Import libraries

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

Read dataset

In [2]:
```
#read the dataset
data=pd.read_csv('health care diabetes.csv')
```

Perform basic EDA

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

Out[3]:

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

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

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
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

In [5]:
```
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 [6]:
```
data.columns
```

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

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

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

In [8]:
```
type(data)
```

Out[8]:
```
pandas.core.frame.DataFrame
```

Data preprocessing/Data Cleaning

Check for Duplicate value

In [9]:
```
data[data.duplicated]
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag

Check for Null Val

In [10]:
```
# will give the boolean value. Just check it.
data.isna().sum()
```

Out[10]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

WWWWWWWWWWWWWWWWWWWWWWWWWW

Data Exploration:

1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

- Glucose

- BloodPressure

- SkinThickness

- Insulin

- BMI

2. Visually explore these variables using histograms. Treat the missing values accordingly.

3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

In [11]:
```
data.describe()
```

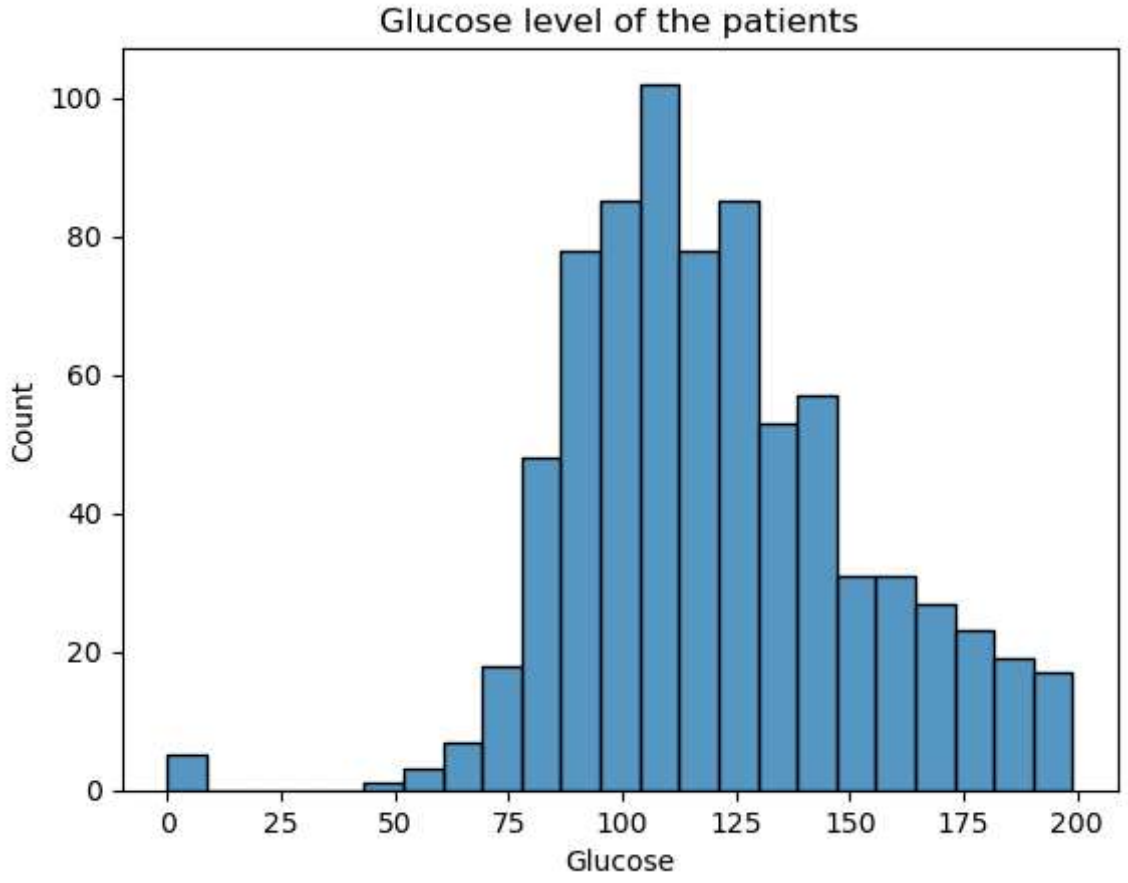
Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
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	

Data Visualization

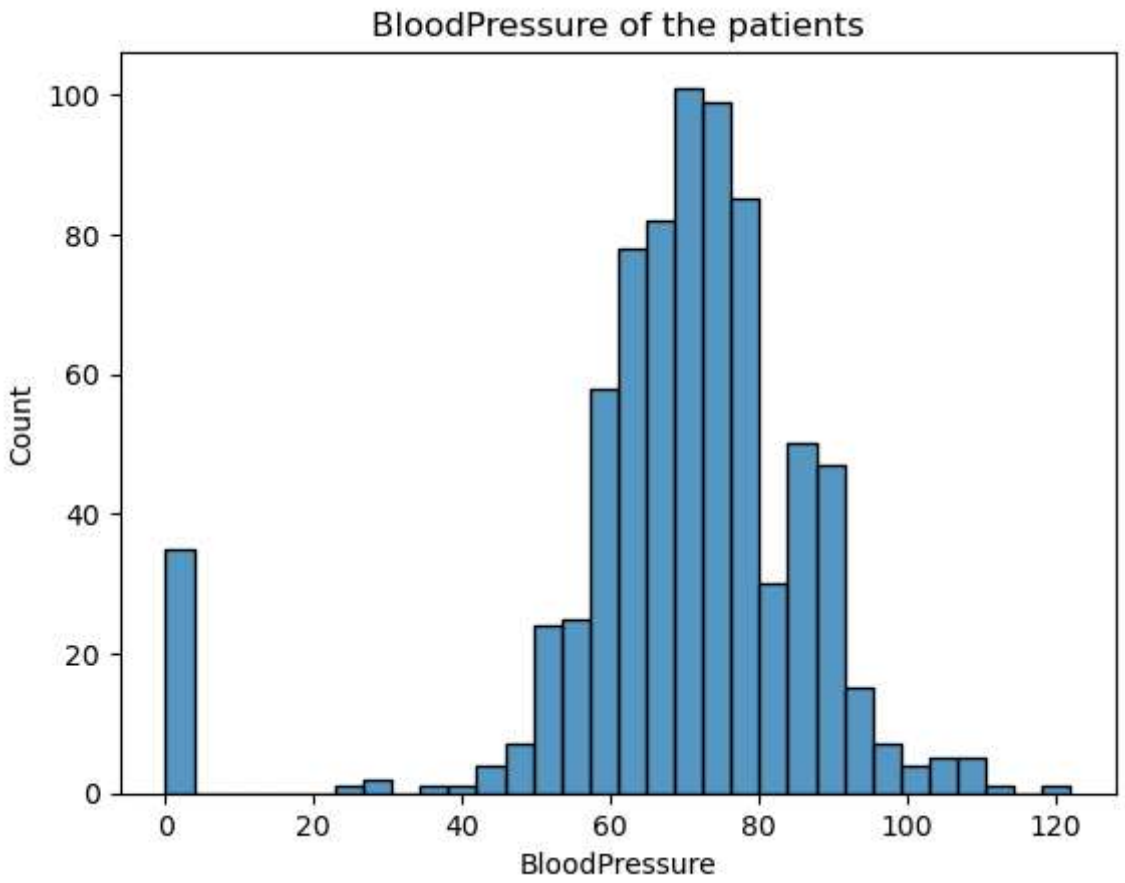
Count Plot

In [12]:
```
#Glucose
# plt.hist(data['Glucose'])
# plt.show()
sns.histplot(data.Glucose)  #sns is seaborn library
plt.title('Glucose level of the patients')
plt.show()
```

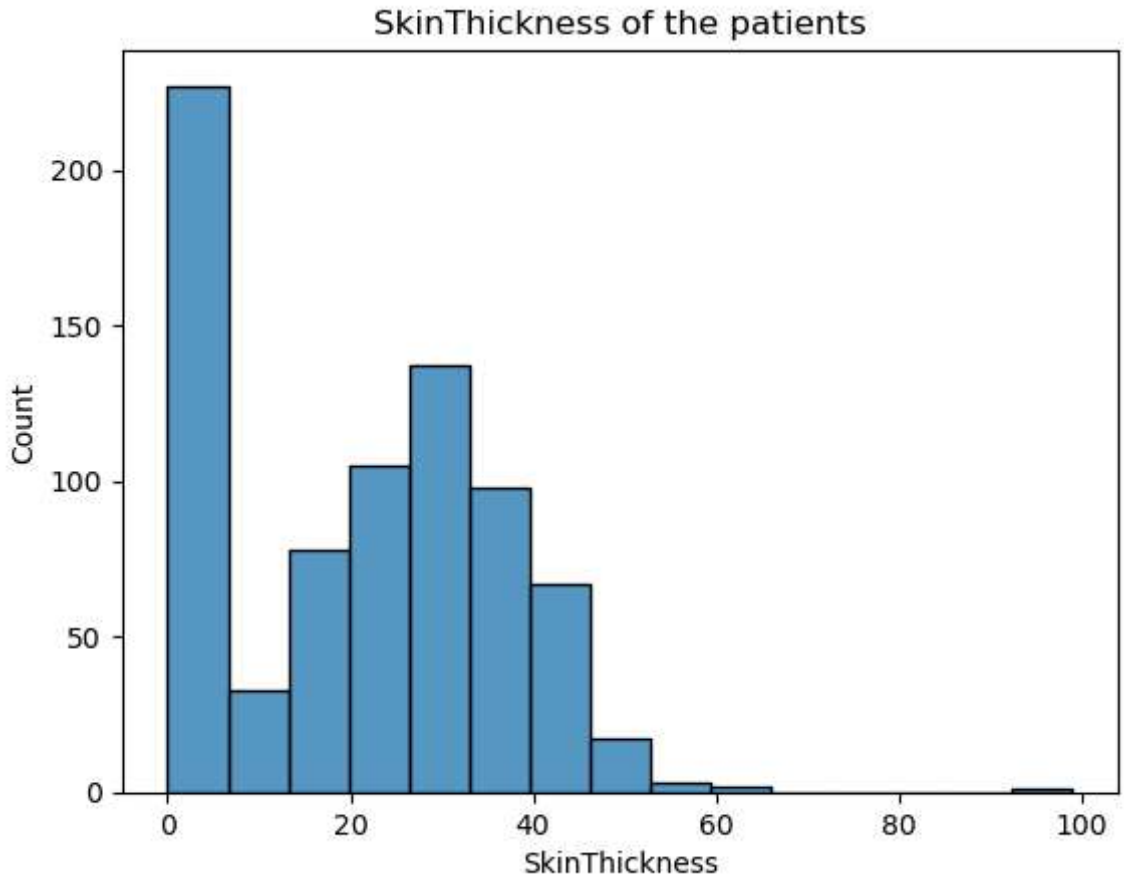


In [13]:
```
# BloodPressure
sns.histplot(data.BloodPressure)  #sns is seaborn library
```

