Pml Labsheet 5: Diabetes Classification Using Logistic Regression

Name: Ezhilarasan C

Roll number: 225229151

Step 1 Understand Data

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:

```
e=pd.read_csv(r"diabetes.csv")
e
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
13	1	189	60	23	846	30.1	0.398	59	1
14	5	166	72	19	175	25.8	0.587	51	1
15	7	100	0	0	0	30.0	0.484	32	1
16	0	118	84	47	230	45.8	0.551	31	1
17	7	107	74	0	0	29.6	0.254	31	1
18	1	103	30	38	83	43.3	0.183	33	0
19	1	115	70	30	96	34.6	0.529	32	1
20	3	126	88	41	235	39.3	0.704	27	0
21	8	99	84	0	0	35.4	0.388	50	0
22	7	196	90	0	0	39.8	0.451	41	1
23	9	119	80	35	0	29.0	0.263	29	1
24	11	143	94	33	146	36.6	0.254	51	1
25	10	125	70	26	115	31.1	0.205	41	1
26	7	147	76	0	0	39.4	0.257	43	1
27	1	97	66	15	140	23.2	0.487	22	0
28	13	145	82	19	110	22.2	0.245	57	0
29	5	117	92	0	0	34.1	0.337	38	0
...	...	...	...	...	...	...	...	...	...
738	2	99	60	17	160	36.6	0.453	21	0
739	1	102	74	0	0	39.5	0.293	42	1
740	11	120	80	37	150	42.3	0.785	48	1
741	3	102	44	20	94	30.8	0.400	26	0
742	1	109	58	18	116	28.5	0.219	22	0
743	9	140	94	0	0	32.7	0.734	45	1
744	13	153	88	37	140	40.6	1.174	39	0
745	12	100	84	33	105	30.0	0.488	46	0
746	1	147	94	41	0	49.3	0.358	27	1
747	1	81	74	41	57	46.3	1.096	32	0
748	3	187	70	22	200	36.4	0.408	36	1
749	6	162	62	0	0	24.3	0.178	50	1
750	4	136	70	0	0	31.2	1.182	22	1
751	1	121	78	39	74	39.0	0.261	28	0
752	3	108	62	24	0	26.0	0.223	25	0
753	0	181	88	44	510	43.3	0.222	26	1
754	8	154	78	32	0	32.4	0.443	45	1
755	1	128	88	39	110	36.5	1.057	37	1
756	7	137	90	41	0	32.0	0.391	39	0
757	0	123	72	0	0	36.3	0.258	52	1
758	1	106	76	0	0	37.5	0.197	26	0
759	6	190	92	0	0	35.5	0.278	66	1
760	2	88	58	26	16	28.4	0.766	22	0
761	9	170	74	31	0	44.0	0.403	43	1

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
762	9	89	62	0	0	22.5	0.142	33	0
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [3]:
```
e.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [4]:
```
e.tail()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

In [5]:
```
e.shape
```

Out[5]:
```
(768, 9)
```

In [6]:
```
e.columns
```

Out[6]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [7]:
```
type(e)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [8]:
```
e.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 [9]:
```
e.value_counts()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-9-18b854b8467d> in <module>()
----> 1 e.value_counts()

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-packages\pandas\core\generic.py in __getattr__(s
elf, name)
   4370             if self._info_axis._can_hold_identifiers_and_holds_name(name):
   4371                 return self[name]
-> 4372             return object.__getattribute__(self, name)
   4373 
   4374     def __setattr__(self, name, value):

AttributeError: 'DataFrame' object has no attribute 'value_counts'
```

Step 2: Build Logistic Regression Model

In [10]:
```
X=e.drop('Outcome',axis=1)
```

In [11]:
```
X.head()
```

Out[11]:

	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

In [12]:
```
y=e['Outcome'].values
```

In [13]:
```
y
```

Out[13]:
```
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, 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, 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 [14]:
```
from sklearn.model_selection import StratifiedShuffleSplit as sp
```

In [15]:
```
d = sp(n_splits=4,test_size=0.25,random_state=42)
```

In [16]:
```
d.get_n_splits(X,y)
```

Out[16]:
```
4
```

In [17]:
```
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,random_state=42)
```

In [18]:
```
from sklearn.linear_model import LogisticRegression
LOR=LogisticRegression(penalty='l2',C=10.0)
LOR=LOR.fit(X_train,y_train)
```

In [19]:
```
y_pred=LOR.predict(X_test)
y_pred
```

Out[19]:
```
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, 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 [20]:
```
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]
```

Step-3. Compute Classification Metrics

In [21]:
```
def accuracy(actual,pred):
    return sum(actual==pred)/float(actual.shape[0])
```

In [22]:
```
accuracy_score=accuracy(y_test,y_pred)
accuracy_score
```

Out[22]:
```
0.734375
```

Precision

In [23]:
```
from sklearn.metrics import precision_score
print(precision_score(y_test,y_pred))
```

```
0.6481481481481481
```

Recall

In [24]:
```
from sklearn.metrics import recall_score
print(recall_score(y_test,y_pred))
```

```
0.5223880597014925
```

AUC scores

In [25]:
```
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y_test,y_pred))
```

```
0.6851940298507463
```

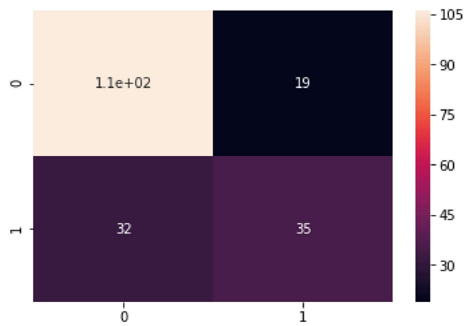
Step 4. Understand Correlation

In [26]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[26]:
```
array([[106,  19],
       [ 32,  35]], dtype=int64)
```

In [27]:
```
import seaborn as sns
sns.heatmap(cm, annot=True)
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1b224f084a8>



Step-5. Normalization using MinMaxScaler and rebuild LOR

In [28]:
```
from sklearn.preprocessing import MinMaxScaler
mm=MinMaxScaler()
mm_X_train=mm.fit_transform(X_train)
mm_X_train
```

Out[28]:
```
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 [29]:
```
mm_X_test=mm.transform(X_test)
mm_X_test
```

Out[29]:
```
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 [30]:
```
mm_lor=LogisticRegression()
mm_lor=mm_lor.fit(mm_X_train,y_train)
```

In [31]:
```
mm_y_pred=mm_lor.predict(mm_X_test)
mm_y_pred
```

Out[31]:
```
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, 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)
```

Accuracy

In [32]:
```
def accuracy(actual,pred):
    return sum(actual==pred)/float(actual.shape[0])
```

In [33]:
```
accuracy_score=accuracy(y_test,mm_y_pred)
accuracy_score
```

Out[33]:
```
0.7395833333333334
```

Precision

In [34]:
```
print(precision_score(y_test,mm_y_pred))
```

```
0.6888888888888889
```

Recall

In [35]:
```
print(recall_score(y_test,mm_y_pred))
```

```
0.4626865671641791
```

AUC Score

In [36]:
```
mm_auc=print(roc_auc_score(y_test,mm_y_pred))
mm_auc
```

```
0.6753432835820895
```

Step-6: Normalization using StandardScaler and rebuild LOR

In [37]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
ss_X_train=ss.fit_transform(X_train)
ss_X_train
```

Out[37]:
```
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 [38]:
```
ss_X_test=ss.transform(X_test)
ss_X_test
```

Out[38]:
```
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 [39]:
```
ss_lor=LogisticRegression()
ss_lor.fit(ss_X_train,y_train)
ss_y_pred=ss_lor.predict(ss_X_test)
```

Accuracy

In [40]:
```
def accuracy(actual,pred):
    return sum(actual==pred)/float(actual.shape[0])
```

In [41]:
```
ss_accuracy_score=accuracy(y_test,ss_y_pred)
ss_accuracy_score
```

Out[41]:
```
0.7291666666666666
```

Precision

In [42]:
```
print(precision_score(y_test,ss_y_pred))
```

```
0.6363636363636364
```

Recall

In [43]:
```
print(recall_score(y_test,ss_y_pred))
```

```
0.5223880597014925
```

AUC Score

In [44]:
```
auc_ss=print(roc_auc_score(y_test,ss_y_pred))
auc_ss
```

