```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df=pd.read_csv('/content/diabetes.csv')
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
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	

```
df.columns
```

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

```
df.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
```

```
df.shape
```

```
(768, 9)
```

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

```
df.isnull().sum()
```

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

```
s = [feature for feature in df.columns if df[feature].dtypes not in ['O', 'o', 'object']]
es = [feature for feature in df.columns if df[feature].dtypes in ['O', 'o', 'object']]
```

```
numerical_features
```

```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
```

```
 'Age',
 'Outcome']
```

```
categorical_fatures
```

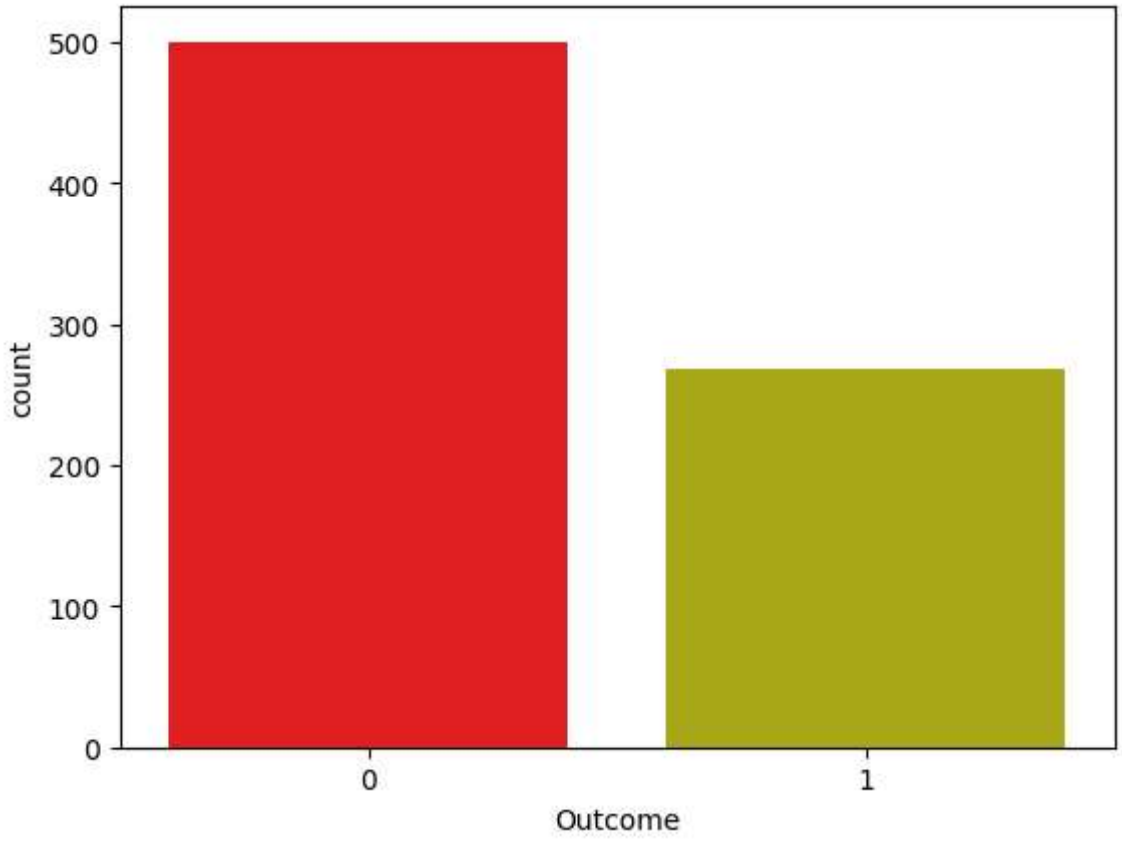
```
[]
```

```
print("Standard Deviation of each variable are: ")
df.apply(np.std)
```

```
Standard Deviation of each variable are:
Pregnancies                  3.367384
Glucose                     30.416127
BloodPressure               12.088468
SkinThickness                8.785217
Insulin                     84.965737
BMI                          6.870674
DiabetesPedigreeFunction     0.331113
Age                         11.752573
Outcome                      0.476641
dtype: float64
```

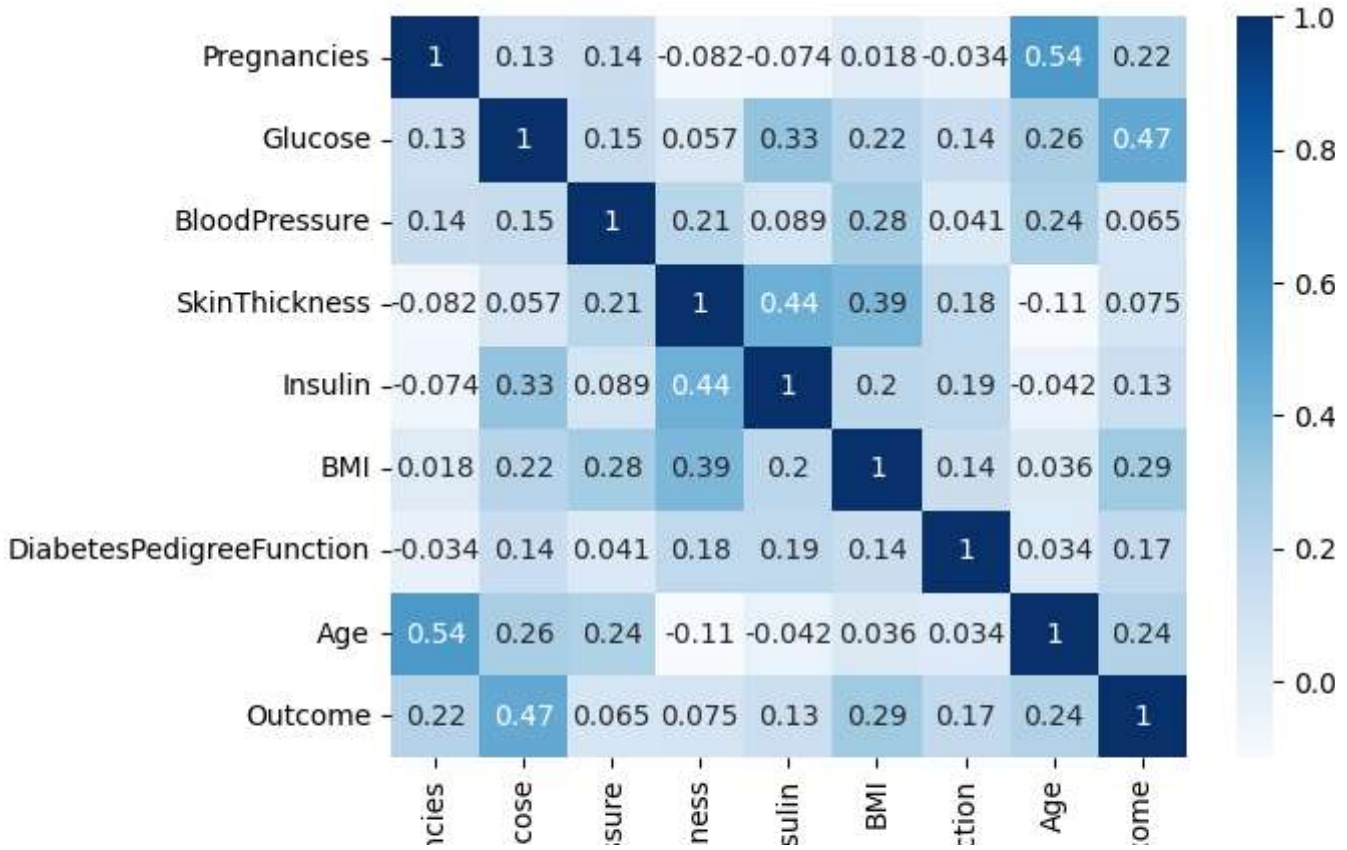
```
sns.countplot(x='Outcome',data=df,palette=['r','y'])
```

```
<Axes: xlabel='Outcome', ylabel='count'>
```



```
sns.pairplot(data=df,hue='Outcome')
plt.show()
```


```
sns.heatmap(df.corr(),annot=True,cmap='Blues')
plt.show()
```




```
# Pregnancies
plt.figure(figsize=(20,6))

plt.subplot(1,3,1)
plt.title("Counter Plot")
sns.countplot(x = 'Pregnancies',data = df)

plt.subplot(1,3,2)
plt.title('Distribution Plot')
sns.distplot(df["Pregnancies"])

plt.subplot(1,3,3)
plt.title('Box Plot')
sns.boxplot(y=df["Pregnancies"])

plt.show()
```

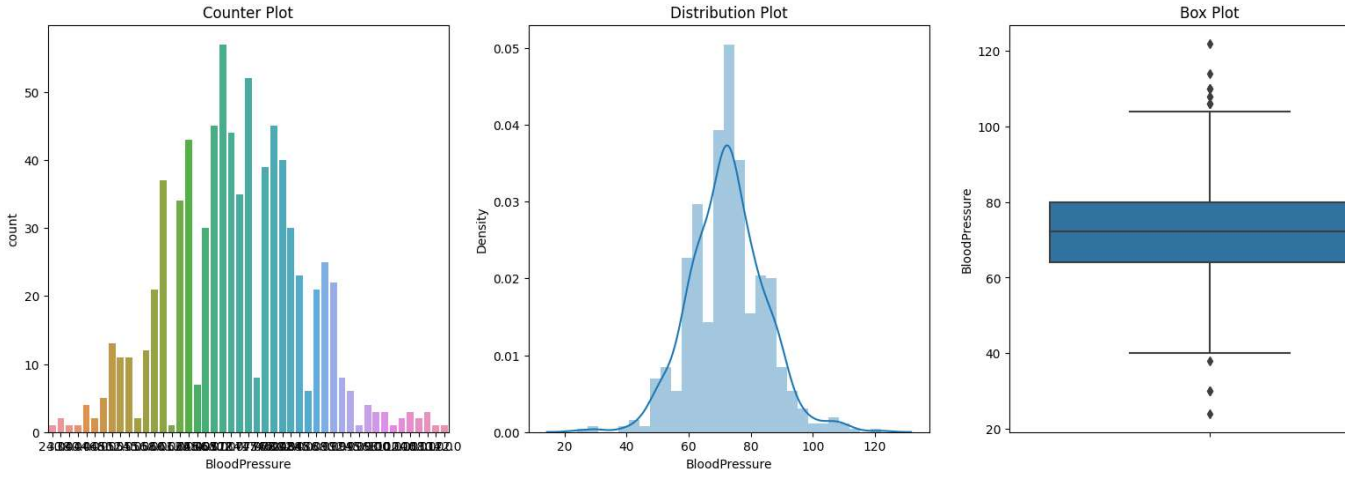
```
<ipython-input-63-f0c96f8f290e>:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Pregnancies"])
```


```
# Glucose
plt.figure(figsize=(20,6))

plt.subplot(1,3,1)
plt.title("Counter Plot")
sns.countplot(x = 'Glucose',data = df)

plt.subplot(1,3,2)
plt.title('Distribution Plot')
sns.distplot(df["Glucose"])

plt.subplot(1,3,3)
plt.title('Box Plot')
sns.boxplot(y=df["Glucose"])

plt.show()
```

```
<ipython-input-52-ad47dbcc942b>:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.
```

```
# BloodPressure
plt.figure(figsize=(20,6))

plt.subplot(1,3,1)
plt.title("Counter Plot")
sns.countplot(x = 'BloodPressure',data = df)

plt.subplot(1,3,2)
plt.title('Distribution Plot')
sns.distplot(df["BloodPressure"])

plt.subplot(1,3,3)
plt.title('Box Plot')
sns.boxplot(y=df["BloodPressure"])

plt.show()
```

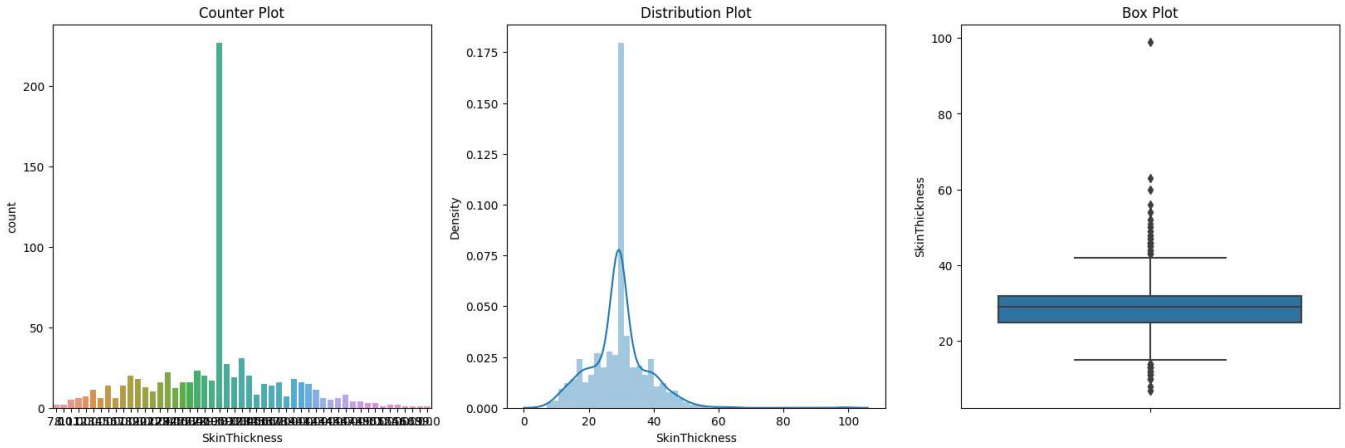
```
<ipython-input-54-3cd952f59b25>:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["BloodPressure"])
```




```
plt.figure(figsize=(20,6))

plt.subplot(1,3,1)
plt.title("Counter Plot")
sns.countplot(x = 'SkinThickness',data = df)
```

```
plt.subplot(1,3,2)
plt.title('Distribution Plot')
sns.distplot(df["SkinThickness"])

plt.subplot(1,3,3)
plt.title('Box Plot')
sns.boxplot(y=df["SkinThickness"])

plt.show()
```

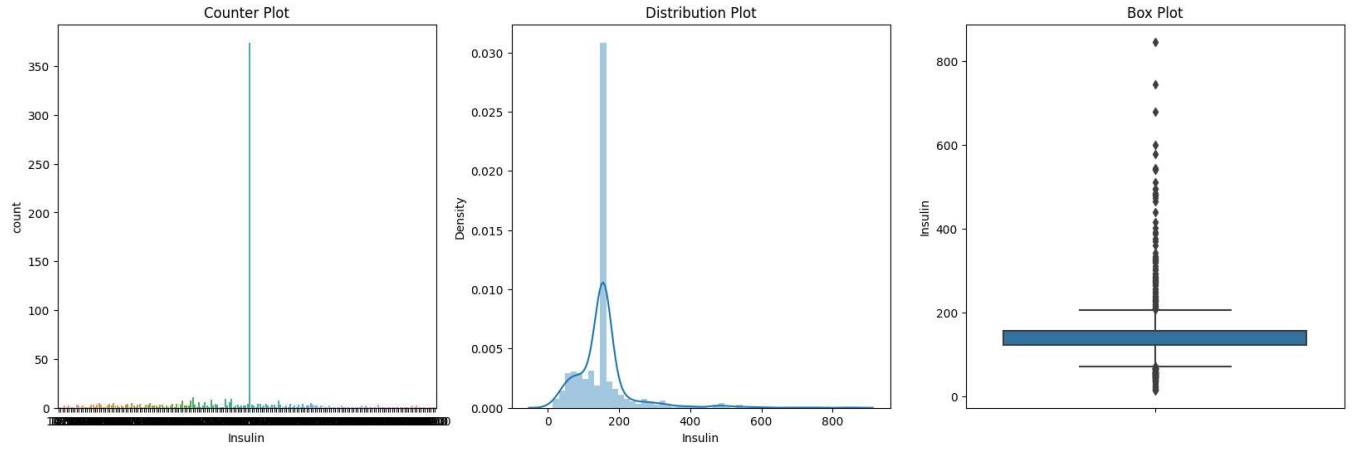
```
<ipython-input-55-dd18503852d4>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["SkinThickness"])
```




```
# Insulin
plt.figure(figsize=(20,6))

plt.subplot(1,3,1)
plt.title("Counter Plot")
sns.countplot(x = 'Insulin',data = df)

plt.subplot(1,3,2)
plt.title('Distribution Plot')
sns.distplot(df["Insulin"])

plt.subplot(1,3,3)
plt.title('Box Plot')
sns.boxplot(y=df["Insulin"])

plt.show()
```

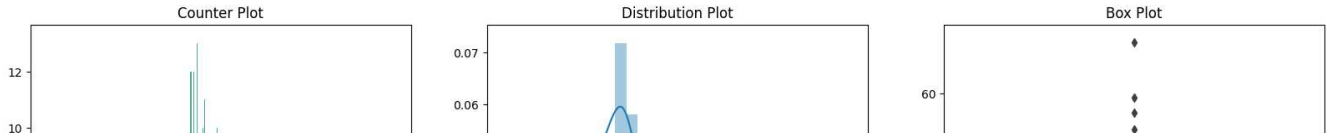
```
<ipython-input-56-b55057659347>:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

```
sns.distplot(df["Insulin"])
```

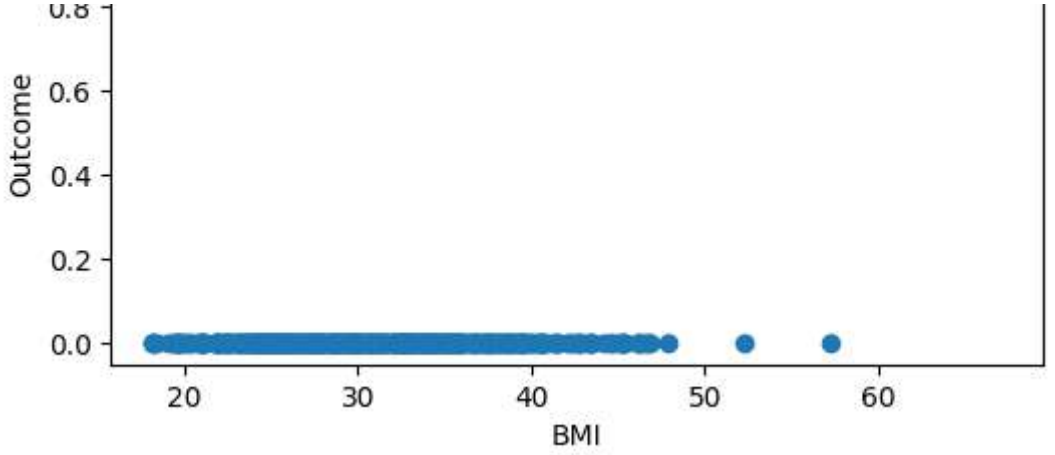


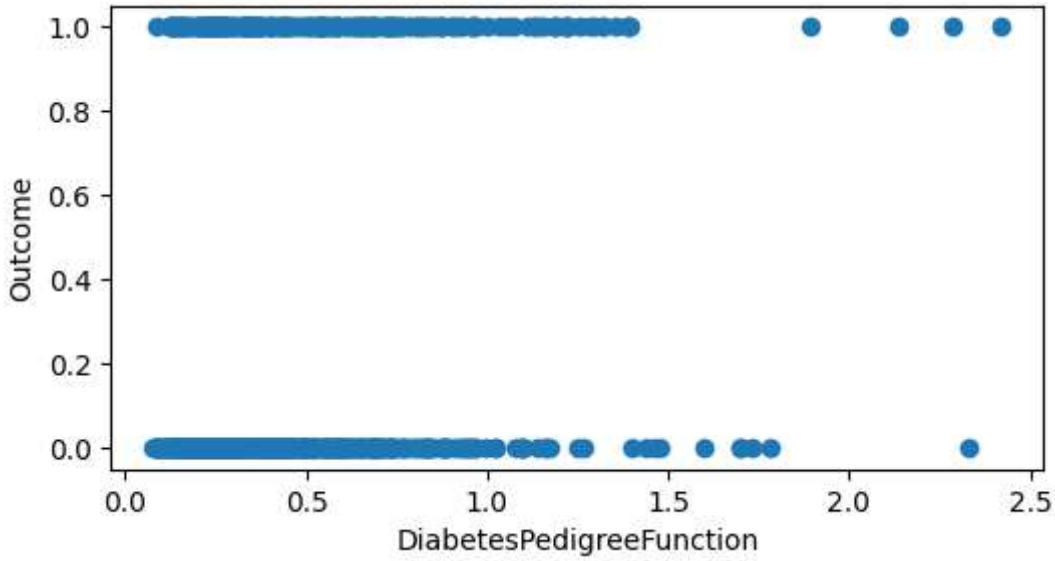

```
# BMI
plt.figure(figsize=(20,6))

plt.subplot(1,3,1)
plt.title("Counter Plot")
sns.countplot(x = 'BMI',data = df)

plt.subplot(1,3,2)
plt.title('Distribution Plot')
sns.distplot(df["BMI"])

plt.subplot(1,3,3)
plt.title('Box Plot')
sns.boxplot(y=df["BMI"])

plt.show()
```

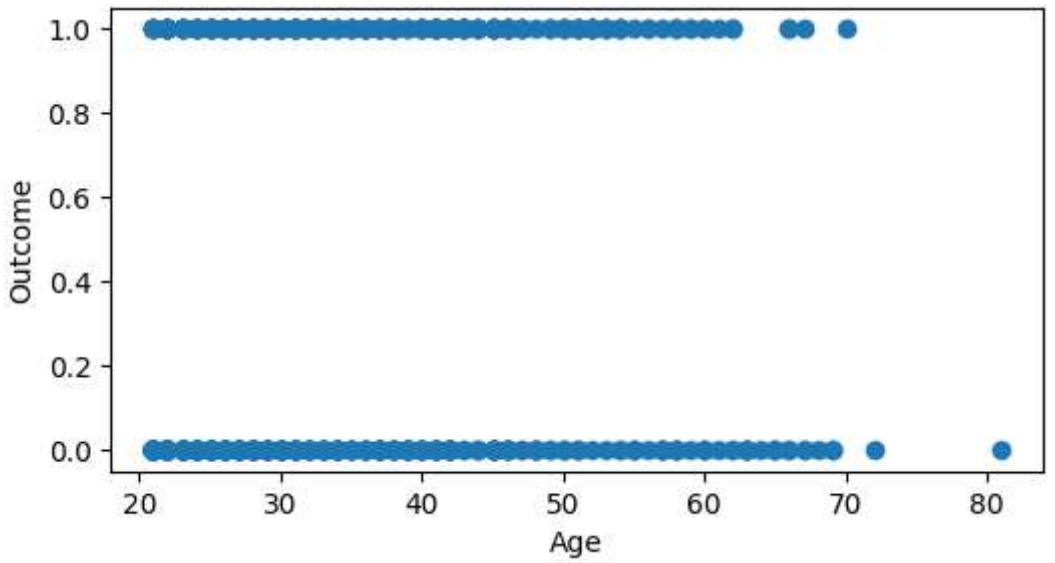
```
<ipython-input-57-a409f7672fe7>:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["BMI"])
```



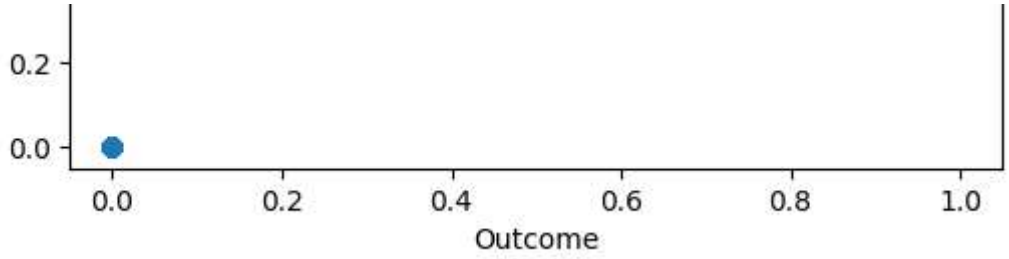

```
for feature in numerical_features:
    plt.figure(figsize=(6,3))
    plt.scatter(y=df["Outcome"], x=df[feature])
    plt.ylabel("Outcome")
    plt.xlabel(feature)
    plt.show()
```


Outcome
0.0
0.2
0.4
0.6
20
30
40
50
60
BMI


Outcome
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
2.0
2.5
DiabetesPedigreeFunction


Outcome
0.0
0.2
0.4
0.6
0.8
1.0
20
30
40
50
60
70
80
Age

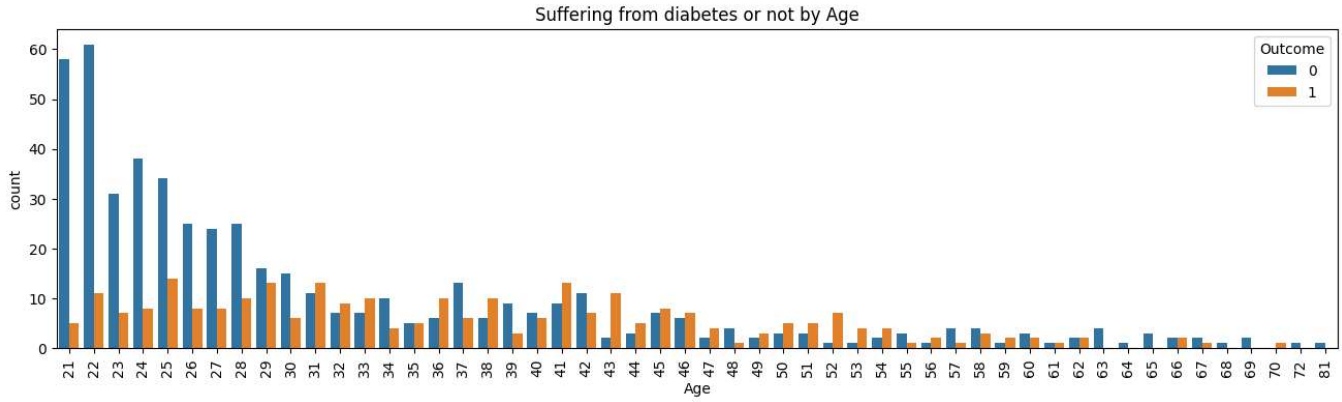

Outcome
0.4
0.6
0.8
1.0


0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Outcome

```
np.sort(df.Age.unique())
```

```
array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
       38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
       55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 72,
       81])
```

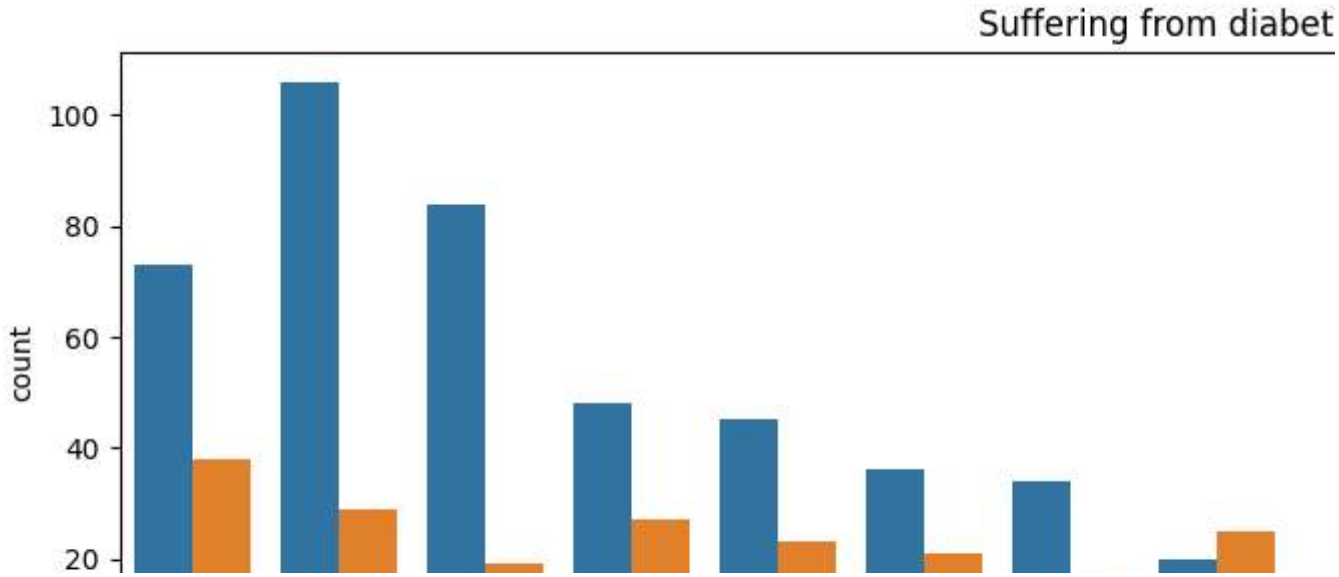
```
plt.figure(figsize=(16,4))
plt.xticks(rotation = 90)
ax = sns.countplot(x= df.Age , hue= df.Outcome)
ax.set_title('Suffering from diabetes or not by Age')
plt.show()
```




```
np.sort(df.Pregnancies.unique())
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 17])
```

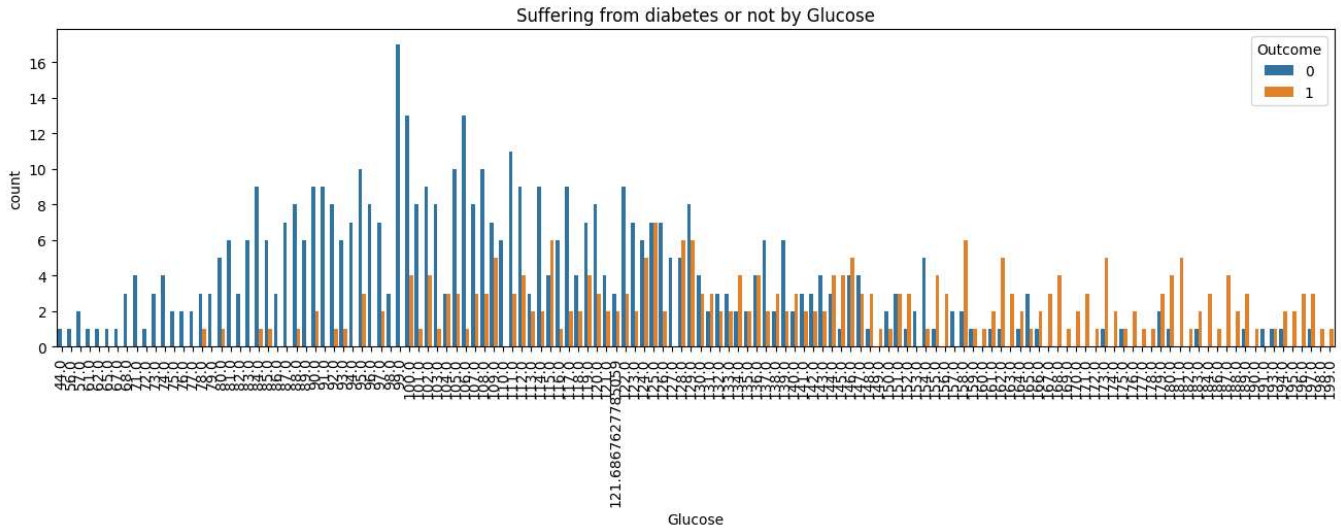
```
plt.figure(figsize=(16,4))
plt.xticks(rotation = 90)
ax = sns.countplot(x= df.Pregnancies , hue= df.Outcome)
ax.set_title('Suffering from diabetes or not by Pregnancies')
plt.show()
```




```
np.sort(df.Glucose.unique())
```

```
array([ 44.        ,  56.        ,  57.        ,  61.        ,
        62.        ,  65.        ,  67.        ,  68.        ,
        71.        ,  72.        ,  73.        ,  74.        ,
        75.        ,  76.        ,  77.        ,  78.        ,
        79.        ,  80.        ,  81.        ,  82.        ,
        83.        ,  84.        ,  85.        ,  86.        ,
        87.        ,  88.        ,  89.        ,  90.        ,
        91.        ,  92.        ,  93.        ,  94.        ,
        95.        ,  96.        ,  97.        ,  98.        ,
        99.        , 100.        , 101.        , 102.        ,
       103.        , 104.        , 105.        , 106.        ,
       107.        , 108.        , 109.        , 110.        ,
       111.        , 112.        , 113.        , 114.        ,
       115.        , 116.        , 117.        , 118.        ,
       119.        , 120.        , 121.        , 121.68676278,
       122.        , 123.        , 124.        , 125.        ,
       126.        , 127.        , 128.        , 129.        ,
       130.        , 131.        , 132.        , 133.        ,
       134.        , 135.        , 136.        , 137.        ,
       138.        , 139.        , 140.        , 141.        ,
       142.        , 143.        , 144.        , 145.        ,
       146.        , 147.        , 148.        , 149.        ,
       150.        , 151.        , 152.        , 153.        ,
       154.        , 155.        , 156.        , 157.        ,
       158.        , 159.        , 160.        , 161.        ,
       162.        , 163.        , 164.        , 165.        ,
       166.        , 167.        , 168.        , 169.        ,
       170.        , 171.        , 172.        , 173.        ,
       174.        , 175.        , 176.        , 177.        ,
       178.        , 179.        , 180.        , 181.        ,
       182.        , 183.        , 184.        , 186.        ,
       187.        , 188.        , 189.        , 190.        ,
       191.        , 193.        , 194.        , 195.        ,
       196.        , 197.        , 198.        , 199.        ])
```

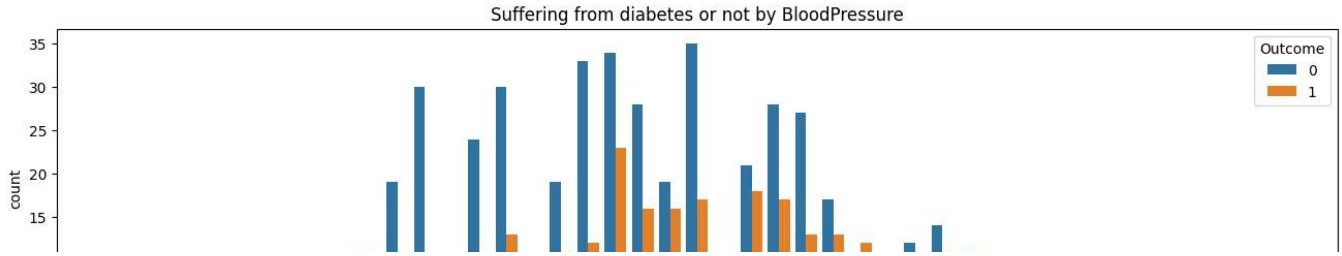
```
plt.figure(figsize=(16,4))
plt.xticks(rotation = 90)
ax = sns.countplot(x= df.Glucose , hue= df.Outcome)
ax.set_title('Suffering from diabetes or not by Glucose')
plt.show()
```




```
np.sort(df.BloodPressure.unique())
```

```
array([ 24.        ,  30.        ,  38.        ,  40.        ,
        44.        ,  46.        ,  48.        ,  50.        ,
        52.        ,  54.        ,  55.        ,  56.        ,
        58.        ,  60.        ,  61.        ,  62.        ,
        64.        ,  65.        ,  66.        ,  68.        ,
        70.        ,  72.        ,  72.40518417,  74.        ,
        75.        ,  76.        ,  78.        ,  80.        ,
        82.        ,  84.        ,  85.        ,  86.        ,
        88.        ,  90.        ,  92.        ,  94.        ,
        95.        ,  96.        ,  98.        , 100.        ,
       102.        , 104.        , 106.        , 108.        ,
       110.        , 114.        , 122.        ])
```

```
plt.figure(figsize=(16,4))
plt.xticks(rotation = 90)
ax = sns.countplot(x= df.BloodPressure , hue= df.Outcome)
ax.set_title('Suffering from diabetes or not by BloodPressure')
plt.show()
```




```
np.sort(df.Insulin.unique())
```

```
array([ 14.        ,  15.        ,  16.        ,  18.        ,
        22.        ,  23.        ,  25.        ,  29.        ,
        32.        ,  36.        ,  37.        ,  38.        ,
        40.        ,  41.        ,  42.        ,  43.        ,
        44.        ,  45.        ,  46.        ,  48.        ,
        49.        ,  50.        ,  51.        ,  52.        ,
        53.        ,  54.        ,  55.        ,  56.        ,
        57.        ,  58.        ,  59.        ,  60.        ,
        61.        ,  63.        ,  64.        ,  65.        ,
        66.        ,  67.        ,  68.        ,  70.        ,
        71.        ,  72.        ,  73.        ,  74.        ,
        75.        ,  76.        ,  77.        ,  78.        ,
        79.        ,  81.        ,  82.        ,  83.        ,
        84.        ,  85.        ,  86.        ,  87.        ,
        88.        ,  89.        ,  90.        ,  91.        ,
        92.        ,  94.        ,  95.        ,  96.        ,
        99.        , 100.        , 105.        , 106.        ,
       108.        , 110.        , 112.        , 114.        ,
       115.        , 116.        , 119.        , 120.        ,
       122.        , 125.        , 126.        , 127.        ,
       128.        , 129.        , 130.        , 132.        ,
       135.        , 140.        , 142.        , 144.        ,
       145.        , 146.        , 148.        , 150.        ,
       152.        , 155.        , 155.54822335, 156.        ,
       158.        , 159.        , 160.        , 165.        ,
       166.        , 167.        , 168.        , 170.        ,
       171.        , 175.        , 176.        , 178.        ,
       180.        , 182.        , 183.        , 184.        ,
       185.        , 188.        , 190.        , 191.        ,
       192.        , 193.        , 194.        , 196.        ,
       200.        , 204.        , 205.        , 207.        ,
       210.        , 215.        , 220.        , 225.        ,
       228.        , 230.        , 231.        , 235.        ,
       237.        , 240.        , 245.        , 249.        ,
       250.        , 255.        , 258.        , 265.        ,
       270.        , 271.        , 272.        , 274.        ,
       275.        , 277.        , 278.        , 280.        ,
       284.        , 285.        , 291.        , 293.        ,
       300.        , 304.        , 310.        , 318.        ,
       321.        , 325.        , 326.        , 328.        ,
       330.        , 335.        , 342.        , 360.        ,
       370.        , 375.        , 387.        , 392.        ,
       402.        , 415.        , 440.        , 465.        ,
       474.        , 478.        , 480.        , 485.        ,
       495.        , 510.        , 540.        , 543.        ,
```

```
       545.        , 579.        , 600.        , 680.        ,
       744.        , 846.        ])
```

```
df.corr().style.background_gradient(cmap="Blues")
```

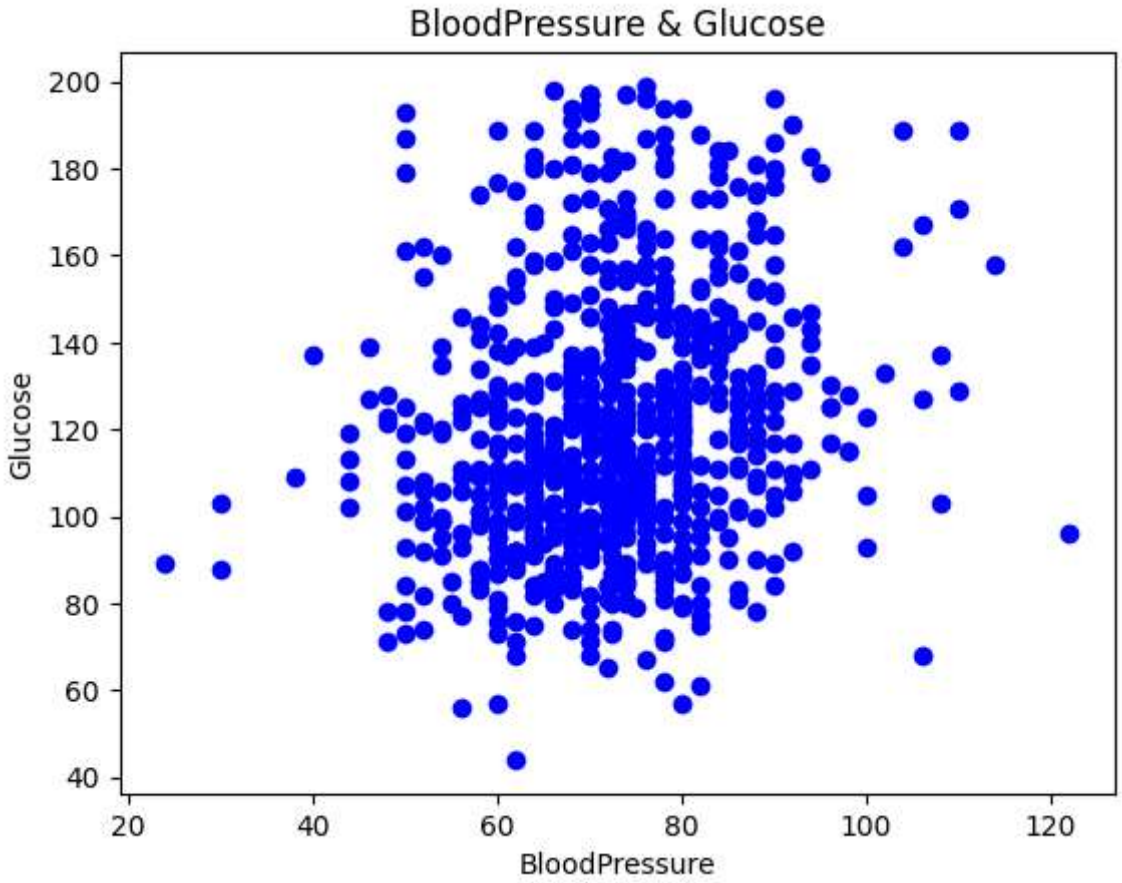
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
Pregnancies	1.000000	0.127911	0.208522	0.082989	0.056027	
Glucose	0.127911	1.000000	0.218367	0.192991	0.420157	
BloodPressure	0.208522	0.218367	1.000000	0.192816	0.072517	
SkinThickness	0.082989	0.192991	0.192816	1.000000	0.158139	
Insulin	0.056027	0.420157	0.072517	0.158139	1.000000	
BMI	0.021565	0.230941	0.281268	0.542398	0.166586	
DiabetesPedigreeFunction	-0.033523	0.137060	-0.002763	0.100966	0.098634	
Age	0.544341	0.266534	0.324595	0.127872	0.136734	
Outcome	0.221898	0.492928	0.166074	0.215299	0.214411	

```
plt.figure(figsize=(5,3),dpi=100)
plt.title('Checking Missing Value with Heatmap')
sns.heatmap(df.isnull(),cmap='plasma',yticklabels=False)
```

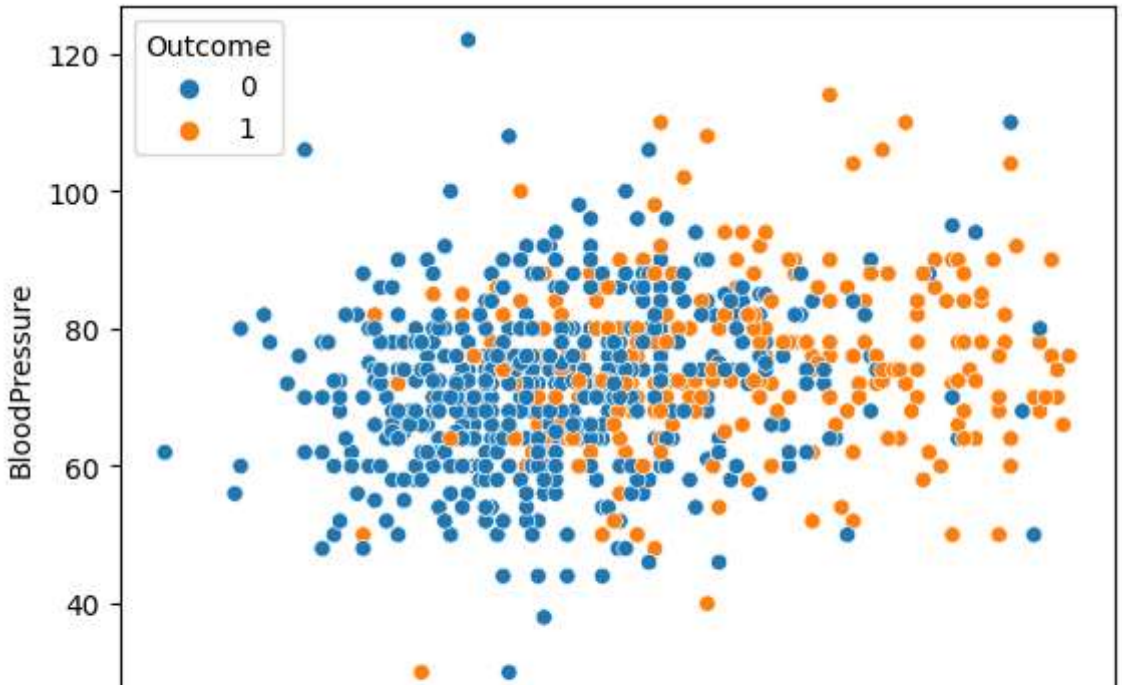
```
<Axes: title={'center': 'Checking Missing Value with Heatmap'}>
```


```
BloodPressure = df['BloodPressure']
Glucose = df['Glucose']
SkinThickness = df['SkinThickness']
Insulin = df['Insulin']
BMI = df['BMI']
```

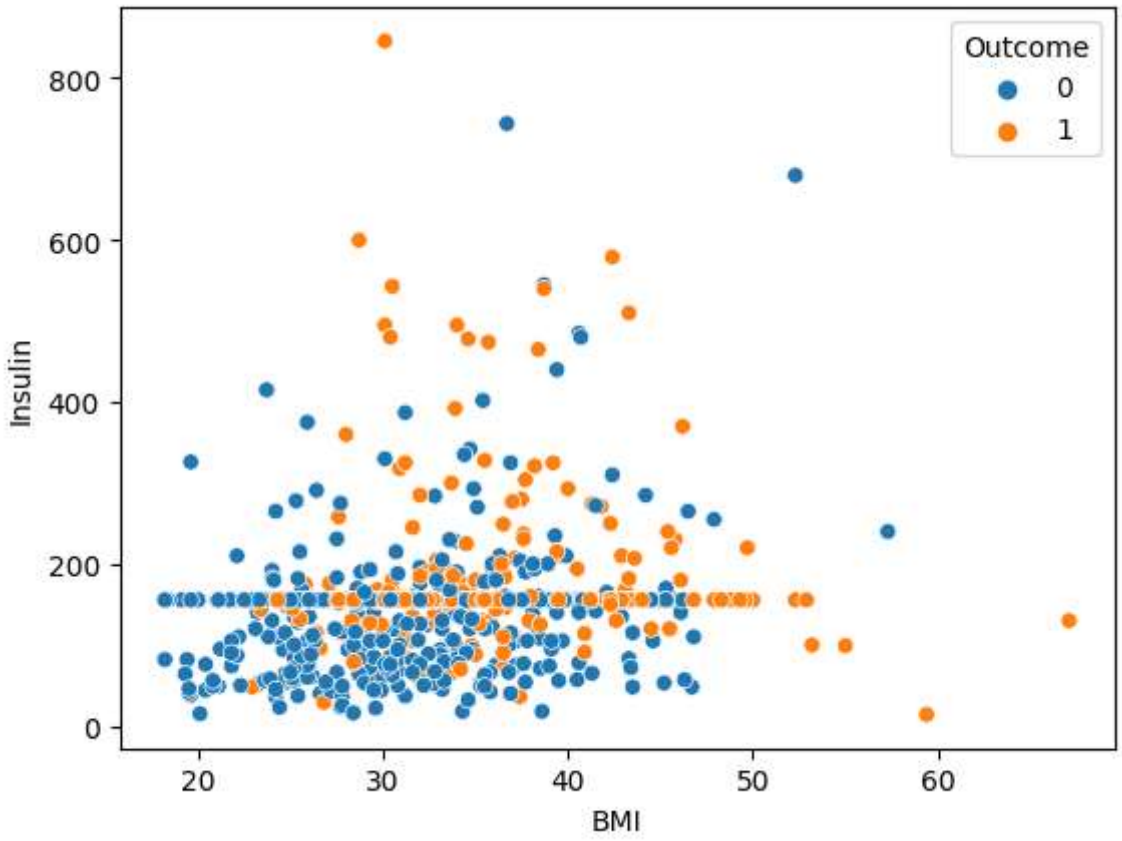
```
plt.scatter(BloodPressure, Glucose, color=['b'])
plt.xlabel('BloodPressure')
plt.ylabel('Glucose')
plt.title('BloodPressure & Glucose')
plt.show()
```



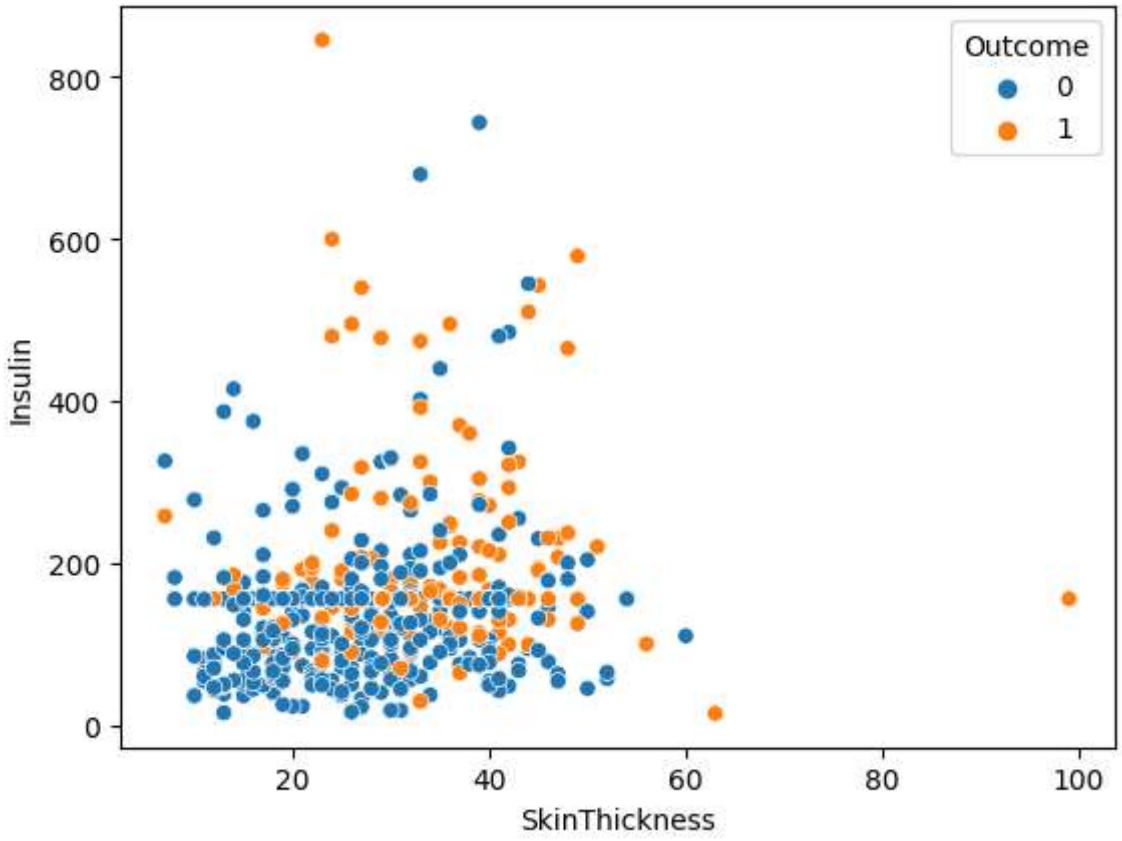

```
g =sns.scatterplot(x= "Glucose" ,y= "BloodPressure",
              hue="Outcome",
              data=df);
```




```
B =sns.scatterplot(x= "BMI" ,y= "Insulin",
                   hue="Outcome",
                   data=df);
```

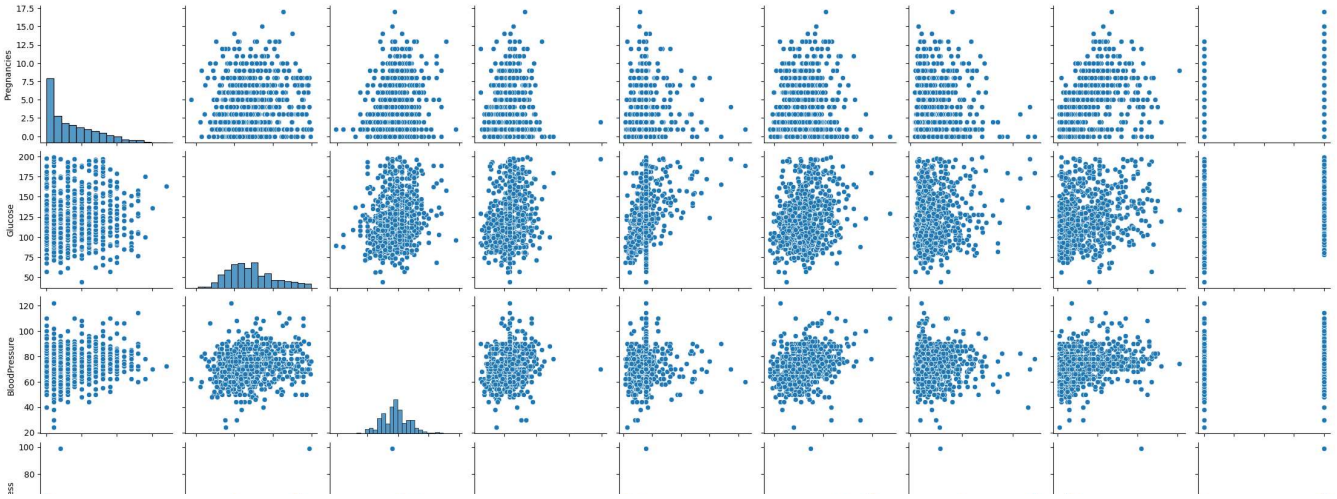



```
S =sns.scatterplot(x= "SkinThickness" ,y= "Insulin",
                   hue="Outcome",
                   data=df);
```




```
sns.pairplot(df)
plt.title('Scatter plot between variables')
```

```
Text(0.5, 1.0, 'Scatter plot between variables')
```




```
df_new=df
df_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] = df_new[["Glucose
```

```
df_new.isnull().sum()
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
df_new["Glucose"].fillna(df_new["Glucose"].mean(), inplace = True)
df_new["BloodPressure"].fillna(df_new["BloodPressure"].mean(), inplace = True)
df_new["SkinThickness"].fillna(df_new["SkinThickness"].mean(), inplace = True)
df_new["Insulin"].fillna(df_new["Insulin"].mean(), inplace = True)
df_new["BMI"].fillna(df_new["BMI"].mean(), inplace = True)
```

```
df_new.isnull().sum()
```

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

```
y=df_new['Outcome']
X=df_new.drop('Outcome',axis=1)


x=(X-np.min(X))/(np.max(X)-np.min(X)).values
```

```
/usr/local/lib/python3.9/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning: In a
  return reduction(axis=axis, out=out, **passkwargs)
/usr/local/lib/python3.9/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning: In a
  return reduction(axis=axis, out=out, **passkwargs)
/usr/local/lib/python3.9/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning: In a
  return reduction(axis=axis, out=out, **passkwargs)
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_state=42)


def initiaize_weights_and_bias(dimension):
    w=np.full((dimension,1),0.01)
    b=0.01
    return w,b


def sigmoid(z):
    y_head= 1/(1+np.exp(-z))
    return y_head


def forward_backward_propagation(w,b,x_train,y_train):
    z=np.dot(w.T,x_train)+b
    y_head=sigmoid(z)
    loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
    cost = (np.sum(loss))/x_train.shape[1]      # x_train.shape[1]  is for scaling

    derivative_weight = (np.dot(x_train,((y_head-y_train).T)))/x_train.shape[1] # x_train.sha
    derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]                 # x_train.shape
    gradients = {"derivative_weight": derivative_weight, "derivative_bias": derivative_bias}

    return cost,gradients


def update(w, b, x_train, y_train, learning_rate,number_of_iterarion):
    cost_list = []
    cost_list2 = []
    index = []

    for i in range(number_of_iterarion):
        cost,gradients = forward_backward_propagation(w,b,x_train,y_train)
        cost_list.append(cost)
        w = w - learning_rate * gradients["derivative_weight"]
        b = b - learning_rate * gradients["derivative_bias"]
```

```
            if i % 10 == 0:
                cost_list2.append(cost)
                index.append(i)
                print ("Cost after iteration %i: %f" %(i, cost))

        parameters = {"weight": w,"bias": b}
        plt.plot(index,cost_list2)
        plt.xticks(index,rotation='vertical')
        plt.xlabel("Number of Iterarion")
        plt.ylabel("Cost")
        plt.show()
        return parameters, gradients, cost_list



    def predict(w,b,x_test):
        # x_test is a input for forward propagation
        z = sigmoid(np.dot(w.T,x_test)+b)
        Y_prediction = np.zeros((1,x_test.shape[1]))
        # if z is bigger than 0.5, our prediction is sign one (y_head=1),
        # if z is smaller than 0.5, our prediction is sign zero (y_head=0),
        for i in range(z.shape[1]):
            if z[0,i]<= 0.5:
                Y_prediction[0,i] = 0
            else:
                Y_prediction[0,i] = 1

        return Y_prediction


    from sklearn.model_selection import train_test_split

    X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.20,random_state=0,stratify=df_


    from sklearn.linear_model import LogisticRegression

    model=LogisticRegression()
    model.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: Convergenc
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

▾ LogisticRegression

LogisticRegression()

```
y_predict=model.predict(X_test)
y_predict
```

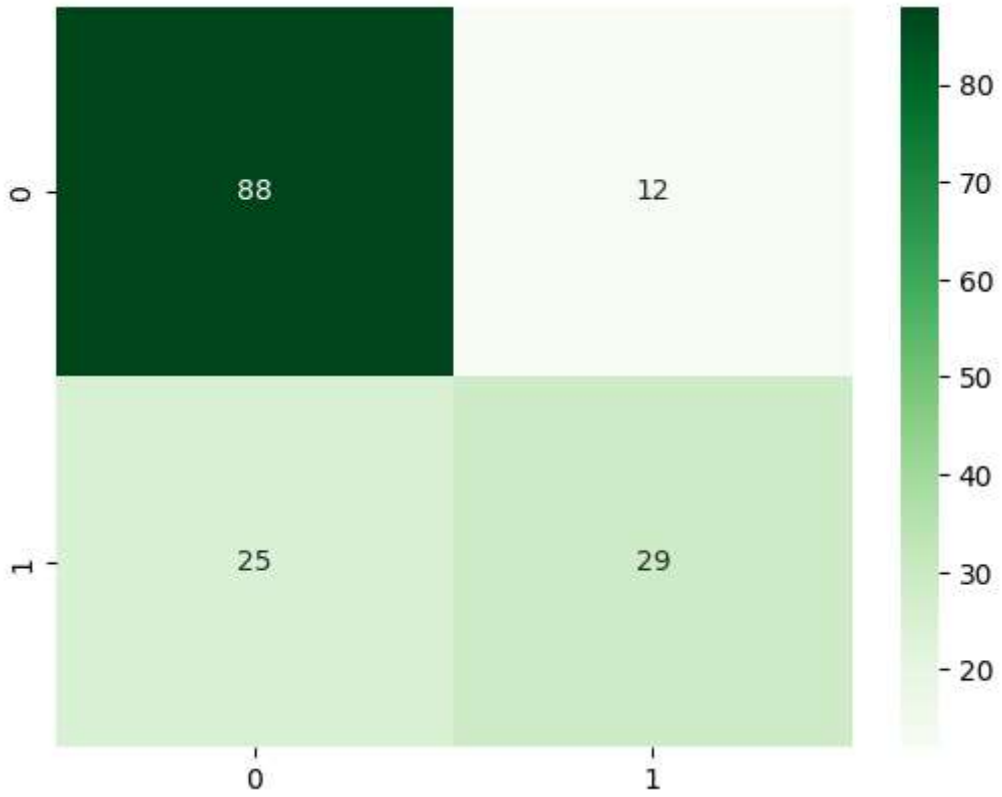
```
array([0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       1,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

```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(Y_test,y_predict)
cm
```

```
array([[88, 12],
       [25, 29]])
```

```
sns.heatmap(pd.DataFrame(cm),annot=True, cmap="Greens")
```

```
<Axes: >
```




```
from sklearn.metrics import accuracy_score

accuracy=accuracy_score(Y_test,y_predict)
print("Accuracy : ",round(accuracy,2)*100,'%')
```

```
Accuracy :  76.0 %
```

```
from imblearn.over_sampling import RandomOverSampler
r1=r1=RandomOverSampler()
x_data,y_data=r1.fit_resample(x,y)
```

```
from collections import Counter
Counter(y_data)
```

```
Counter({1: 500, 0: 500})
```

```
from sklearn.preprocessing import StandardScaler
sd=StandardScaler()
x_scaled=sd.fit_transform(x_data)
x_scaled
```

```
array([[ 0.55623253,  0.68910043, -0.06995388, ...,  0.08987138,
         0.38353449,  1.39038526],
       [-0.87735646, -1.31067768, -0.56581757, ..., -0.94686039,
        -0.4162322 , -0.26183314],
       [ 1.12966813,  1.80008826, -0.73110547, ..., -1.43560537,
         0.51393123, -0.17487427],
       ...,
       [ 1.12966813, -0.19968984,  1.08706142, ..., -0.68027222,
        -0.68282109, -1.0444629 ],
       [ 0.84295033,  1.32395062,  1.25234932, ...,  0.77115226,
         0.84716735,  0.52079663],
       [-1.16407426,  0.2764478 , -0.40052967, ...,  1.37838087,
        -0.37566432, -0.87054518]])
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=10)
```

```
from collections import Counter
print(Counter(y_data))
```

```
Counter({1: 500, 0: 500})
```

```
from sklearn.neighbors import KNeighborsClassifier
k1=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
k1.fit(x_train,y_train)
y_pred2=k1.predict(x_test)
y_pred2
from sklearn.metrics import accuracy_score
ac=accuracy_score(y_test,y_pred2)*100
print(ac)
```

```
74.67532467532467
```

```
from sklearn.ensemble import RandomForestClassifier
r1=RandomForestClassifier()
r1.fit(x_train,y_train)
y_pred4=r1.predict(x_test)
y_pred4
from sklearn.metrics import accuracy_score
ac=accuracy_score(y_test,y_pred4)*100
print(ac)
```

```
75.97402597402598
```

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred5=dt.predict(x_test)
y_pred5
accuracy_score(y_test,y_pred5)*100
```

```
70.12987012987013
```

```
from sklearn.model_selection import StratifiedKFold
skf=StratifiedKFold(n_splits=5,random_state=10,shuffle=True)
skf.get_n_splits(x)
print(skf)
```

```
StratifiedKFold(n_splits=5, random_state=10, shuffle=True)
```

```
from sklearn.model_selection import StratifiedKFold
skf=StratifiedKFold(n_splits=5,random_state=10,shuffle=True)
skf.get_n_splits(x)
print(skf)
```

```
StratifiedKFold(n_splits=5, random_state=10, shuffle=True)
```

```
X= df.drop(['Outcome'] , axis=1)
y= df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20)
```

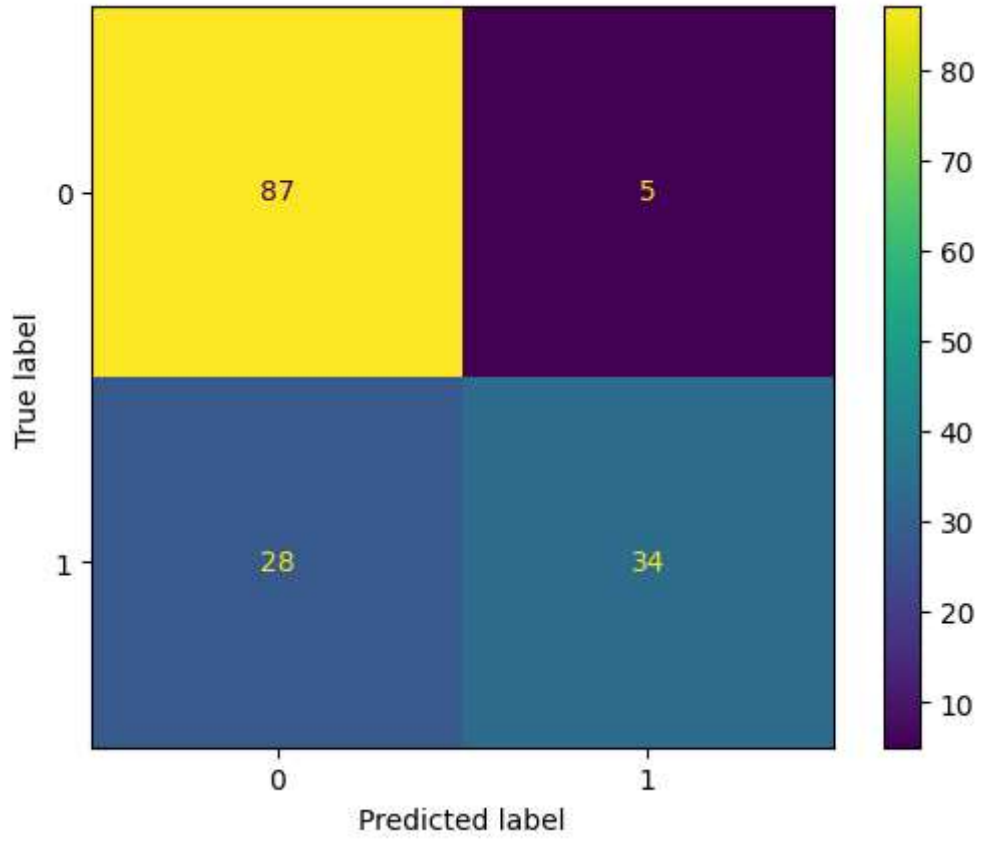
```
from sklearn import svm
```

```
 , ConfusionMatrixDisplay , accuracy_score , precision_score , recall_score , f1_score
```

```
clf = svm.SVC().fit(X_train, y_train)
acc train = clf.score(X train, y train)
```

```
y_pred = clf.predict(X_test)
acc_test = accuracy_score(y_pred , y_test)
print('accuracy_train: ',round(acc_train*100,2),'%')
print('accuracy_test: ', round(acc_test*100,2),'%')
print('precision_score: ', round(precision_score(y_test, y_pred),2))
print('recall_score: ', round(recall_score(y_test, y_pred),2))
print('f1_score: ', round(f1_score(y_test, y_pred),2))
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

```
accuracy_train:  75.9 %
accuracy_test:  78.57 %
precision_score:  0.87
recall_score:  0.55
f1_score:  0.67
```




```
from sklearn.ensemble import VotingClassifier
vote=VotingClassifier(estimators=[("Logistic",model),("KNN",k1),("SVM",clf),("random fores
vote.fit(x_train,y_train)
y_vote=vote.predict(x_test)
print("Voting Classifier",y_vote)
```

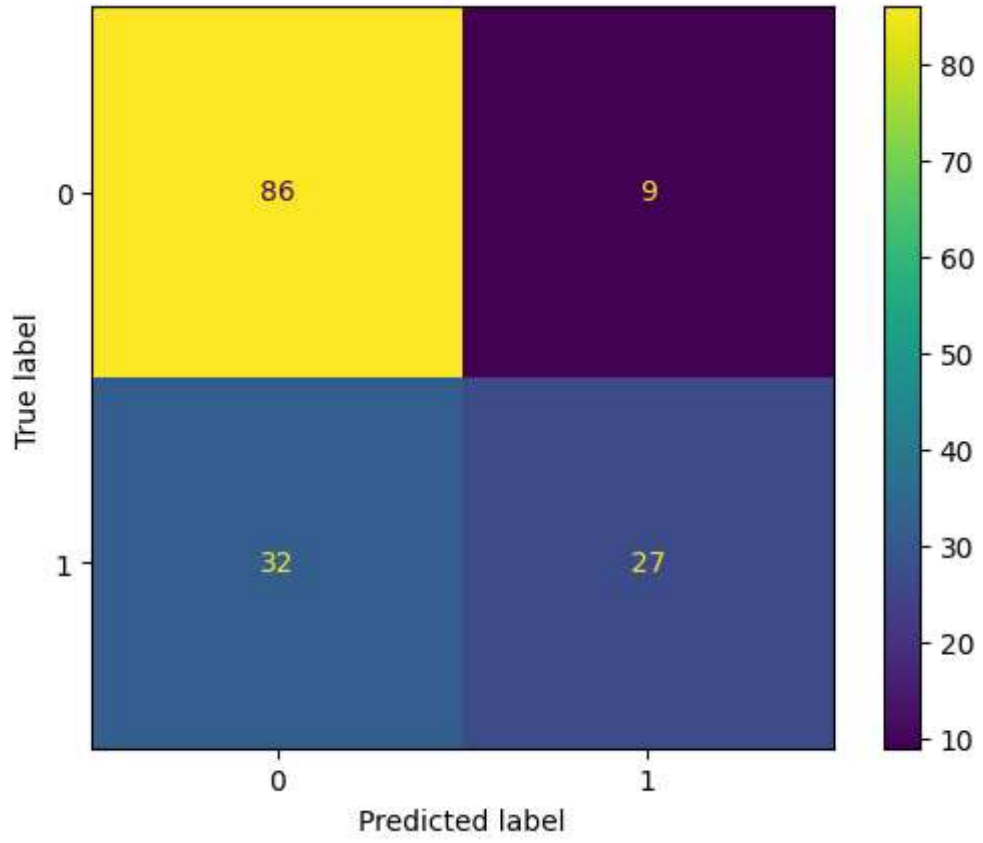
```
Voting Classifier [0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0
 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 1 0 1 0
```

```
 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1
 1 0 0 0 1 0]
```

```
Final_accuracy=accuracy_score(y_test,y_vote)
Final_accuracy*100
```

```
73.37662337662337
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
Confusion_Matrix=metrics.confusion_matrix(y_test,y_vote)
cm_display=metrics.ConfusionMatrixDisplay(Confusion_Matrix)
cm_display.plot()
plt.show()
```




```
Precision=metrics.precision_score(y_test,y_vote)*100
Precision
```

```
75.0
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_vote)
print(cm)
```

```
[[86  9]
 [32 27]]
```

```
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_vote)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.73      0.91      0.81        95
           1       0.75      0.46      0.57        59

    accuracy                           0.73       154
   macro avg       0.74      0.68      0.69       154
weighted avg       0.74      0.73      0.72       154
```

```
Sensitivity_recall = metrics.recall_score(y_test,y_vote)*100
print(Sensitivity_recall)
```

```
45.76271186440678
```

```
Specificity = metrics.recall_score(y_test,y_vote, pos_label=0)*100
print(Specificity)
```

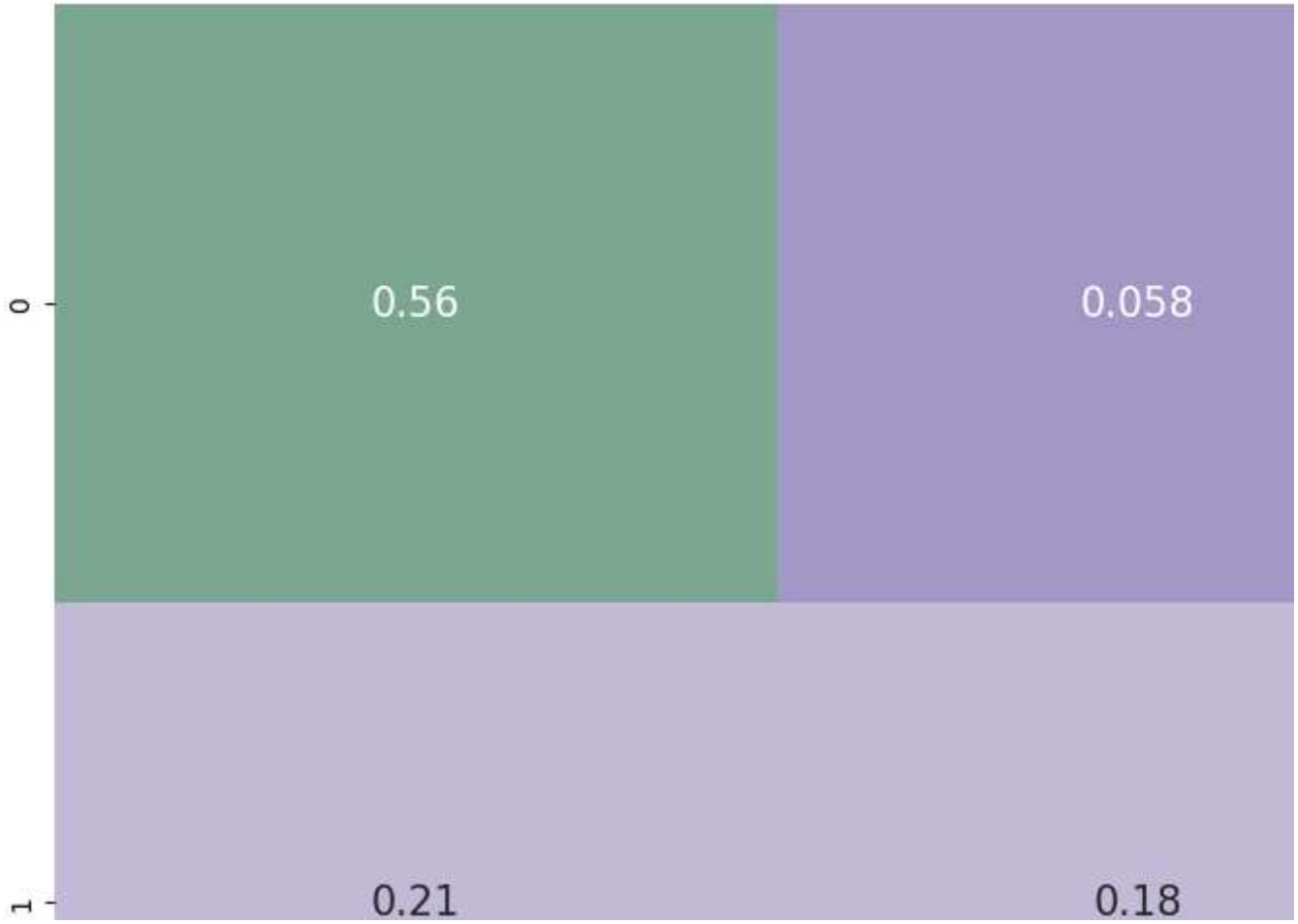
```
90.52631578947368
```

```
F1_score = metrics.f1_score(y_test,y_vote)*100
print(F1_score)
```

```
56.84210526315788
```

```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_r
cmap1 = sns.diverging_palette(275,150,  s=40, l=65, n=6)
plt.subplots(figsize=(12,8))
cf_matrix = confusion_matrix(y_test, y_vote)
sns.heatmap(cf_matrix/np.sum(cf_matrix), cmap = cmap1, annot = True, annot_kws = {'size':15})
```

```
<Axes: >
```




```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=0)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(x_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(614, 8)
(154, 8)
(614,)
(154,)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y,model.predict(x))
cm
```

```
array([[448,  52],
       [123, 145]])
```

```
from sklearn.preprocessing import StandardScaler
```

```
Scale=StandardScaler()
x_train_std=Scale.fit_transform(x_train)
x_test_std=Scale.transform(x_test)
```

```
norm=lambda a:(a-min(a))/(max(a)-min(a))
```

```
df_norm=df.iloc[:,:-1]
```

```
df_normalized=df_norm.apply(norm)
```

```
x_train_norm,x_test_norm,y_train_norm,y_test_norm=train_test_split(df_normalized.values,y,tes
```

```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=25)
#Using 25 Neighbors just as thumb rule sqrt of observation
knn_model.fit(x_train_std,y_train)
knn_pred=knn_model.predict(x_test_std)
```

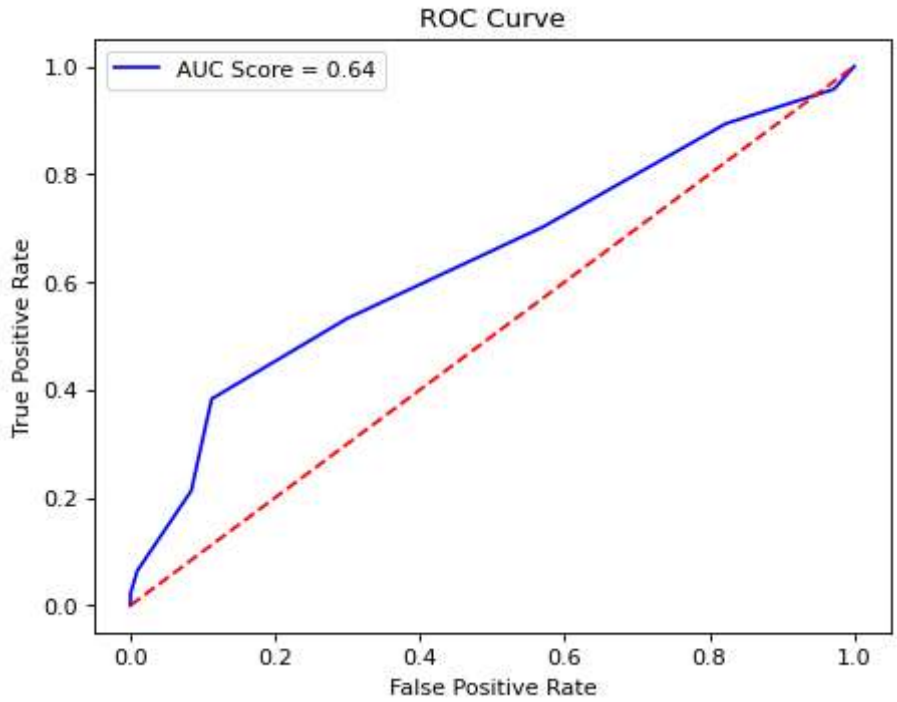
```
%matplotlib inline
```

```
from sklearn.neighbors import KNeighborsClassifier
knn_model_norm = KNeighborsClassifier(n_neighbors=25)
#Using 25 Neighbors just as thumb rule sqrt of observation
knn_model_norm.fit(x_train_norm,y_train_norm)
knn_pred_norm=knn_model_norm.predict(x_test_norm)
```

```
print("Model Validation ==>\n")
print("Accuracy Score of KNN Model with Normalization::")
print(metrics.accuracy_score(y_test_norm,knn_pred_norm))
print("\n","Classification Report::")
print(metrics.classification_report(y_test_norm,knn_pred_norm),'\n')
print("\n","ROC Curve")
knn_prob_norm=knn_model.predict_proba(x_test_norm)
knn_prob_norm1=knn_prob_norm[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test_norm,knn_prob_norm1)
```

```
roc_auc_knn=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_knn)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of KNN Model with Normalization::
0.8116883116883117

 Classification Report::
              precision    recall  f1-score   support

           0       0.84      0.90      0.87       107
           1       0.72      0.62      0.67        47

    accuracy                           0.81       154
   macro avg       0.78      0.76      0.77       154
weighted avg       0.81      0.81      0.81       154



 ROC Curve
<ipython-input-152-92ec9e2b26b3>:16: UserWarning: color is redundantly defined by the 'c
  plt.plot(fpr,fpr,'r--',color='red')
<matplotlib.legend.Legend at 0x7fd7892ab6a0>
```




```
from sklearn.svm import SVC
```

```
svc_model_linear = SVC(kernel='linear',random_state=0,probability=True,C=0.01)
svc_model_linear.fit(x_train_std,y_train)
svc_pred=svc_model_linear.predict(x_test_std)


print("Model Validation ==>\n")
print("Accuracy Score of SVC Model with Linear Kernel::")
print(metrics.accuracy_score(y_test,svc_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,svc_pred),'\n')
print("\n","ROC Curve")
svc_prob_linear=svc_model_linear.predict_proba(x_test_std)
svc_prob_linear1=svc_prob_linear[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,svc_prob_linear1)
roc_auc_svc=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_svc)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

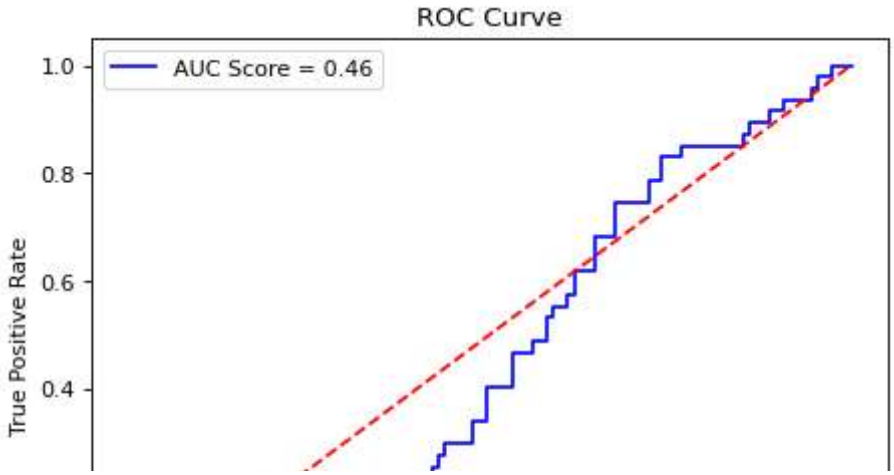
```
Model Validation ==>

Accuracy Score of SVC Model with Linear Kernel::
0.6948051948051948

 Classification Report::
              precision    recall  f1-score   support

           0       0.69      1.00      0.82       107
           1       0.00      0.00      0.00        47

    accuracy                           0.69       154
   macro avg       0.35      0.50      0.41       154
weighted avg       0.48      0.69      0.57       154



 ROC Curve
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefine
  warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.svm import SVC
svc_model_rbf = SVC(kernel='rbf',random_state=0,probability=True,C=1)
svc_model_rbf.fit(x_train_std,y_train)
svc_pred_rbf=svc_model_rbf.predict(x_test_std)
```

```
plt.plot(fpr,fpr,'r--',color='red')
```

```
print("Model Validation ==>\n")
print("Accuracy Score of SVC Model with RBF Kernel::")
print(metrics.accuracy_score(y_test,svc_pred_rbf))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,svc_pred_rbf),'\n')
print("\n","ROC Curve")
svc_prob_rbf=svc_model_linear.predict_proba(x_test_std)
svc_prob_rbf1=svc_prob_rbf[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,svc_prob_rbf1)
roc_auc_svc=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_svc)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of SVC Model with RBF Kernel::
0.7012987012987013

 Classification Report::
              precision    recall  f1-score   support

           0       0.70      1.00      0.82       107
           1       1.00      0.02      0.04        47

    accuracy                           0.70       154
   macro avg       0.85      0.51      0.43       154
weighted avg       0.79      0.70      0.58       154



 ROC Curve
<ipython-input-156-cc5662fbc78c>:16: UserWarning: color is redundantly defined by the 'c
  plt.plot(fpr,fpr,'r--',color='red')
<matplotlib.legend.Legend at 0x7fd789276d60>
```




```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(C=0.01)
lr_model.fit(x_train_std,y_train)
lr_pred=lr_model.predict(x_test_std)
```

False Positive Rate

```
print("Model Validation ==>\n")
print("Accuracy Score of Logistic Regression Model::")
print(metrics.accuracy_score(y_test,lr_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,lr_pred),'\n')
print("\n","ROC Curve")
lr_prob=lr_model.predict_proba(x_test_std)
lr_prob1=lr_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,lr_prob1)
roc_auc_lr=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_lr)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of Logistic Regression Model::
0.6948051948051948

 Classification Report::
              precision    recall  f1-score   support

           0       0.69      1.00      0.82       107
           1       0.00      0.00      0.00        47

    accuracy                           0.69       154
   macro avg       0.35      0.50      0.41       154
weighted avg       0.48      0.69      0.57       154



 ROC Curve
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
<ipython-input-158-b93c11c88ffa>:16: UserWarning: color is redundantly defined by the 'c
  plt.plot(fpr,fpr,'r--',color='red')
<matplotlib.legend.Legend at 0x7fd78ac388b0>
```