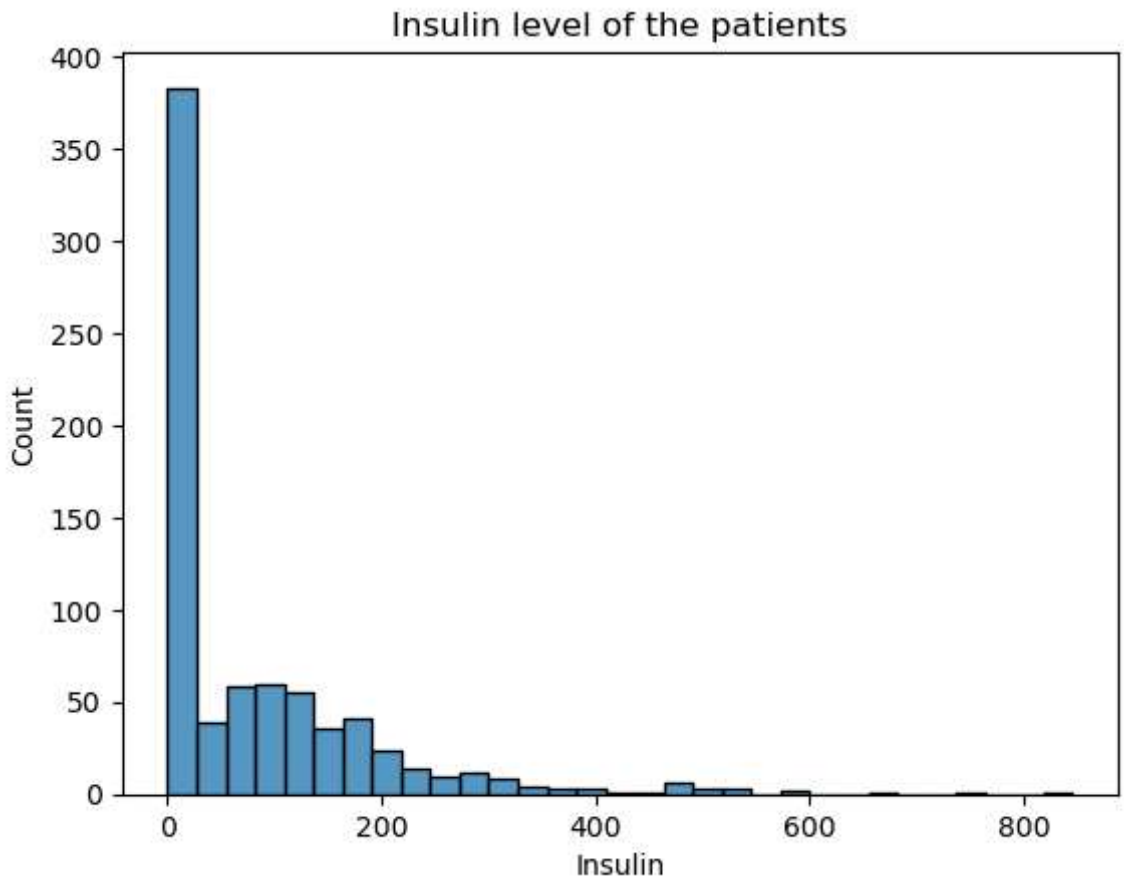
```
plt.title('BloodPressure of the patients')
plt.show()
```



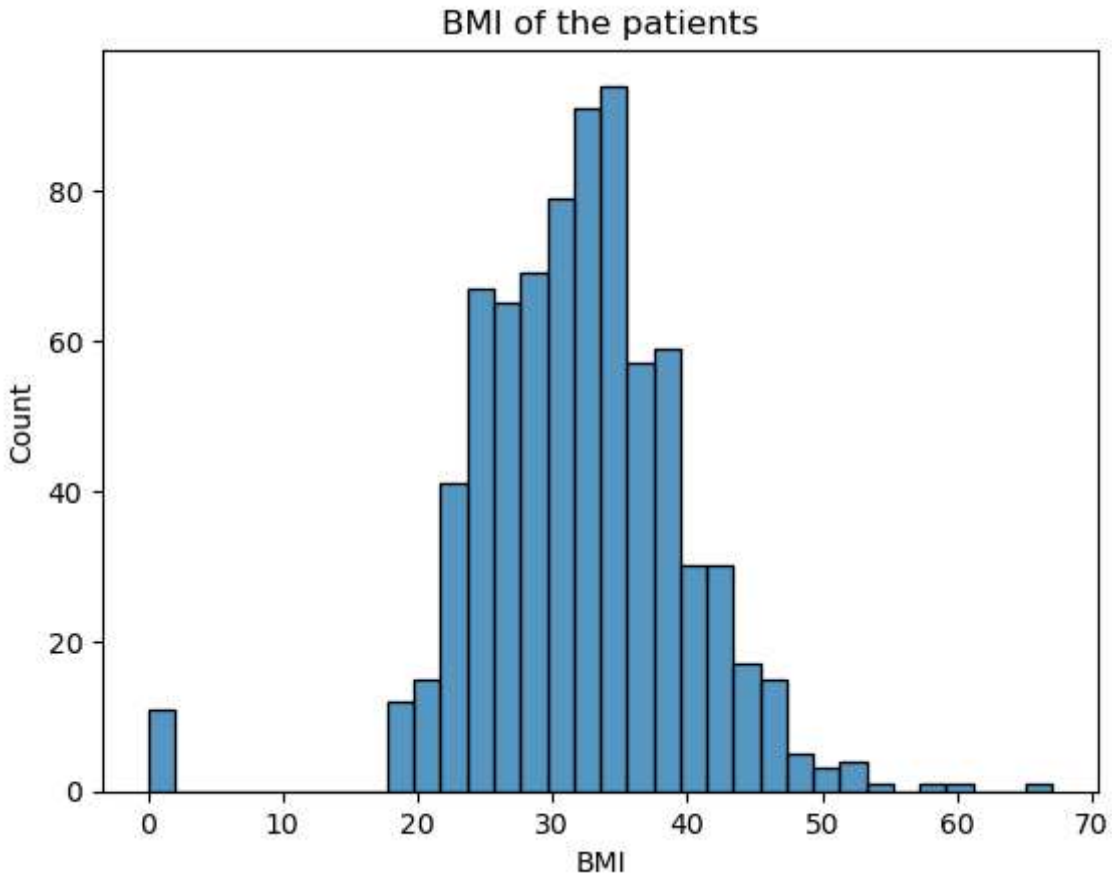
In [14]:
```
# SkinThickness
sns.histplot(data.SkinThickness)  #sns is seaborn library
plt.title('SkinThickness of the patients')
plt.show()
```



In [15]:
```
# Insulin
sns.histplot(data.Insulin)  #sns is seaborn library
plt.title('Insulin level of the patients')
plt.show()
```



In [16]:
```
# BMI
sns.histplot(data.BMI)  #sns is seaborn library
plt.title('BMI of the patients')
plt.show()
```



As shown in the point above, we can see missing (zero) values in the above 5 histograms. We have to handle them.Because, for example glucose value is zero means there is no meaning in it, similarly for others. So we have to handle these missing values (zeroes).

We can see that there is no outliers in these columns. So we can use median to replace all these zeroes.

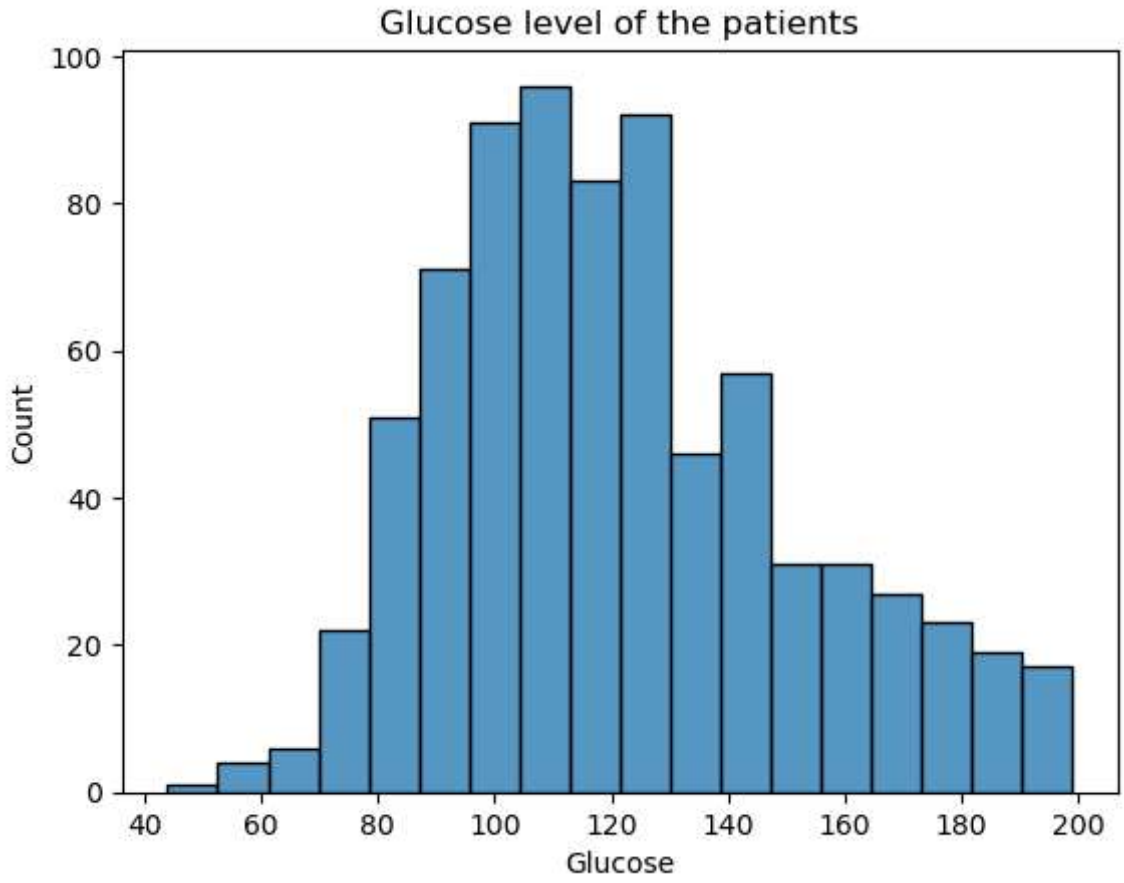
Replace these zero values with the median.

