Diabetes Detection Using Logistic Regression

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import Ridge,Lasso,RidgeCV, LassoCV, ElasticNet, ElasticNetCV, L
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
import scikitplot as skl
sns.set()
```

In [2]:

```
data=pd.read_csv('E:\\Machine Learning\\Logistic-regression_final\\diabetes.csv')
data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [3]:

```
data.describe()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabet
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

There are no missing values

In [4]:

```
df=data.copy()
df.head()
```

Out[4]:

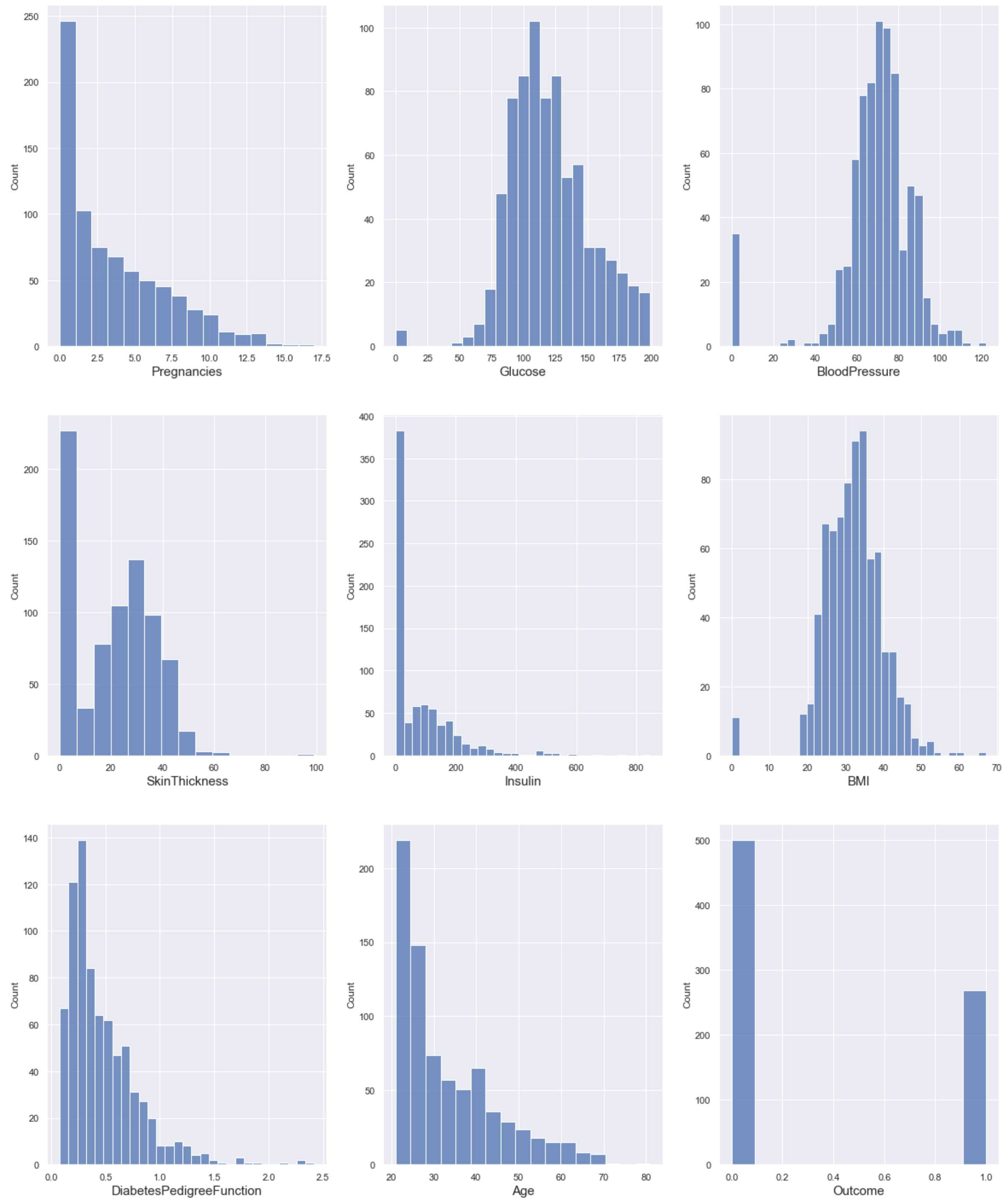
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [5]:

```
#Now let us look at how the data is distributed for every column

plt.figure(figsize=(20,25),facecolor='white')
plotnumber=1

for column in df:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.histplot(df[column])
        plt.xlabel(column,fontsize=15)
    plotnumber+=1
plt.show()
```



We can see there is some skewness in the data, let's deal with data.

Also, we can see there few data for columns Glucose, Insulin, skin thickness, BMI and Blood Pressure which have value as 0. That's not possible. You can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

In [6]:

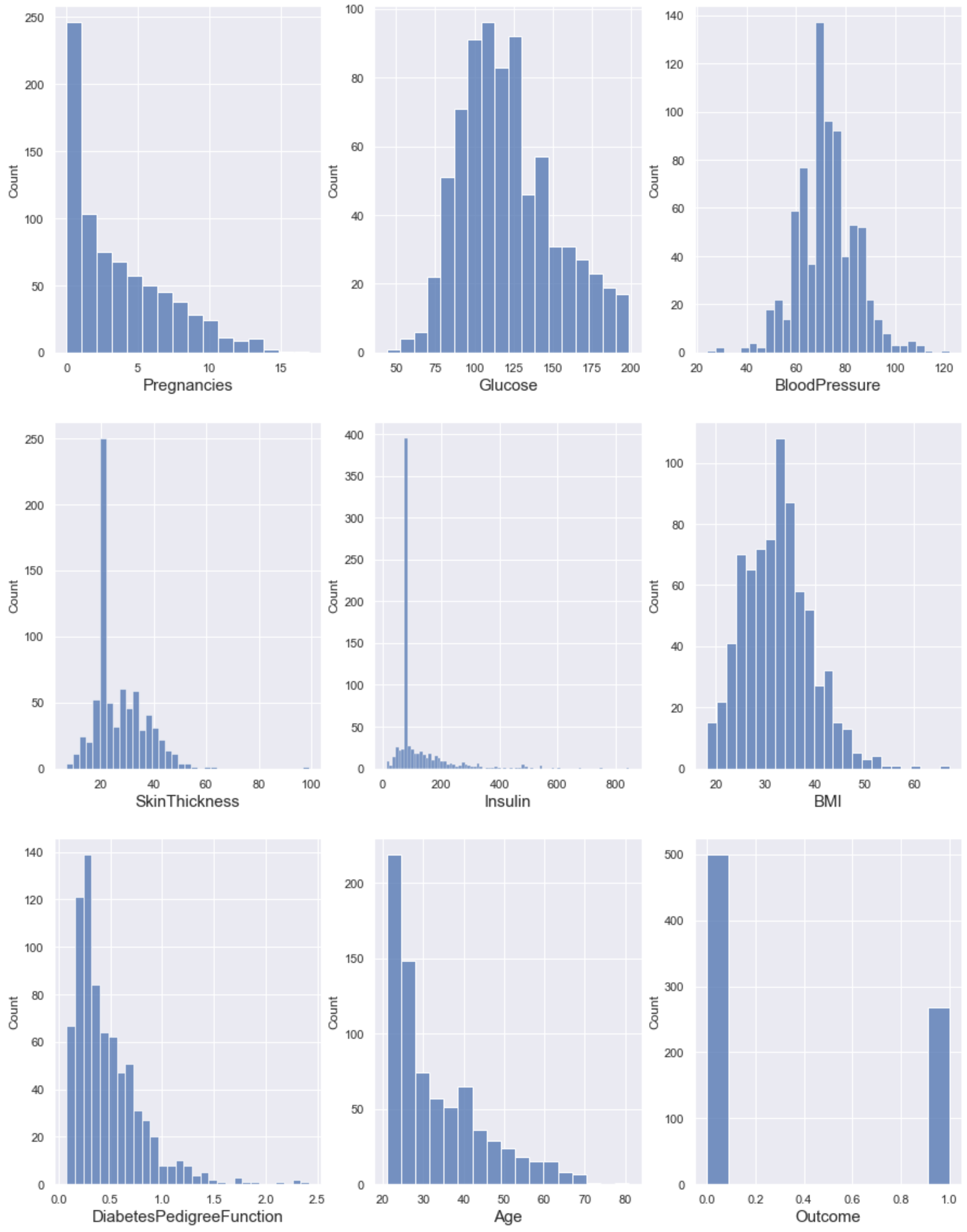
```
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].mean())
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].mean())
df['BMI']=df['BMI'].replace(0,df['BMI'].mean())
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())
```

In [7]:

```
##Now let us again see how the distribution for different column looks

plt.figure(figsize=(15,20),facecolor='white')
plotnumber=1

for column in df:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.histplot(df[column])
        plt.xlabel(column,fontsize=15)
    plotnumber+=1
plt.show()
```



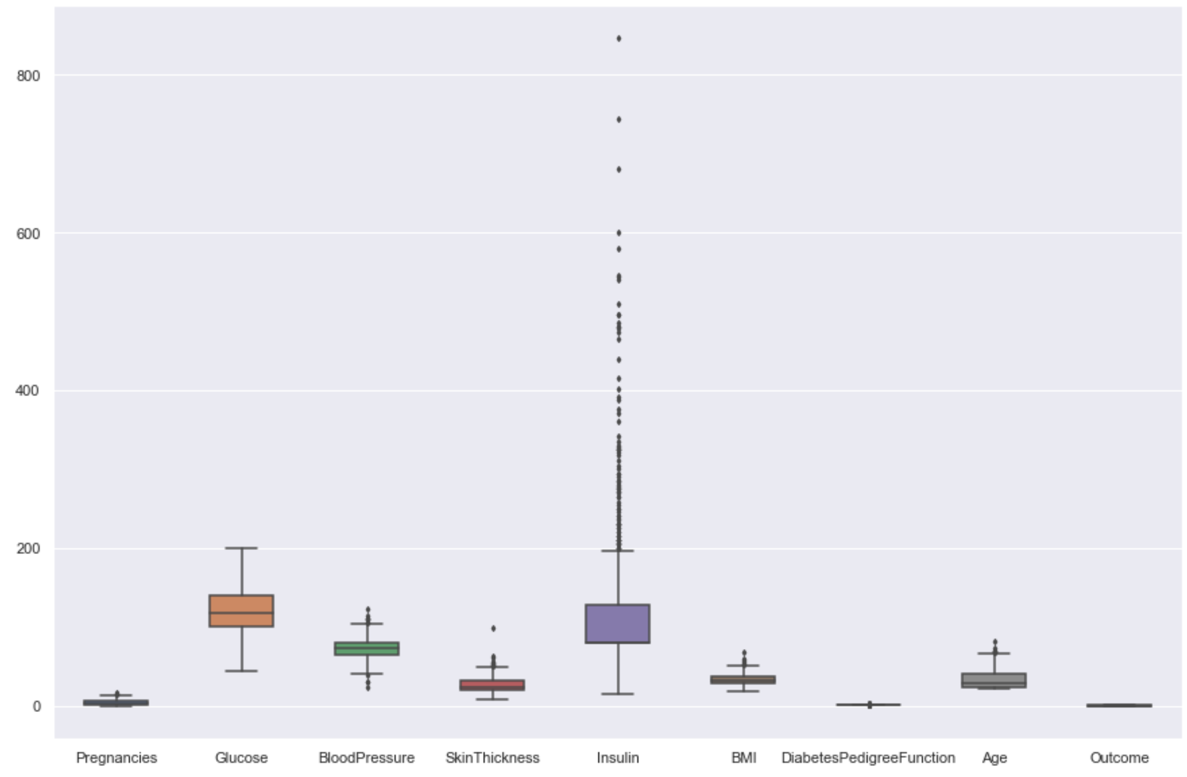
We notice that there are outliers.It is time to remove them

In [8]:

```
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=df, width= 0.5,ax=ax,  fliersize=3)
```

Out[8]:

```
<AxesSubplot:>
```



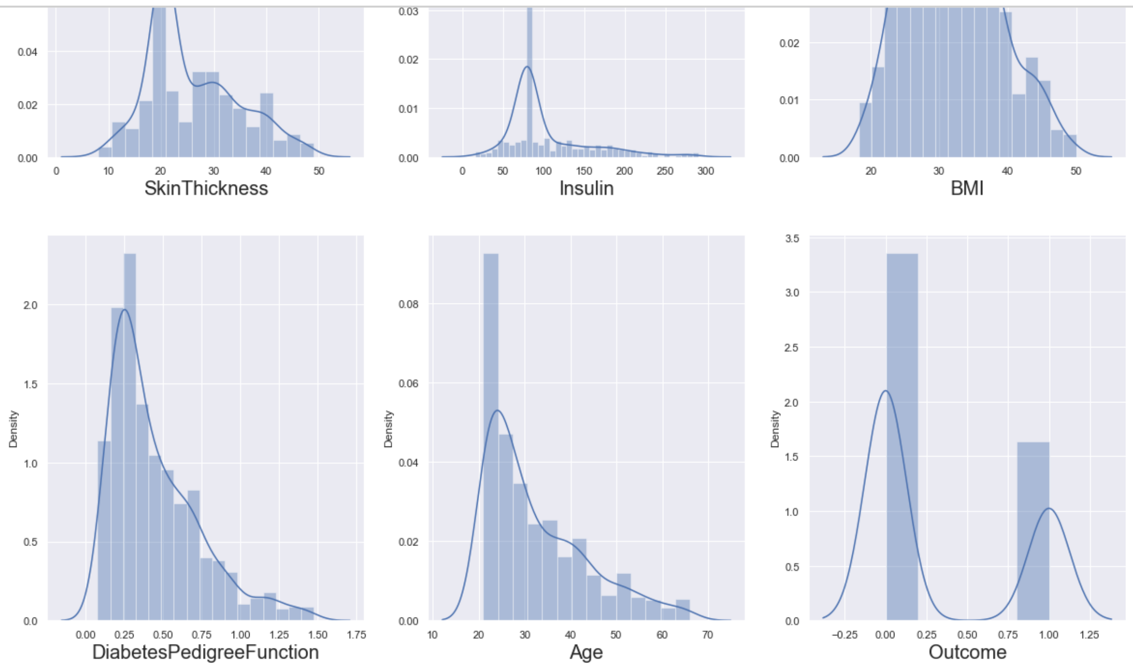
In [9]:

```
q = df['Pregnancies'].quantile(0.98)
# we are removing the top 2% data from the Pregnancies column
data_cleaned = df[df['Pregnancies']<q]
q = data_cleaned['BMI'].quantile(0.99)
# we are removing the top 1% data from the BMI column
data_cleaned  = data_cleaned[data_cleaned['BMI']<q]
q = data_cleaned['SkinThickness'].quantile(0.99)
# we are removing the top 1% data from the SkinThickness column
data_cleaned  = data_cleaned[data_cleaned['SkinThickness']<q]
q = data_cleaned['Insulin'].quantile(0.95)
# we are removing the top 5% data from the Insulin column
data_cleaned  = data_cleaned[data_cleaned['Insulin']<q]
q = data_cleaned['DiabetesPedigreeFunction'].quantile(0.99)
# we are removing the top 1% data from the DiabetesPedigreeFunction column
data_cleaned  = data_cleaned[data_cleaned['DiabetesPedigreeFunction']<q]
q = data_cleaned['Age'].quantile(0.99)
# we are removing the top 1% data from the Age column
data_cleaned  = data_cleaned[data_cleaned['Age']<q]
```

In [10]:

```
# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data_cleaned:
    if plotnumber<=9 :
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(data_cleaned[column])
        plt.xlabel(column,fontsize=20)
        #plt.ylabel('Salary',fontsize=20)
    plotnumber+=1
plt.show()
```



The data looks much better now than before. We will start our analysis with this data now as we don't want to lose important information. If our model doesn't work with accuracy, we will come back for more preprocessing.

In [11]:

```
df=data_cleaned
```

In [12]:

```
X= df.drop(columns=["Outcome"])
y=df["Outcome"]
```

Now we should scale our data. Let's use the standard scaler for that.

In [13]:

```
scalar=StandardScaler()
X_scaled=scalar.fit_transform(X)
```

In [14]:

```
X_scaled
```

Out[14]:

```
array([[ 7.96753910e-01,  9.83984062e-01,  4.52611463e-04, ...,
         2.65819648e-01,  6.30484542e-01,  1.60141519e+00],
       [-8.64793539e-01, -1.16977621e+00, -5.04474494e-01, ...,
        -8.31445036e-01, -3.38078670e-01, -1.32706484e-01],
       [ 1.46137289e+00,  2.18051755e+00, -6.72783529e-01, ...,
        -1.34872696e+00,  7.88402456e-01, -4.14369227e-02],
       ...,
       [ 4.64444420e-01,  6.09439465e-02,  4.52611463e-04, ...,
        -8.94145875e-01, -7.10063091e-01, -2.23976046e-01],
       [-8.64793539e-01,  2.31877301e-01, -1.00940160e+00, ...,
        -2.82812694e-01, -3.45097244e-01,  1.32760650e+00],
       [-8.64793539e-01, -8.96282840e-01, -1.67856424e-01, ...,
        -2.35787064e-01, -4.64413001e-01, -8.62862978e-01]])
```

Let us check for multicollinearity now

In [15]:

```
vif = pd.DataFrame()
vif["vif"] = [variance_inflation_factor(X_scaled,i) for i in range(X_scaled.shape[1])]
vif["Features"] = X.columns

#let's check the values
vif
```

Out[15]:

	vif	Features
0	1.449056	Pregnancies
1	1.304263	Glucose
2	1.262686	BloodPressure
3	1.470049	SkinThickness
4	1.271017	Insulin
5	1.513160	BMI
6	1.042300	DiabetesPedigreeFunction
7	1.662728	Age

All the VIF values are less than 5 and are very low. That means no multicollinearity. Now, we can go ahead with fitting our data to the model. Before that, let's split our data in test and training set.

In [16]:

```
x_train,x_test,y_train,y_test= train_test_split(X_scaled,y,test_size= 0.25, random_state =
```

In [17]:

```
log_reg= LogisticRegression()
log_reg.fit(x_train,y_train)
```

Out[17]:

```
LogisticRegression()
```

Model Evaluation

Let's see how well our model performs on the test data set.

In [18]:

```
y_pred=log_reg.predict(x_test)
```

In [19]:

```
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

Out[19]:

```
0.834319526627219
```

In [20]:

```
##Let us find the Confusion Matrix

conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

Out[20]:

```
array([[109,   8],
       [ 20,  32]], dtype=int64)
```

In [21]:

```
true_positive=conf_mat[0][0]
false_positive=conf_mat[0][1]
false_negative=conf_mat[1][0]
true_negative=conf_mat[1][1]
```

In [22]:

```
# Breaking down the formula for Accuracy
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negativ
Accuracy
```

Out[22]:

```
0.834319526627219
```

In [23]:

```
# Precison
Precision = true_positive/(true_positive+false_positive)
Precision
```

Out[23]:

```
0.9316239316239316
```

In [24]:

```
# Recall
Recall = true_positive/(true_positive+false_negative)
Recall
```

Out[24]:

```
0.8449612403100775
```

In [25]:

```
#F1 Score

F1_score=2*Precision*Recall/(Precision+Recall)
F1_score
```

Out[25]:

```
0.8861788617886178
```

AUC

In [26]:

```
#Area Under Curve (AUC)

auc= roc_auc_score(y_test,y_pred)
auc
```

Out[26]:

```
0.7735042735042735
```

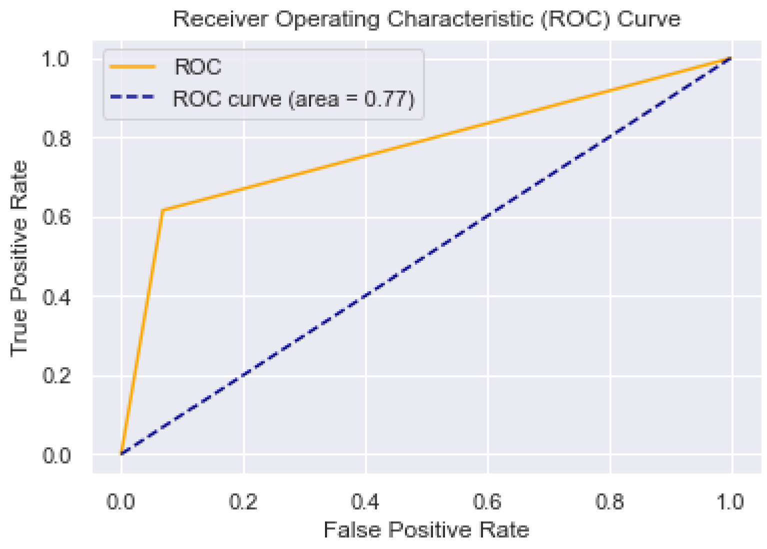
ROC

In [27]:

```
fpr,tpr, thresholds= roc_curve(y_test,y_pred)
```

In [28]:

```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)'
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



In [ ]:

```
```