In [42]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import LogisticRegression
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_
from sklearn.metrics import log_loss, f1_score, precision_score, recall_score
import matplotlib.pyplot as plt
from pandas_profiling import ProfileReport
import seaborn as sns
import pickle
%matplotlib inline
```

Diabeties Dataset

In [124]:
```
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/diabet
```

In [126]:
```
df
```

Out[126]:

	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
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

768 rows × 9 columns

EDA and Feature Engineering

In [127]:
```python
# Profile report
df.profile_report(minimal=True)
```

Summarize dataset: 100% 15/15 [00:00<00:00, 38.89it/s, Completed]

Generate report structure: 100% 1/1 [00:12<00:00, 12.72s/it]

Render HTML: 100% 1/1 [00:08<00:00, 8.25s/it]

Observation:- Features like glucose, BloodPressure, skinThickness, Insulin, BMI have zeros which, not possible for living body. These features are almost normally distributed so we can use mean to fill the zeros.

In [128]:
```python
df.columns # checking columns names
```

Out[128]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [129]:
```python
# replacing the zeros with mean
for col in ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']:
    df[col].replace(0,df[col].mean(),inplace=True)
```

In [130]:
```
ProfileReport(df,minimal=True)
```

Summarize dataset: 100% 15/15 [00:00<00:00, 33.24it/s, Completed]

Generate report structure: 100% 1/1 [3:11:09<00:00, 11469.19s/it]

Render HTML: 100% 1/1 [00:35<00:00, 35.25s/it]

Overview

Dataset statistics

Number of variables	9
Number of observations	768
Missing cells	0
Missing cells (%)	0.0%
Total size in memory	54.1 KiB
Average record size in memory	72.2 B

Variable types

Numeric	9

Alerts

`Pregnancies` has 111 (14.5%) zeros	Zeros
`Outcome` has 500 (65.1%) zeros	Zeros

Reproduction

Analysis started	2022-09-09 13:01:31.240407
Analysis finished	2022-09-09 13:01:31.379103

Out[130]:

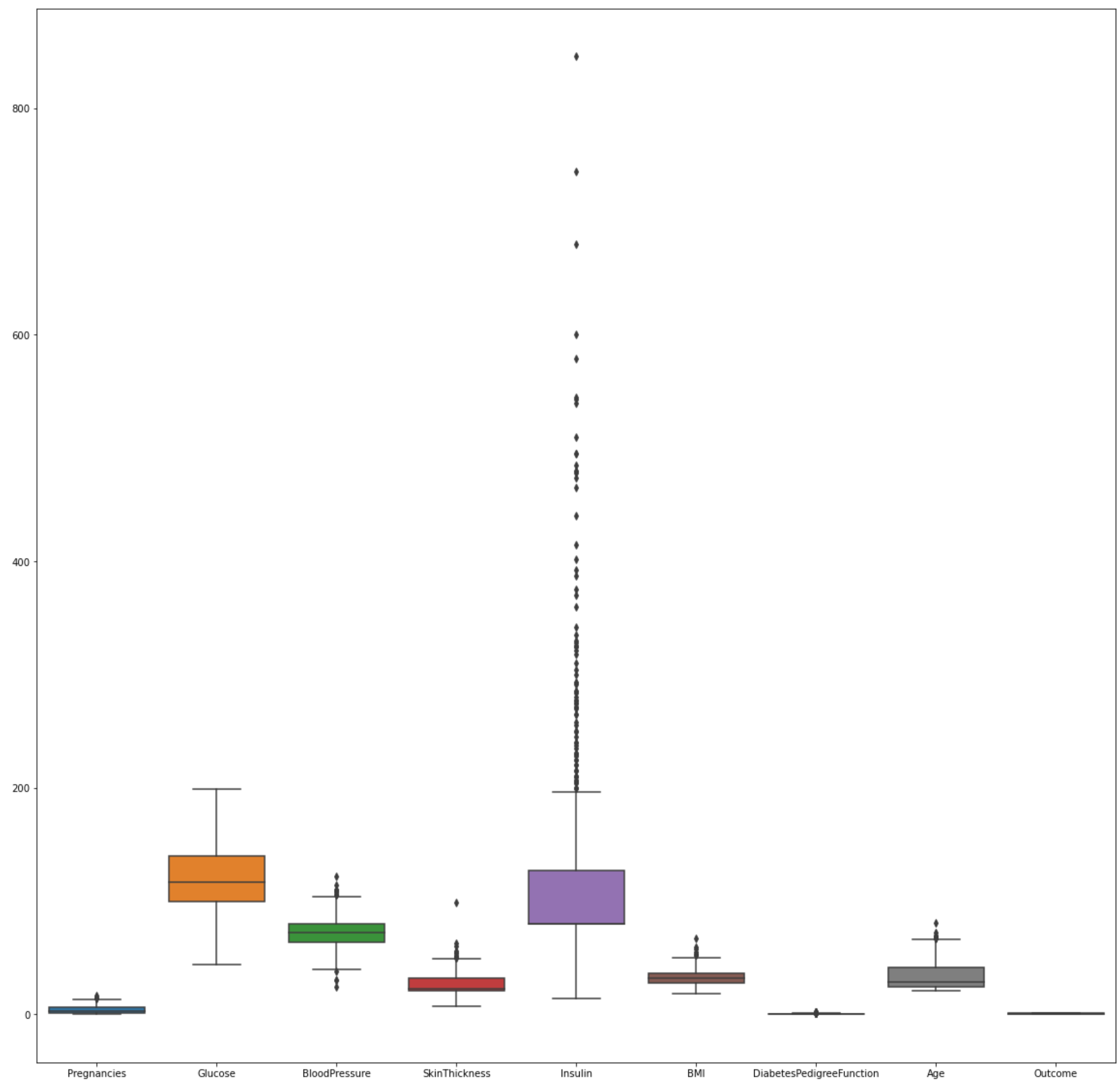
Observation:- Zeros are replaced. There is no null values in the dataset.

In [131]:
```
df.isnull().sum()
```

Out[131]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [132]:
```
## Checking Outliers using box plots
fig ,ax  = plt.subplots(figsize = (20,20))
sns.boxplot(data = df , ax = ax)
```

Out[132]: <AxesSubplot:>



Observation:- There are outliers in many features of the dataset.

In [133]:
```
df.shape
```

Out[133]:
```
(768, 9)
```

In [134]:
```
df.columns
```

Out[134]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [151]:
```
# As we have less numbers of data rows so we are carefully choosing our quantiles
q = df['Pregnancies'].quantile(.99)
df_new = df[df['Pregnancies'] < q]