In [17]:
```
var=['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI']
for i in var:
    data[i].replace(0,data[i].median(),inplace=True)
```

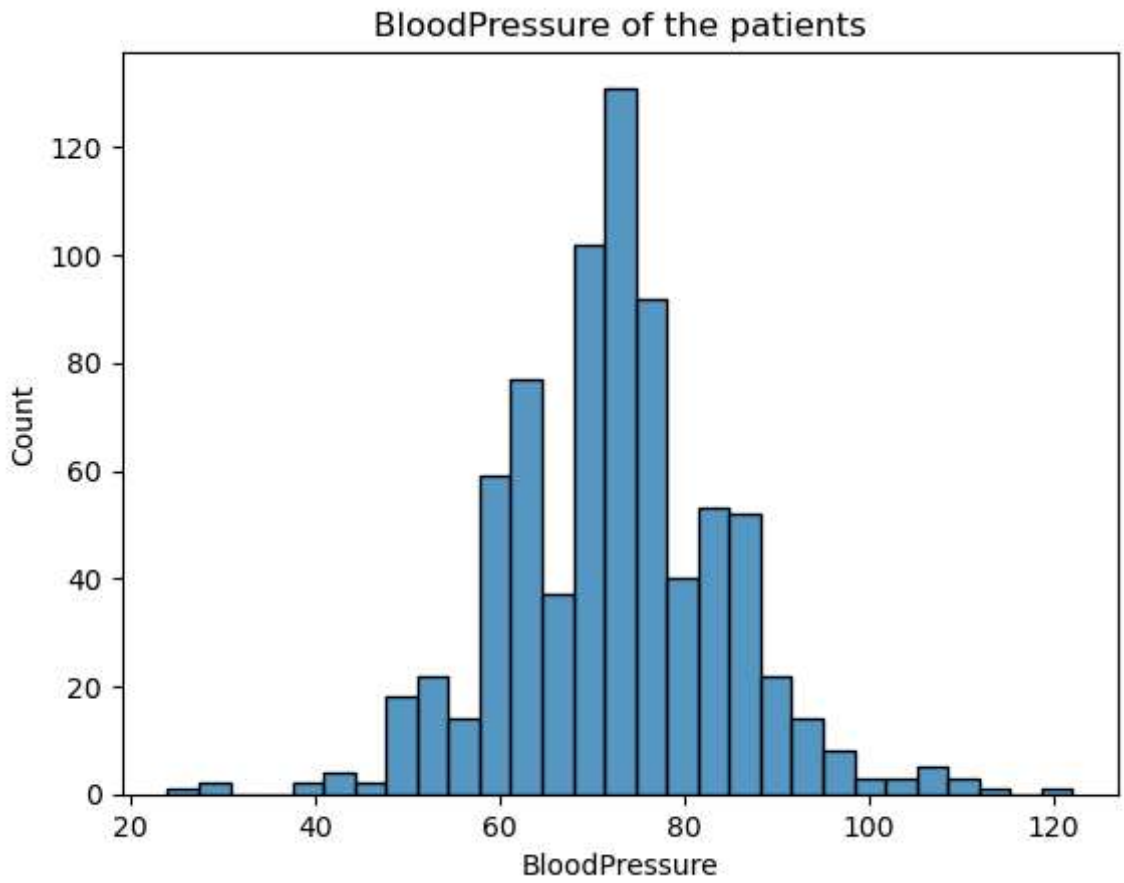
Count plots after zero values replaced.

Now check whether the zero values are replaced or not.

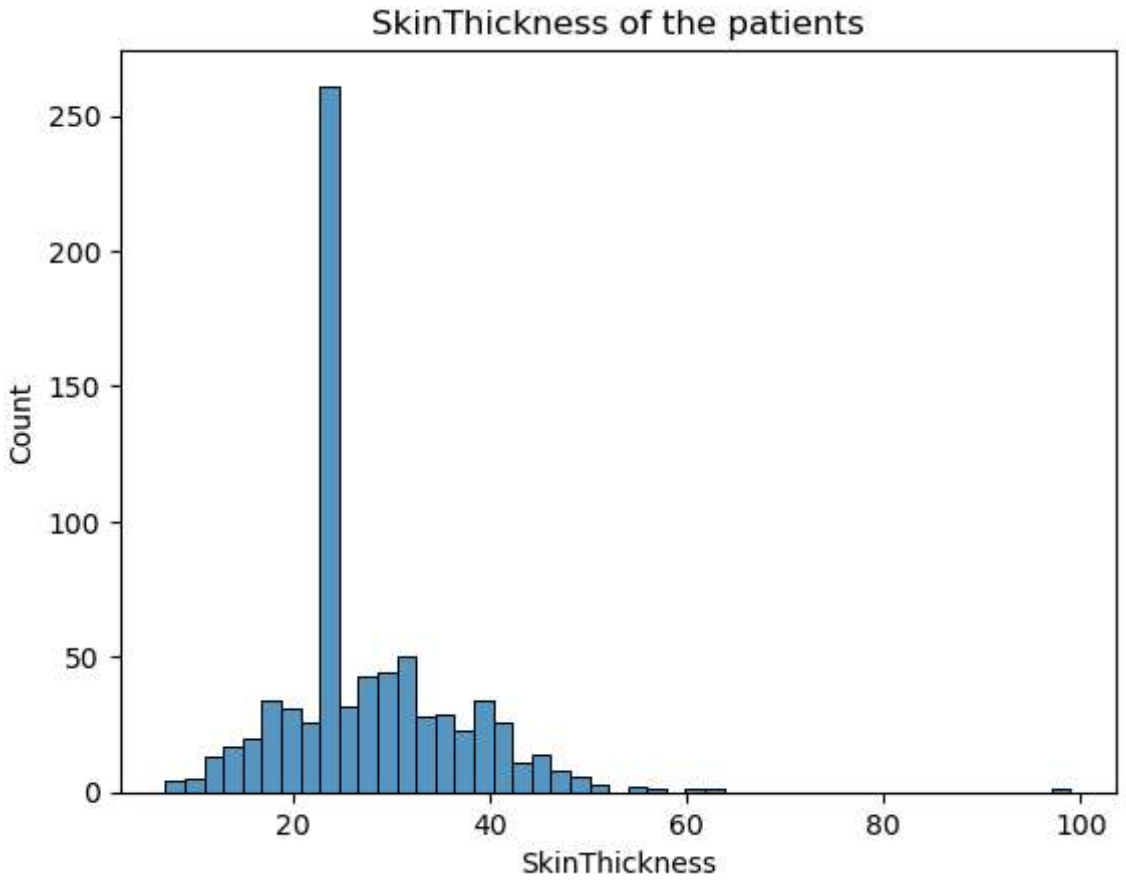
In [18]:
```
# Glucose after zero replaced
sns.histplot(data.Glucose)  #sns is seaborn library
plt.title('Glucose level of the patients')
plt.show()
```

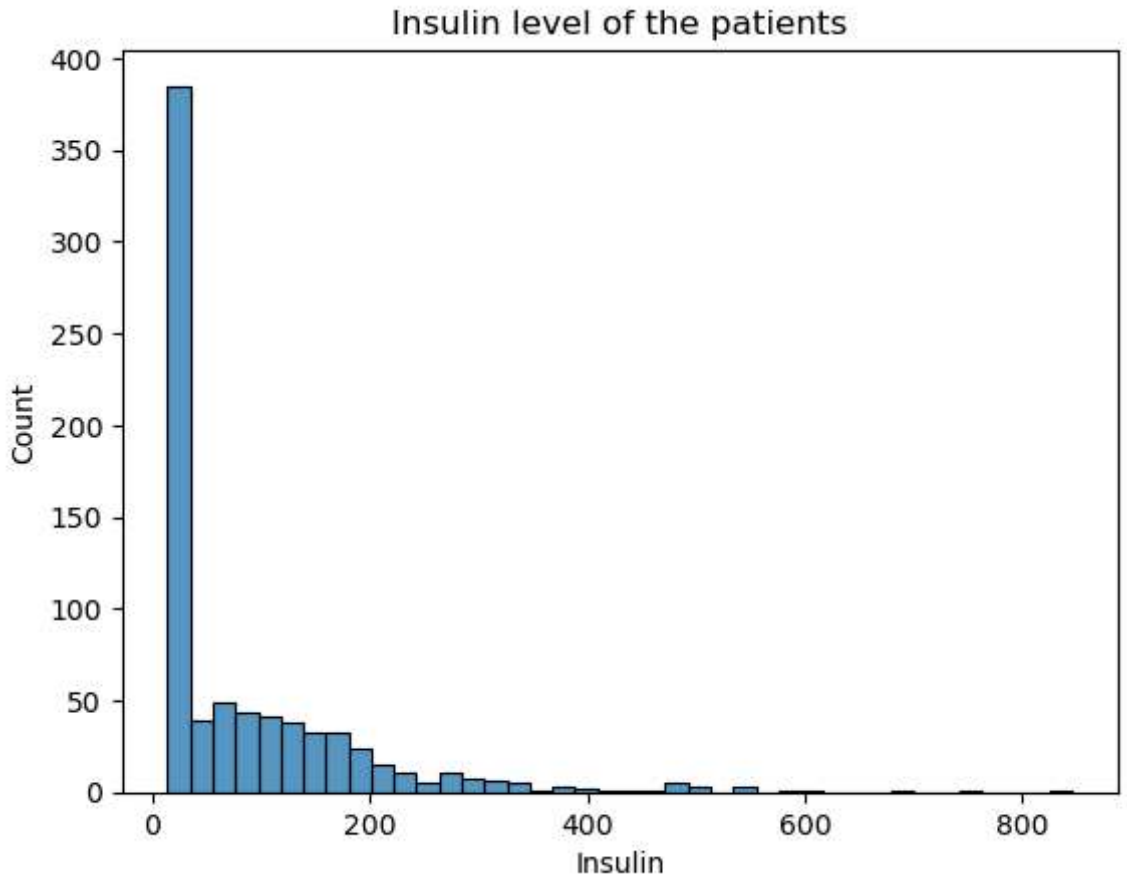


In [19]:
```
# BloodPressure after zero replaced
sns.histplot(data.BloodPressure)  #sns is seaborn library
plt.title('BloodPressure of the patients')
plt.show()
```

