MINIOR PROJECT

Problem Statement :

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

```

```
df = pd.read_excel("diabetes.csv.xlsx")
```

Exploratory Data Analysis

```
# DISPLAY FIRST FIVE RECORDS OF DATA
df.head(5)
```

```
# Display last five reacords of data
df.tail(5)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
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	

```
# Display randomlay any number of records od data
df.sample(5)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
15	7	100	0	0	0	30.0	
330	8	118	72	19	0	23.1	
461	1	71	62	0	0	21.8	
102	0	125	96	0	0	22.5	
352	3	61	82	28	0	34.4	

The shape of the dataset

```
#number of rows and columns
df.shape
```

```
(768, 9)
```

No. of Rows = 768

No. of Columns = 9

List of all Columns

```
#list the types of all columns
df.dtypes
```

```
Pregnancies                int64
Glucose                    int64
BloodPressure              int64
SkinThickness              int64
Insulin                    int64
BMI                      float64
```

```
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

Info of the dataset

```
1 #finding out if the dataset countains any null values
2 df.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
```

Summary of the dataset

```
1 # Statistical summary
2 df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
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

Observation :

In the above table,the main value of columns 'Glucose','BloodPressure','skinthickness','insuline','BMI'is zero(0).It is clear that this values cant be zero. So i am going to impute mean values of these respective columns insted of zero.

Data cleaning

```
#check the shape before drop the duplicates
df.shape
```

```
(768, 9)
```

```
df=df.drop_duplicates()
```

```
# check the shape after drop the duplicates
df.shape
```

```
(768, 9)
```

Before drop and after drop the duplicates the data set has same shape which means no duplicates in the dataset.

Check the Null Values

```
#count of null values
#check the missing values in any column
# #display number of null values is very column in dataset
df.isnull().sum()
```

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

There is no null values in the given dataset.

```
df.columns
```

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

Check the no. of zero values in dataset.

```
print('No. of zero values in Glucose',df[df['Glucose']==0].shape[0])
```

```
No. of zero values in Glucose 5
```

```
print('no. of zero values in Bloodpressure',df[df['BloodPressure']==0].shape[0])
```

```
no. of zero values in Bloodpressure 35
```

```
print('no. of zero values in skinThickness',df[df['SkinThickness']==0].shape[0])
```

```
no. of zero values in skinThickness 227
```

```
print('no. of zero values in Insulin',df[df['Insulin']==0].shape[0])
```

```
no. of zero values in Insulin 374
```

```
print('no. of zero values in BMI',df[df['BMI']==0].shape[0])
```

```
no. of zero values in BMI 11
```

Replace no. of zero values with mean of the columns

```
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
print('No.of zero values in Glucose',df[df['Glucose']==0].shape[0])
```

```
No.of zero values in Glucose 0
```

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

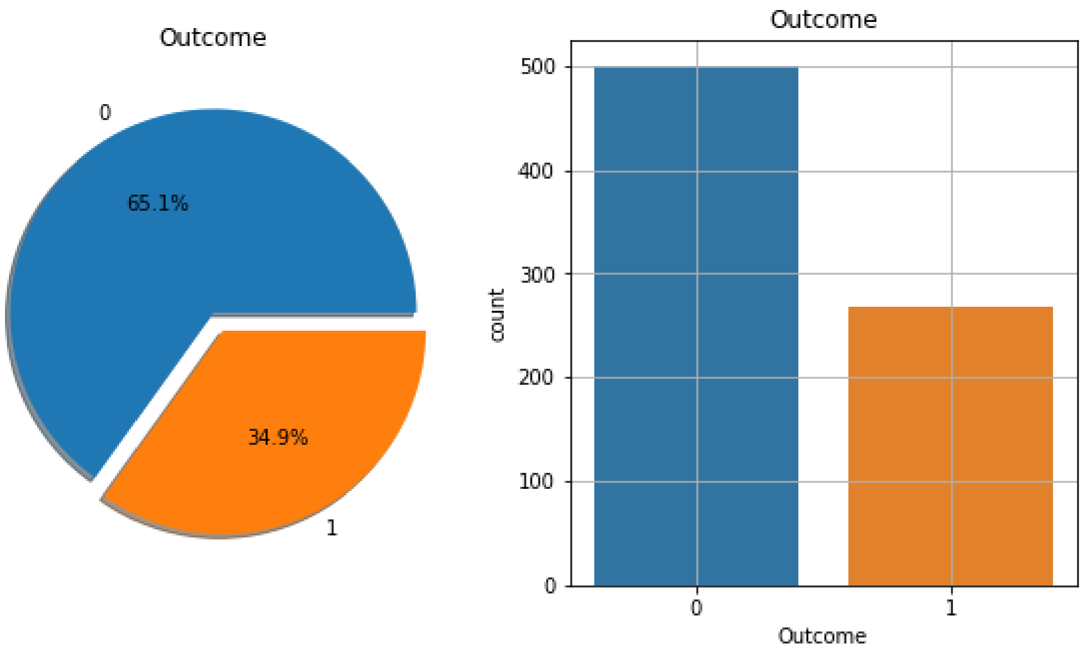
```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	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

Count plot

```
#outcome count plot
f,ax=plt.subplots(1,2,figsize=(10,5))
df['Outcome'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow
ax[0].set_title('Outcome')
ax[0].set_ylabel('')
sns.countplot('Outcome',data=df,ax=ax[1])
ax[1].set_title('Outcome')
N,P = df['Outcome'].value_counts()
print('Negative (0): ',N)
print('Positive (1): ',P)
plt.grid()
plt.show()
```

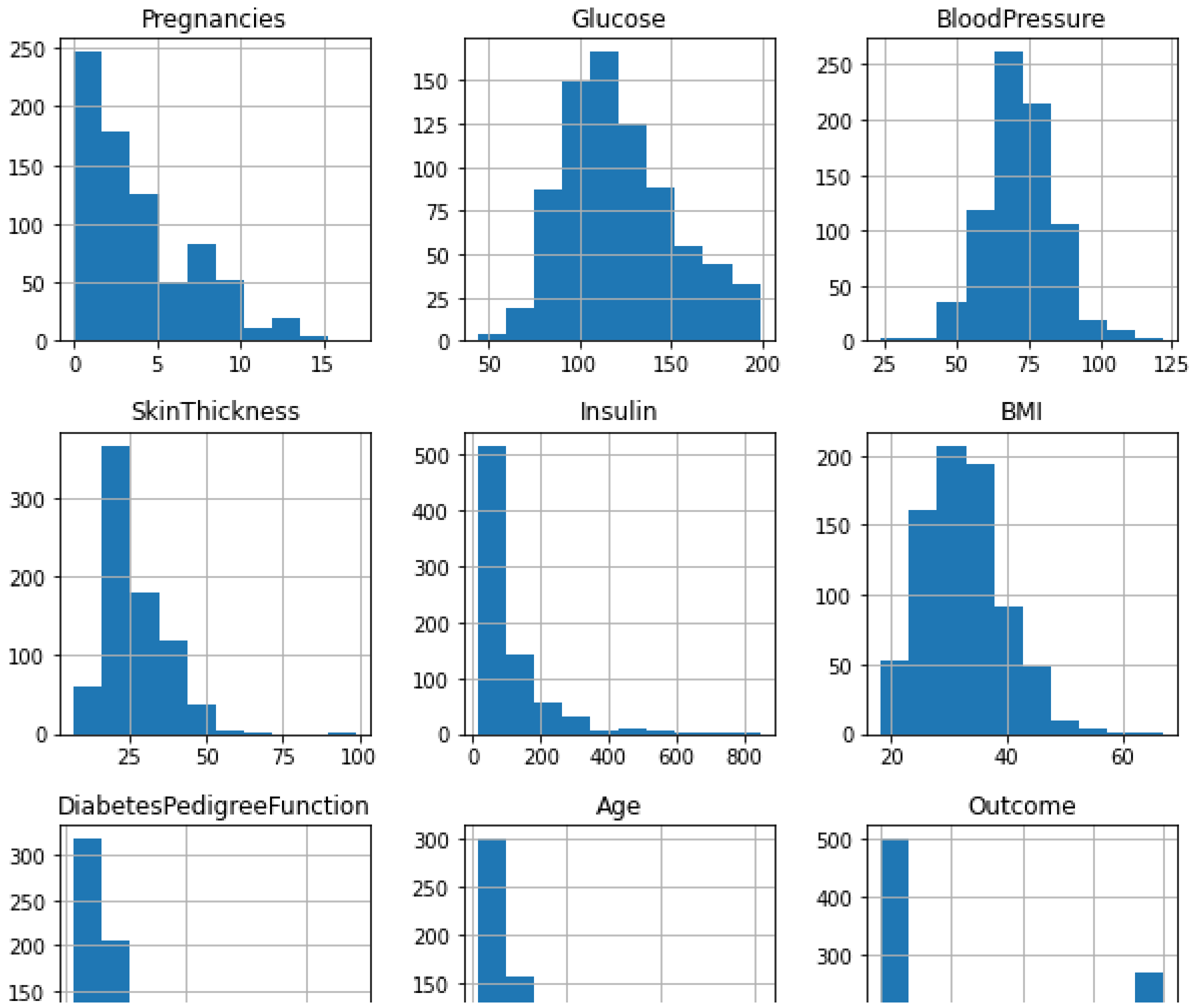
```
Negative (0):  500
Positive (1):  268
```



Out of total 768 people, 268 are dibetic (Positive(1))and 500 are non-dibetic(Negative(0)). In the outcome coulmn, **1** represents **diabetes positive** and **0** represents **diabetes negative**. The countplot tells us that the dataset is Imbalanced,as number of patients who dont have diabetes is more than those who have diabets.