```
0.6811940298507462
```

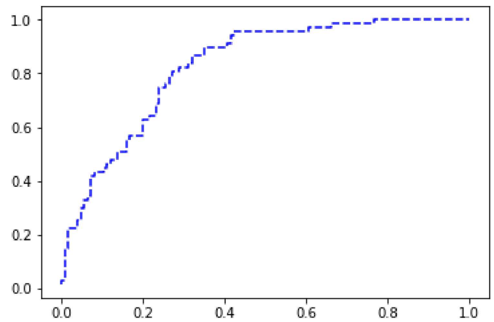
Step-7. Plot ROC Curve

In [45]:
```
pred_prob1=mm_lor.predict_proba(mm_X_test)
```

In [46]:
```
from sklearn.metrics import roc_curve
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
```

In [48]:
```
import matplotlib.pyplot as plt
plt.plot(fpr1,tpr1,linestyle='--',color='blue',label='MinMaxScaler values')
```

Out[48]: [<matplotlib.lines.Line2D at 0x1b224e23780>]



Step-8. Comparison with KNN classifier

In [49]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=4)
knn=knn.fit(X_train,y_train)
```

In [50]:
```
knn_y_pred=knn.predict(X_test)
```

In [51]:
```
from sklearn.preprocessing import MinMaxScaler
m=MinMaxScaler()
m_X_train=m.fit_transform(X_train)
m_X_train
```

Out[51]:
```
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 [52]:
```
m_X_test=m.transform(X_test)
m_X_test
```

Out[52]:
```
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 [53]:
```
m_knn=KNeighborsClassifier()
m_knn=m_knn.fit(m_X_train,y_train)
```

In [54]:
```
m_y_pred=m_knn.predict(m_X_test)
m_y_pred
```

Out[54]:
```
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)
```

Classification Metrics

Accuracy

In [55]:
```
def accuracy(actual,pred):
    return sum(actual==pred)/float(actual.shape[0])
```

In [56]:
```
m_accuracy_score=accuracy(y_test,m_y_pred)
m_accuracy_score
```

Out[56]: 0.703125

Precision

In [57]:
```
print(precision_score(y_test,m_y_pred))
```

0.5806451612903226

Recall

In [58]:
```
print(recall_score(y_test,m_y_pred))
```

0.5373134328358209

AUC Scores

In [59]:
```
knn_auc=print(roc_auc_score(y_test,m_y_pred))
knn_auc
```

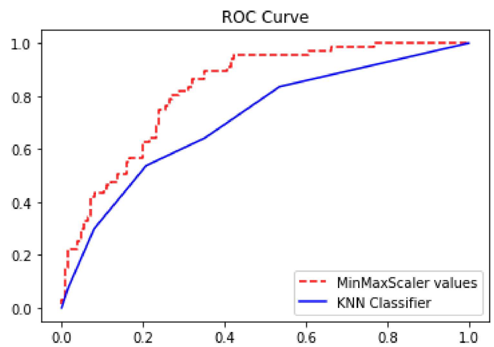
0.6646567164179105

Step-9. Update ROC Curve

In [62]:
```
pred_p2=m_knn.predict_proba(m_X_test)
```

In [63]:
```
from sklearn.metrics import roc_curve
fpr2,tpr2,thresh2=roc_curve(y_test,pred_p2[:,1],pos_label=1)
```

In [78]:
```
plt.plot(fpr1,tpr1,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

In [79]:
```
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 [80]:
```
model1.fit(mm_X_train,y_train)
model2.fit(mm_X_train,y_train)
```

Out[80]:
```
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 [81]:
```
rg_y_pred1 = model1.predict(mm_X_test)
rg_y_pred2 = model2.predict(mm_X_test)
```

AUC SCORE OF L1

In [82]:
```
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[82]:
```
(' LOR L1 MINMAX AUC', 0.6811940298507462)
```

AUC SCORE OF L2

In [83]:
```
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
```

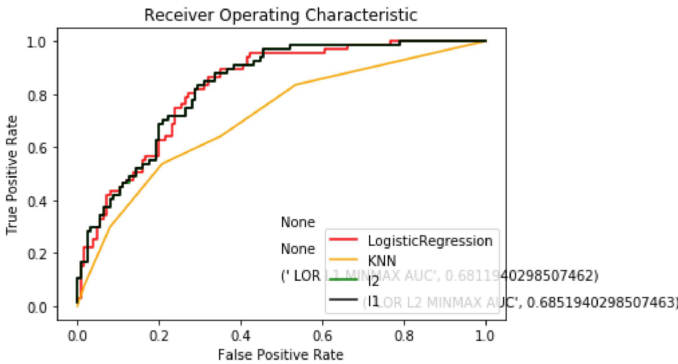
Out[83]:
```
(' LOR L2 MINMAX AUC', 0.6851940298507463)
```

Step 11: Update ROC curve

In [84]:
```
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 [88]:

```python
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()
```



In [ ]:

```python

```