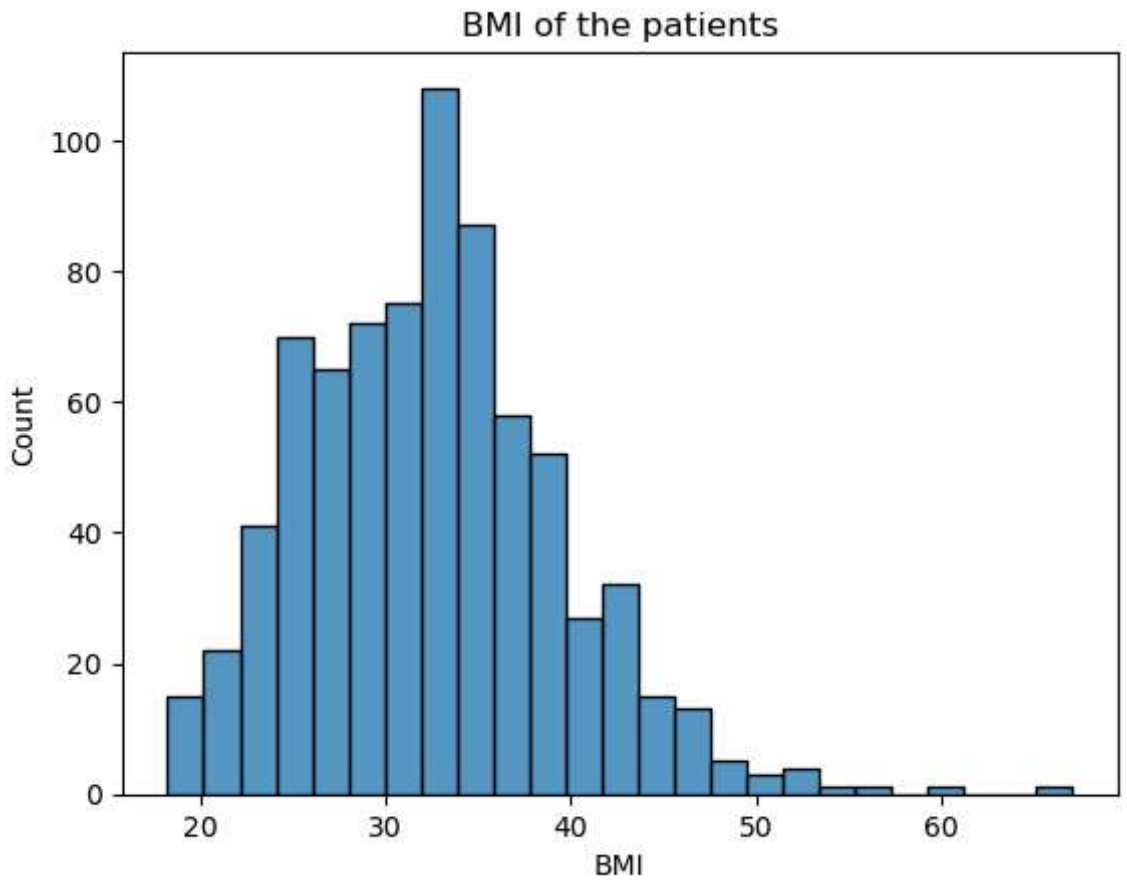


In [20]:
```
# SkinThickness after zero replaced
sns.histplot(data.SkinThickness)  #sns is seaborn library
plt.title('SkinThickness of the patients')
plt.show()

# Insulin after zero replaced
sns.histplot(data.Insulin)  #sns is seaborn library
plt.title('Insulin level of the patients')
plt.show()
```

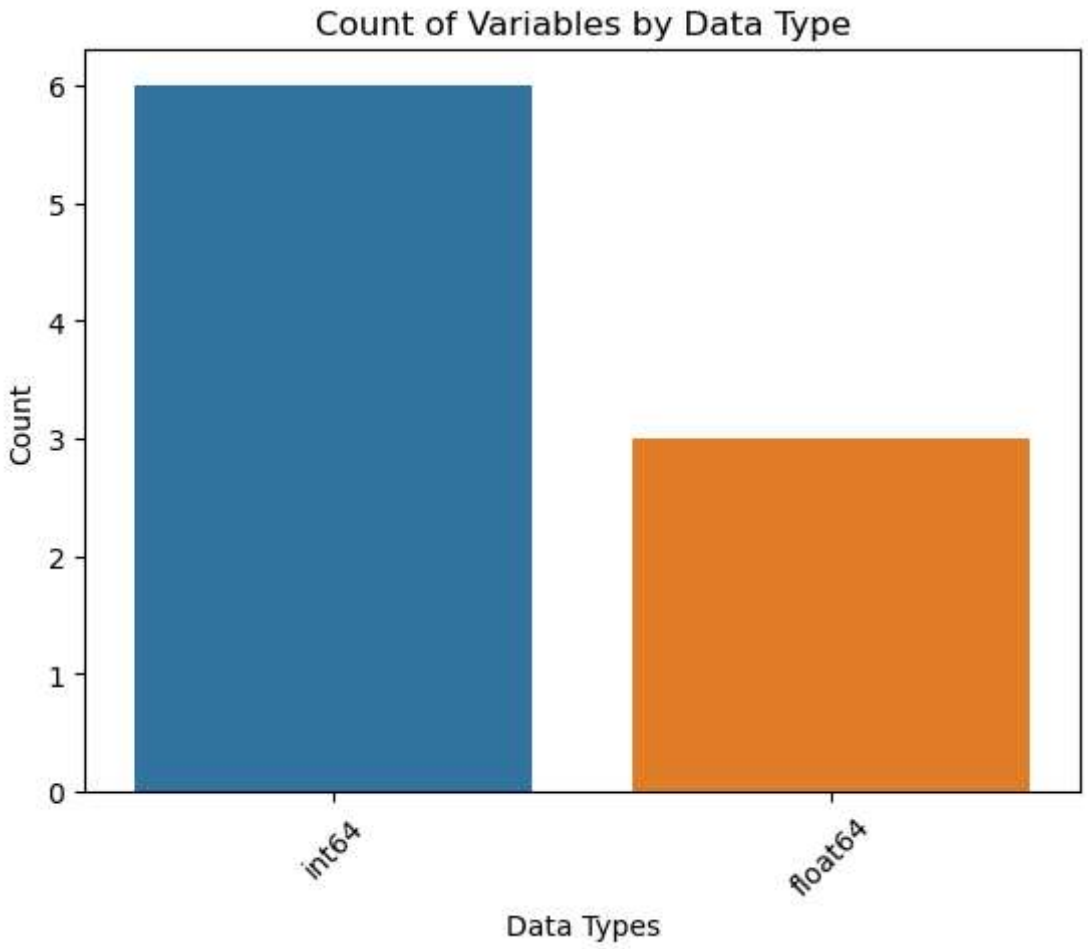




In [21]:
```
# BMI after zero replaced
sns.histplot(data.BMI)  #sns is seaborn library
plt.title('BMI of the patients')
plt.show()
```