q = df_new['Insulin'].quantile(.95)
df_new = df_new[df_new['Insulin']< q]

q = df_new['Age'].quantile(.97)
df_new = df_new[df_new['Age']< q]

q = df_new['BloodPressure'].quantile(.98)
df_new = df_new[df_new['BloodPressure']< q]

q = df_new['SkinThickness'].quantile(.98)
df_new = df_new[df_new['SkinThickness']< q]

q = df_new['BMI'].quantile(.99)
df_new = df_new[df_new['BMI']< q]

q = df_new['Glucose'].quantile(.99)
df_new = df_new[df_new['Glucose']< q]
```

In [152]:
```
df_new.shape
```

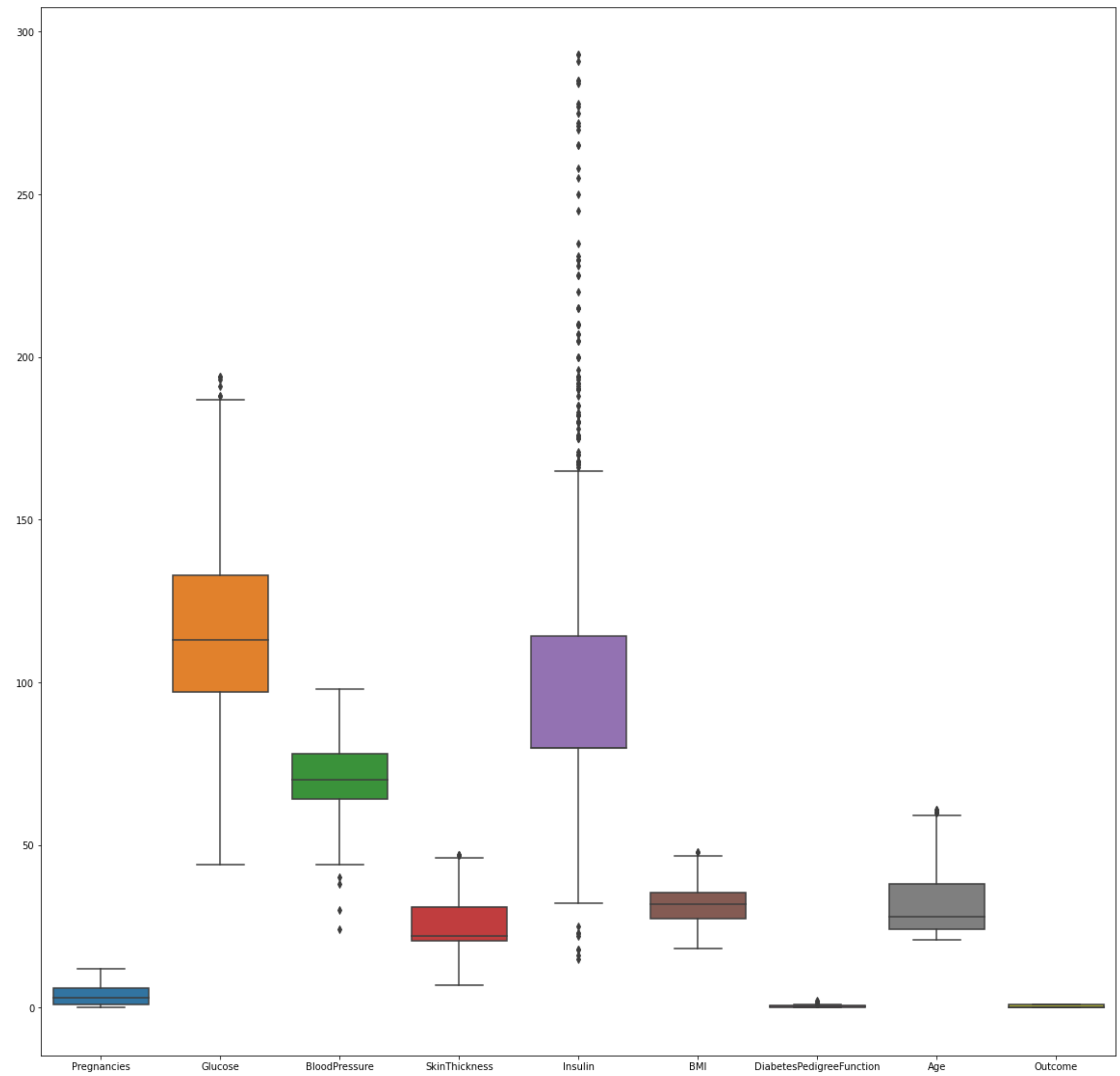
Out[152]:
```
(648, 9)
```

In [153]:
```
df_new.isnull().sum()
```

Out[153]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [154]:
```
fig ,ax  = plt.subplots(figsize = (20,20))
sns.boxplot(data = df_new , ax = ax)
```

Out[154]: <AxesSubplot:>



Observation:- Quantile technique to remove outlier removed outliers from some columns. We can try standardscaler as well.

In [155]:
```
df_new.isnull().sum()
```

Out[155]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [156]:
```
# Segregating output column
y = df_new['Outcome']
y
```

Out[156]:
```
0      1
1      0
2      1
3      0
4      1
      ..
762    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 648, dtype: int64
```

In [157]:
```
X = df_new.drop(columns=['Outcome'])
```

In [158]:
```
X
```

Out[158]:

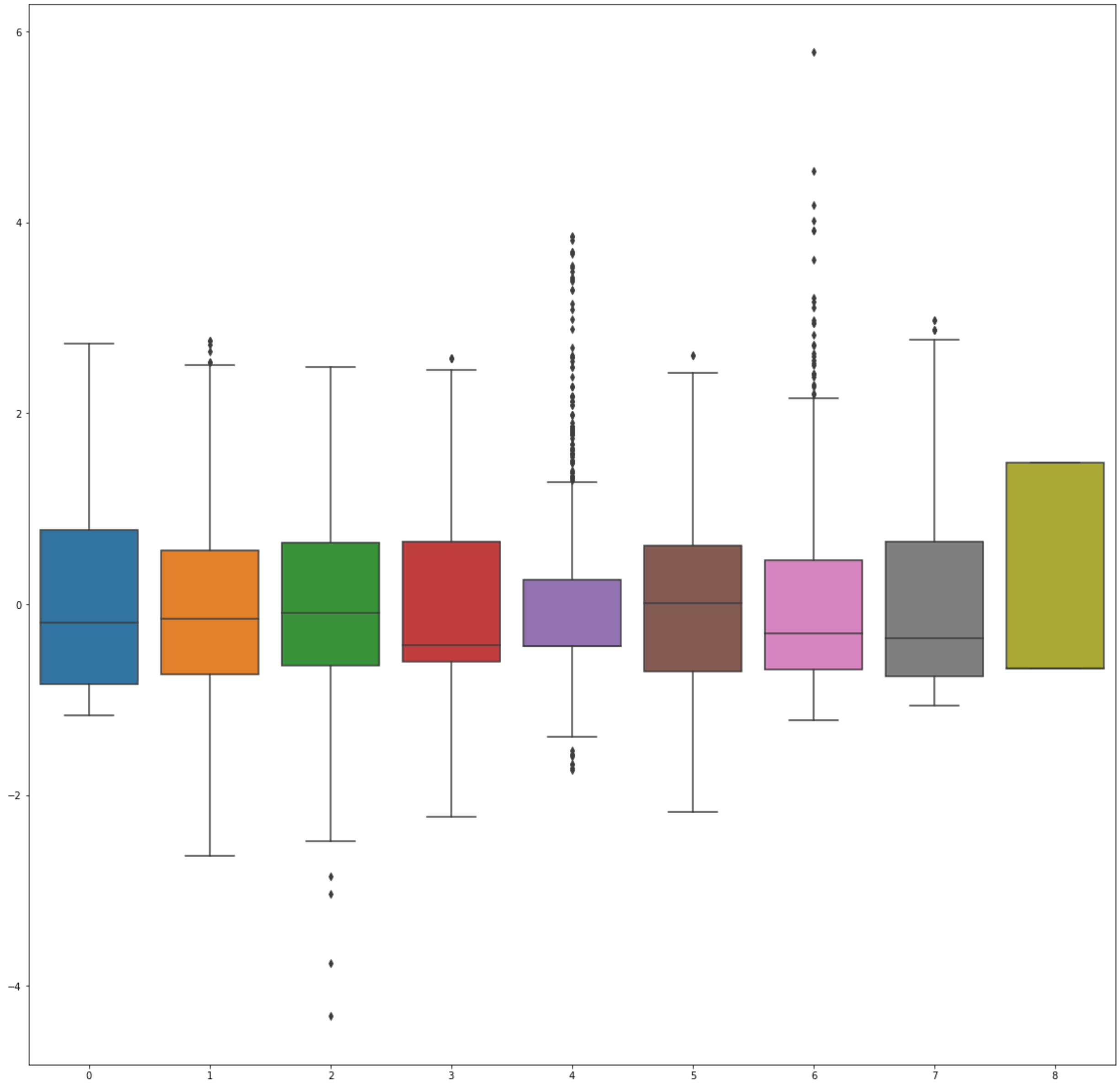
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148.0	72.0	35.000000	79.799479	33.6	0
1	1	85.0	66.0	29.000000	79.799479	26.6	0
2	8	183.0	64.0	20.536458	79.799479	23.3	0
3	1	89.0	66.0	23.000000	94.000000	28.1	0
