In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
data=pd.read_csv("diabetes .csv")
```

In [3]:
```
data
```

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
...	...	...	...	...	...	...	...	...	...
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 [4]:
```
data.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
```

In [5]:
```
data.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Ou
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.3
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.4
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.0
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.0
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.0

In [6]:
```
1 data.head()
```

Out[6]:

	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 [7]:
```
1 data.tail()
```

Out[7]:

	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 [8]:
```
1 data["Outcome"].value_counts(normalize=True)
```

Out[8]:
```
Outcome
0    0.651042
1    0.348958
Name: proportion, dtype: float64
```

In [9]:
```
1 x=data.drop(["Outcome"],axis=1)
```

In [10]:
```
1 y=data["Outcome"]
```

In [11]:
```
1 x
```

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
...	...	...	...	...	...	...	...	...
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

In [12]:
```
1 y
```

Out[12]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [13]:
```
1 # spliting of data
```

In [14]:
```
1 from sklearn.model_selection import train_test_split
```

In [15]:
```
1 train_x, test_x, train_y , test_y =train_test_split(x,y,random_state =56)
```

In [17]:
```
1 train_x
```

Out[17]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
536	0	105	90	0	0	29.6	0.197	46
547	4	131	68	21	166	33.1	0.160	28
307	0	137	68	14	148	24.8	0.143	21
45	0	180	66	39	0	42.0	1.893	25
196	1	105	58	0	0	24.3	0.187	21
...	...	...	...	...	...	...	...	...
235	4	171	72	0	0	43.6	0.479	26
418	1	83	68	0	0	18.2	0.624	27
192	7	159	66	0	0	30.4	0.383	36
399	3	193	70	31	0	34.9	0.241	25
484	0	145	0	0	0	44.2	0.630	31

576 rows × 8 columns

In [18]:
```
1 test_x
```

Out[18]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
123	5	132	80	0	0	26.8	0.186	69
295	6	151	62	31	120	35.5	0.692	28
370	3	173	82	48	465	38.4	2.137	25
300	0	167	0	0	0	32.3	0.839	30
155	7	152	88	44	0	50.0	0.337	36
...	...	...	...	...	...	...	...	...
443	8	108	70	0	0	30.5	0.955	33
134	2	96	68	13	49	21.1	0.647	26
181	0	119	64	18	92	34.9	0.725	23
588	3	176	86	27	156	33.3	1.154	52
737	8	65	72	23	0	32.0	0.600	42

192 rows × 8 columns

In [19]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [20]:
```
scaler=MinMaxScaler()
scaler
```

Out[20]:
```
MinMaxScaler
MinMaxScaler()
```

In [21]:
```
cols=train_x.columns
cols
```

Out[21]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

In [22]:
```
train_x_scaled=scaler.fit_transform(train_x)
train_x_scaled
```

Out[22]:
```
array([[0.        , 0.53030303, 0.73770492, ..., 0.44113264, 0.05081127,
        0.41666667],
       [0.23529412, 0.66161616, 0.55737705, ..., 0.49329359, 0.03501281,
        0.11666667],
       [0.        , 0.69191919, 0.55737705, ..., 0.36959762, 0.02775406,
        0.        ],
       ...,
       [0.41176471, 0.8030303 , 0.54098361, ..., 0.45305514, 0.13023057,
        0.25      ],
       [0.17647059, 0.97474747, 0.57377049, ..., 0.52011923, 0.06959863,
        0.06666667],
       [0.        , 0.73232323, 0.        , ..., 0.65871833, 0.23569599,
        0.16666667]])
```

In [23]:
```
train_x_scaled=pd.DataFrame(train_x_scaled,columns=cols)
```

In [24]:
```
train_x_scaled
```

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.000000	0.530303	0.737705	0.000000	0.000000	0.441133	0.050811	0.416667
1	0.235294	0.661616	0.557377	0.212121	0.196217	0.493294	0.035013	0.116667
2	0.000000	0.691919	0.557377	0.141414	0.174941	0.369598	0.027754	0.000000
3	0.000000	0.909091	0.540984	0.393939	0.000000	0.625931	0.774979	0.066667
4	0.058824	0.530303	0.475410	0.000000	0.000000	0.362146	0.046541	0.000000
...	...	...	...	...	...	...	...	...
571	0.235294	0.863636	0.590164	0.000000	0.000000	0.649776	0.171221	0.083333
572	0.058824	0.419192	0.557377	0.000000	0.000000	0.271237	0.233134	0.100000
573	0.411765	0.803030	0.540984	0.000000	0.000000	0.453055	0.130231	0.250000
574	0.176471	0.974747	0.573770	0.313131	0.000000	0.520119	0.069599	0.066667
575	0.000000	0.732323	0.000000	0.000000	0.000000	0.658718	0.235696	0.166667

576 rows × 8 columns

In [25]:
```
from sklearn.linear_model import LogisticRegression as LogReg
```

In [26]:
```
logreg=LogReg()
```

In [27]:
```
1 logreg.fit(train_x,train_y)
```

Out[27]:
```
▾ LogisticRegression

LogisticRegression()
```

In [28]:
```
1 train_predict=logreg.predict(train_x)
2 test_predict=logreg.predict(test_x)
```

In [29]:
```
1 train_predict
```

Out[29]:
```
array([0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 1, 1, 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, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 1], dtype=int64)
```

In [30]:
```
1 test_predict
```

Out[30]:
```
array([0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0]], dtype=int64)
```

In [31]:
```
1 from sklearn.metrics import f1_score, confusion_matrix, roc_auc_score, roc_curve
```

In [33]:
```
1 f1_score(train_predict,train_y)
```

Out[33]:
```
0.6304347826086956
```

In [34]:
```
1 f1_score(test_predict,test_y)
```

Out[34]:
```
0.7008547008547009
```

In [35]:
```
1 conf1=confusion_matrix(train_y,train_predict)
```

In [36]:
```
1 conf1
```

Out[36]:
```
array([[324,  42],
       [ 94, 116]], dtype=int64)
```

In [37]:
```
conf=confusion_matrix(test_y,test_predict)
```

In [38]:
```
conf
```

Out[38]:
```
array([[116,  18],
       [ 17,  41]], dtype=int64)
```

In [39]:
```
true_negative =conf[0][0]
false_negative =conf[1][0]
false_positive =conf[0][1]
true_positive =conf[1][1]
```

In [59]:
```
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negative + true_neg
Accuracy
# Precison
Precision = true_positive/(true_positive+false_positive)
Precision
# Recall
Recall = true_positive/(true_positive+false_negative)
Recall
# F1 Score
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

Out[59]:
```
0.7008547008547009
```

In [68]:
```
Accuracy
```

Out[68]:
```
0.8177083333333334
```

In [69]:
```
Precision
```

Out[69]:
```
0.6949152542372882
```

In [70]:
```
Recall
```

Out[70]:
```
0.7068965517241379
```

In [71]:
```
F1_Score
```

Out[71]:
```
0.7008547008547009
```

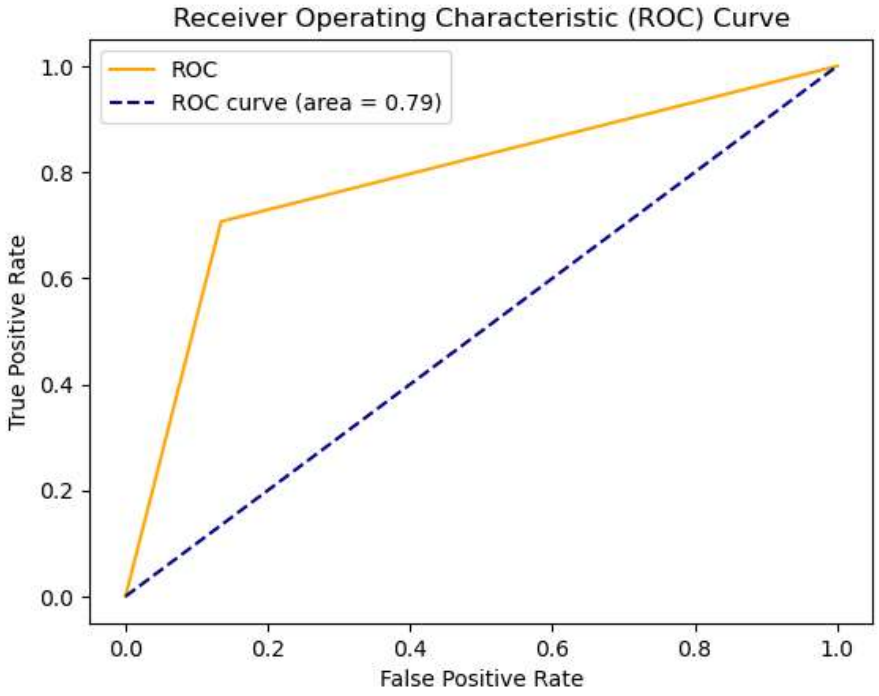
In [72]:
```
auc_score=roc_auc_score(test_y,test_predict)
```

In [73]:
```
fpr,tpr,threasholds=roc_curve(test_y,test_predict)
```

In [74]:
```
threasholds
```

Out[74]:
```
array([inf,  1.,  0.])
```

In [75]:
```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)' % auc_score
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```




```
# Swayambhu Bhapkar
# Roll no:- 13121
```