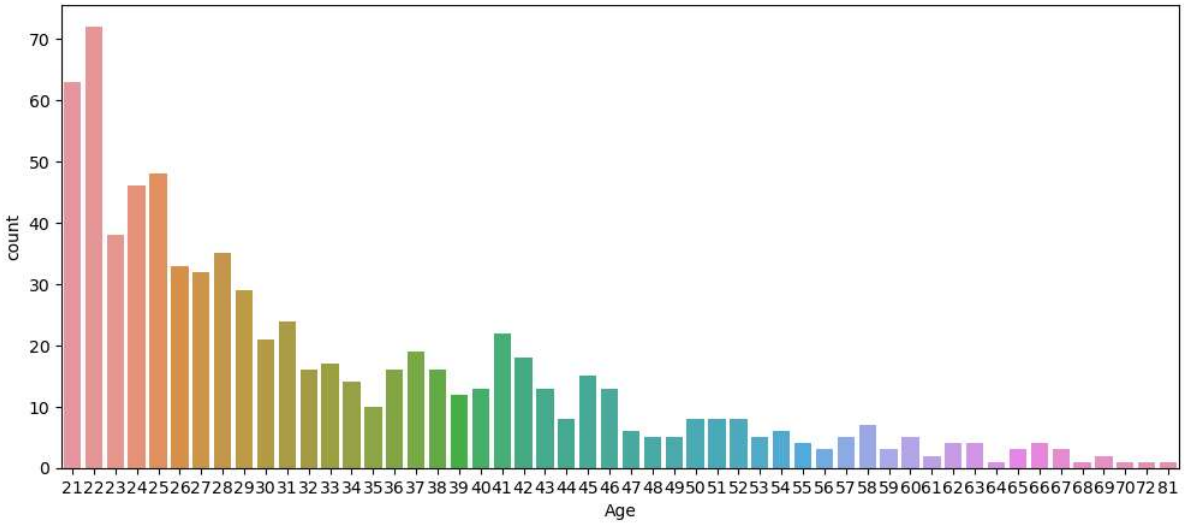


In [22]:
```
# Counting occurrences of each data type
data_type_counts = data.dtypes.value_counts()

# Plotting the count of variables by data type
sns.barplot(x=data_type_counts.index, y=data_type_counts.values)
plt.title('Count of Variables by Data Type')
plt.xlabel('Data Types')
plt.ylabel('Count')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```



4. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

In [23]:
```
#Age
plt.figure(figsize=(12,5))
#sns.countplot(x=data['Age'],hue='Outcome',data=data)
sns.countplot(x=data['Age'],data=data)
plt.show()
```



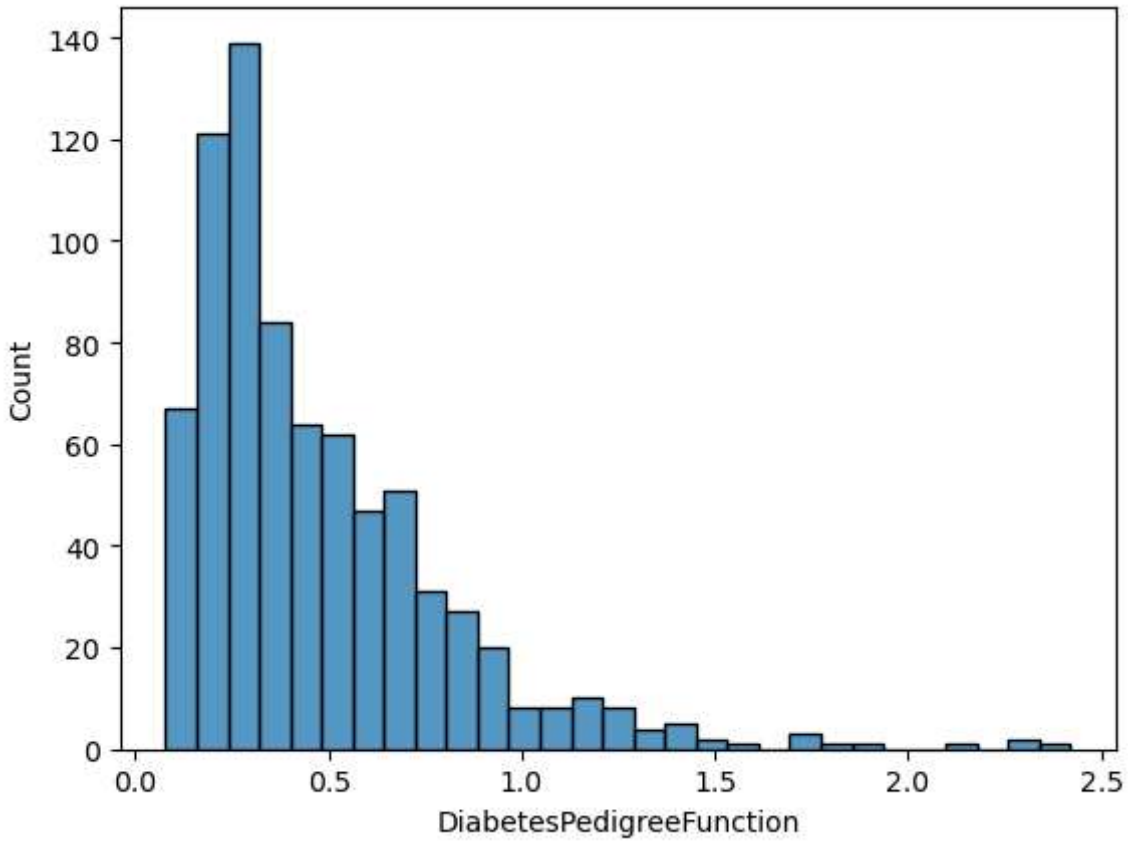
Observation: Most of the Age groups are there in youngsters.

In [24]:
```
#Outcome
data.Outcome.value_counts()
```

Out[24]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

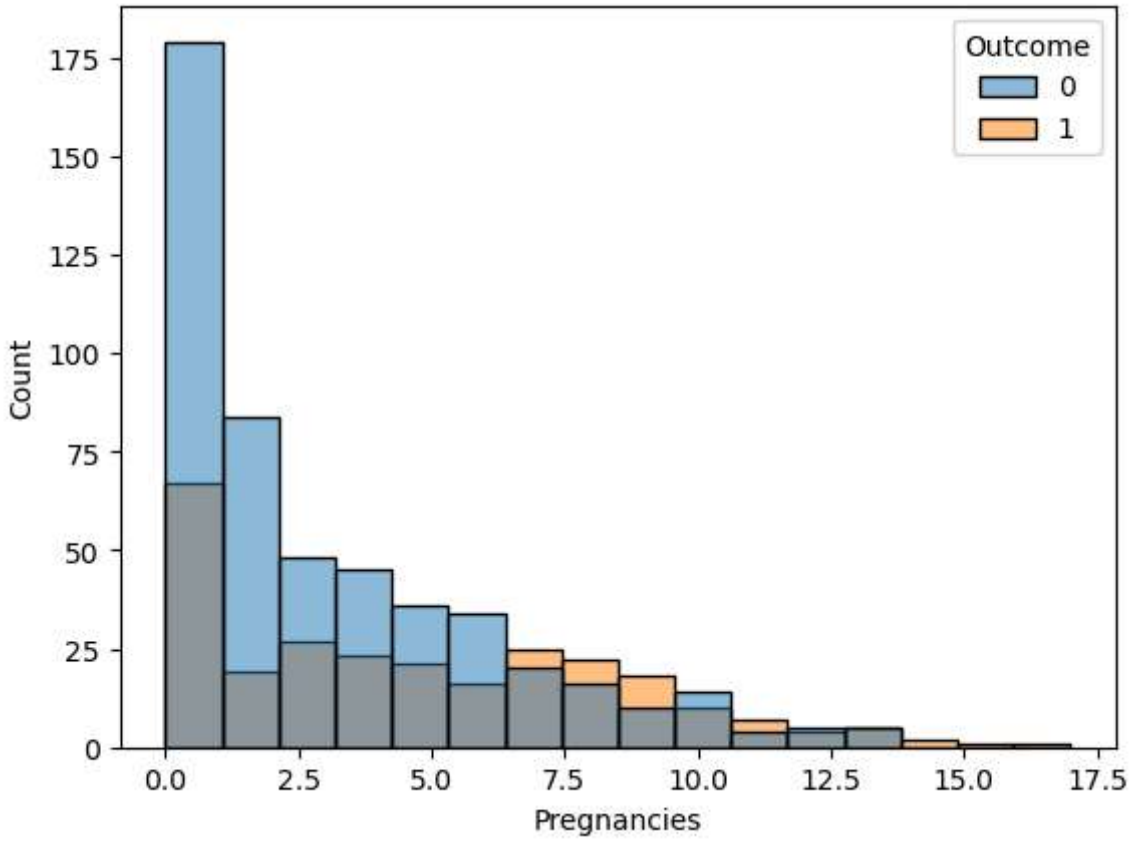
0= don't have Diabetic, 1= Diabetic

In [25]:
```
sns.histplot(x=data['DiabetesPedigreeFunction'])
plt.show()
```



DiabetsPedigreeFunction is nothing but coming of diabetics due to heridetary, and its most values ranges in between 0 to 0.5. Its meaning is that in this dataset the diabetic problem is not most due to heriditary.

In [26]:
```
#Pregnancies
#sns.histplot(x=data['Pregnancies'])
sns.histplot(x=data['Pregnancies'],hue='Outcome',data=data)
plt.show()
```



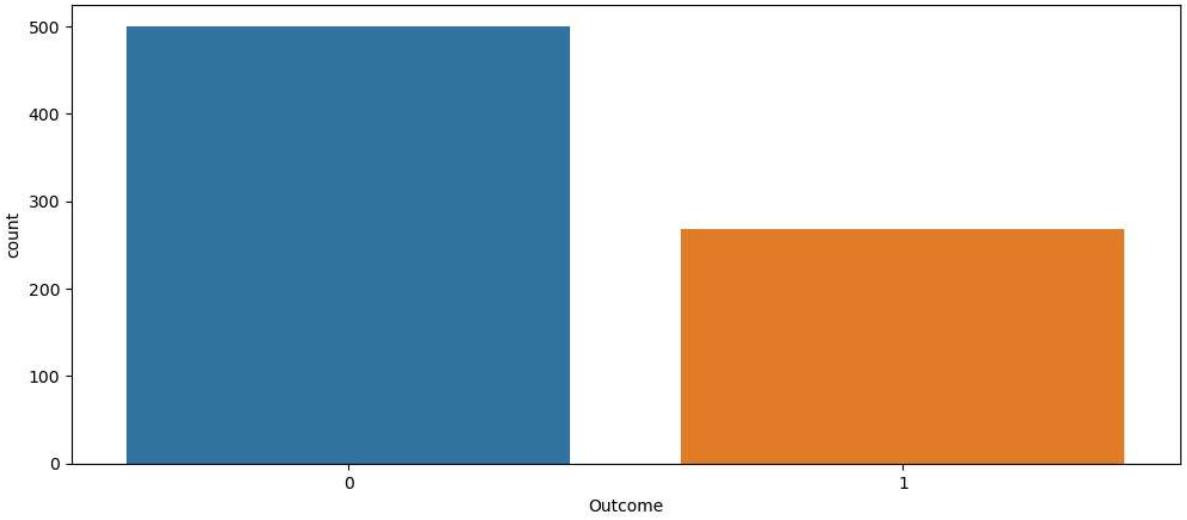
Observation: diabetics due to pregnancies is less in this dataset.

In [ ]:

In [ ]:

In [ ]:

In [27]:
```
#Outcome
plt.figure(figsize=(12,5))
sns.countplot(x=data['Outcome'],data=data)
plt.show()
```



From above we could identify that there is a data imbalancing.The value of 0 is much higher than the value of 1. We have to solve it, for that we are using SMOTE here.

SMOTE

Before that we have to split the data as X and y

In [28]:
```
# Extract the Independent & Dependent Variables
# X = data.iloc[:, :-1].values
# y = data.iloc[:, -1].values
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

In [29]:
```
from imblearn.over_sampling import SMOTE
```

In [44]:
```
smk = SMOTE()
X_res,y_res=smk.fit_resample(X,y)
```

In [45]:
```
from collections import Counter
print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))
```

```
Original dataset shape Counter({0: 500, 1: 268})
Resampled dataset shape Counter({1: 500, 0: 500})
```

In [46]:
```
print(X_res.shape)  # 'res' means resampled
print(y_res.shape)
```

```
(1000, 8)
(1000,)
```

In [ ]:

In [ ]:

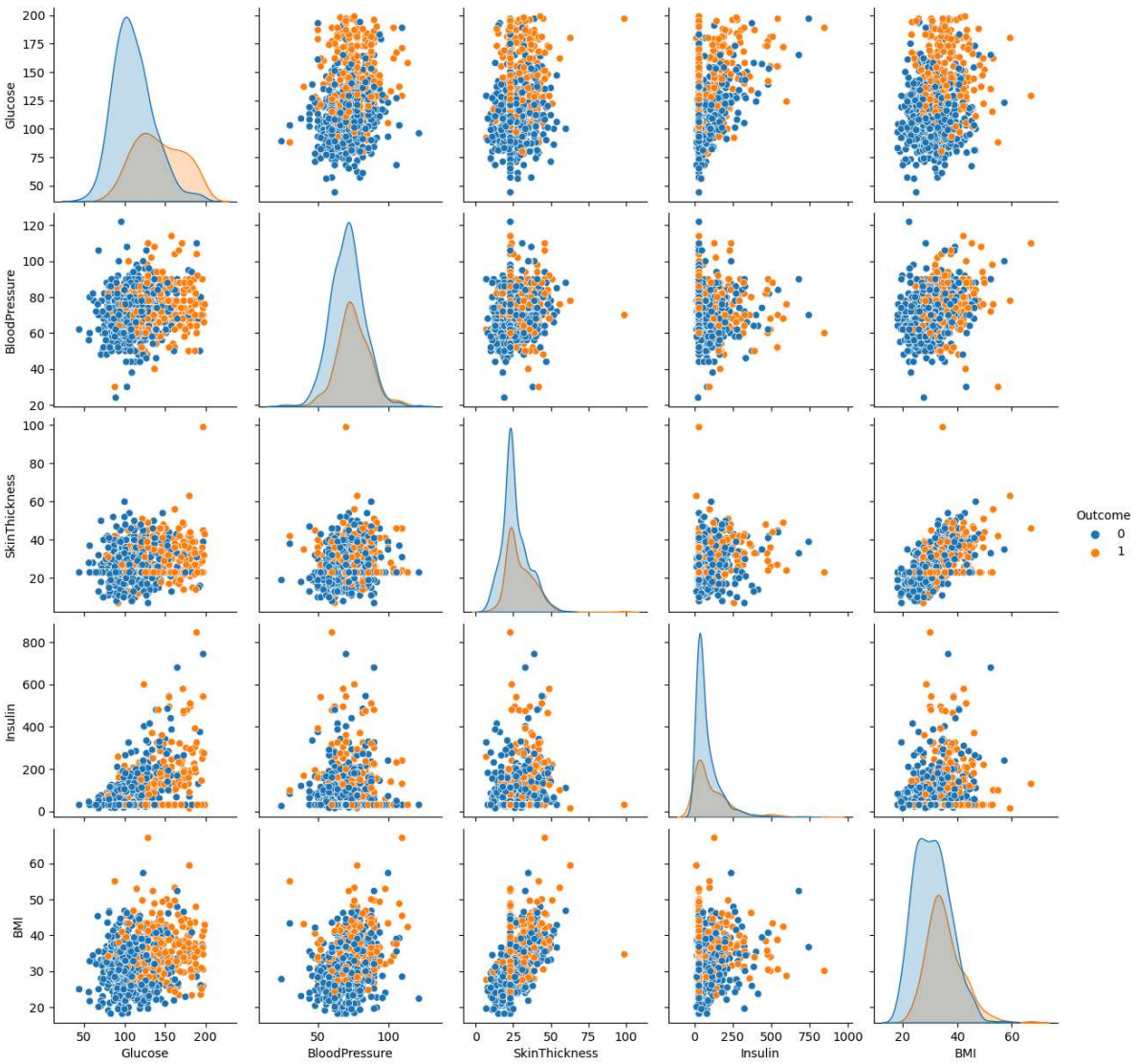
In [ ]:

In [ ]:

5. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

In [39]:
```
# sns.pairplot(data[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI']]

# plt.show()

# sns.pairplot(data[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
# plt.show()

# Selecting the desired columns along with the 'Outcome' column
selected_columns = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
selected_data = data[selected_columns]

# Plotting pairplot
```

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



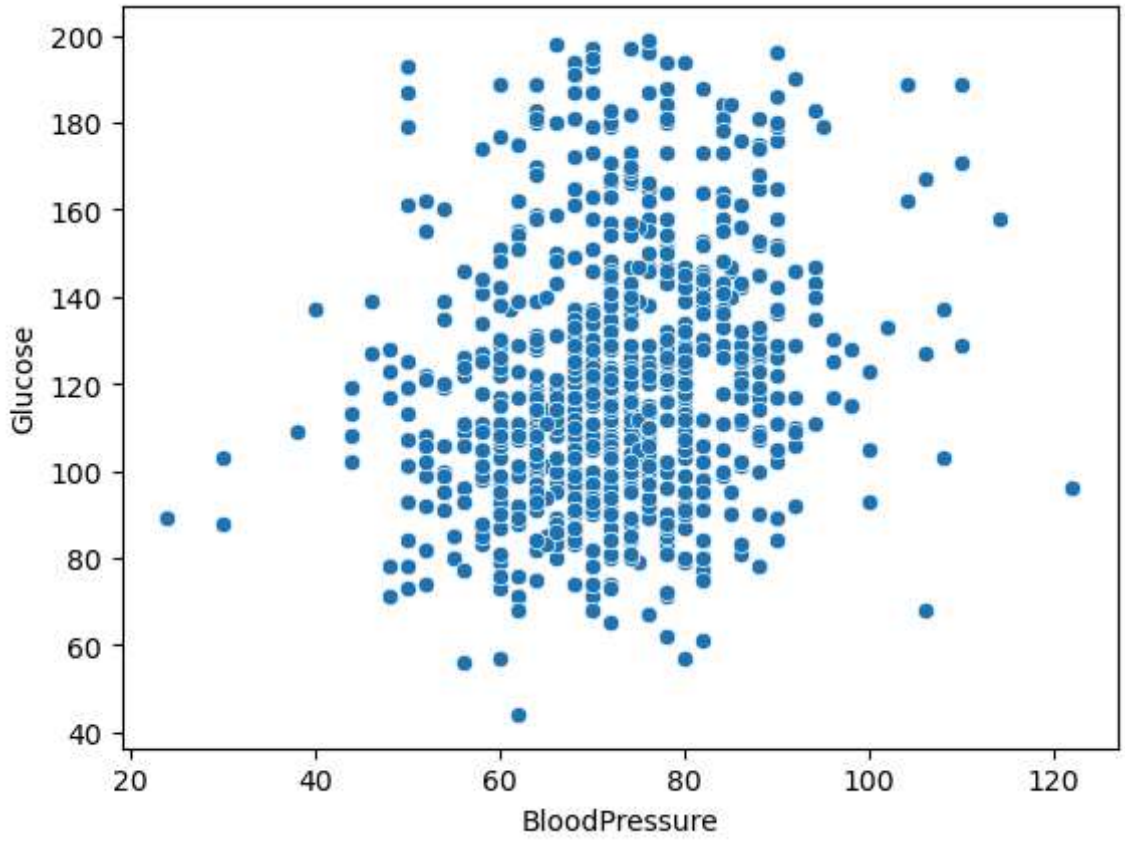
Observation:

1. If Blood Pressure and Glucose levels are high which leads to diabetics.

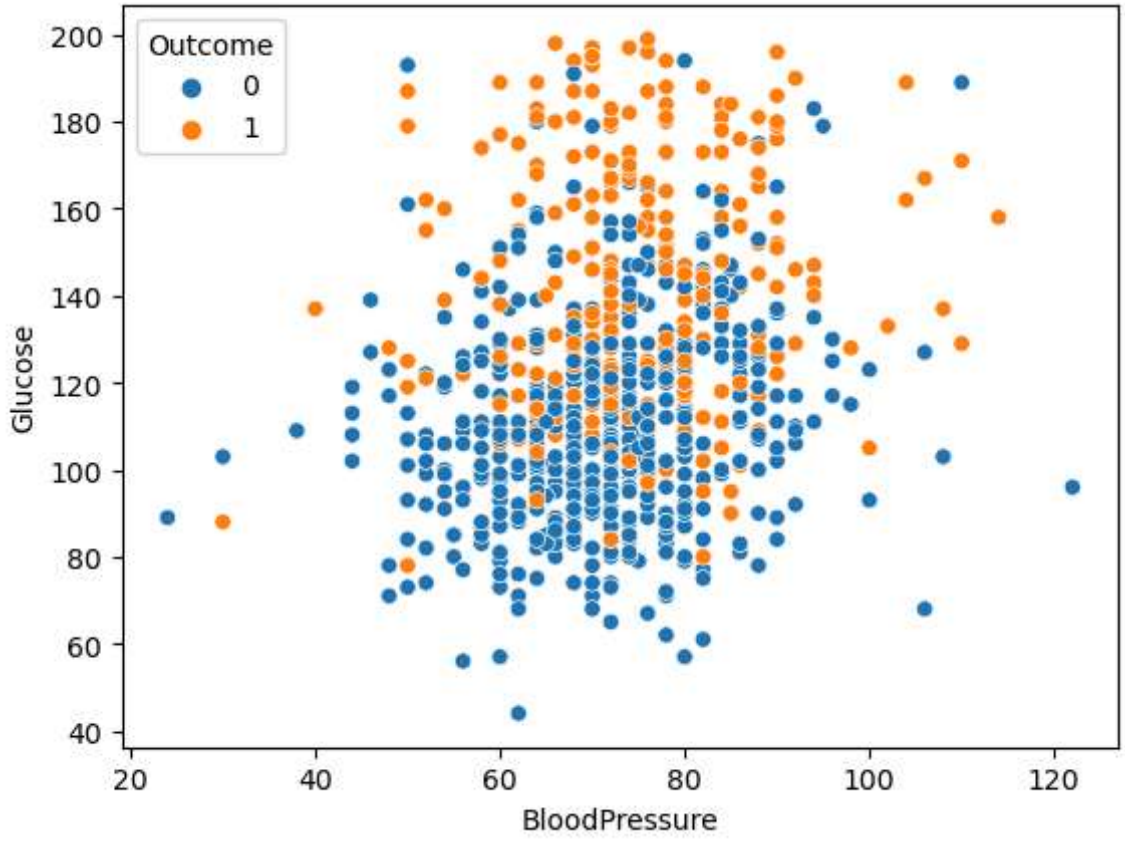
2. Evern with small values of skninthickness and high Glucose level may lead to diabetics.

