SURUTHI S

225229141

PML LAB 5

Diabetes Classification Using Logistic Regression

```
import pandas as pd
import numpy as np
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
db = pd.read_csv("/content/diabetes.csv")
```

+ Code + Text

STEP 1 : UNDERSTAND DATA

```
db.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

```
db.shape
```

```
(768, 9)
```

Pregnancies - Number of time Pregnant

Glucose- Plasma Glucose concentration over 2 hours in an oral glucose tolerance test

BloodPressure - Diastolic BloodPressure

SkinThickness - Triceps Skin Fold Thickness

Insulin - 2 - Hour Serum Insulin

BMI - Body Mass Index (weight in kg / (height in m)2)

DiabetesPedigreeFunction - a function which scores likelihood of diabetes based on Family History

```
db.columns
```

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

```
db.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
```

```
db.dtypes
```

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

```
db.value_counts()
```

```
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI   DiabetesPedigreeFunction  Age  Outcome
0            57       60             0              0        21.7  0.735                     67   0          1
             67       76             0              0        45.3  0.194                     46   0          1
5            103      108            37             0        39.2  0.305                     65   0          1
             104      74             0              0        28.8  0.153                     48   0          1
             105      72             29             325      36.9  0.159                     28   0          1
                                                                                                            ..
2            84       50             23             76       30.4  0.968                     21   0          1
             85       65             0              0        39.6  0.930                     27   0          1
             87       0              23             0        28.9  0.773                     25   0          1
                      58             16             52       32.7  0.166                     25   0          1
17           163      72             41             114      40.9  0.817                     47   1          1
Length: 768, dtype: int64
```

```
db.isnull().sum()
```

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

STEP 2 : BUILDING LOGISTIC REGRESSION MODEL

```
db.columns
```

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

```
features = db.drop("Outcome",axis = 1)
features
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

```
label = db[['Outcome']]
label
```

	Outcome
0	1
1	0
2	1
3	0
4	1
...	...
763	0
767	0

```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import accuracy_score
```

```
sss= StratifiedShuffleSplit(n_splits=4,test_size = 0.25,random_state = 42)
```

```
sss.get_n_splits(features,label)
```

```
4
```

```
scores = []
for train_index,test_index in sss.split(features,label):
  x_train,x_test = features.iloc[train_index],features.iloc[test_index]
  y_train,y_test = label.iloc[train_index],label.iloc[test_index]
```

```
from sklearn.linear_model import LogisticRegression
```

```
log_reg = LogisticRegression()
```

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

```
LogisticRegression()
```

```
log_reg.fit(x_train, y_train)
y_pred = log_reg.predict(x_test)
```

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

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
```

```
mean_squared_error(y_pred,y_test)
```

```
0.234375
```

STEP 3 - PREDICT ON NEW SAMLPE

```
new_person = [[6,200,90,10,25,23.3,0.672,42]]

log_reg.predict(new_person)
```

```
array([1])
```

COMPUTE CLASIFICATION METRICS

```
def calculate_score(test,pred):
  accuracy = accuracy_score(test, pred)
  precision =precision_score(test,pred)
  recall  = recall_score(test, pred)
  roc_auc = roc_auc_score(test, pred)

  return accuracy,precision,recall,roc_auc
```

```
calculate_score(y_test, y_pred)
```

```
(0.765625, 0.6896551724137931, 0.5970149253731343, 0.7265074626865671)
```

```
print('accuracy_score : ',accuracy_score(y_test, y_pred))
print('precision_score : ',precision_score(y_test,y_pred))
print('recall_score : ',recall_score(y_test, y_pred))
print('roc_auc_score : ',roc_auc_score(y_test, y_pred))
```

```
accuracy_score :  0.765625
precision_score :  0.6896551724137931
recall_score :  0.5970149253731343
roc_auc_score :  0.7265074626865671
```

STEP 4 - CORRELATION

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import  ConfusionMatrixDisplay
```

```
db.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesF
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	

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

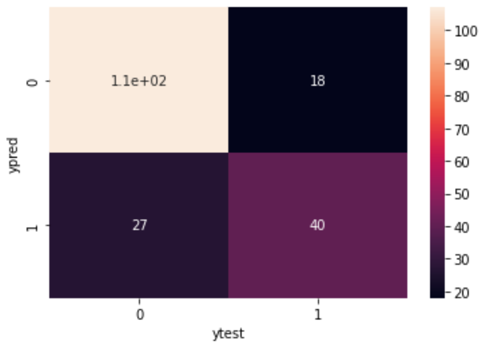
```
cm
```