4	0	137.0	40.0	35.000000	168.000000	43.1	2
...	...	...	...	...	...	...	
762	9	89.0	62.0	20.536458	79.799479	22.5	0
764	2	122.0	70.0	27.000000	79.799479	36.8	0
765	5	121.0	72.0	23.000000	112.000000	26.2	0
766	1	126.0	60.0	20.536458	79.799479	30.1	0
767	1	93.0	70.0	31.000000	79.799479	30.4	0

648 rows × 8 columns

In [159]:
```
# Transformation of data using standardScaler()
scalar = StandardScaler()
X_scaled = scalar.fit_transform(X)
```

In [160]:
```
# Boxplot ploting to check the dataset after standard scaler
df_new_scalar = pd.DataFrame(scalar.fit_transform(df_new))
fig ,ax  = plt.subplots(figsize = (20,20))
sns.boxplot(data = df_new_scalar , ax = ax)
```

Out[160]: <AxesSubplot:>



Observation:- After StandardScaling data looks normalized.

In [162]:
```
X_scaled
```

Out[162]:
```
array([[ 0.78080278,  1.10205322,  0.09293203, ...,  0.30463329,
         0.52496906,  1.8616411 ],
       [-0.84190473, -1.16279909, -0.4581753 , ..., -0.82128998,
        -0.3490655 , -0.05334973],
       [ 1.42988579,  2.36030449, -0.64187774, ..., -1.35208237,
         0.66747469,  0.04743926],
       ...,
       [ 0.45626128,  0.13140223,  0.09293203, ..., -0.88562845,
        -0.68474543, -0.15413872],
       [-0.84190473,  0.31115241, -1.00928262, ..., -0.25832834,
        -0.35539908,  1.55927413],
       [-0.84190473, -0.87519879, -0.09077041, ..., -0.21007449,
        -0.46307   , -0.85966166]])
```

In [163]:
```
y
```

Out[163]:
```
0      1
1      0
2      1
3      0
4      1
      ..
762    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 648, dtype: int64