3. We can conclude that high Glucose level lead to diabetic.

In [33]:
```
sns.scatterplot(x=data['BloodPressure'],y=data['Glucose'])
plt.show()
```

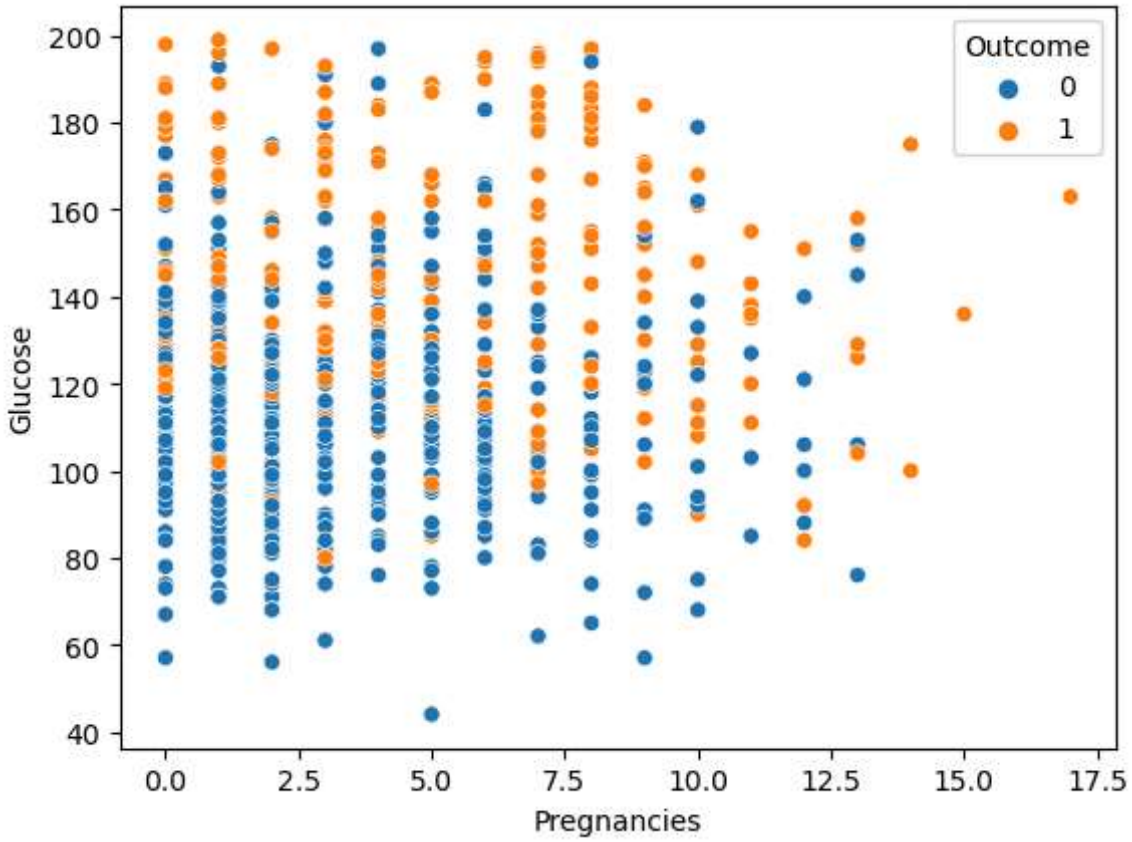


In [34]:
```
sns.scatterplot(x=data['BloodPressure'],y=data['Glucose'],hue='Outcome',data=data)
plt.show()
```



From the above graph it is understood that if Glucose level is more there is more chances of getting diabetics.

In [35]:
```
sns.scatterplot(x=data['Pregnancies'],y=data['Glucose'],hue='Outcome',data=data)
plt.show()
```

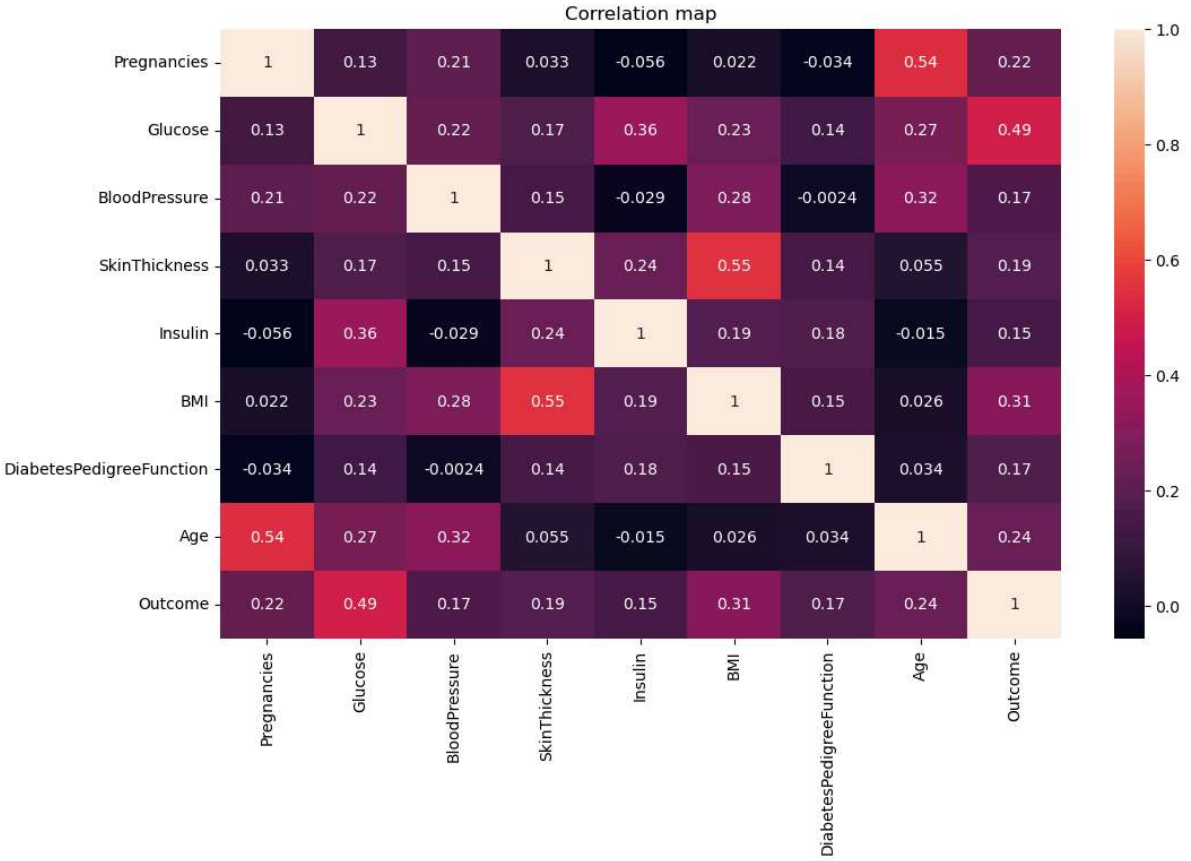


Observation: There is a chance of diabetic due to pregnancies.

6. Perform correlation analysis. Visually explore it using a heat map.

In [36]:
```
#Finding the correlation
plt.figure(figsize=(12,7));
sns.heatmap(data.corr(),annot=True);
# sns.heatmap()
plt.title("Correlation map")
```

Out[36]:
```
Text(0.5, 1.0, 'Correlation map')
```



From the Heat map above, we can say that 'Glucose' and 'BMI' are more correlated to Output variable Outcome.

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM

WWWWWWWWWWWWWWWWWWWWWWWWWWWWW

Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

In [48]:
```
# Create train set & test test
from sklearn.model_selection import train_test_split
X_res_train, X_res_test, y_res_train, y_res_test = train_test_split(X_res, y_res, t
# print(X_train.shape)
# print(X_test.shape)
# print(y_train.shape)
# print(y_test.shape)

print(X_res_train.shape)
print(X_res_test.shape)
print(y_res_train.shape)
print(y_res_test.shape)
```

```
(800, 8)
(200, 8)
(800,)
(200,)
```

2. Apply an appropriate classification algorithm to build a model.

In [49]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

In [50]:
```
logreg.fit(X_res_train,y_res_train)
```

```
C:\Users\RIJO K GEORGE\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.
py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
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(
```

Out[50]:
```
▾ LogisticRegression
LogisticRegression()
```

In [51]:
```
y_res_pred = logreg.predict(X_res_test)
y_res_pred
```

Out[51]:
```
array([0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0,
       0, 1], dtype=int64)
```

In [52]:
```
y_res_test
```

Out[52]:
```
521    0
737    0
740    1
660    0
411    0
      ..
408    1
332    1
208    0
613    0
78     1
Name: Outcome, Length: 200, dtype: int64
```

In [53]:
```
type(y_res_test)
```

Out[53]:
```
pandas.core.series.Series
```

Performance Evaluation

Here we evaluate the performance of the Logistic Regression Algorithm

In [54]:
```
from sklearn import metrics
```

We can find the accuracy directly by accuracy_score function as below.

In [55]:
```
metrics.accuracy_score(y_res_test, y_res_pred)
```

Out[55]:
```
0.73
```

Confusion Metrics:

In [56]:
```
metrics.confusion_matrix(y_res_test, y_res_pred)
```

Out[56]:
```
array([[70, 29],
       [25, 76]], dtype=int64)
```

In [57]:
```
print(metrics.classification_report(y_res_test, y_res_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.71      0.72        99
           1       0.72      0.75      0.74       101

    accuracy                           0.73       200
   macro avg       0.73      0.73      0.73       200
weighted avg       0.73      0.73      0.73       200
```

So, here we get an accuracy of 75% using Logistic Regression.

Here support represents the total record (ie, 99+101=200)

0=are not diabetic, 1=are diabetic

3. Compare various models with the results from KNN algorithm.

In [58]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [59]:
```
knn = KNeighborsClassifier(n_neighbors=3)
```

In [60]:
```
knn.fit(X_res_train,y_res_train)
```

Out[60]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [61]:
```
pred = knn.predict(X_res_test)
```

In [62]:
```
pred
```

Out[62]:
```
array([0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1], dtype=int64)
```

Predictions and Evaluations

Let's evaluate our KNN model!