```
array([[107,  18],
       [ 27,  40]])
```

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sns.heatmap(cm,annot=True)
plt.xlabel("ytest")
plt.ylabel("ypred")
```

```
Text(33.0, 0.5, 'ypred')
```



INTERPRETATION -

True negative : 40 Samples are correctly identified as 1(having diabetes)

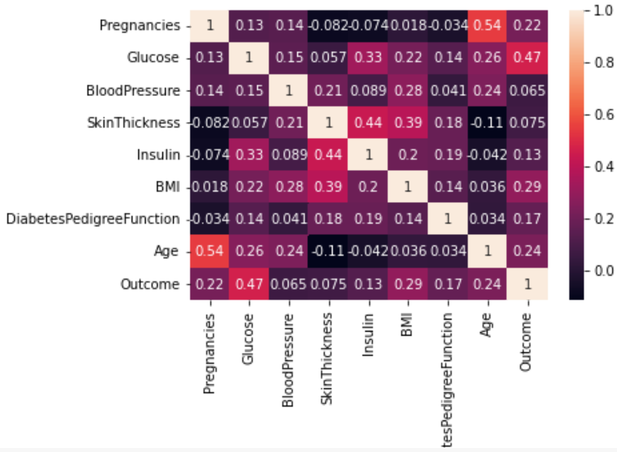
True positive : 107 Samples are correctly identified as 0 (not having diabetes)

False negative :27 samples are misclassified as 1(having diabetes) despite being 0(not having diabetes)

False positive : 18 samples are misclassified as 0 (not having diabetes)despite being 1(having diabetes)

```
sns.heatmap(db.corr(),annot=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f8d60062a00>




```
from sklearn.metrics import classification_report

print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       125
           1       0.69      0.60      0.64        67

    accuracy                           0.77       192
   macro avg       0.74      0.73      0.73       192
weighted avg       0.76      0.77      0.76       192
```

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

STEP 5 - MINMAX SCALER

```
minmax = MinMaxScaler()
```

```
#xtrain
scaled_x_train_minmax = minmax.fit_transform(x_train)
```

```
# xtest
scaled_x_test_minmax = minmax.transform(x_test)
```

```
log_reg_minmax = LogisticRegression()
```

```
log_reg_minmax.fit(scaled_x_train_minmax,y_train)
```

```
LogisticRegression()
```

```
y_pred_minmax = log_reg_minmax.predict(scaled_x_test_minmax)
```

```
calculate_score(y_test,y_pred_minmax)
```

```
(0.765625, 0.7115384615384616, 0.5522388059701493, 0.7161194029850746)
```

STEP 6 - STANDARD SCALER

```
scale = StandardScaler()
```

```
#xtrain
scaled_x_train_norm = scale.fit_transform(x_train)
```

```
# xtest
scaled_x_test_norm = scale.transform(x_test)
```

```
log_reg_norm = LogisticRegression()
```

```
log_reg_norm.fit(scaled_x_train_norm,y_train)
```

```
LogisticRegression()
```

```
y_pred_norm = log_reg_norm.predict(scaled_x_test_norm)
```

```
calculate_score(y_test,y_pred_norm)
```

    (0.765625, 0.6964285714285714, 0.582089552238806, 0.7230447761194031)

```
calculate_score(y_test,y_pred)
```

    (0.765625, 0.6896551724137931, 0.5970149253731343, 0.7265074626865671)

```
calculate_score(y_test,y_pred_minmax)
```

    (0.765625, 0.7115384615384616, 0.5522388059701493, 0.7161194029850746)

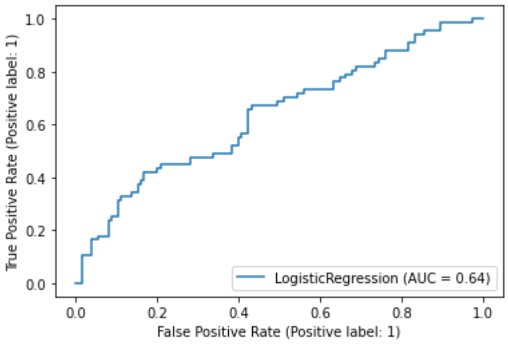
```
auc_ss = roc_auc_score(y_test,y_pred_norm)
auc_ss
```

    0.7230447761194031

STEP 7 - PLOT ROC CURVE

```
from sklearn.metrics import plot_roc_curve, auc
plot_roc_curve(log_reg,scaled_x_test_norm,y_test)
```

    <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f8d5fedc5e0>



predict() is used to predict the actual class (in your case one of 0, 1, or 2).

predict_proba() is used to predict the class probabilities From the example output that you shared,

predict() would output class 0 since the class probability for 0 is 0.6. [0.6, 0.2, 0.2] is the output of predict_proba that simply denotes that the class probability for classes 0, 1, and 2 are 0.6, 0.2, and 0.2 respectively

```
pred_prob=log_reg_minmax.predict_proba(scaled_x_test_norm)
```

```
# scaled_x_test_norm # co eff
# y_test
# pred_prob1 # calculating class - probability for each sample
```

```
from sklearn.metrics import roc_curve
```

Returns: --fpr ndarray of shape (>2,)

Increasing false positive rates such that element i is the false positive rate of predictions with score >= thresholds[i].

--tpr ndarray of shape (>2,)

Increasing true positive rates such that element i is the true positive rate of predictions with score >= thresholds[i].

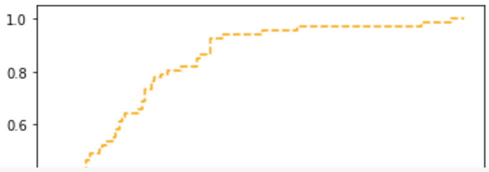
--thresholds ndarray of shape = (n_thresholds,)

Decreasing thresholds on the decision function used to compute fpr and tpr. thresholds[0] represents no instances being predicted and is arbitrarily set to max(y_score) + 1.

```
fpr, tpr, thresholds = roc_curve(y_test, pred_prob[:,1], pos_label=1)
```

```
plt.plot(fpr,tpr,linestyle='--',color='orange',label='MinMaxScaler values')
```

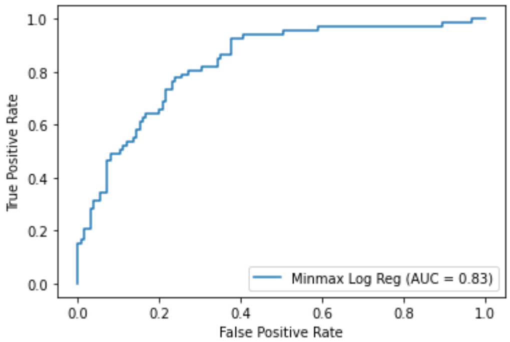
```
[<matplotlib.lines.Line2D at 0x7f8d73794a00>]
```




```
roc_auc = auc(fpr, tpr)
```

```
from sklearn.metrics import RocCurveDisplay
```

```
display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc,
                                  estimator_name='Minmax Log Reg')
display.plot()
plt.show()
```



Step-8. Comparison with KNN classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=4)
knn=knn.fit(x_train,y_train)
```

```
knn_y_pred=knn.predict(x_test)
```

```
from sklearn.preprocessing import MinMaxScaler
m=MinMaxScaler()
m_x_train=m.fit_transform(x_train)
m_x_train
```

```
array([[0.        , 0.52525253, 0.52459016, ..., 0.50074516, 0.18201285,
        0.01666667],
       [0.47058824, 0.5959596 , 0.59016393, ..., 0.3442623 , 0.59571734,
        0.41666667],
       [0.58823529, 0.9040404 , 0.57377049, ..., 0.52309985, 0.04925054,
        0.26666667],
       ...,
       [0.        , 0.54040404, 0.49180328, ..., 0.39344262, 0.02055675,
        0.03333333],
       [0.05882353, 0.61616162, 0.52459016, ..., 0.52309985, 0.25995717,
        0.15      ],
       [0.64705882, 0.42929293, 0.60655738, ..., 0.4485842 , 0.09207709,
        0.23333333]])
```

```
m_x_test=m.transform(x_test)
m_x_test
```

```
array([[0.17647059, 0.65151515, 0.75409836, ..., 0.54247392, 0.37815846,
        0.18333333],
       [0.05882353, 0.46969697, 0.45901639, ..., 0.33532042, 0.14218415,
        0.01666667],
       [0.11764706, 0.78282828, 0.42622951, ..., 0.57675112, 0.06638116,
        0.06666667],
       ...,
       [0.05882353, 0.43939394, 0.49180328, ..., 0.55439642, 0.18158458,
        0.01666667],
       [0.05882353, 0.57575758, 0.54098361, ..., 0.56780924, 0.08736617,
        0.        ],
       [0.58823529, 0.67171717, 0.55737705, ..., 0.4023845 , 0.06852248,
        0.25      ]])
```

```
m_knn=KNeighborsClassifier()
m_knn=m_knn.fit(m_x_train,y_train)
```

```
m_y_pred=m_knn.predict(m_x_test)
m_y_pred
```

```
array([1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1])
```

Classification Metrics

```
calculate_score(y_test,m_y_pred)
```

```
(0.734375, 0.66, 0.4925373134328358, 0.6782686567164178)
```

AUC Scores

```
knn_auc=print(roc_auc_score(y_test,m_y_pred))
knn_auc
```

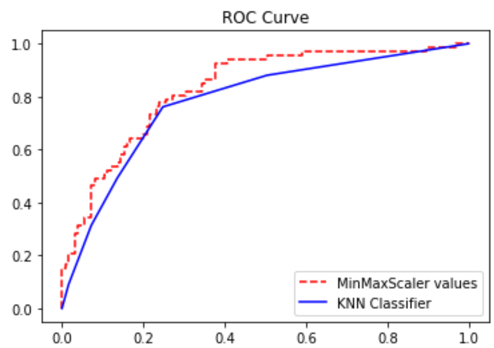
```
0.6782686567164178
```

Step-9. Update ROC Curve

```
pred_p2=m_knn.predict_proba(m_x_test)
```

```
from sklearn.metrics import roc_curve
fpr2,tpr2,thresh2=roc_curve(y_test,pred_p2[:,1],pos_label=1)
```

```
plt.plot(fpr,tpr,linestyle='--',color='red',label='MinMaxScaler values')
plt.plot(fpr2,tpr2,linestyle='-',color='blue',label='KNN Classifier')
plt.title('ROC Curve')
plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()
```



Step-10. Regularization

```
from sklearn.linear_model import LogisticRegressionCV
model1=LogisticRegressionCV(Cs=10,cv=4,penalty='l1',solver='liblinear')
model2=LogisticRegressionCV(Cs=10,cv=4,penalty='l2')
```

```
model1.fit(scaled_x_train_minmax,y_train)
model2.fit(scaled_x_train_minmax,y_train)
```

```
LogisticRegressionCV(cv=4)
```

```
rg_y_pred1 = model1.predict(scaled_x_test_minmax)
rg_y_pred2 = model2.predict(scaled_x_test_minmax)
```

AUC SCORE OF L1

```
from sklearn.metrics import roc_auc_score
l1_auc = roc_auc_score(y_test, rg_y_pred1)
l1_auc = (' LOR L1 MINMAX AUC', l1_auc)
l1_auc
```

```
(' LOR L1 MINMAX AUC', 0.7230447761194031)
```

AUC SCORE OF L2

```
from sklearn.metrics import roc_auc_score
l2_auc = roc_auc_score(y_test, rg_y_pred2)
l2_auc = (' LOR L2 MINMAX AUC', l2_auc)
l2_auc
```

```
(' LOR L2 MINMAX AUC', 0.7155820895522387)
```

Step 11: Update ROC curve

```
pred_prb7 = model1.predict_proba(scaled_x_test_minmax)
pred_prb8 = model2.predict_proba(scaled_x_test_minmax)
fpr,tbr,threshold = roc_curve(y_test, pred_prob[:,1],pos_label=1)
fpr1,tbr1,threshold1 = roc_curve(y_test, pred_p2[:,1],pos_label=1)
fpr2,tbr2,threshold2= roc_curve(y_test, pred_prb7[:,1],pos_label=1)
fpr3,tbr3,threshold3 = roc_curve(y_test, pred_prb8[:,1],pos_label=1)
```

```
plt.plot(fpr, tbr, linestyle='-', color='red', label='LogisticRegression')
plt.plot(fpr1, tbr1, linestyle='-', color='orange', label='KNN')
plt.plot(fpr3, tbr3, linestyle='-', color='green', label='l2')
plt.plot(fpr2, tbr2, linestyle='-', color='black', label='l1')
plt.annotate(xy=[0.5,0.3],s= auc_ss)
plt.annotate(xy=[0.5,0.2],s= knn_auc)
plt.annotate(xy=[0.5,0.1],s= l1_auc)
plt.annotate(xy=[0.7,0],s= l2_auc)
plt.title('Receiver Operating Characteristic')
plt.legend(loc = 'best')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```