```

In [180]:
```
# To check Multicollinearity we use vif
def vif_score(x):
    scaler = StandardScaler()
    arr = scaler.fit_transform(x)
    return pd.DataFrame([[x.columns[i], variance_inflation_factor(arr,i)] for i i
```

In [181]:
```
vif_score(X)
```

Out[181]:

	FEATURE	VIF_SCORE
0	Pregnancies	1.566153
1	Glucose	1.277012
2	BloodPressure	1.223138
3	SkinThickness	1.458437
4	Insulin	1.252390
5	BMI	1.520853
6	DiabetesPedigreeFunction	1.056995
7	Age	1.759953

Observation:- No Multicollinearity among features

Train Test Split

In [187]:
```
x_train, x_test, y_train, y_test = train_test_split(X_scaled , y , test_size = .2
```

In [188]:
```
x_train
```

Out[188]:
```
array([[-1.16644624, -1.59419952, -0.1729342 , ..., -1.70594397,
        -0.37756662, -0.65808368],
       [-0.19282173,  0.52685263,  0.8277418 , ...,  0.43331023,
        -0.18755911,  1.25690716],
       [-0.84190473,  0.27520238, -0.09077041, ..., -1.19123619,
        -0.76074844, -0.65808368],
       ...,
       [-0.84190473, -0.15619806, -0.64187774, ...,  0.30463329,
         0.25895854, -1.06123964],
       [-0.19282173, -0.04834795,  0.27663448, ..., -0.86954383,
        -1.12176271, -0.75887267],
       [-1.16644624, -0.83924876, -0.1729342 , ...,  0.04608562,
        -0.64991072, -0.65808368]])
```

In [189]:
```
x_test
```

Out[189]:
```
array([[-0.84190473, -0.62354854,  0.27663448, ..., -1.96329786,
        -0.98875745, -0.3557167 ],
       [ 2.4035103 ,  0.92230303,  2.1136589 , ...,  0.78717183,
        -0.65624431,  1.9624301 ],