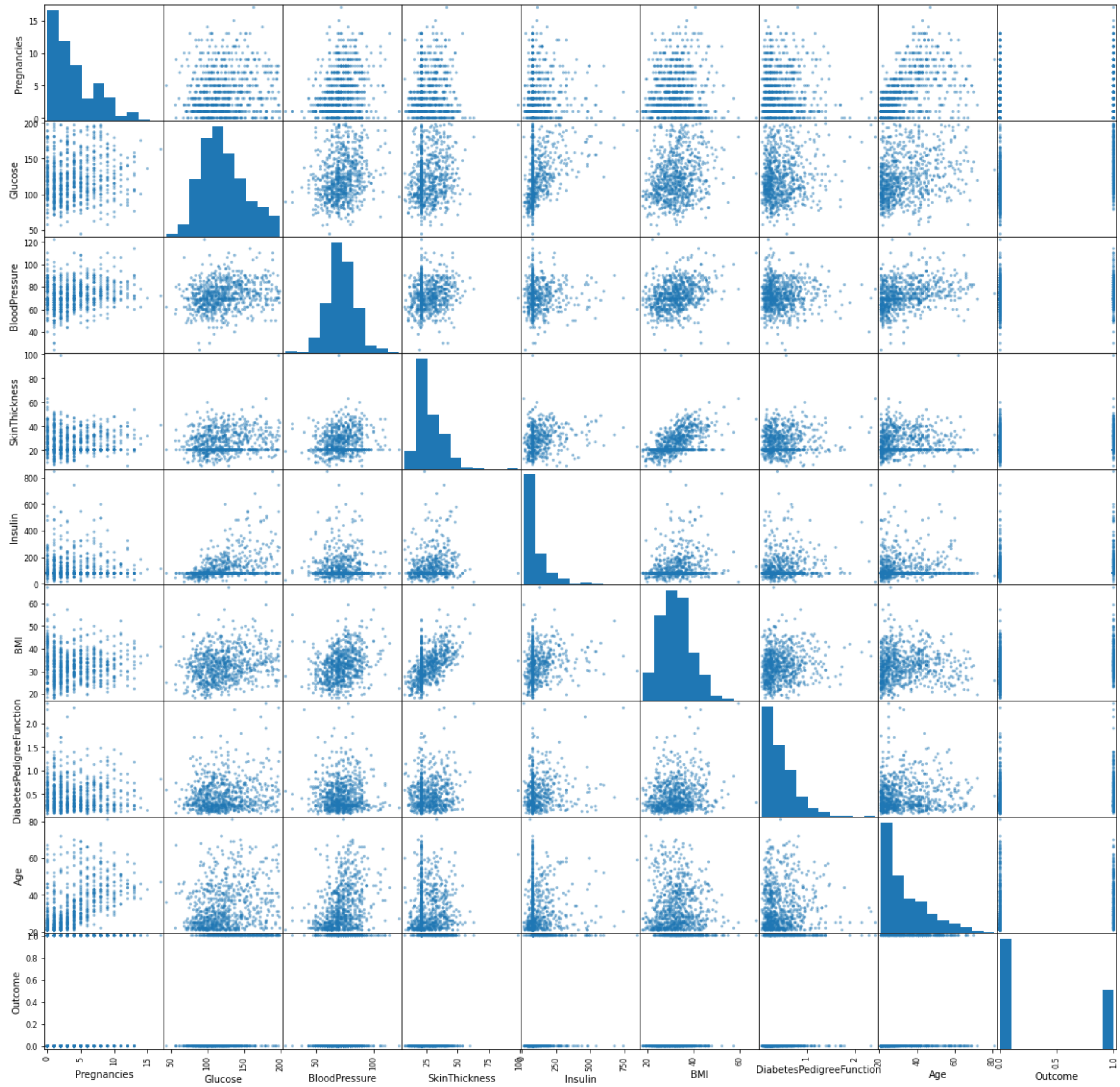
Histograms

```
#Histogram of each feature
df.hist(bins=10,figsize=(10,10))
plt.show()
```



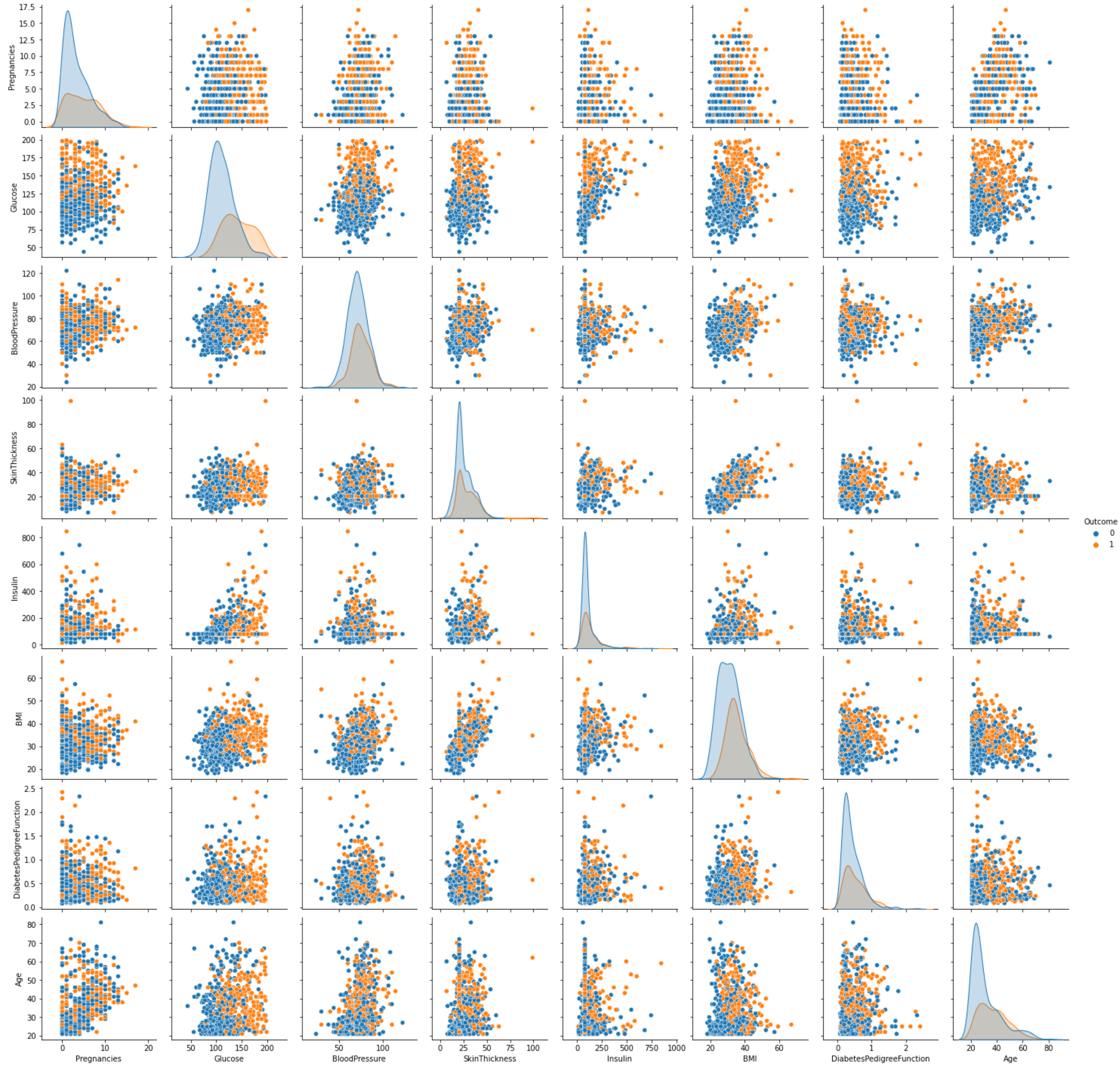
Scatter plot

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



Pair plot

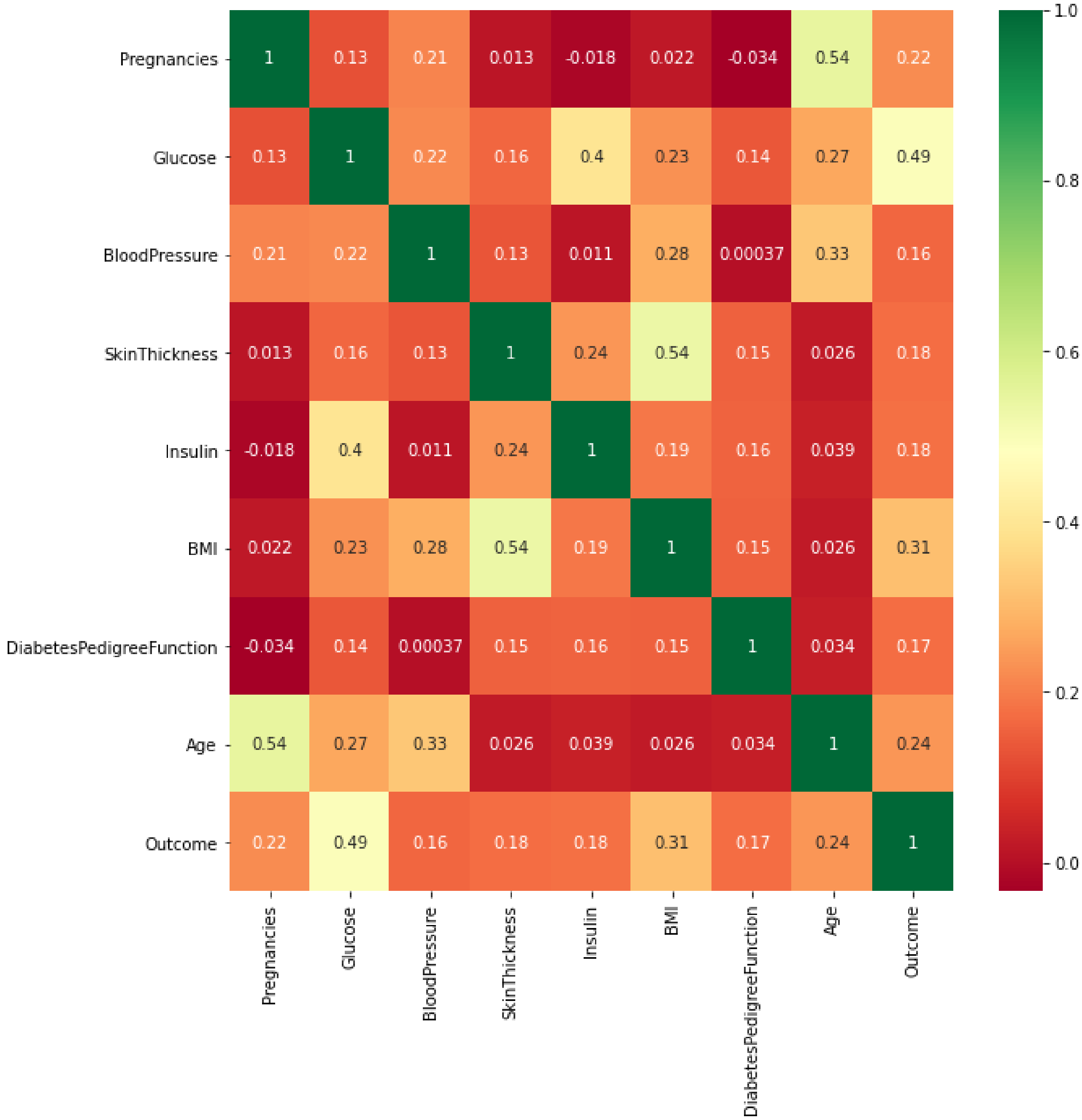
```
1 #Pairplot
2 sns.pairplot(data = df, hue = 'Outcome')
3 plt.show()
```



Analyzing relationships between variables

Correlation analysis

```
import seaborn as sns
#get correlation of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize = (10,10))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



Observations:

From the correlation heatmap. we can see that there is a high correlation between between Outcome and [Pregnancies, Glucose,BMI,Age,Insulin].We can select these features to accept input from the user and predict the outcome.

Split the data frame into X & y

```
target_name = 'Outcome'

# Separate object for target feature
y = df[target_name]

# Separate Object for Input Features
X = df.drop(target_name, axis=1)
```

```
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	85.0	66.0	29.000000	79.799479	26.6	
2	8	183.0	64.0	20.536458	79.799479	23.3	
3	1	89.0	66.0	23.000000	94.000000	28.1	
4	0	137.0	40.0	35.000000	168.000000	43.1	

```
y.head()
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Apply Feature Scaling

```
#Apply Standard Scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
SSX = scaler.transform(X)
```

Train Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(SSX, y , test_size=0.2, random_stat
```

```
X_train.shape,y_train.shape
```

```
((614, 8), (614,))
```

```
X_test.shape,y_test.shape
```

```
X_test.shape,y_test.shape
```

```
((154, 8), (154,))
```

Bulid the Classification Algorithams

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(X_train,y_train)
```

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

KNeighbors(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

*Naive-Bayes *

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

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Support Vector Machine(SVM)

```
from sklearn.svm import SVC
sv=SVC()
sv.fit(X_train,y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

*Decision Tree *

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(criterion='entropy')
rf.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Making Prediction

```
X_test.shape
```

```
(154, 8)
```

Making Prediction using Logistic Regression

```
## Making predictions on test dataset
lr_pred=lr.predict(X_test)
```

```
lr_pred.shape
```

```
(154,)
```

Making Prediction using KNN

```
## Making predictions on test dataset
knn_pred=knn.predict(X_test)
```

```
knn_pred.shape
```

```
(154,)
```

Making Prediction using Naive-Byes

```
## Making predictions on test dataset
nb_pred=nb.predict(X_test)
```

```
nb_pred.shape
```

```
(154,)
```

Making Prediction using SVM

```
#Making predictions on test dataset
sv_pred=sv.predict(X_test)
```

```
sv_pred.shape
```

```
(154,)
```

Making Prediction using Decision Tree

```
#Making predictions on test dataset
dt_pred=dt.predict(X_test)
```

```
dt_pred.shape
```

```
(154,)
```

Making Prediction using Random Forest

```
#Making predictions on test dataset
rf_pred=rf.predict(X_test)
```

```
rf_pred.shape
```

```
(154,)
```

Model Evaluation

Train Score & Test Score

```
# Train score & Test score of logistic regression
from sklearn.metrics import accuracy_score
print("Train Accuracy of Logistic Regression",lr.score(X_train,y_train)*100)
```

```
print("Train Accuracy of Logistic Regression",lr.score(X_train,y_train)*100)
print("Accuracy (test) score of logistic Regression",lr.score(X_test, y_test)*100)
print("Accuracy (Test) score of Logistic Regression",accuracy_score(y_test, lr_pred)*10
```

```
Train Accuracy of Logistic Regression 77.36156351791531
Accuracy (test) score of logistic Regression 77.27272727272727
Accuracy (Test) score of Logistic Regression 77.27272727272727
```

```
# Train score & Test score of KNN
print("Train Accuracy of KNN",knn.score(X_train,y_train)*100)
print("Accuracy (Test) score of KNN",knn.score(X_test, y_test)*100)
print("Accuracy score of KNN",accuracy_score(y_test, knn_pred)*100)
```

```
Train Accuracy of KNN 81.10749185667753
Accuracy (Test) score of KNN 74.67532467532467
Accuracy score of KNN 74.67532467532467
```

```
# Train score & Test score of Naive-Bayes
print("Train Accuracy of Naive Bayes",nb.score(X_train,y_train)*100)
print("Accuracy (Test) score of Naive Bayes",nb.score(X_test, y_test)*100)
print("Accuracy score of Naive Bayes",accuracy_score(y_test,nb_pred)*100)
```

```
Train Accuracy of Naive Bayes 74.2671009771987
Accuracy (Test) score of Naive Bayes 74.02597402597402
Accuracy score of Naive Bayes 74.02597402597402
```

```
# Train score & Test score of SVM
print("Train Accuracy of SVM",sv.score(X_train, y_train)*100)
print("Accuracy (Test) score of SVM",sv.score(X_test, y_test)*100)
print("Accuracy score of SVM",accuracy_score(y_test,sv_pred)*100)
```

```
Train Accuracy of SVM 81.92182410423453
Accuracy (Test) score of SVM 83.11688311688312
Accuracy score of SVM 83.11688311688312
```

```
# Train score & Test score of Decesion Tree
print("Train Accuracy of Decesion Tree",dt.score(X_train,y_train)*100)
print("Accuracy(Test) score of Decesion Tree",dt.score(X_test,y_test)*100)
print("Accuracy score of Decesion Tree",accuracy_score(y_test,dt_pred)*100)
```

```
Train Accuracy of Decesion Tree 100.0
Accuracy(Test) score of Decesion Tree 79.87012987012987
Accuracy score of Decesion Tree 79.87012987012987
```

```
# Train scrore & Test score of Random Forest
print("Train Accuracy of Random Forest",rf.score(X_train,y_train)*100)
print("Accuracy (Test) score of Random Forest",rf.score(X_test,y_test)*100)
print("Accuracy score of Random Forest",accuracy_score(y_test,rf_pred)*100)
```

```
Train Accuracy of Random Forest 100.0
Accuracy (Test) score of Random Forest 81.16883116883116
Accuracy score of Random Forest 81.16883116883116
```

Confusion Matrix

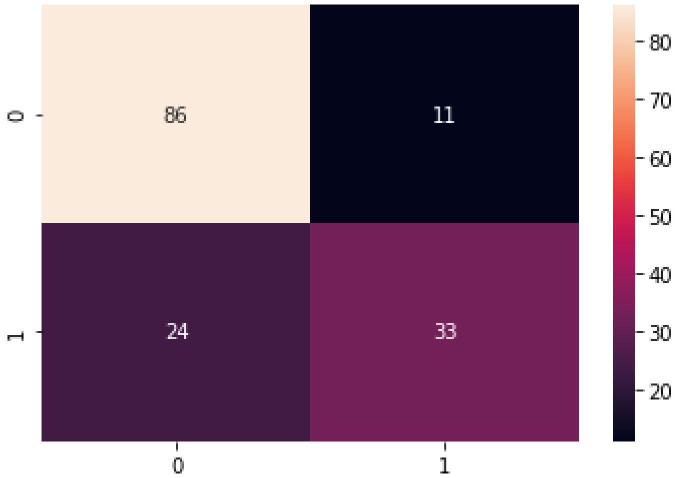
Confusion Matrix of "Logistic Regression"

```
from sklearn.metrics import classification_report,confusion_matrix
# confusion Matrix of Logistic Regression
cm=confusion_matrix(y_test,lr_pred)
cm
```

```
array([[86, 11],
       [24, 33]])
```

```
sns.heatmap(confusion_matrix(y_test,lr_pred),annot=True,fmt="d")
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f7c1d14b4d0>




```
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TP = cm[1,1]
```

```
TN, FP, FN,TP
```

```
(86, 11, 24, 33)
```

```
####
```

```
print('Classification Report of Logistic Regression: \n',classification_report(y_test,l
```

```
Classification Report of Logistic Regression:
               precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
```

```
weighted avg     0.7700    0.7727    0.7652       154
```

```
# Making the Confusion Matrix Of Logistic Regression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, lr_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm
```

```
TN - True Negative 86
FP - False Positive 11
FN - False Negative 24
TP - True Positive 33
Accuracy Rate: 77.27272727272727
Misclassification Rate: 22.727272727272727
```

```
77.27272727272727+22.727272727272727
```

```
100.0
```

```
import matplotlib.pyplot as plt
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0','1']
plt.title('Confusion Matrix of Logistic Regression')
plt.ylabel('Actual(true) values')
plt.xlabel('Predicated values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'],['FN','TP']]
for i in range(2):
  for j in range(2):
     plt.text(j,i,str(s[i][j])+"="+str(cm[i][j]))
plt.show()
```


```
pd.crosstab(y_test, lr_pred, margins=False)
```

col_0	0	1
Outcome		
0	86	11
1	24	33

```
pd.crosstab(y_test, lr_pred, margins=True)
```

col_0	0	1	All
Outcome			
0	86	11	97
1	24	33	57
All	110	44	154

```
pd.crosstab(y_test, lr_pred, rownames=['Actual values'], colnames=['Predicted values'],
```

Predicted values	0	1	All
Actual values			
0	86	11	97
1	24	33	57
All	110	44	154

```
TP,FP
```

```
(33, 11)
```

```
Precision=TP/(TP+FP)
Precision
```

```
0.75
```

```
33/(33+11)
```

```
0.75
```

```
# print precision score

precision_Score = TP/float(TP+FP)*100
print('Precision score : {0:0.4f}'.format(precision_Score))
```

```
print('Precision score : {0:0.4f}'.format(precision_Score))
```

```
Precision score : 75.0000
```

```
from sklearn.metrics import precision_score
print("precision Score is:", precision_score(y_test,lr_pred,average='micro')*100)
print("Mircro Average precision Score is:",precision_score(y_test,lr_pred,average='micr
print("Marcro Average precision Score is:",precision_score(y_test,lr_pred,average='macr
print("Weighted Average precision Score is:",precision_score(y_test,lr_pred,average='we
print("precision Score on Non weighted score is:", precision_score(y_test,lr_pred,avera
```

```
precision Score is: 77.27272727272727
Mircro Average precision Score is: 77.27272727272727
Marcro Average precision Score is: 76.5909090909091
Weighted Average precision Score is: 77.00413223140497
precision Score on Non weighted score is: [78.18181818 75.        ]
```

```
print('Classification Report of Logistic Regression: \n',classification_report(y_test,l
```

```
Classification Report of Logistic Regression:
               precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

```
recall_score=TP/float(TP+FN)*100
print('recall_score',recall_score)
```

```
recall_score 57.89473684210527
```

```
TP,FN
```

```
(33, 24)
```

```
33/(33+24)
```

```
0.5789473684210527
```

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_score:',recall_score(y_test,lr_pred)*100)
```

```
Recall or Sensitivity_score: 57.89473684210527
```

```
print("Mircro Average Recall Score is:", recall_score(y_test,lr_pred,average='micro')*1
print("Marcro Average Recall Score is:", recall_score(y_test,lr_pred,average='macro')*1
print("Weighted Average Recall Score is:",recall_score(y_test,lr_pred,average='weighted
print("Recall Score on Non weighted score is:", recall_score(y_test,lr_pred,average=Non
```

```
Mircro Average Recall Score is: 77.27272727272727
Marcro Average Recall Score is: 73.27726532826912
Weighted Average Recall Score is: 77.27272727272727
Recall Score on Non weighted score is: [88.65979381 57.89473684]
```

```
1 print('Classification Report of logistic Regression: \n',classification_report(y_test,l
```

```
Classification Report of logistic Regression:
               precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

False Positive Rate(FPR)

```
1 FPR=FP/float(FP+TN)*100
2 print('False Positive Rate : {0:0.4f}'.format(FPR))
```

```
False Positive Rate : 11.3402
```

```
1 FP,TN
```

```
(11, 86)
```

```
1 11/(11+86)
```

```
0.1134020618556701
```

Specificity

```
1 specificity=TN/(TN+FP)*100
2 print('Specificity :{0:0.4f}'.format(specificity))
```

```
Specificity :88.6598
```

F1-Score

```
1 from sklearn.metrics import f1_score
2 print('f1_score of macro :',f1_score(y_test, lr_pred)*100)
```

```
f1_score of macro : 65.34653465346535
```

```
1 print("Mircro Average F1_Score is:",f1_score(y_test,lr_pred,average='micro')*100)
2 print("marcro Average F1_Score is:",f1_score(y_test,lr_pred,average='macro')*100)
```

```
print("Weighted Average F1_score is:",f1_score(y_test,lr_pred,average='weighted')*100)
print("F1_Score on Non weighted score is:",f1_score(y_test,lr_pred,average=None)*100)
```

```
Mircro Average F1_Score is: 77.27272727272727
marcro Average F1_Score is: 74.21916104653944
Weighted Average F1_score is: 76.52373933045479
F1_Score on Non weighted score is: [83.09178744 65.34653465]
```

*Classification Report of Logistic Regression *

```
from sklearn.metrics import classification_report
print('Classification Report of logistic Regression: \n', classification_report(y_test,
```

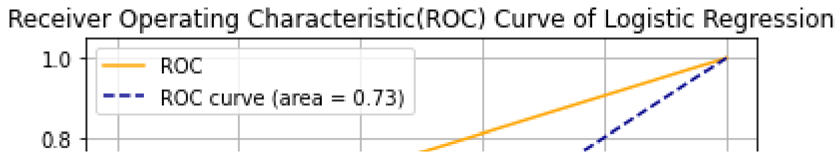
```
Classification Report of logistic Regression:
              precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

ROC Curve & ROC AUC

```
# Area Under Curve
auc = roc_auc_score(y_test,lr_pred)
print("ROC AUC SCORE of logistic Regression is",auc)
```

```
ROC AUC SCORE of logistic Regression is 0.7327726532826913
```

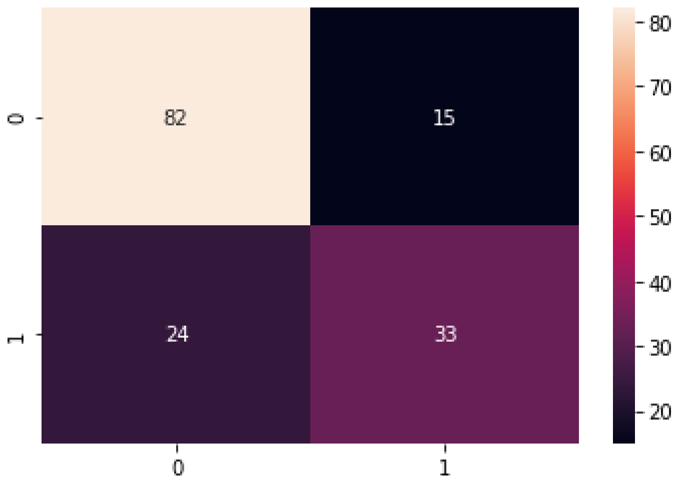
```
fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0,1],[0,1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)'
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic(ROC)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



Confusion Matrix of "KNN"

```
sns.heatmap(confusion_matrix(y_test,knn_pred),annot=True,fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c1d2fcd90>
```




```
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TP = cm[1,1]
```

```
TN, FP, FN, TP
```

```
(86, 11, 24, 33)
```

```
####
```

```
# classification Report of KNN
print('Classification Report of KNN: \n', classification_report(y_test,knn_pred,digits=
```

```
Classification Report of KNN:
               precision    recall  f1-score   support

           0     0.7736    0.8454    0.8079        97
           1     0.6875    0.5789    0.6286        57

    accuracy                         0.7468       154
   macro avg     0.7305    0.7122    0.7182       154
weighted avg     0.7417    0.7468    0.7415       154
```

```
# Making the confusion Matrix of KNN
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score,roc_curve
```

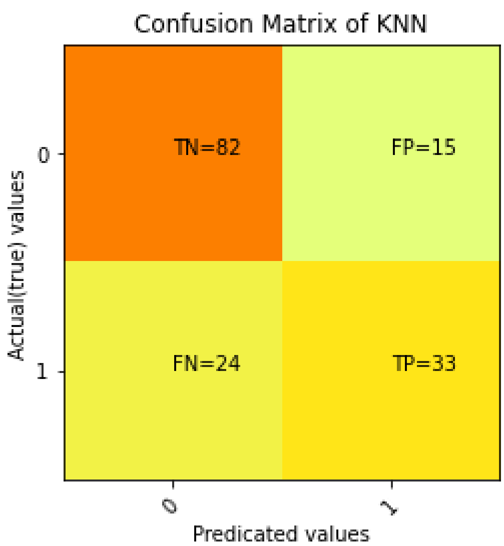
```
cm = confusion_matrix(y_test,knn_pred)


print('TN - True Negative: {}'.format(cm[0,0]))
print('FP - False Positive: {}'.format (cm[0,1]))
print('FN - False Negative: {}'.format(cm[1,0]))
print('TP - True Positive: {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[0,1],cm[1,0],cm[1,1]]),np
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm
```

```
TN - True Negative: 82
FP - False Positive: 15
FN - False Negative: 24
TP - True Positive: 33
Accuracy Rate: 100.0
Misclassification Rate: 25.324675324675322
```

```
74.67532467532467+25.324675324675322
```

```
100.0
```

```
import matplotlib.pyplot as plt
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0','1']
plt.title('Confusion Matrix of KNN')
plt.ylabel('Actual(true) values')
plt.xlabel('Predicated values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'],['FN','TP']]
for i in range(2):
  for j in range(2):
      plt.text(j,i,str(s[i][j])+"="+str(cm[i][j]))
plt.show()
```




```
pd.crosstab(y_test, knn_pred, margins=False)
```

```
pd.crosstab(y_test, knn_pred, margins=False)
```

col_0	0	1
Outcome		
0	82	15
1	24	33

```
pd.crosstab(y_test, knn_pred, margins=True)
```

col_0	0	1	All
Outcome			
0	82	15	97
1	24	33	57
All	106	48	154

```
pd.crosstab(y_test, knn_pred, rownames=['Actual values'], colnames=['Predicted values']
```

Predicted values	0	1	All
Actual values			
0	82	15	97
1	24	33	57
All	106	48	154

```
TP,FP
```

```
(33, 11)
```

```
Precision=TP/(TP+FP)
Precision
```

```
0.75
```

```
33/(33+15)
```

```
0.6875
```

```
# print precision score

precision_Score = TP/float(TP+FP)*100
print('Precision score : {0:0.4f}'.format(precision_Score))
```

```
Precision score : 75.0000
```

```
from sklearn.metrics import precision_score
print("precision Score is:", precision_score(y_test,knn_pred,average='micro')*100)
print("Mircro Average precision Score is:",precision_score(y_test,knn_pred,average='mic
print("Marcro Average precision Score is:",precision_score(y_test,knn_pred,average='mac
print("Weighted Average precision Score is:",precision_score(y_test,knn_pred,average='w
print("precision Score on Non weighted score is:", precision_score(y_test,knn_pred,aver
```

```
precision Score is: 74.67532467532467
Mircro Average precision Score is: 74.67532467532467
Marcro Average precision Score is: 73.05424528301887
Weighted Average precision Score is: 74.17223107081597
precision Score on Non weighted score is: [77.35849057 68.75      ]
```

```
print('Classification Report of KNN: \n',classification_report(y_test,knn_pred,digits=4
```

```
Classification Report of KNN:
               precision    recall  f1-score   support

           0     0.7736    0.8454    0.8079        97
           1     0.6875    0.5789    0.6286        57

    accuracy                         0.7468       154
   macro avg     0.7305    0.7122    0.7182       154
weighted avg     0.7417    0.7468    0.7415       154
```

```
recall_score=TP/float(TP+FN)*100
print('recall_score',recall_score)
```

```
recall_score 57.89473684210527
```

```
TP,FN
```

```
(33, 24)
```

```
33/(33+24)
```

```
0.5789473684210527
```

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_score:',recall_score(y_test,knn_pred)*100)
```

```
Recall or Sensitivity_score: 57.89473684210527
```

```
print("Mircro Average Recall Score is:", recall_score(y_test,knn_pred,average='micro')*
print("Marcro Average Recall Score is:", recall_score(y_test,knn_pred,average='macro')*
print("Weighted Average Recall Score is:",recall_score(y_test,knn_pred,average='weighte
print("Recall Score on Non weighted score is:", recall_score(y_test,knn_pred,average=No
```

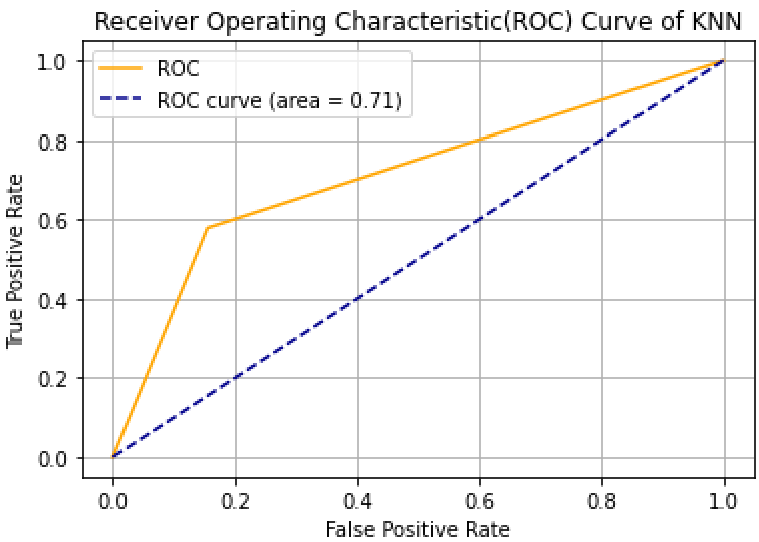
```
Mircro Average Recall Score is: 74.67532467532467
Marcro Average Recall Score is: 71.21540965816604
Weighted Average Recall Score is: 74.67532467532467
Recall Score on Non weighted score is: [84.53608247 57.89473684]
```

ROC Curve & ROC AUC

```
# Area Under the Curve
auc = roc_auc_score(y_test, knn_pred)
print("ROC AUC SCORE of KNN is",auc)
```

```
ROC AUC SCORE of KNN is 0.7121540965816603
```

```
fpr, tpr, thresholds = roc_curve(y_test, knn_pred)
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 of KNN')
plt.legend()
plt.grid()
plt.show()
```



Confusion Matrix of "Naive-Byes"

```
sns.heatmap(confusion_matrix(y_test, nb_pred),annot=True,fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c1d897c50>
```

```
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TP = cm[1,1]
```

```
TN, FP, FN,TP
```

```
(82, 15, 24, 33)
```

```
####
```

```
# classification Report of Naive Byes
print('Classification Report of Naive Byes: \n', classification_report(y_test,nb_pred,d
```

```
Classification Report of Naive Byes:
               precision    recall  f1-score   support

           0     0.7879    0.8041    0.7959        97
           1     0.6545    0.6316    0.6429        57

    accuracy                         0.7403       154
   macro avg     0.7212    0.7179    0.7194       154
weighted avg     0.7385    0.7403    0.7393       154
```

```
# Making the confusion Matrix of Naive Bayes
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score,roc_curve
cm = confusion_matrix(y_test,nb_pred)


print('TN - True Negative: {}'.format(cm[0,0]))
print('FP - False Positive: {}'.format (cm[0,1]))
print('FN - False Negative: {}'.format(cm[1,0]))
print('TP - True Positive: {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[0,1],cm[1,0],cm[1,1]]),np
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm
```

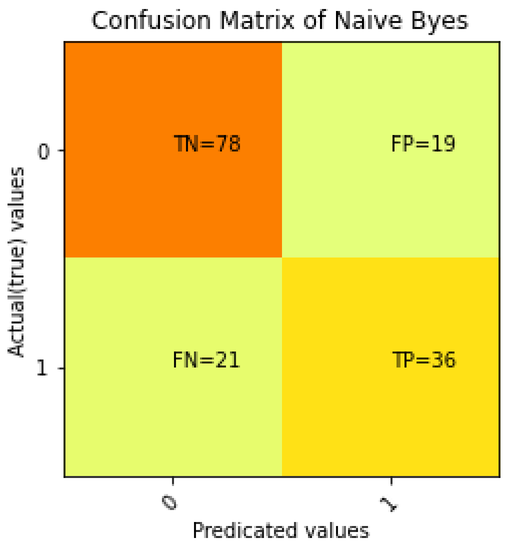
```
TN - True Negative: 78
FP - False Positive: 19
FN - False Negative: 21
TP - True Positive: 36
Accuracy Rate: 100.0
Misclassification Rate: 25.97402597402597
```

```
74.02597402597402+25.97402597402597
```

```
100.0
```

```
import matplotlib.pyplot as plt
```

```
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0','1']
plt.title('Confusion Matrix of Naive Byes')
plt.ylabel('Actual(true) values')
plt.xlabel('Predicated values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'],['FN','TP']]
for i in range(2):
  for j in range(2):
      plt.text(j,i,str(s[i][j])+"="+str(cm[i][j]))
plt.show()
```




```
pd.crosstab(y_test, nb_pred, margins=False)
```

col_0	0	1
Outcome		
0	78	19
1	21	36

```
pd.crosstab(y_test, nb_pred, margins=True)
```

col_0	0	1	All
Outcome			
0	78	19	97
1	21	36	57
All	99	55	154

```
pd.crosstab(y_test, nb_pred, rownames=['Actual values'], colnames=['Predicted values'],
```

Predicted values	0	1	All
Actual values			
0	78	19	97
1	21	36	57
All	99	55	154

```
TP,FP
```

```
(44, 18)
```

```
Precision=TP/(TP+FP)
Precision
```

```
0.6875
```

```
36/(36+19)
```

```
0.6545454545454545
```

```
# print precision score

precision_Score = TP/float(TP+FP)*100
print('Precision score : {0:0.4f}'.format(precision_Score))
```

```
Precision score : 68.7500
```

```
from sklearn.metrics import precision_score
print("precision Score is:", precision_score(y_test,nb_pred,average='micro')*100)
print("Mircro Average precision Score is:",precision_score(y_test,nb_pred,average='micr
print("Marcro Average precision Score is:",precision_score(y_test,nb_pred,average='macr
print("Weighted Average precision Score is:",precision_score(y_test,nb_pred,average='we
print("precision Score on Non weighted score is:", precision_score(y_test,nb_pred,avera
```

```
precision Score is: 74.02597402597402
Mircro Average precision Score is: 74.02597402597402
Marcro Average precision Score is: 72.12121212121212
Weighted Average precision Score is: 73.85281385281385
precision Score on Non weighted score is: [78.78787879 65.45454545]
```

```
# classification Report of Naive Bayes
print('Classification Report of Naive Bayes: \n', classification_report(y_test,nb_pred,
```

```
Classification Report of Naive Bayes:
               precision    recall  f1-score   support

           0     0.7879    0.8041    0.7959        97
           1     0.6545    0.6316    0.6429        57

    accuracy                         0.7403       154
```

```
   macro avg     0.7212    0.7179    0.7194       154
weighted avg     0.7385    0.7403    0.7393       154
```

```
recall_score=TP/float(TP+FN)*100
print('recall_score',recall_score)
```

```
recall_score 57.89473684210527
```

```
TP,FN
```

```
(33, 24)
```

```
36/(36+21)
```

```
0.631578947368421
```

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_score:',recall_score(y_test,nb_pred)*100)
```

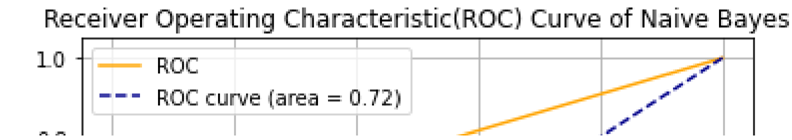
```
Recall or Sensitivity_score: 63.1578947368421
```

ROC Curve & ROC AUC

```
# Area Under the Curve
auc = roc_auc_score(y_test, nb_pred)
print("ROC AUC SCORE of Naive Bayes is",auc)
```

```
ROC AUC SCORE of Naive Bayes is 0.7178513293543136
```

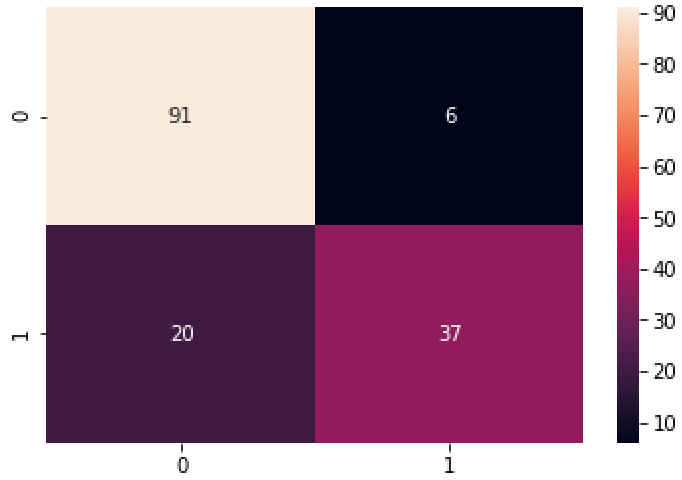
```
fpr, tpr, thresholds = roc_curve(y_test, nb_pred)
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0,1],[0,1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)'
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic(ROC)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



Confusion Matrix of "SVM"

```
sns.heatmap(confusion_matrix(y_test, sv_pred),annot=True,fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c1d9dc210>
```




```
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TP = cm[1,1]
```

```
TN, FP, FN,TP
```

```
(78, 19, 21, 36)
```

```
####
```

```
# classification Report of SVM
print('Classification Report of SVM: \n', classification_report(y_test,sv_pred,digits=4
```

```
Classification Report of SVM:
               precision    recall  f1-score   support

           0     0.8198    0.9381    0.8750        97
           1     0.8605    0.6491    0.7400        57

    accuracy                         0.8312       154
   macro avg     0.8401    0.7936    0.8075       154
weighted avg     0.8349    0.8312    0.8250       154
```

```
# Making the confusion Matrix of SVM
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score,roc_curve
```

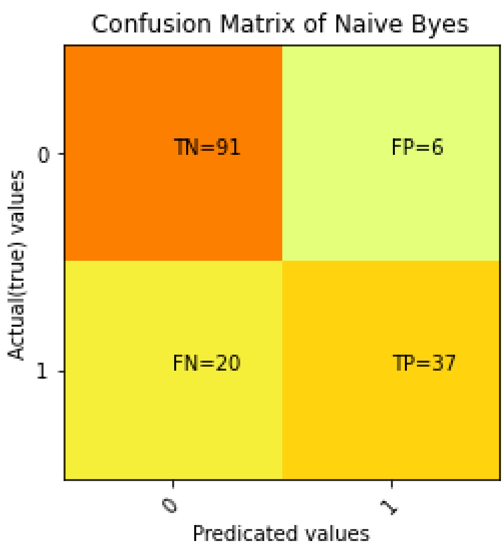
```
cm = confusion_matrix(y_test,sv_pred)


print('TN - True Negative: {}'.format(cm[0,0]))
print('FP - False Positive: {}'.format (cm[0,1]))
print('FN - False Negative: {}'.format(cm[1,0]))
print('TP - True Positive: {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[0,1],cm[1,0],cm[1,1]]),np
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm
```

```
TN - True Negative: 91
FP - False Positive: 6
FN - False Negative: 20
TP - True Positive: 37
Accuracy Rate: 100.0
Misclassification Rate: 16.883116883116884
```

```
83.11688311688312+16.883116883116884
```

```
100.0
```

```
import matplotlib.pyplot as plt
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0','1']
plt.title('Confusion Matrix of Naive Byes')
plt.ylabel('Actual(true) values')
plt.xlabel('Predicated values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'],['FN','TP']]
for i in range(2):
  for j in range(2):
     plt.text(j,i,str(s[i][j])+"="+str(cm[i][j]))
plt.show()
```




```
pd.crosstab(y_test, sv_pred, margins=False)
```

col_0	0	1
Outcome		
0	91	6
1	20	37

```
pd.crosstab(y_test, sv_pred, margins=True)
```

col_0	0	1	All
Outcome			
0	91	6	97
1	20	37	57
All	111	43	154

```
pd.crosstab(y_test, sv_pred, rownames=['Actual values'], colnames=['Predicted values'],
```

Predicted values	0	1	All
Actual values			
0	91	6	97
1	20	37	57
All	111	43	154

```
TP, FP
```

```
(36, 19)
```

```
Precision=TP/(TP+FP)
Precision
```

```
0.6545454545454545
```

```
37/(37+6)
```

```
0.8604651162790697
```

```
#print 'Precision score'

precision_Score = TP/float(TP+FP)*100
print('Precision score : {0:0.4f}'.format(precision_Score))
```

```
Precision score : 65.4545
```

```
from sklearn.metrics import precision_score
print("precision Score is:", precision_score(y_test,sv_pred,average='micro')*100)
print("Mircro Average precision Score is:",precision_score(y_test,sv_pred,average='micr
print("Marcro Average precision Score is:",precision_score(y_test,sv_pred,average='macr
print("Weighted Average precision Score is:",precision_score(y_test,sv_pred,average='we
print("precision Score on Non weighted score is:", precision_score(y_test,sv_pred,avera
```

```
precision Score is: 83.11688311688312
Mircro Average precision Score is: 83.11688311688312
Marcro Average precision Score is: 84.01424680494446
Weighted Average precision Score is: 83.48638581196721
precision Score on Non weighted score is: [81.98198198 86.04651163]
```

```
# classification Report of SVM
print('Classification Report of SVM: \n', classification_report(y_test,sv_pred,digits=4
```

```
Classification Report of SVM:
               precision    recall  f1-score   support

           0     0.8198    0.9381    0.8750        97
           1     0.8605    0.6491    0.7400        57

    accuracy                         0.8312       154
   macro avg     0.8401    0.7936    0.8075       154
weighted avg     0.8349    0.8312    0.8250       154
```

```
recall_score=TP/float(TP+FN)*100
print('recall_score',recall_score)
```

```
recall_score 63.1578947368421
```

```
TP, FN
```

```
(36, 21)
```

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_score:',recall_score(y_test,sv_pred)*100)
```

```
Recall or Sensitivity_score: 64.91228070175438
```

```
37/(37+20)
```

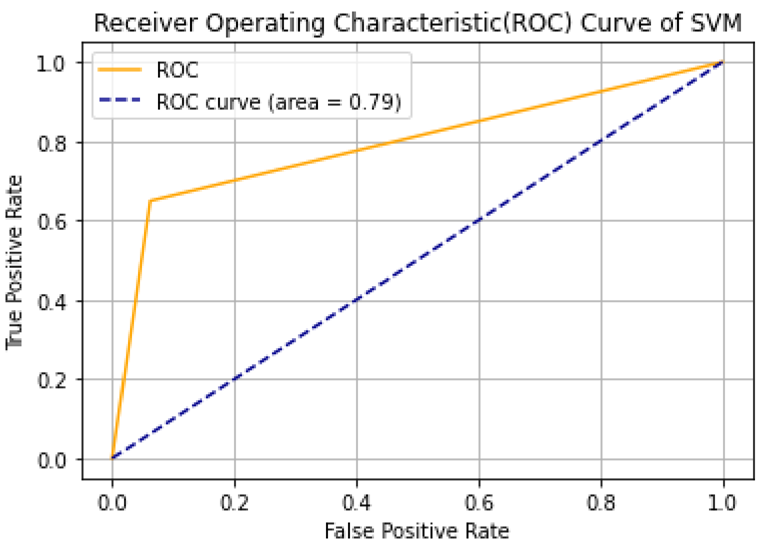
```
0.6491228070175439
```

ROC Curve & ROC AUC

```
# Area Under the Curve
auc = roc_auc_score(y_test, sv_pred)
print("ROC AUC SCORE of SVM is",auc)
```

```
ROC AUC SCORE of SVM is 0.7936335684572255
```

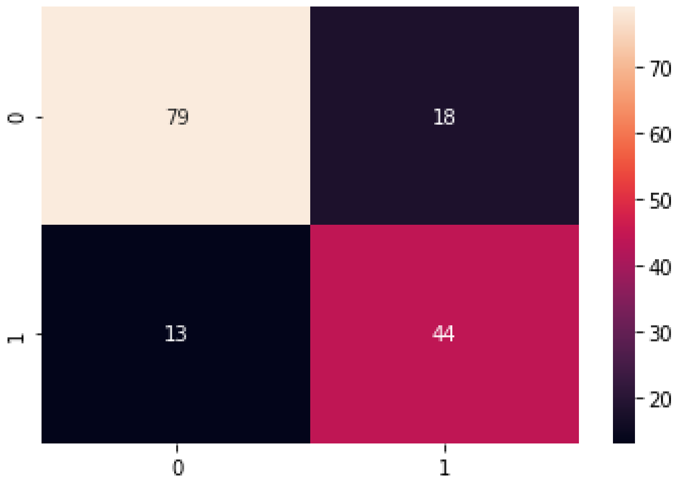
```
fpr, tpr, thresholds = roc_curve(y_test, sv_pred)
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0,1],[0,1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)'
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic(ROC)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



Confusion Matrix of "Decision Tree"

```
sns.heatmap(confusion_matrix(y_test, dt_pred),annot=True,fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c1d407e90>
```




```
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TP = cm[1,1]
```

```
TN, FP, FN,TP
```

```
(91, 6, 20, 37)
```

```
####
```

```
# classification Report of Decesion Tree
print('Classification Report of Decesion Tree: \n', classification_report(y_test,dt_pre
```

```
Classification Report of Decesion Tree: 
               precision    recall  f1-score   support

           0     0.8587    0.8144    0.8360        97
           1     0.7097    0.7719    0.7395        57

    accuracy                         0.7987       154
   macro avg     0.7842    0.7932    0.7877       154
weighted avg     0.8035    0.7987    0.8003       154
```

```
# Making the confusion Matrix of Decesion Tree
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score,roc_curve
cm = confusion_matrix(y_test,dt_pred)


print('TN - True Negative: {}'.format(cm[0,0]))
print('FP - False Positive:{}'.format (cm[0,1]))
print('FN - False Negative:{}'.format(cm[1,0]))
print('TP - True Positive:{}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[0,1],cm[1,0],cm[1,1]]),np
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm
```

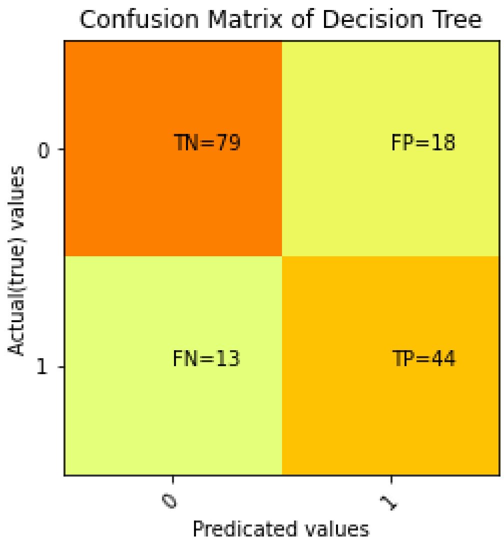
```
TN - True Negative: 79
FP - False Positive:18
FN - False Negative:13
TP - True Positive:44
Accuracy Rate: 100.0
Misclassification Rate: 20.12987012987013
```

```
78.57142857142857+21.428571428571427
```

```
100.0
```

```
import matplotlib.pyplot as plt
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0','1']
plt.title('Confusion Matrix of Decision Tree')
plt.ylabel('Actual(true) values')
plt.xlabel('Predicated values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'],['FN','TP']]
```

```
for i in range(2):
  for j in range(2):
      plt.text(j,i,str(s[i][j])+"="+str(cm[i][j]))
plt.show()
```




```
pd.crosstab(y_test, dt_pred, margins=False)
```

col_0	0	1
Outcome		
0	79	18
1	13	44

```
pd.crosstab(y_test, dt_pred, margins=True)
```

col_0	0	1	All
Outcome			
0	79	18	97
1	13	44	57
All	92	62	154

```
pd.crosstab(y_test, dt_pred, rownames=['Actual values'], colnames=['Predicted values'],
```

Predicted values	0	1	All
Actual values			
0	79	18	97
1	13	44	57
All	92	62	154

```
TP, FP
```

```
(37, 6)
```

```
Precision=TP/(TP+FP)
Precision
```

```
0.8604651162790697
```

```
42/(42+17)
```

```
0.711864406779661
```

```
#print precision score

precision_Score = TP/float(TP+FP)*100
print('Precision score : {0:0.4f}'.format(precision_Score))
```

```
Precision score : 86.0465
```

```
from sklearn.metrics import precision_score
print("precision Score is:", precision_score(y_test,dt_pred,average='micro')*100)
print("Mircro Average precision Score is:",precision_score(y_test,dt_pred,average='micr
print("Marcro Average precision Score is:",precision_score(y_test,dt_pred,average='macr
print("Weighted Average precision Score is:",precision_score(y_test,dt_pred,average='we
print("precision Score on Non weighted score is:", precision_score(y_test,dt_pred,avera
```

```
precision Score is: 79.87012987012987
Mircro Average precision Score is: 79.87012987012987
Marcro Average precision Score is: 78.41865357643759
Weighted Average precision Score is: 80.35395530136064
precision Score on Non weighted score is: [85.86956522 70.96774194]
```

```
# classification Report of Decesion Tree
print('Classification Report of Decesion Tree: \n', classification_report(y_test,dt_pre
```

```
Classification Report of Decesion Tree:
               precision    recall  f1-score   support

           0     0.8587    0.8144    0.8360        97
           1     0.7097    0.7719    0.7395        57

    accuracy                         0.7987       154
   macro avg     0.7842    0.7932    0.7877       154
weighted avg     0.8035    0.7987    0.8003       154
```

```
recall_score=TP/float(TP+FN)*100
print('recall_score',recall_score)
```

```
recall_score 64.91228070175438
```

```
TP, FN
```

```
1 TP, FN
```

```
(37, 20)
```

```
1 from sklearn.metrics import recall_score
2 print('Recall or Sensitivity_score:',recall_score(y_test,dt_pred)*100)
```

```
Recall or Sensitivity_score: 77.19298245614034
```

```
1 42/(42+15)
```

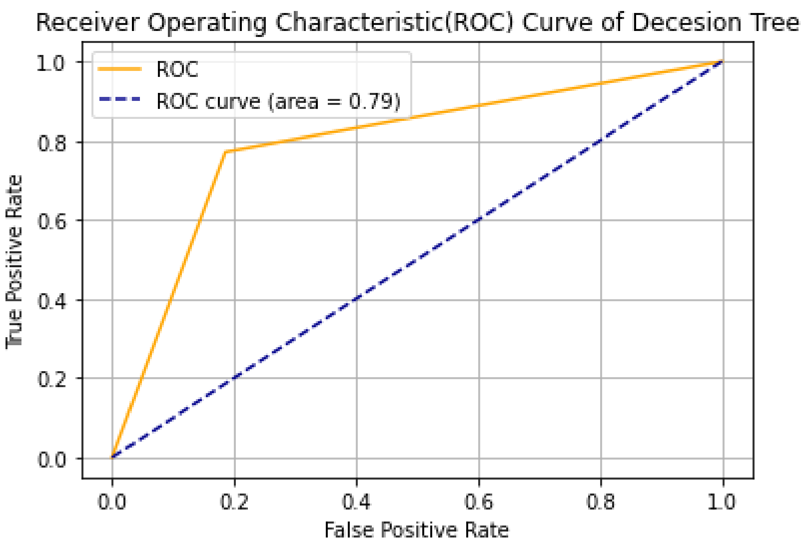
```
0.7368421052631579
```

ROC Curve & ROC AUC

```
1 # Area Under the Curve
2 auc = roc_auc_score(y_test, dt_pred)
3 print("ROC AUC SCORE of Decesion Tree is",auc)
```

```
ROC AUC SCORE of Decesion Tree is 0.7931814071260626
```

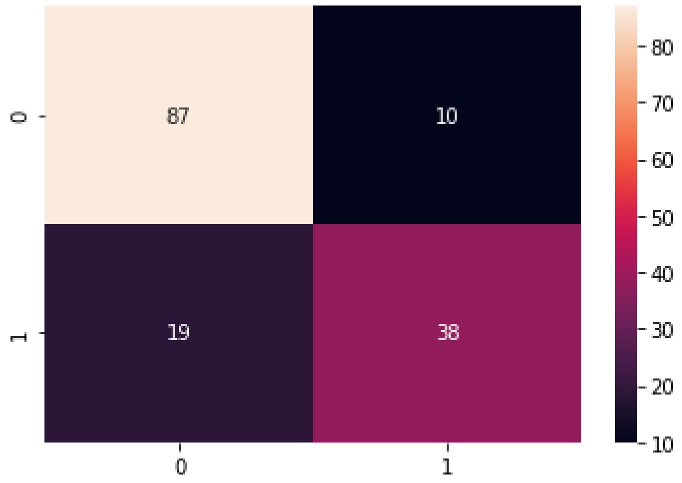
```
1 fpr, tpr, thresholds = roc_curve(y_test, dt_pred)
2 plt.plot(fpr, tpr, color='orange', label='ROC')
3 plt.plot([0,1],[0,1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)'
4 plt.xlabel('False Positive Rate')
5 plt.ylabel('True Positive Rate')
6 plt.title('Receiver Operating Characteristic(ROC) Curve of Decesion Tree')
7 plt.legend()
8 plt.grid()
9 plt.show()
```



Confusion Matrix of "Random Forest"

```
1 sns.heatmap(confusion_matrix(y_test, rf_pred),annot=True,fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c1d7b8c90>
```




```
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TP = cm[1,1]
```

```
TN, FP, FN,TP
```

```
(87, 10, 19, 38)
```

```
####
```

```
# classification Report of Random Forest
print('Classification Report of Random Forest: \n', classification_report(y_test,rf_pre
```

```
Classification Report of Random Forest:
               precision    recall  f1-score   support

           0     0.8208    0.8969    0.8571        97
           1     0.7917    0.6667    0.7238        57

    accuracy                         0.8117       154
   macro avg     0.8062    0.7818    0.7905       154
weighted avg     0.8100    0.8117    0.8078       154
```

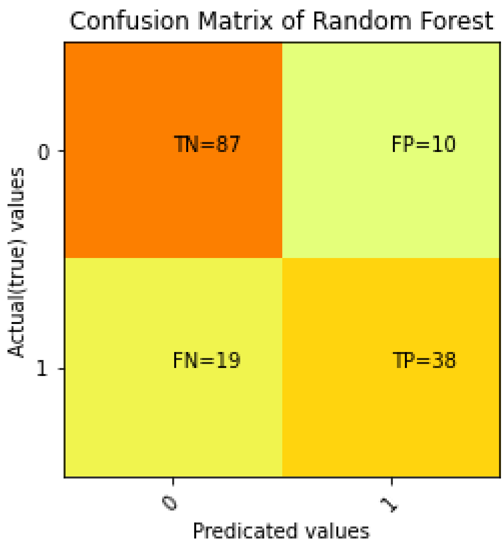
```
# Making the confusion Matrix of Random Forest
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score,roc_curve
cm = confusion_matrix(y_test,rf_pred)


print('TN - True Negative:  {}'.format(cm[0,0]))
print('FP - False Positive: {}'.format (cm[0,1]))
print('FN - False Negative: {}'.format(cm[1,0]))
print('TP - True Positive: {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[0,1],cm[1,0],cm[1,1]]),np
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm
```

```
TN - True Negative:  87
FP - False Positive: 10
FN - False Negative: 19
TP - True Positive: 38
Accuracy Rate: 100.0
Misclassification Rate: 18.83116883116883
```

```
79.22077922077922+20.77922077922078
```

```
100.0
```

```
import matplotlib.pyplot as plt
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0','1']
plt.title('Confusion Matrix of Random Forest')
plt.ylabel('Actual(true) values')
plt.xlabel('Predicated values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'],['FN','TP']]
for i in range(2):
  for j in range(2):
      plt.text(j,i,str(s[i][j])+"="+str(cm[i][j]))
plt.show()
```




```
pd.crosstab(y_test, rf_pred, margins=False)
```

col_0	**0**	**1**
Outcome		
0	87	10
1	19	38

```
pd.crosstab(y_test, rf_pred, margins=True)
```

col_0	0	1	All
Outcome			
0	87	10	97
1	19	38	57
All	106	48	154

```
pd.crosstab(y_test, rf_pred, rownames=['Actual values'], colnames=['Predicted values'],
```

Predicted values	0	1	All
Actual values			
0	87	10	97
1	19	38	57
All	106	48	154

```
TP, FP
```

```
(38, 10)
```

```
Precision=TP/(TP+FP)
Precision
```

```
0.7916666666666666
```

```
38/(38+10)
```

```
0.7916666666666666
```

```
#print precision score

precision_Score = TP/float(TP+FP)*100
print('Precision score : {0:0.4f}'.format(precision_Score))
```

```
Precision score : 79.1667
```

```
from sklearn.metrics import precision_score
print("precision Score is:", precision_score(y_test,dt_pred,average='micro')*100)
print("Mircro Average precision Score is:",precision_score(y_test,dt_pred,average='micr
print("Marcro Average precision Score is:",precision_score(y_test,dt_pred,average='macr
print("Weighted Average precision Score is:",precision_score(y_test,dt_pred,average='we
print("precision Score on Non weighted score is:", precision_score(y_test,dt_pred,avera
```

```
precision Score is: 79.87012987012987
Mircro Average precision Score is: 79.87012987012987
Marcro Average precision Score is: 78.41865357643759
```

```
Weighted Average precision Score is: 80.35395530136064
precision Score on Non weighted score is: [85.86956522 70.96774194]
```

```
# classification Report of Random Forest
print('Classification Report of Random Forest: \n', classification_report(y_test,rf_pre
```

```
Classification Report of Random Forest:
               precision    recall  f1-score   support

           0     0.8208    0.8969    0.8571        97
           1     0.7917    0.6667    0.7238        57

    accuracy                         0.8117       154
   macro avg     0.8062    0.7818    0.7905       154
weighted avg     0.8100    0.8117    0.8078       154
```

```
recall_score=TP/float(TP+FN)*100
print('recall_score',recall_score)
```

```
recall_score 66.66666666666666
```

```
TP, FN
```

```
(38, 19)
```

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_score:',recall_score(y_test,rf_pred)*100)
```

```
Recall or Sensitivity_score: 66.66666666666666
```

```
39/(39+19)
```

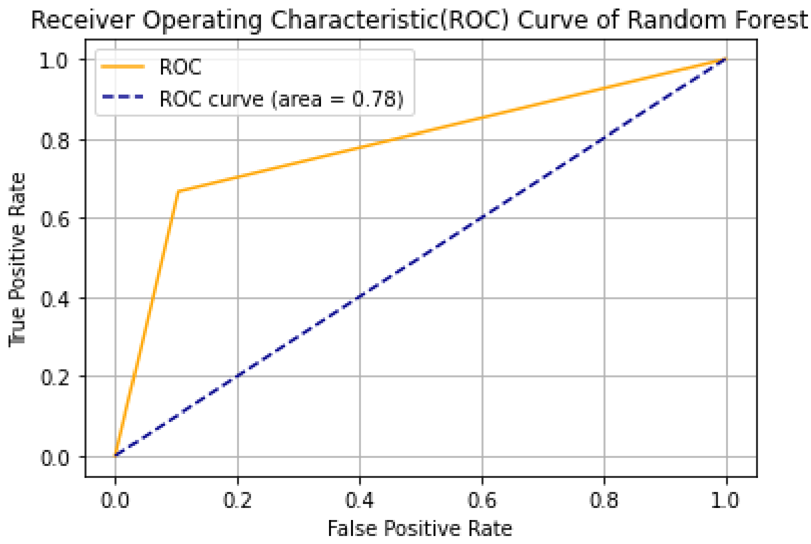
```
0.6724137931034483
```

ROC Curve & ROC AUC

```
# Area Under the Curve
auc = roc_auc_score(y_test, rf_pred)
print("ROC AUC SCORE of Random Forest is",auc)
```

```
ROC AUC SCORE of Random Forest is 0.781786941580756
```

```
fpr, tpr, thresholds = roc_curve(y_test, rf_pred)
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 of Random Forest')
plt.legend()
plt.grid()
plt.show()
```




```
# Confusion Matrix function

def conf_mtx(y_act,y_pred):
    cm=metrics.confusion_matrix(y_act, y_pred, labels=[1, 0])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Diabetic","Non-Diabetic"]],
                  columns = [i for i in ["Predict Diabetic","Predict Non-Diabetic"]])
    plt.figure(figsize = (6,6))
    plt.title("Confusion Matrix")
    sns.heatmap(df_cm, annot=True ,fmt='g')

    Score_Accuracy = "%.2f%%" %(metrics.accuracy_score(y_act,y_pred)*100)
    Score_Recall = "%.2f%%" %(metrics.recall_score(y_act,y_pred)*100)
    Score_Precision = "%.2f%%" %(metrics.precision_score(y_act,y_pred)*100)

    print("Model Accuracy Score: " + Score_Accuracy)
    print("Model Recall Score: " + Score_Recall)
    print("Model Precision Score: " + Score_Precision)

    return Score_Accuracy,Score_Recall,Score_Precision
```

```
# Prepare an empty summary dataframe to append the data of the various models for compa
summary = pd.DataFrame(columns=('Model', 'Training Accuracy', 'Test Accuracy Score','Te
                                'Test Precision Score', 'AUC'))
```

```
# For building a function for performing ML algos testing

def ML_test(Mdl,Param_grid):
    if bool(Param_grid):
        Mdl = GridSearchCV(Mdl,Param_grid,cv=10)
        Mdl.fit(X_train_sm,y_train_sm)
        Mdl_params = Mdl.best_params_
        Mdl_train_sc = Mdl.cv_results_['mean_test_score'].mean()
        Mdl_test_sc = Mdl.score(X_test_scaled,y_test)
        probas = Mdl.predict_proba(X_test_scaled)

        print("Best fit parameter is: " + str(Mdl_params))
```

```
    else:
        Mdl = Mdl
        Mdl.fit(X_train_sm,y_train_sm)
        Mdl_train_sc = round(Mdl.score(X_train_sm,y_train_sm),4)
        Mdl_test_sc = round(Mdl.score(X_test_scaled,y_test),4)
        probas = Mdl.predict_proba(X_test_scaled)

    y_pred = Mdl.predict(X_test_scaled)

    print("Training score is: " + str(Mdl_train_sc))
    print("Test Mean score is: " + str(Mdl_test_sc))

    Score_Accuracy,Score_Recall,Score_Precision = conf_mtx(y_test,y_pred)
    Mdl_train_sc = "%.2f%%" % (Mdl_train_sc*100)

    # Calculating AUC
    fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])
    roc_auc = round(auc(fpr, tpr),4)
    print("Area under the ROC curve : " + str(roc_auc))

    return Mdl_train_sc, Score_Accuracy, Score_Recall, Score_Precision, roc_auc
```

```
# Scaling the x training and testing dataset
scaler = preprocessing.StandardScaler().fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Logistic Regression Model

Mdl_LogReg = LogisticRegression(solver="liblinear")

model_name = "LogisticRegression"
Mdl_train_sc = "77.36156351791531"
Score_Accuracy = "77.27272727272727"
Score_Recall = "57.89473684210527"
Score_Precision = "75.27272727272727"
roc_auc = "0.7327726532826913"

Param_grid_LogReg =  {'penalty': ['l1','l2'], 'C': np.linspace(0.1,1.1,10)}


summary = summary.append({'Model' : model_name, 'Training Accuracy' : Mdl_train_sc, 'Te
                       'Test Recall Score' : Score_Recall, 'Test Precision Score' : Sco
                         ignore_index=True)
```

```
# KNN Model

Mdl = KNeighborsClassifier()

model_name = "KNN"
Mdl_train_sc = "81.10749185667753"
```

```
Score_Accuracy = "74.67532467532467"
Score_Recall = "57.89473684210527"
Score_Precision = "68.7500"
roc_auc = "0.71215409865816603"

Param_grid_kNeigh =  {'n_neighbors': list(np.arange(3,8)), 'metric': ['euclidean','manh

summary = summary.append({'Model' : model_name, 'Training Accuracy' : Mdl_train_sc, 'Te
                       'Test Recall Score' : Score_Recall, 'Test Precision Score' : Sco
                         ignore_index=True)
```

```
#Naive Bayes Model

Mdl = GaussianNB()

model_name = "Naive Byes"
Mdl_train_sc = "74.2671009771987"
Score_Accuracy = "74.025974025977402"
Score_Recall = "63.1578947368421"
Score_Precision = "65.4545"
roc_auc = "0.7178513293543136"

summary = summary.append({'Model' : model_name, 'Training Accuracy' : Mdl_train_sc, 'Te
                       'Test Recall Score' : Score_Recall, 'Test Precision Score' : Sco
                         ignore_index=True)
```

```
#SVM Model

Mdl = SVC(probability=True)

model_name = "SVM"
Mdl_train_sc = "81.92182410423453"
Score_Accuracy = "83.11688311688312"
Score_Recall = "64.9122807017543"
Score_Precision = "86.0465"
roc_auc = "0.79386335684572255"

Param_grid_SVC =  {'C': np.linspace(0.1,1.1,10), 'kernel': ['linear','poly','rbf',]}

summary = summary.append({'Model' : model_name, 'Training Accuracy' : Mdl_train_sc, 'Te
                       'Test Recall Score' : Score_Recall, 'Test Precision Score' : Sco
                         ignore_index=True)
```

```
# Decision Tree Model

Mdl = DecisionTreeClassifier(random_state=1)

model_name = "Decision Tree"
Mdl_train_sc = "100.0"
Score_Accuracy = "75.97402597402598"
Score_Recall = "73.68421052631578"
Score_Precision = "79.22077922077922"
roc_auc = "0.7807921866521975"
```

```

Param_grid_dt = {'criterion':['gini','entropy'],'max_depth': [3, 4, 5, 6, 7, 8],
             'min_impurity_decrease': [0.0001, 0.0003, 0.0005, 0.0007, 0.009]}


summary = summary.append({'Model' : model_name, 'Training Accuracy' : Mdl_train_sc, 'Te
                       'Test Recall Score' : Score_Recall, 'Test Precision Score' : Sco
                         ignore_index=True)
```

```
# Random Forest Model

Mdl = RandomForestClassifier(random_state=1,n_estimators=100)

model_name = "Random Forest Classifier"
Mdl_train_sc = "100.0"
Score_Accuracy = "81.16883116883116"
Score_Recall = "66.66666666666666"
Score_Precision = "79.1667"
roc_auc = "0.781786941580756"

Param_grid_rf = {'criterion':['gini','entropy'],'max_depth': [3, 4, 5, 6, 7, 8],
             'min_impurity_decrease': [0.0001, 0.0003, 0.0005, 0.0007, 0.009]}


summary = summary.append({'Model' : model_name, 'Training Accuracy' : Mdl_train_sc, 'Te
                       'Test Recall Score' : Score_Recall, 'Test Precision Score' : Sco
                         ignore_index=True)
```

```
summary
```

	Model	Training Accuracy	Test Accuracy Score	Test Recall Score	Test
0	LogisticRegression	77.36156351791531	77.27272727272727	57.89473684210527	75.2727
1	KNN	81.10749185667753	74.67532467532467	57.89473684210527	
2	Naive Byes	74.2671009771987	74.025974025977402	63.1578947368421	
3	Naive Byes	74.2671009771987	74.025974025977402	63.1578947368421	
4	SVM	81.92182410423453	83.11688311688312	64.9122807017543	
5	Decision Tree	100.0	75.97402597402598	73.68421052631578	79.2207
6	Random Forest	100.0	81.16883116883116	66.66666666666666	

Conclusion:

Based on the comparison between the various algorithms used, SVM seems to produce the best results to me.

-By P.Akanksh

IBS(ICFAI BUSSINES SCHOOL),Hyderabad