
    #out.append(data_out)
```

```
88
[[0.29 0.61 0.59 0.23 0.13 0.39 0.07 0.15]]
[0.]
------------------------
72
[[0.12 0.44 0.61 0.19 0.06 0.43 0.06 0.02]]
[0.]
------------------------
95
[[0.59 0.54 0.54 0.   0.   0.48 0.08 0.35]]
[0.]
------------------------
93
[[0.06 0.82 0.67 0.43 0.08 0.49 0.11 0.48]]
[1.]
------------------------
2
[[0.   0.59 0.52 0.23 0.11 0.   0.71 0.  ]]
[0.]
------------------------
129
[[0.12 0.72 0.48 0.33 0.16 0.47 0.15 0.07]]
[0.]
------------------------
90
```

```
[[0.71 0.46 0.51 0.07 0.3  0.41 0.36 0.38]]
[0.]
------------------------
46
[[0.06 0.72 0.67 0.46 0.21 0.69 0.11 0.42]]
[1.]
------------------------
41
[[0.59 0.58 0.   0.   0.   0.53 0.02 0.13]]
[1.]
------------------------
106
[[0.06 0.53 0.48 0.   0.   0.36 0.05 0.  ]]
[0.]
------------------------
```