In [63]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

In [64]:
```
print(accuracy_score(y_res_test,pred))
```

```
0.775
```

In [65]:
```
print(classification_report(y_res_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.71      0.76        99
           1       0.75      0.84      0.79       101

    accuracy                           0.78       200
   macro avg       0.78      0.77      0.77       200
weighted avg       0.78      0.78      0.77       200
```

So, we get an accuracy of 78%

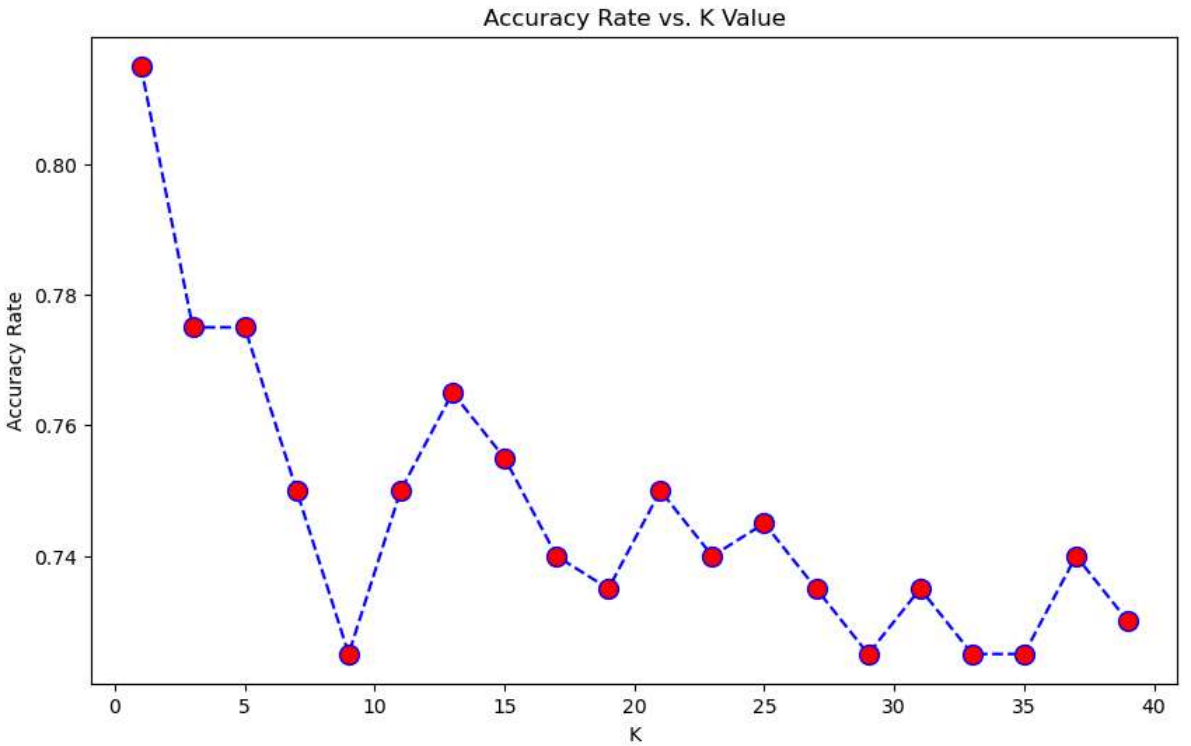
Choosing a K Value

In [60]:
```
accuracy_rate = []

for i in range(1,40,2):  # (starting value,ending value, increment as 2), it will i
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_res_train,y_res_train)
    pred = knn.predict(X_res_test)
    score = accuracy_score(y_res_test,pred)
    print('Value of i',i,'accuracy is',score)
    accuracy_rate.append(score)
```

```
Value of i 1 accuracy is 0.815
Value of i 3 accuracy is 0.775
Value of i 5 accuracy is 0.775
Value of i 7 accuracy is 0.75
Value of i 9 accuracy is 0.725
Value of i 11 accuracy is 0.75
Value of i 13 accuracy is 0.765
Value of i 15 accuracy is 0.755
Value of i 17 accuracy is 0.74
Value of i 19 accuracy is 0.735
Value of i 21 accuracy is 0.75
Value of i 23 accuracy is 0.74
Value of i 25 accuracy is 0.745
Value of i 27 accuracy is 0.735
Value of i 29 accuracy is 0.725
Value of i 31 accuracy is 0.735
Value of i 33 accuracy is 0.725
Value of i 35 accuracy is 0.725
Value of i 37 accuracy is 0.74
Value of i 39 accuracy is 0.73
```

From above we can see that the better value of K is 1 (accuracy is 0.815)

In [61]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40,2),accuracy_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Accuracy Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy Rate')
plt.show()
```



In [62]:
```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_res_train,y_res_train)
pred = knn.predict(X_res_test)

print('WITH K=3')
print('\n')
print(accuracy_score(y_res_test,pred))
```

```
WITH K=3


0.775
```

with K=3 accuracy is 77.5%

In [63]:
```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_res_train,y_res_train)
pred = knn.predict(X_res_test)

print('WITH K=1')
print('\n')
print(accuracy_score(y_res_test,pred))
```

```
WITH K=1


0.815
```

with K=1 accuracy is 81.5%

So, we can say that accuracy decreased with KNN Algorithm.

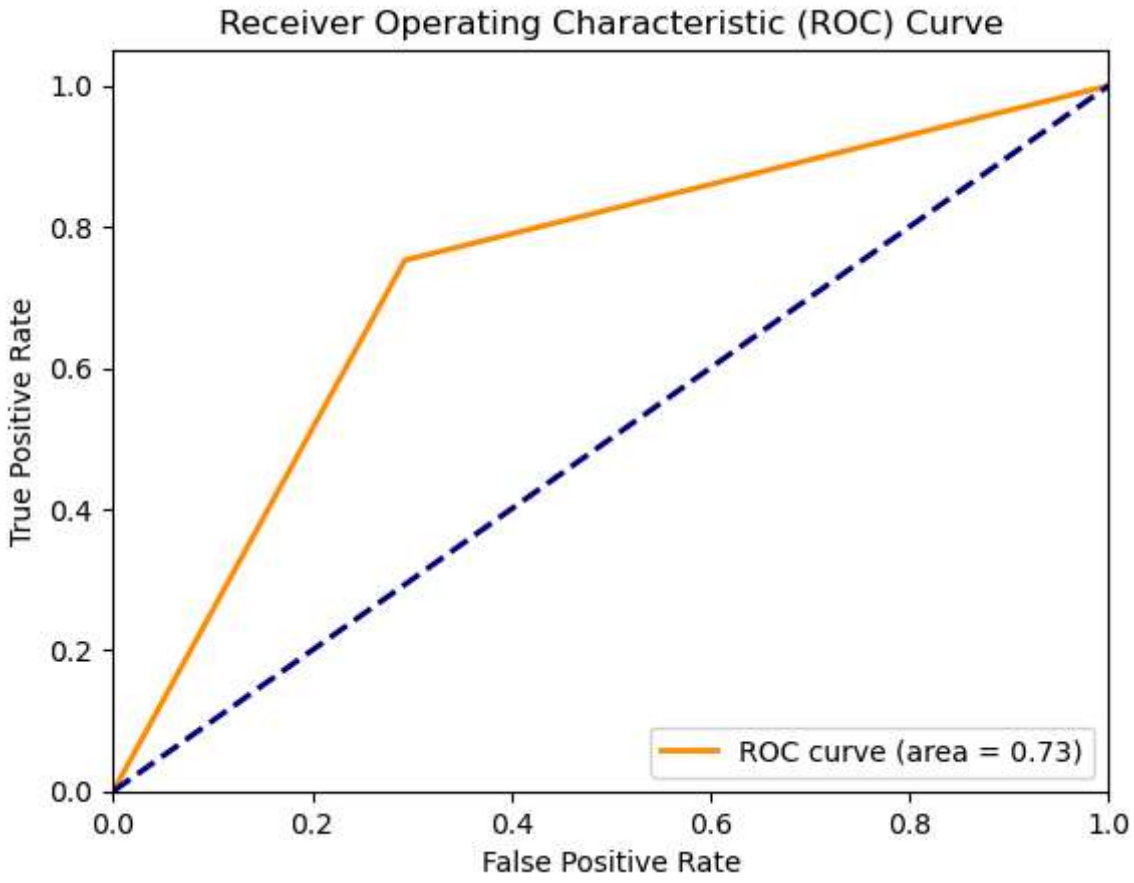
4. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc.

the AUC (ROC) curve provides a comprehensive evaluation of a binary classification model's performance, capturing its ability to distinguish between the positive and negative classes across various decision thresholds. Higher AUC values indicate better model performance.

In [68]:
```
# Importing necessary libraries
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc

# Define a function to calculate sensitivity and specificity
def sensitivity_specificity(y_res_true, y_res_pred):
    cm = confusion_matrix(y_res_true, y_res_pred)  #cm=confusion matri

    sensitivity = cm[1, 1] / (cm[1, 1] + cm[1, 0])  # True Positive Rate (Sensitivi
    specificity = cm[0, 0] / (cm[0, 0] + cm[0, 1])  # True Negative Rate (Specifici
    # cm[1, 1] is TP.
    # cm[0, 0] is TN.
    # cm[1, 0] is FN.
    # cm[0, 1] is FP.
    return sensitivity, specificity

# Classification report for logistic regression
print("Logistic Regression Classification Report:")
print(classification_report(y_res_test, y_res_pred))

# Sensitivity and Specificity for logistic regression
logreg_sensitivity, logreg_specificity = sensitivity_specificity(y_res_test, y_res_
print("Logistic Regression Sensitivity:", logreg_sensitivity)
print("Logistic Regression Specificity:", logreg_specificity)

# ROC Curve and AUC for logistic regression
fpr, tpr, thresholds = roc_curve(y_res_test, y_res_pred)
roc_auc = auc(fpr, tpr)
```

```
# Plot ROC curve for logistic regression
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()


auc_score=roc_auc_score(y_res_test,prob_pos)
print('AUC Score',auc_score)
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.71      0.72        99
           1       0.72      0.75      0.74       101

    accuracy                           0.73       200
   macro avg       0.73      0.73      0.73       200
weighted avg       0.73      0.73      0.73       200

Logistic Regression Sensitivity: 0.7524752475247525
Logistic Regression Specificity: 0.7070707070707071
```




```
AUC Score 0.8346834683468346
```

the AUC value approaches 1, it means the model's ability to distinguish between positive and negative classes is excellent, and it has a strong discriminatory power.

Other method to find AUC & ROC score

In [69]:
```
# # Prepare AUC & ROC Score
# from sklearn.metrics import roc_auc_score,roc_curve
# prob=logreg.predict_proba(X_res_test)
# prob_pos=prob[:,-1]
```

In [70]:
```
# auc_score=roc_auc_score(y_res_test,prob_pos)
# print('AUC Score',auc_score)
```

In [71]:
```
# fpr,tpr,thrs=roc_curve(y_res_test,prob_pos)
# plt.plot([0,1],[0,1],linestyle='--')

# plt.plot(fpr,tpr,marker='.')
# plt.show
```

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM

In [ ]: