Name:P.Asha Belcilda

Roll no:225229104

Lab5: Diabetes Classification using Logistic Regression

Step1.[Understand Data]

In [3]:
```
import pandas as pd
```

In [4]:
```
data=pd.read_csv("diabetes.csv")
print(data)
```

```
     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
5              5      116             74              0        0  25.6
6              3       78             50             32       88  31.0
7             10      115              0              0        0  35.3
8              2      197             70             45      543  30.5
9              8      125             96              0        0   0.0
10             4      110             92              0        0  37.6
11            10      168             74              0        0  38.0
12            10      139             80              0        0  27.1
13             1      189             60             23      846  30.1
14             5      166             72             19      175  25.8
15             7      100              0              0        0  30.0
16             0      118             84             47      230  45.8
17             7      107             74              0        0  29.6
```

In [5]:
```
#head
data.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
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	

In [7]:
```
#shape
data.shape
```

Out[7]:
```
(768, 9)
```

In [8]:
```
#columns
data.shape[1]
```

Out[8]:
```
9
```

In [11]:
```
#dtype
data.dtypes
```

Out[11]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [12]:
```
#info
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [52]:
```
#value_counts.
data.count()
```

Out[52]:
```
Pregnancies                 768
Glucose                     768
BloodPressure               768
SkinThickness               768
Insulin                     768
BMI                         768
DiabetesPedigreeFunction    768
Age                         768
Outcome                     768
dtype: int64
```

Step2.[Build Logistic Regression Model]

In [32]:
```
X=data.drop('Outcome',axis=1)
```

In [34]:
```
X.head()
```

Out[34]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
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	

In [37]:
```
y=data['Outcome'].values
y
```

Out[37]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0],
      dtype=int64)
```

In [40]:
```
from sklearn.model_selection import StratifiedShuffleSplit
a=StratifiedShuffleSplit(n_splits=4,test_size=0.25,random_state=42)
```

In [41]:
```
a.get_n_splits(X,y)
```

Out[41]:
```
4
```

In [47]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,stratify=y,test_size=.25,rando
```

In [48]:
```
from sklearn.linear_model import LogisticRegression
```

In [53]:
```
LOR=LogisticRegression(penalty='l2',C=10.0)
LOR=LOR.fit(X_train,y_train)
```

In [54]:
```
y_pred=LOR.predict(X_test)
y_pred
```

Out[54]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0], dtype=int64)
```

Step-3. [Predict on a new sample]

In [58]:
```
new=LOR.predict([[6,200,90,10,25,23.3,.672,42]])
if new==0:
    print("Non-diabetic patient",new)
else:
    print("Diabetic patient",new)
```

```
Diabetic patient [1]
```

Step3.[Compute Classification Metrics]

In [60]:
```
#accuracy
def accuracy(actual,pred):
    return sum(actual==pred)/float(actual.shape[0])
```

In [61]:
```
accuracy_score=accuracy(y_test,y_pred)
accuracy_score
```

Out[61]:
```
0.734375
```

In [62]:
```
#precision
from sklearn.metrics import precision_score
print(precision_score(y_test,y_pred))
```

```
0.6481481481481481
```

In [63]:
```
#recall
from sklearn.metrics import recall_score
print(recall_score(y_test,y_pred))
```

```
0.5223880597014925
```

In [68]:
```
#AUC scores
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y_test,y_pred))
```

```
0.6851940298507463
```

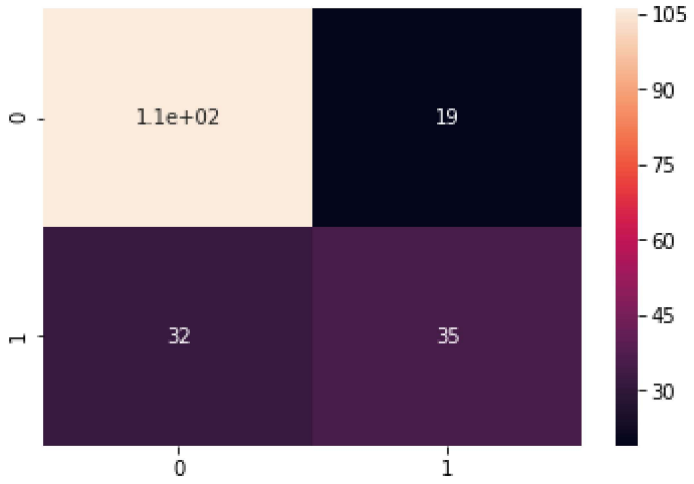
Step-4. [Understand Correlation]

In [69]:
```
from sklearn.metrics import confusion_matrix
b=confusion_matrix(y_test,y_pred)
b
```

Out[69]:
```
array([[106,  19],
       [ 32,  35]], dtype=int64)
```

In [70]:
```
import seaborn as sns
sns.heatmap(b, annot=True)
```

Out[70]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2434875f5c0>
```



Step-5. [Normalization using MinMaxScaler and rebuild LOR]

In [75]:
```
from sklearn.preprocessing import MinMaxScaler
mm=MinMaxScaler()
mm_X_train=mm.fit_transform(X_train)
mm_X_train
```

Out[75]:
```
array([[0.05882353, 0.6080402 , 0.63934426, ..., 0.58122206, 0.07884187,
        0.11666667],
       [0.70588235, 0.44221106, 0.60655738, ..., 0.52608048, 0.13095768,
        0.45      ],
       [0.05882353, 0.54271357, 0.49180328, ..., 0.5290611 , 0.14743875,
        0.05      ],
       ...,
       [0.05882353, 0.48743719, 0.57377049, ..., 0.56780924, 0.0596882 ,
        0.15      ],
       [0.52941176, 0.7839196 , 0.70491803, ..., 0.51117735, 0.4922049 ,
        0.35      ],
       [0.23529412, 0.72361809, 0.47540984, ..., 0.43964232, 0.09042316,
        0.26666667]])
```

In [76]:
```
mm_X_test=mm.transform(X_test)
mm_X_test
```

Out[76]:
```
array([[0.76470588, 0.52261307, 0.59016393, ..., 0.46497765, 0.16971047,
        0.28333333],
       [0.23529412, 0.63819095, 0.72131148, ..., 0.51415797, 0.22895323,
        0.11666667],
       [0.11764706, 0.47236181, 0.62295082, ..., 0.4709389 , 0.25167038,
        0.03333333],
       ...,
       [0.        , 0.53266332, 0.57377049, ..., 0.58718331, 0.23207127,
        0.01666667],
       [0.29411765, 0.62311558, 0.60655738, ..., 0.50670641, 0.06057906,
        0.28333333],
       [0.17647059, 0.64321608, 0.59016393, ..., 0.4828614 , 0.20712695,
        0.1       ]])
```

In [77]:
```
mm_lor=LogisticRegression()
mm_lor=mm_lor.fit(mm_X_train,y_train)
```

In [78]:
```
mm_y_pred=mm_lor.predict(mm_X_test)
mm_y_pred
```

Out[78]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0], dtype=int64)
```

In [80]:
```
#accuracy
def accuracy(actual,pred):
    return sum(actual==pred)/float(actual.shape[0])
```

In [81]:
```
accuracy_score=accuracy(y_test,mm_y_pred)
accuracy_score
```

Out[81]:
```
0.7395833333333334
```

In [82]:
```
#precision
print(precision_score(y_test,mm_y_pred))
```

```
0.6888888888888889
```

In [83]:
```
#recall
print(recall_score(y_test,mm_y_pred))
```

```
0.4626865671641791
```

In [84]:
```
#auc scores
mm_auc=print(roc_auc_score(y_test,mm_y_pred))
mm_auc
```

```
0.6753432835820895
```

Step-6: [Normalization using StandardScaler and rebuild LOR]

In [85]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
ss_X_train=ss.fit_transform(X_train)
ss_X_train
```

Out[85]:
```
array([[-0.85547074,  0.00732864,  0.47259835, ...,  0.88301955,
        -0.65845729, -0.46648591],
       [ 2.46780492, -1.03224482,  0.2585074 , ...,  0.41193433,
        -0.30699915,  1.21865604],
       [-0.85547074, -0.4022003 , -0.49081095, ...,  0.43739839,
        -0.19585426, -0.8035143 ],
       ...,
       [-0.85547074, -0.74872478,  0.04441644, ...,  0.76843126,
        -0.78762567, -0.29797171],
       [ 1.56145701,  1.10990656,  0.90078026, ...,  0.28461399,
         2.12917653,  0.71311346],
       [ 0.05087717,  0.73187984, -0.59785643, ..., -0.3265236 ,
        -0.58035548,  0.29182797]])
```

In [86]:
```
ss_X_test=ss.transform(X_test)
ss_X_test
```

Out[86]:
```
array([[ 2.76992089, -0.5282092 ,  0.15146192, ..., -0.11007904,
        -0.04565848,  0.37608507],
       [ 0.05087717,  0.196342  ,  1.00782574, ...,  0.31007806,
         0.35386232, -0.46648591],
       [-0.55335477, -0.84323146,  0.36555287, ..., -0.0591509 ,
         0.50706202, -0.8877714 ],
       ...,
       [-1.15758671, -0.46520475,  0.04441644, ...,  0.93394769,
         0.37488973, -0.97202849],
       [ 0.35299314,  0.10183532,  0.2585074 , ...,  0.24641789,
        -0.78161784,  0.37608507],
       [-0.2512388 ,  0.22784423,  0.15146192, ...,  0.04270536,
         0.20667045, -0.55074301]])
```

In [88]:
```
ss_lor=LogisticRegression()
ss_lor.fit(ss_X_train,y_train)
ss_y_pred=ss_lor.predict(ss_X_test)
```

In [90]:
```
#accuracy
def accuracy(actual,pred):
    return sum(actual==pred)/float(actual.shape[0])
```

In [91]:
```
ss_accuracy_score=accuracy(y_test,ss_y_pred)
ss_accuracy_score
```

Out[91]: 0.7291666666666666

In [92]:
```
#precision
print(precision_score(y_test,ss_y_pred))
```

```
0.6363636363636364
```

In [93]:
```
#recall
print(recall_score(y_test,ss_y_pred))
```

```
0.5223880597014925
```

In [95]:
```
#aucscores
auc_ss=print(roc_auc_score(y_test,ss_y_pred))
auc_ss
```

```
0.6811940298507462
```

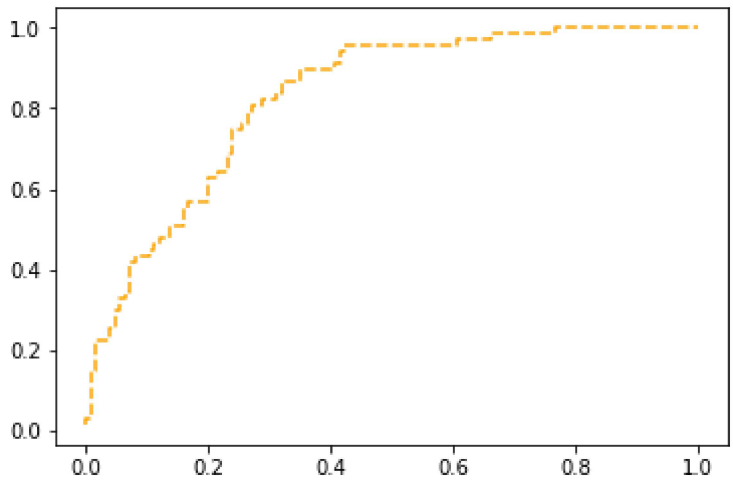
Step-7. [Plot ROC Curve]

In [121]:
```
pred_prob1=mm_lor.predict_proba(mm_X_test)
```

In [122]:
```
from sklearn.metrics import roc_curve
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
```

In [123]:
```
import matplotlib.pyplot as plt
plt.plot(fpr1,tpr1,linestyle='--',color='orange',label='MinMaxScaler values')
```

Out[123]: [<matplotlib.lines.Line2D at 0x243488be828>]



Step-8. [Comparison with KNN classifier]

In [124]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=4)
knn=knn.fit(X_train,y_train)
```

In [125]:
```
knn_y_pred=knn.predict(X_test)
```

In [126]:
```
from sklearn.preprocessing import MinMaxScaler
m=MinMaxScaler()
m_X_train=m.fit_transform(X_train)
m_X_train
```

Out[126]:
```
array([[0.05882353, 0.6080402 , 0.63934426, ..., 0.58122206, 0.07884187,
        0.11666667],
       [0.70588235, 0.44221106, 0.60655738, ..., 0.52608048, 0.13095768,
        0.45      ],
       [0.05882353, 0.54271357, 0.49180328, ..., 0.5290611 , 0.14743875,
        0.05      ],
       ...,
       [0.05882353, 0.48743719, 0.57377049, ..., 0.56780924, 0.0596882 ,
        0.15      ],
       [0.52941176, 0.7839196 , 0.70491803, ..., 0.51117735, 0.4922049 ,
        0.35      ],
       [0.23529412, 0.72361809, 0.47540984, ..., 0.43964232, 0.09042316,
        0.26666667]])
```

In [127]:
```
m_X_test=m.transform(X_test)
m_X_test
```

Out[127]:
```
array([[0.76470588, 0.52261307, 0.59016393, ..., 0.46497765, 0.16971047,
        0.28333333],
       [0.23529412, 0.63819095, 0.72131148, ..., 0.51415797, 0.22895323,
        0.11666667],
       [0.11764706, 0.47236181, 0.62295082, ..., 0.4709389 , 0.25167038,
        0.03333333],
       ...,
       [0.        , 0.53266332, 0.57377049, ..., 0.58718331, 0.23207127,
        0.01666667],
       [0.29411765, 0.62311558, 0.60655738, ..., 0.50670641, 0.06057906,
        0.28333333],
       [0.17647059, 0.64321608, 0.59016393, ..., 0.4828614 , 0.20712695,
        0.1       ]])
```

In [128]:
```
m_knn=KNeighborsClassifier()
m_knn=m_knn.fit(m_X_train,y_train)
```

In [129]:
```
m_y_pred=m_knn.predict(m_X_test)
m_y_pred
```

Out[129]:
```
array([0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0], dtype=int64)
```

In [131]:
```
#accuracy
def accuracy(actual,pred):
    return sum(actual==pred)/float(actual.shape[0])
```

In [132]:
```
m_accuracy_score=accuracy(y_test,m_y_pred)
m_accuracy_score
```

Out[132]:
```
0.703125
```

In [133]:
```
#precision
print(precision_score(y_test,m_y_pred))
```

```
0.5806451612903226
```

In [134]:
```
#recall
print(recall_score(y_test,m_y_pred))
```

```
0.5373134328358209
```

In [135]:
```
#auc scores
knn_auc=print(roc_auc_score(y_test,m_y_pred))
knn_auc
```

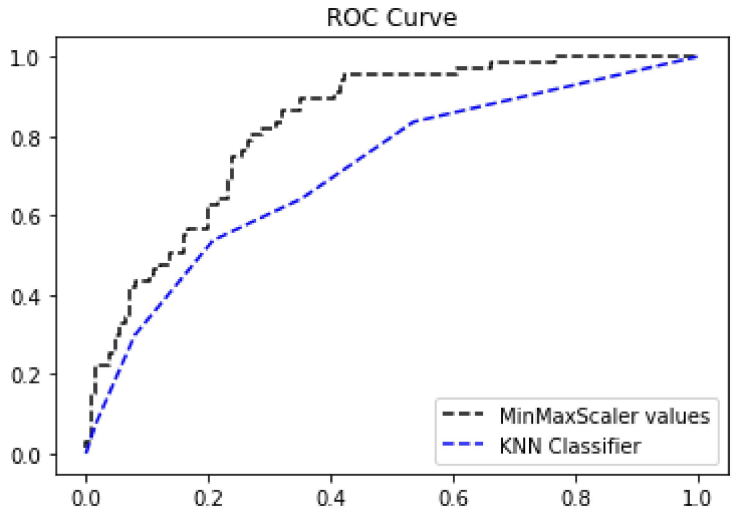
```
0.6646567164179105
```

Step-9. [Update ROC Curve]

In [136]:
```
pred_prob2=m_knn.predict_proba(m_X_test)
```

In [137]:
```
from sklearn.metrics import roc_curve
fpr2,tpr2,thresh2=roc_curve(y_test,pred_prob2[:,1],pos_label=1)
```

In [141]:
```
import matplotlib.pyplot as plt
plt.plot(fpr1,tpr1,linestyle='--',color='black',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]

In [139]:
```
from sklearn.linear_model import LogisticRegressionCV
model1=LogisticRegressionCV(Cs=10,cv=4,penalty='l1',solver='liblinear')
model2=LogisticRegressionCV(Cs=10,cv=4,penalty='l2')
```

In [142]:
```
model1.fit(mm_X_train,y_train)
model2.fit(mm_X_train,y_train)
```

Out[142]:
```
LogisticRegressionCV(Cs=10, class_weight=None, cv=4, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
           refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
```

In [143]:
```
rg_y_pred1 = model1.predict(mm_X_test)
rg_y_pred2 = model2.predict(mm_X_test)
```

In [144]:
```
#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
```

Out[144]:
```
(' LOR L1 MINMAX AUC', 0.6811940298507462)
```

In [146]:
```
#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
```

Out[146]:
```
(' LOR L2 MINMAX AUC', 0.6851940298507463)
```

STEP 11 : UPDATE ROC CURVE

In [147]:
```
pred_prb7 = model1.predict_proba(mm_X_test)
pred_prb8 = model2.predict_proba(mm_X_test)
fpr,tbr,threshold = roc_curve(y_test, pred_prob1[:,1],pos_label=1)
fpr1,tbr1,threshold1 = roc_curve(y_test, pred_prob2[:,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)
```

In [151]:
```
plt.plot(fpr, tbr, linestyle='-', color='black', label='LogisticRegression')
plt.plot(fpr1, tbr1, linestyle='-', color='blue', label='KNN')
plt.plot(fpr3, tbr3, linestyle='-', color='red', label='l2')
plt.plot(fpr2, tbr2, linestyle='-', color='orange', 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()
```