       [ 2.7280518 , -1.19874912,  0.09293203, ..., -0.32266682,
        -0.52007226,  1.45848514],
       ...,
       [ 0.13171978,  0.38305248, -0.09077041, ...,  0.41722561,
        -0.50107151, -0.75887267],
       [-1.16644624,  0.23925234, -0.09077041, ..., -0.69261303,
        -0.65624431,  0.45059523],
       [-0.84190473,  0.77850289, -2.29519972, ..., -0.483513  ,
         0.61047244, -0.96045065]])
```

Logistics Regression

In [208]:
```
# logis object by default it uses lbfgs solver and penalty= l2
logis = LogisticRegression()
```

In [209]:
```
logis.fit(x_train,y_train )
```

Out[209]:
```
LogisticRegression()
```

In [210]:
```
# All default parameters we can see
logis.get_params()
```

Out[210]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

Observation:- By default, lbfgs solver and penalty=l2 also fit_intercept=True

In [211]:
```
# slopes values
logis.coef_
```

Out[211]:
```
array([[ 0.28596086,  1.18507163, -0.09769945,  0.01972989, -0.10540602,
         0.60765528,  0.47934611,  0.22849868]])
```

In [212]:
```
# Binary classes
logis.classes_
```

Out[212]:
```
array([0, 1], dtype=int64)
```

In [219]:
```
# Intercept of the best fit line
logis.intercept_
```

Out[219]:
```
array([-1.12817895])
```

In [236]:
```
# Prediction probabilty of class 0 and 1
pred_prob = logis.predict_proba([x_test[1]])
```

In [237]:
```
# let's predict just one values from test data
print("predicted value: ",logis.predict([x_test[1]]))
print(f"prediction probabilty of 0 is {pred_prob[0][0]} and of 1 is {pred_prob[0]
```

```
predicted value:  [1]
prediction probabilty of 0 is 0.2726911636295054 and of 1 is 0.7273088363704946
```

In [239]:
```
# log probabilty
logis.predict_log_proba([x_test[1]])
```

Out[239]:
```
array([[-1.29941539, -0.31840408]])
```

In [246]:
```
# Model accuracy using x_test dataset
logis.score(x_test,y_test)
```

Out[246]:
```
0.7461538461538462
```

In [250]:
```
# Making prediction on x_test
y_pred = logis.predict(x_test)
```

Checking Acurracy,confusion_matrix,f1_score etc using sklearn.metrics

In [266]:
```
# function shows accuracy,confusion_matrix using sklearn.metrics
def model_metrics(y_test,y_pred):
    print("Accuracy: ",accuracy_score(y_test,y_pred))         # Model Accuracy
    print("Confusion Matrix: ",confusion_matrix(y_test,y_pred))     # confusion m
    print("log loss :",log_loss(y_test,y_pred))       # log loss
    print("precision score: ", precision_score(y_test,y_pred))     # precision sc
    print("recall score : ",recall_score(y_test,y_pred))      # recall score
    print("F1 score : ",f1_score(y_test,y_pred))
model_metrics(y_test,y_pred)
```

```
Accuracy:  0.7461538461538462
Confusion Matrix:  [[80 10]
 [23 17]]
log loss : 8.767597053895035
precision score:  0.6296296296296297
recall score :  0.425
F1 score :  0.5074626865671642
```

Checking model accuracy,confusion_matrix using formula

In [265]:
```
def model_eval(y_true,y_pred):
    tn, fp, fn, tp = confusion_matrix(y_test,y_pred).ravel()
    accuracy=(tp+tn)/(tp+tn+fp+fn)
    precision=tp/(tp+fp)
    recall=tp/(tp+fn)
    specificity=tn/(fp+tn)
    F1_Score = 2*(recall * precision) / (recall + precision)
    result={"Accuracy":accuracy,"Precision":precision,"Recall":recall,'Specficity
    return result
model_eval(y_test,y_pred)
```

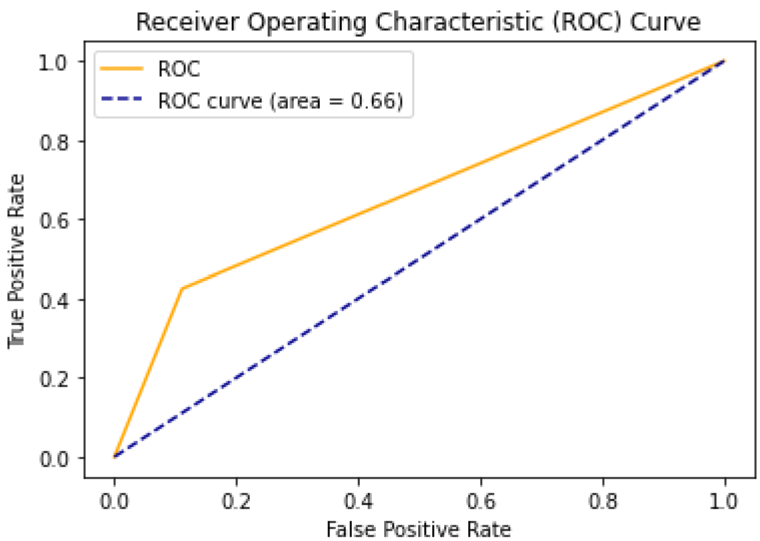
Out[265]:
```
{'Accuracy': 0.7461538461538462,
 'Precision': 0.6296296296296297,
 'Recall': 0.425,
 'Specficity': 0.8888888888888888,
 'F1': 0.5074626865671642}
```

ROC_AUC_SCORE AND GRAPH

In [328]:
```
roc_auc_score(y_test,y_pred)
```

Out[328]:
```
0.6569444444444444
```

In [331]:
```
# plot
fpr, tpr, thresholds  = roc_curve(y_test,y_pred)
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



In [ ]:

Creating and testing model using Solver=liblinear, penalty='l1'

In [275]:
```
logis_lib = LogisticRegression(verbose=1,solver='liblinear',penalty='l1')
```

In [276]:
```
logis_lib.fit(x_train,y_train)
```

```
[LibLinear]
```

Out[276]:
```
LogisticRegression(penalty='l1', solver='liblinear', verbose=1)
```

In [278]:
```
# see all parameter
logis_lib.get_params()
```

Out[278]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l1',
 'random_state': None,
 'solver': 'liblinear',
 'tol': 0.0001,
 'verbose': 1,
 'warm_start': False}
```

In [280]:
```
# slopes
logis_lib.coef_
```

Out[280]:
```
array([[ 0.27969855,  1.17511063, -0.07810108,  0.00365199, -0.08997112,
         0.59970065,  0.46837545,  0.21484829]])
```

In [281]:
```
# intercepts
logis_lib.intercept_
```

Out[281]:
```
array([-1.10817926])
```

In [283]:
```
y_pred_lib = logis_lib.predict(x_test)
```

In [284]:
```
# Checking accuracy etc
model_metrics(y_test,y_pred_lib)
```

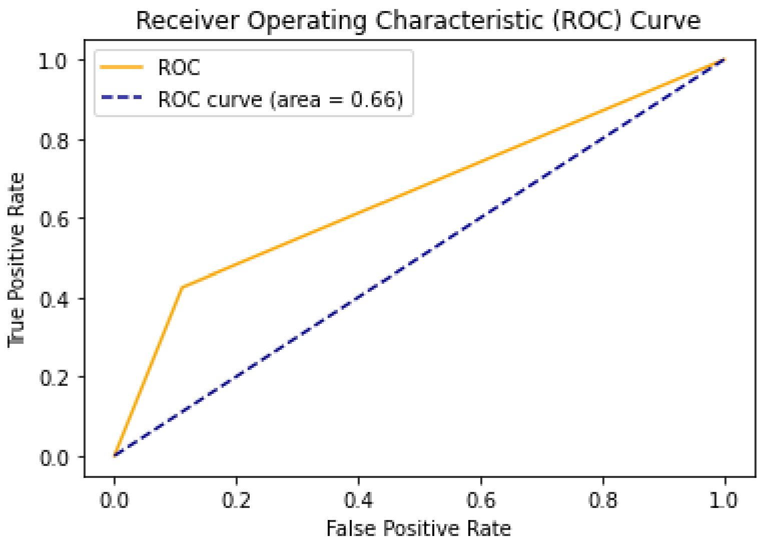
```
Accuracy:  0.7461538461538462
Confusion Matrix:  [[80 10]
 [23 17]]
log loss : 8.767597053895035
precision score:  0.6296296296296297
recall score :  0.425
F1 score :  0.5074626865671642
```

ROC_AUC_SCORE

In [333]:
```
roc_auc_score(y_test,y_pred_lib)
```

Out[333]:
```
0.6569444444444444
```

In [334]:
```
# plot
fpr, tpr, thresholds  = roc_curve(y_test,y_pred_lib)
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



Observation:- Model of solver='lbfgs' ,penalty='l2' AND Model of solver='liblinear',penalty='l1' have almost same performance

Creating and testing model with solver='saga' and penalty='elasticnet'

In [316]:
```
logis_saga = LogisticRegression(solver='saga',penalty='elasticnet',l1_ratio=0.5,f
```

In [317]:
```
logis_saga.fit(x_train,y_train)
```

Out[317]:
```
LogisticRegression(fit_intercept=False, l1_ratio=0.5, penalty='elasticnet',
                   solver='saga')
```

In [318]:
```
# see all parameter
logis_saga.get_params()
```

Out[318]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': False,
 'intercept_scaling': 1,
 'l1_ratio': 0.5,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'elasticnet',
 'random_state': None,
 'solver': 'saga',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [319]:
```
logis_saga.coef_  # slopes
```

Out[319]:
```
array([[ 0.26350835,  1.11069258, -0.07539441, -0.03970395, -0.16687024,
         0.52891693,  0.43414594,  0.16602705]])
```

In [320]:
```
logis_saga.intercept_ #intercept will 0
```

Out[320]:
```
array([0.])
```

In [321]:
```
y_pred_saga = logis_saga.predict(x_test)
```

In [322]:
```
model_metrics(y_test,y_pred_saga)
```

```
Accuracy:  0.6923076923076923
Confusion Matrix:  [[60 30]
 [10 30]]
log loss : 10.627500336302564
precision score:  0.5
recall score :  0.75
F1 score :  0.6
```

Observation:- model solver='saga',penalty='elastinet' and fit_intercept=False has low accuracy than above models

AUC_ROC_SCORE

In [335]:
```
roc_auc_score(y_test,y_pred_saga)
```

Out[335]:
```
0.7083333333333334
```

In [337]:
```
fpr, tpr, thresholds  = roc_curve(y_test,y_pred_saga)
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```