CAPSTONE PROJECT

November 14, 2023

0.0.1 PROBLEM STATEMENT

Healthcare Course-end Project 2 Description NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases. • The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. • Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Dataset Description The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Variables Description

Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration in an oral glucose tolerance test

BloodPressure: Diastolic blood pressure (mm Hg)

SkinThickness: Triceps skinfold thickness (mm)

Insulin: Two hour serum insulin

BMI: Body Mass Index

DiabetesPedigreeFunction: Diabetes pedigree function

Age: Age in years

Outcome: Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

Project Task: Week 1 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

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

2. 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.

Data Exploration: 1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action. 1. Create scatter charts between the pair of variables to understand the relationships. Describe your findings. 1. Perform correlation analysis. Visually explore it using a heat map.

Project Task: Week 2 Data Modeling: 1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process. 2. Apply an appropriate classification algorithm to build a model. 3. Compare various models with the results from KNN algorithm. 4. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

Data Reporting: 1. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following: • Pie chart to describe the diabetic or non-diabetic population • Scatter charts between relevant variables to analyze the relationships • Histogram or frequency charts to analyze the distribution of the data • Heatmap of correlation analysis among the relevant variables • Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.

1 Importing Important Libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

2 Reading the Dataset

[2]:
```
df= pd.read_csv("health care diabetes.csv")
```

3 Previewing Dataset

[3]:
```
df.head()
```

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
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	

```
   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[4]:
```
#Previewing dataset
df.tail()
```

[4]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

4 Checking the shape of the Dataset

[5]:
```
#Checking the shape of the dataset
df.shape
```

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

5 Populating the features

[6]:
```
df.columns
```

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

6 Undertanding The Dataset

[7]:
```
#Looking up datset information
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
```

The dataset has 7 integers and 2 float values.

7 Checking for Null or Missing Values

[8]:
```
#Checking for missing values
Missing_values=df.isnull().sum()
Missing_values
```

[8]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

There are no null values in our dataset.

7.0.1 Project Task: Week 1 Data Exploration:

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

[9]:
```
#Understanding the dispersion of the data
df.describe()
```

[9]:

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

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

7.0.2 Central Tedencies Inferences for the Dataset

From the above table

This summary provides valuable information about the dataset's core trends, variability, and how the data is distributed, all of which are crucial when exploring datasets and understanding their attributes.

The dataset consists of 768 rows. Unusually high mean values for Glucose, Blood Pressure, Insulin, BMI, and Age suggest possible outliers, as indicated by their respective standard deviations. The minimum values of 0 in Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, and BMI indicate skewness in these features.

Skin Thickness and Insulin have 25% quartile values of 0, marking the lower boundary of the first quartile. The 50% percentile, or median, shows an even split in data. The 75% percentile, when subtracted from the 25% percentile, defines the Interquartile Range (IQR), hinting at potential outliers.

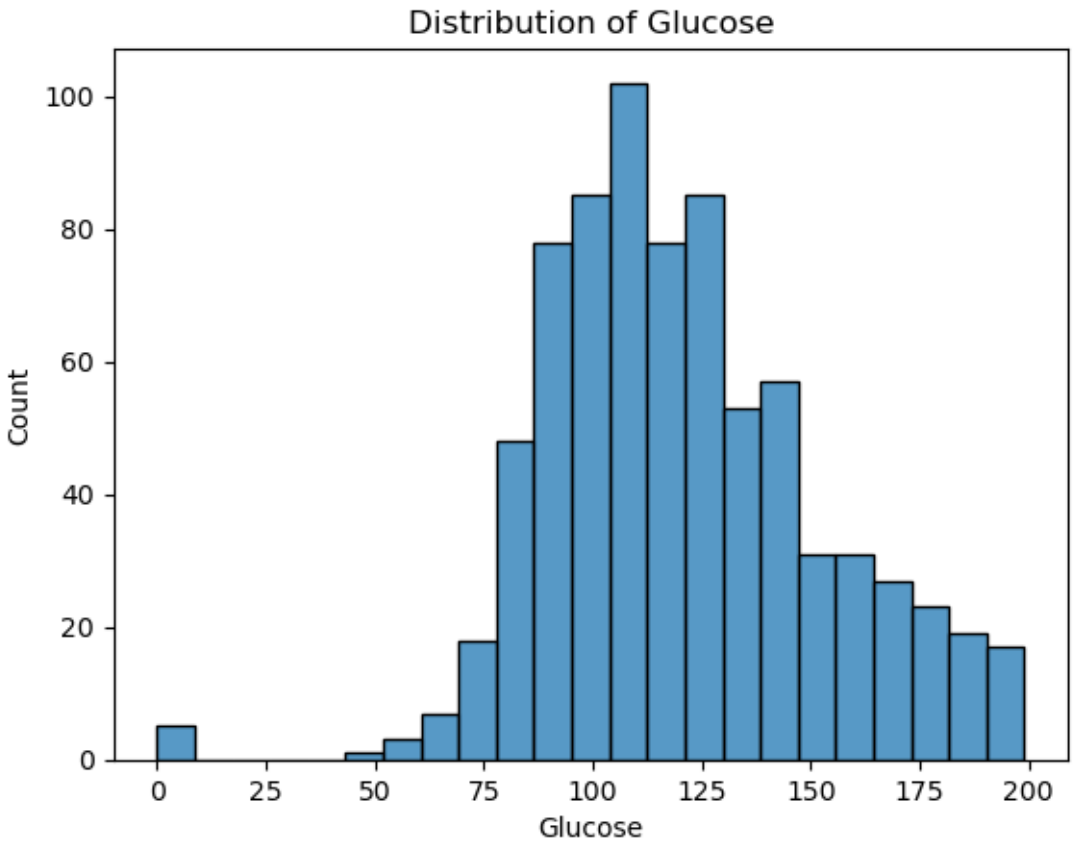
The maximum number of pregnancies is 17, prompting further investigation into whether this impacts diabetes development alongside other predictor variables.

CONDUCTING UNIVARIATE ANALYSIS

We shall be Visually exploring these variables using histograms, and we shall be treating the missing values accordingly.

FEATURE ANALYSIS

[10]:
```
#create histogram for Glucose

sns.histplot(x=df['Glucose'])
plt.title('Distribution of Glucose')
plt.show()
```



We can visually see the presence of 0 in the dataset which indicates the presence of missing values.This has partial normal distribution.

[11]:
```
#Looking up the number of zero's in Glucose variable
count_zeros = (df['Glucose'] == 0).sum()
count_zeros
```

[11]:
```
5
```

There are a total of 5 zeros in the Glucose variable.

[12]:
```
#Checking Frequency count of Glucose in Plasma among patients.
Glucose_counts=df['Glucose'].value_counts()
Glucose_counts
```

[12]:
```
99     17
100    17
111    14
129    14
125    14
       ..
191     1
177     1
44      1
62      1
190     1
Name: Glucose, Length: 136, dtype: int64
```

17 patients have 99 and 100 glucose in plasma.

[13]:
```
# Checking for Mean of Glucose
Glucose_Mean=df['Glucose'].mean()
Glucose_Mean
```

[13]:
```
120.89453125
```

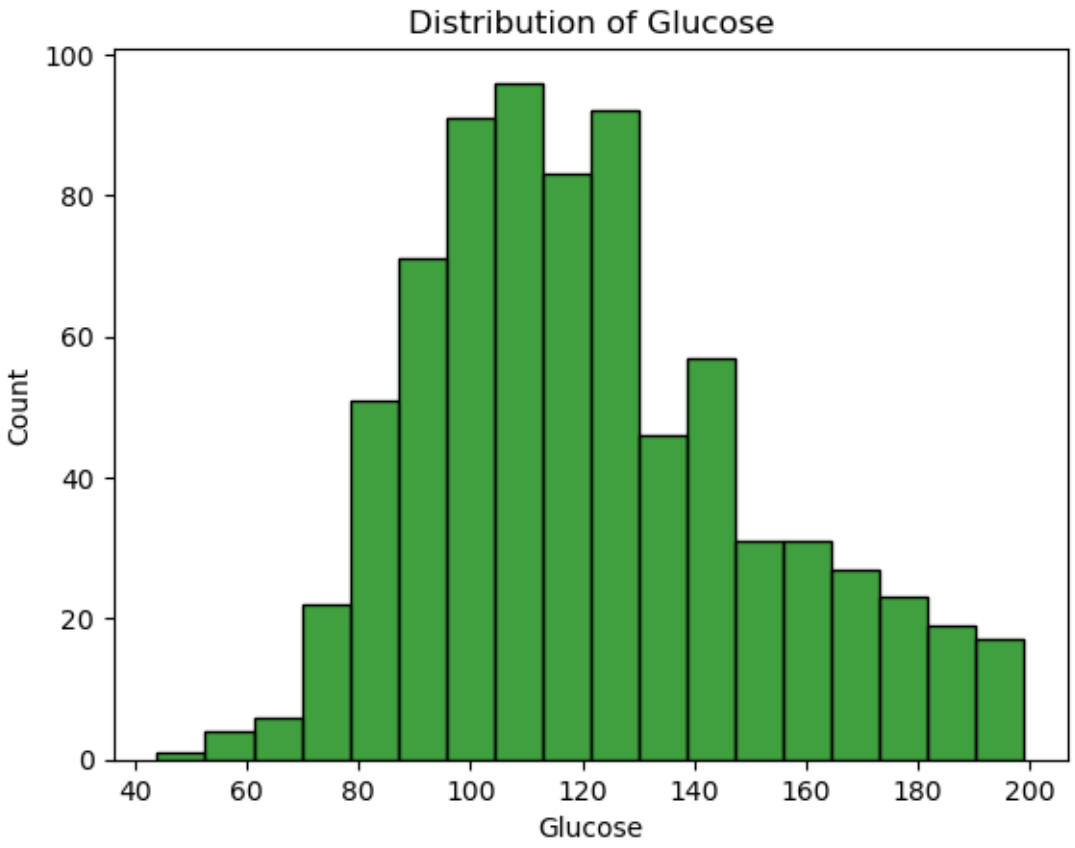
We shall be replacing the missing values which are zeros using mean, due to the partial normal disribution in the glucose variable as observed above

[14]:
```
#Replacing the zeros with the mean of Glucose
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
```

[15]:
```
#Rechecking for zeros in the Glucose variable
count_zeros = (df['Glucose'] == 0).sum()
count_zeros
```

[15]:
```
0
```

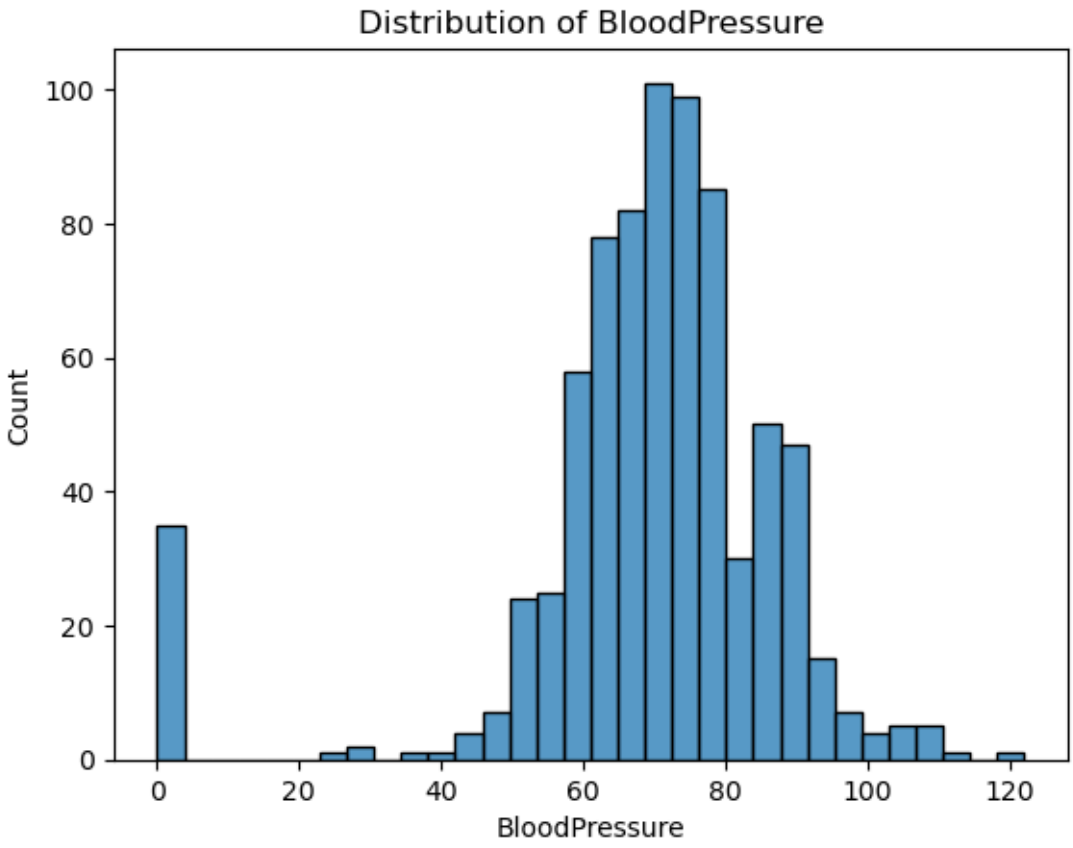
[16]:
```
#create  histogram

sns.histplot(x='Glucose',color='green',data=df)

plt.title('Distribution of Glucose')

plt.show()
```



We can infer that the zeros have been replaced with the mean of Glucose variable.

BLOOD PRESSURE FEATURE ANALYSIS

[17]:
```
#create histogram for BloodPressure

sns.histplot(x=df['BloodPressure'])
plt.title('Distribution of BloodPressure')
plt.show()
```



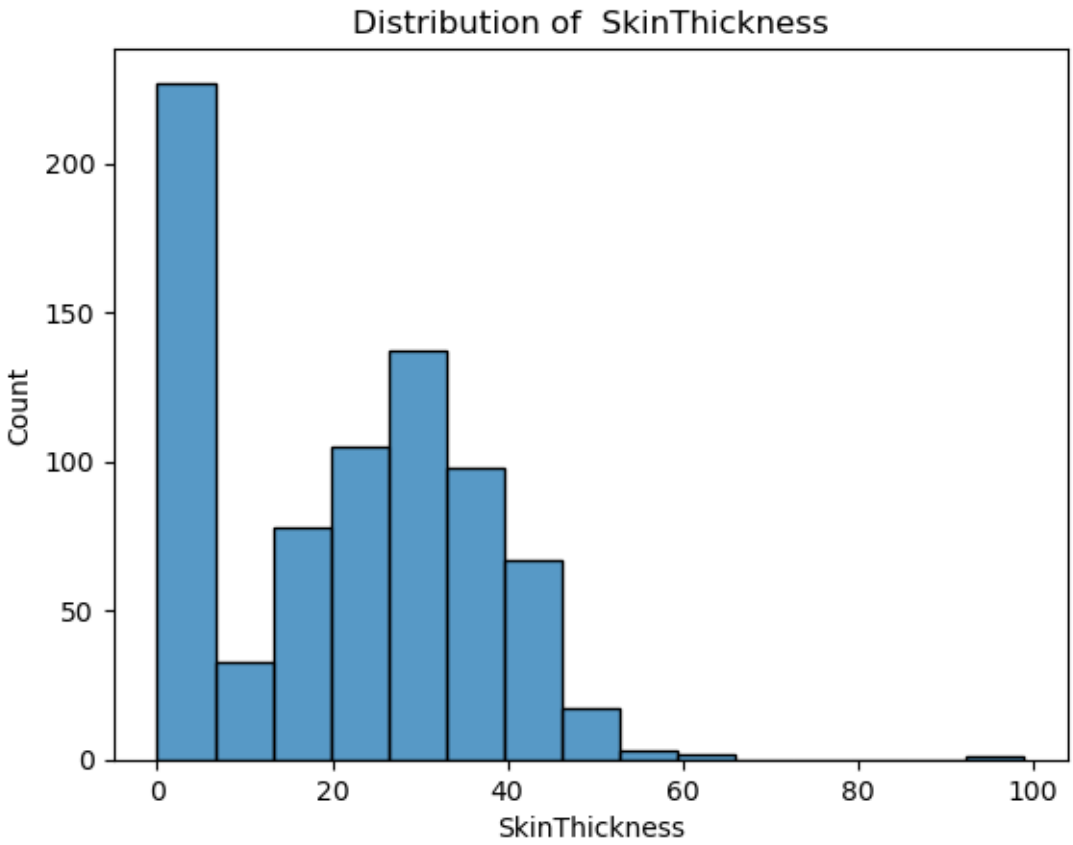
We can visually see the presence of 0 in the BloodPressure variable.

[18]:
```
#Looking up the number of zero's in BloodPressure variable
BP_zero_count = (df['BloodPressure'] == 0).sum()
BP_zero_count
```

[18]: 35

A total of 35 zeros are present in the BloodPressure column.

[19]:
```
#create histogram for SkinThickness

sns.histplot(x=df['SkinThickness'])
plt.title('Distribution of  SkinThickness')
plt.show()
```



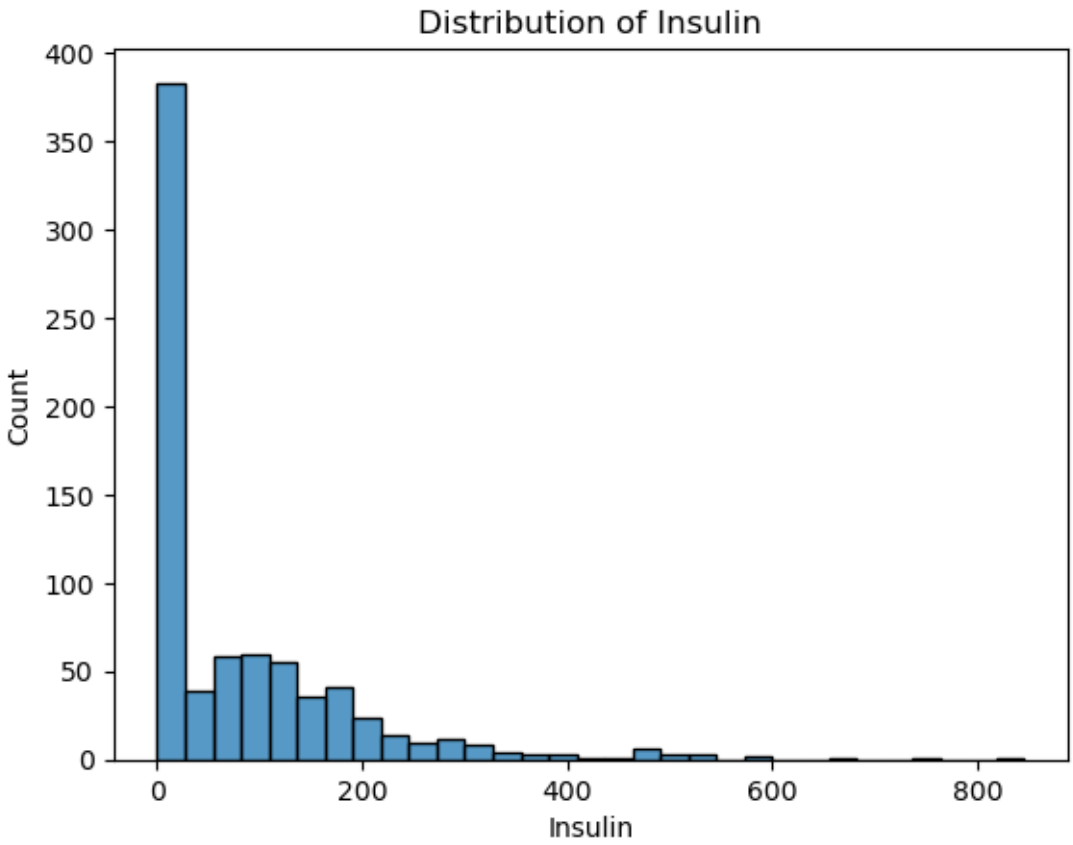
Visually zeros are present in SkinThickness columnn. the data does not follow a normal distribution.

[20]:
```
#Looking up the number of zero's in SkinThickness variable
ST_zero_count = (df['SkinThickness'] == 0).sum()
ST_zero_count
```

[20]: 227

227, zeros are there in the SkinThickness column.

[21]:
```
#create histogram for Insulin

sns.histplot(x=df['Insulin'])
plt.title('Distribution of Insulin')
plt.show()
```



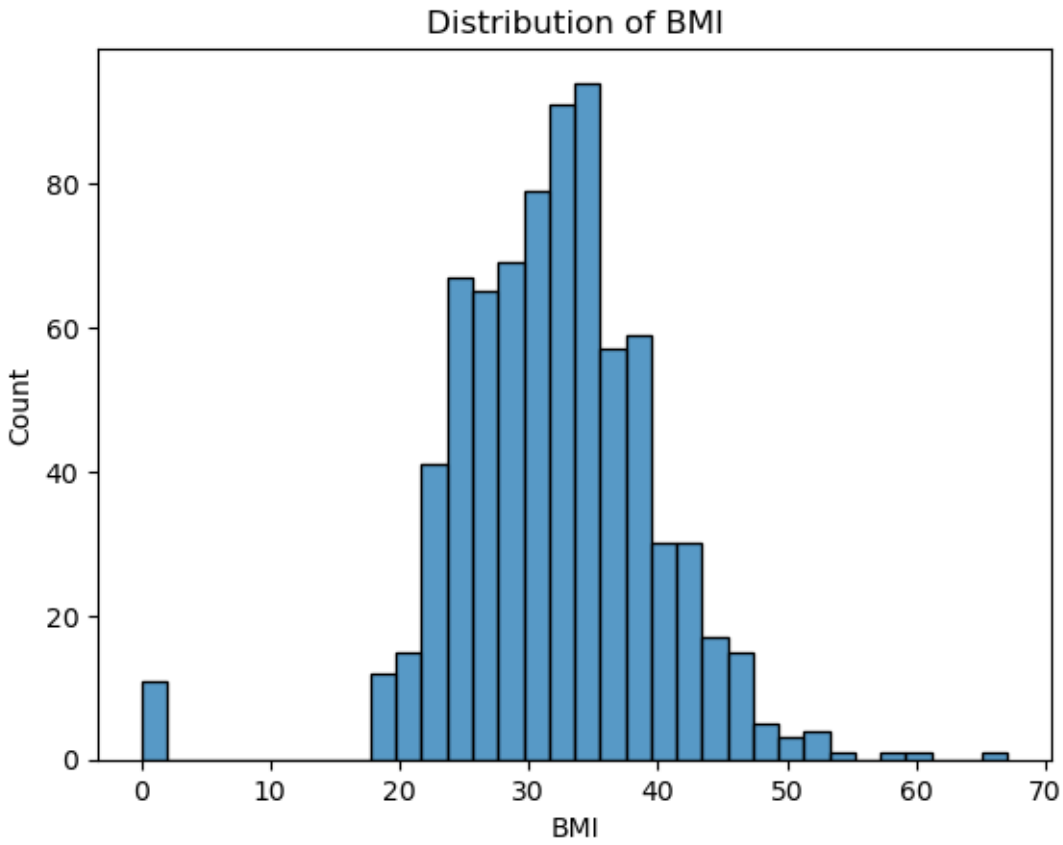
Visually there is zero in Insulin variable,the data does not follow a normal distribution.

[22]:
```
#Looking up the number of zero's in Insulin variable
IN_zero_count = (df['Insulin'] == 0).sum()
IN_zero_count
```

[22]: 374

Atotal of 374 are there in insulin variable

[23]:
```
#create histogram for BMI

sns.histplot(x=df['BMI'])
plt.title('Distribution of BMI')
plt.show()
```



Zeros are prrsent,the dataset look normally distributed.

[24]:
```
#Looking up the number of zero's in BMI variable
BMI_zero_count = (df['BMI'] == 0).sum()
BMI_zero_count
```

[24]:
```
11
```

There are 11 zeros in BMI variable

[25]:
```
#Creating a variable to replace the zeros in all our features.
variables=['BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
```

[26]:
```
#collectively replacing all zero values with median using for loop

for i in variables:
    df[i].replace(0, df[i].median(), inplace=True)
```

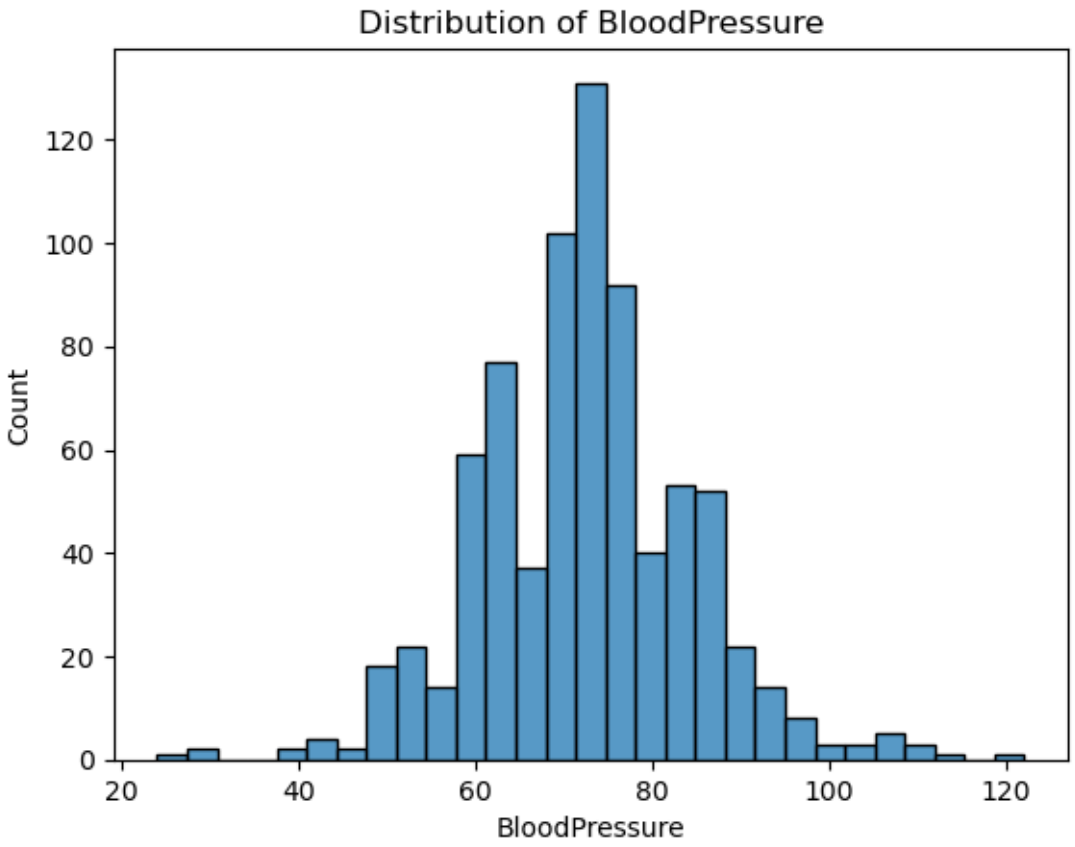
We replaced all the missing values represented as zeros in our dataset with the median. The median is often a robust choice for imputing missing values in non-normally distributed data. It's not affected by extreme values (outliers) and represents the middle value of the data.

[27]:
```
##Rechecking the presence of zero after analysis
(df[['BloodPressure', 'SkinThickness', 'Insulin', 'BMI']]==0).sum()
```

[27]:
```
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
dtype: int64
```

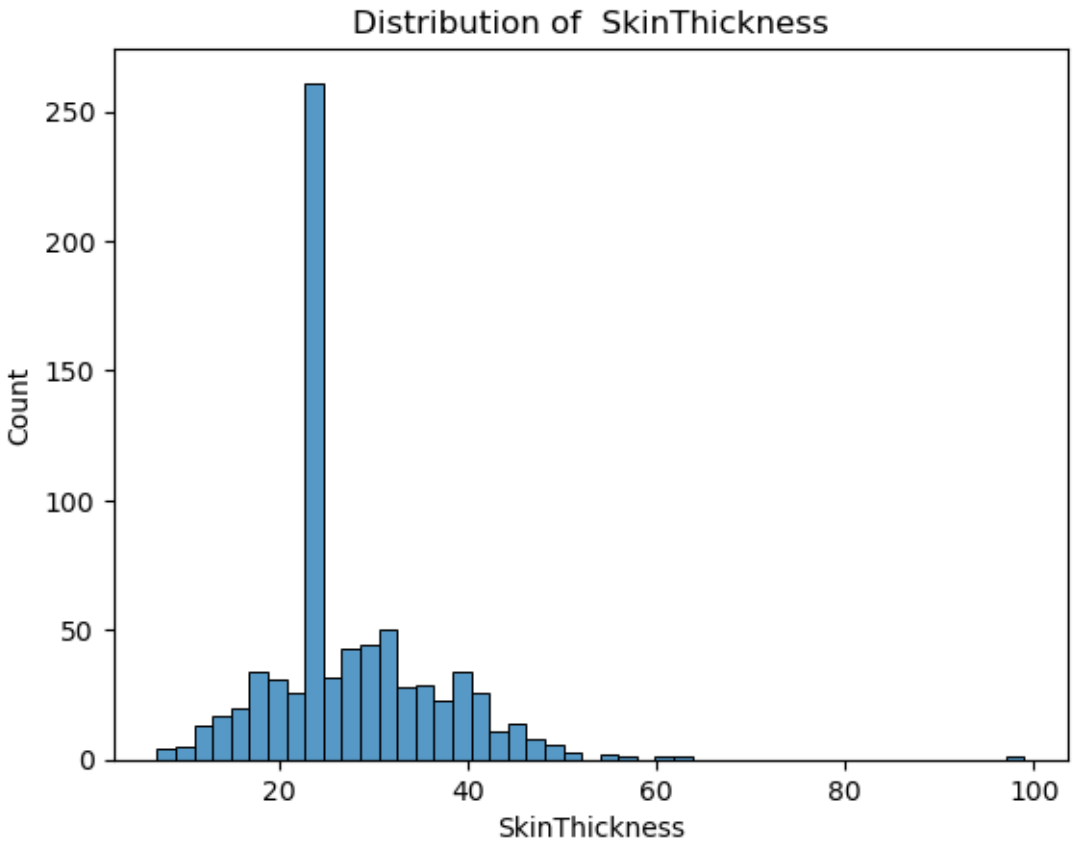
From the above output we can see that there are no zeros in the above features.

[28]:
```
#create histogram for BMI

sns.histplot(x=df['BloodPressure'])
plt.title('Distribution of BloodPressure')
plt.show()
```



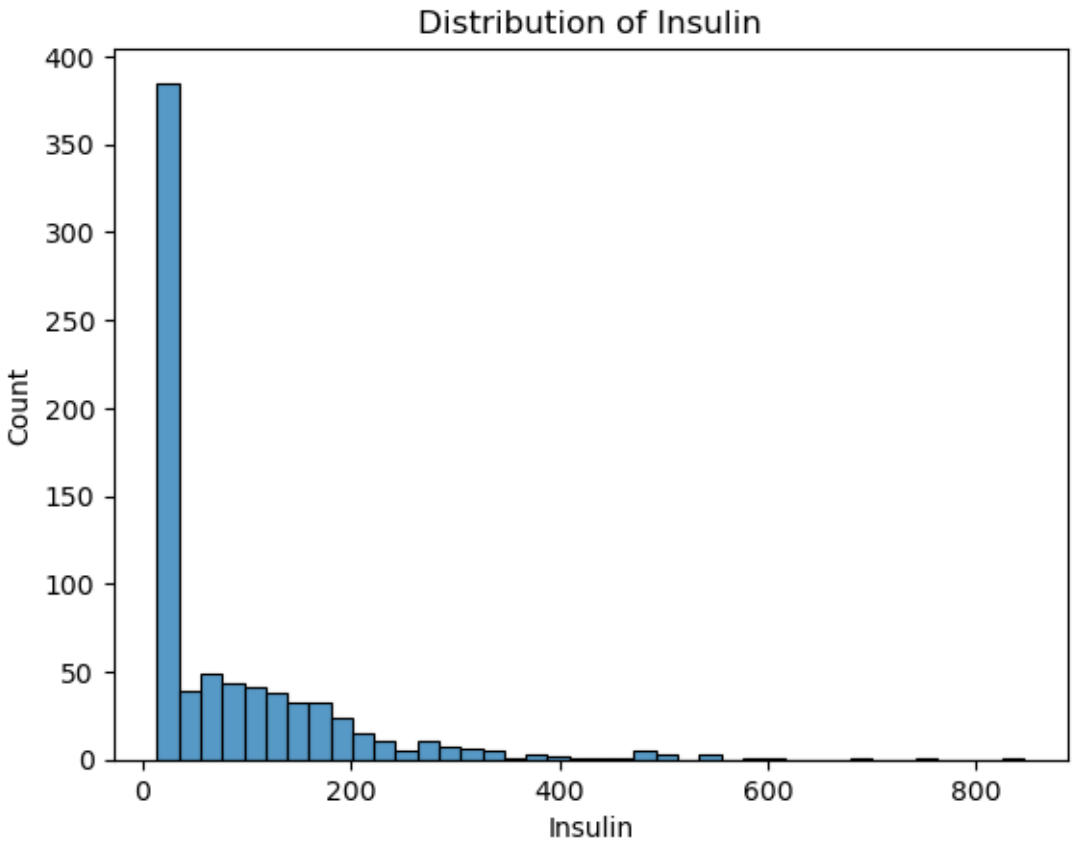
Blood Pressure variable with zeros has been replaced with median.

[29]:
```
#create histogram for SkinThickness

sns.histplot(x=df['SkinThickness'])
plt.title('Distribution of  SkinThickness')
plt.show()
```



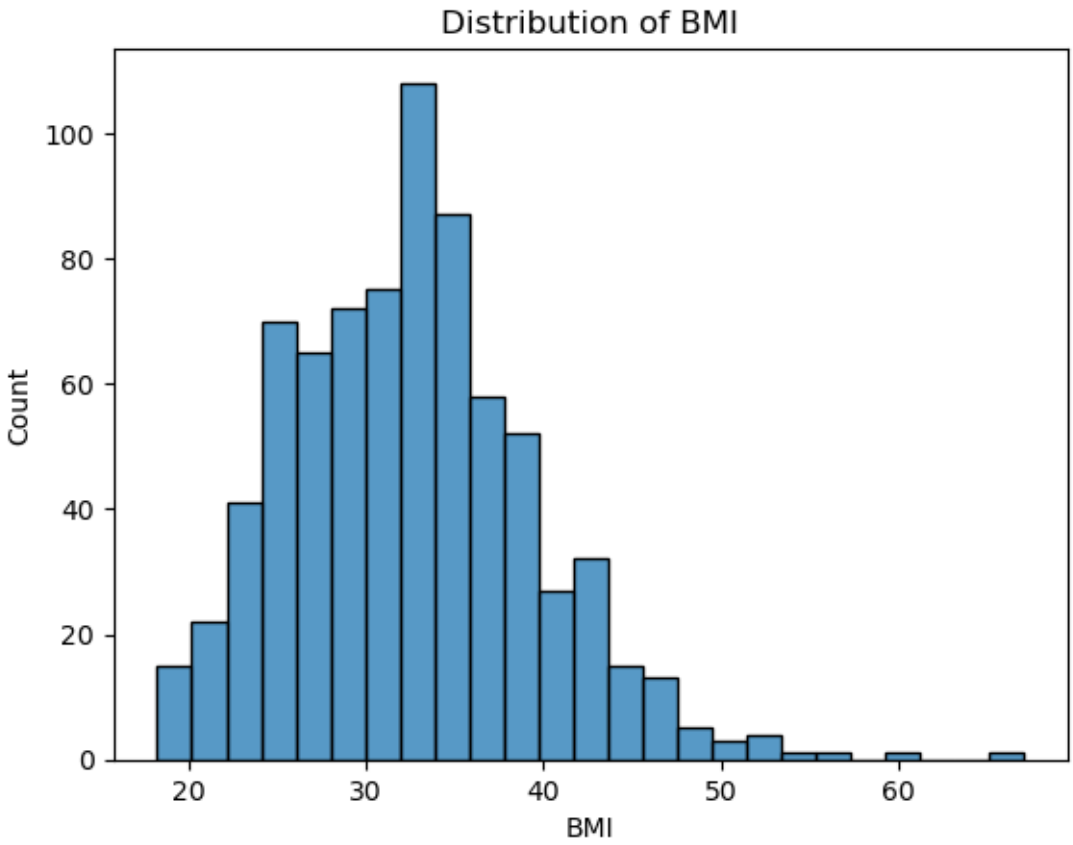
The SkinThickness variable, which initially contained zero values, has now had those zeros substituted with the median value.

[30]:
```
#create histogram for Insulin

sns.histplot(x=df['Insulin'])
plt.title('Distribution of Insulin')
plt.show()
```



Zeros in Insulin variable are replaced with median.

[31]:
```
#create histogram for BMI

sns.histplot(x=df['BMI'])
plt.title('Distribution of BMI')
plt.show()
```



BMI columns with zero values has been replaced with median.

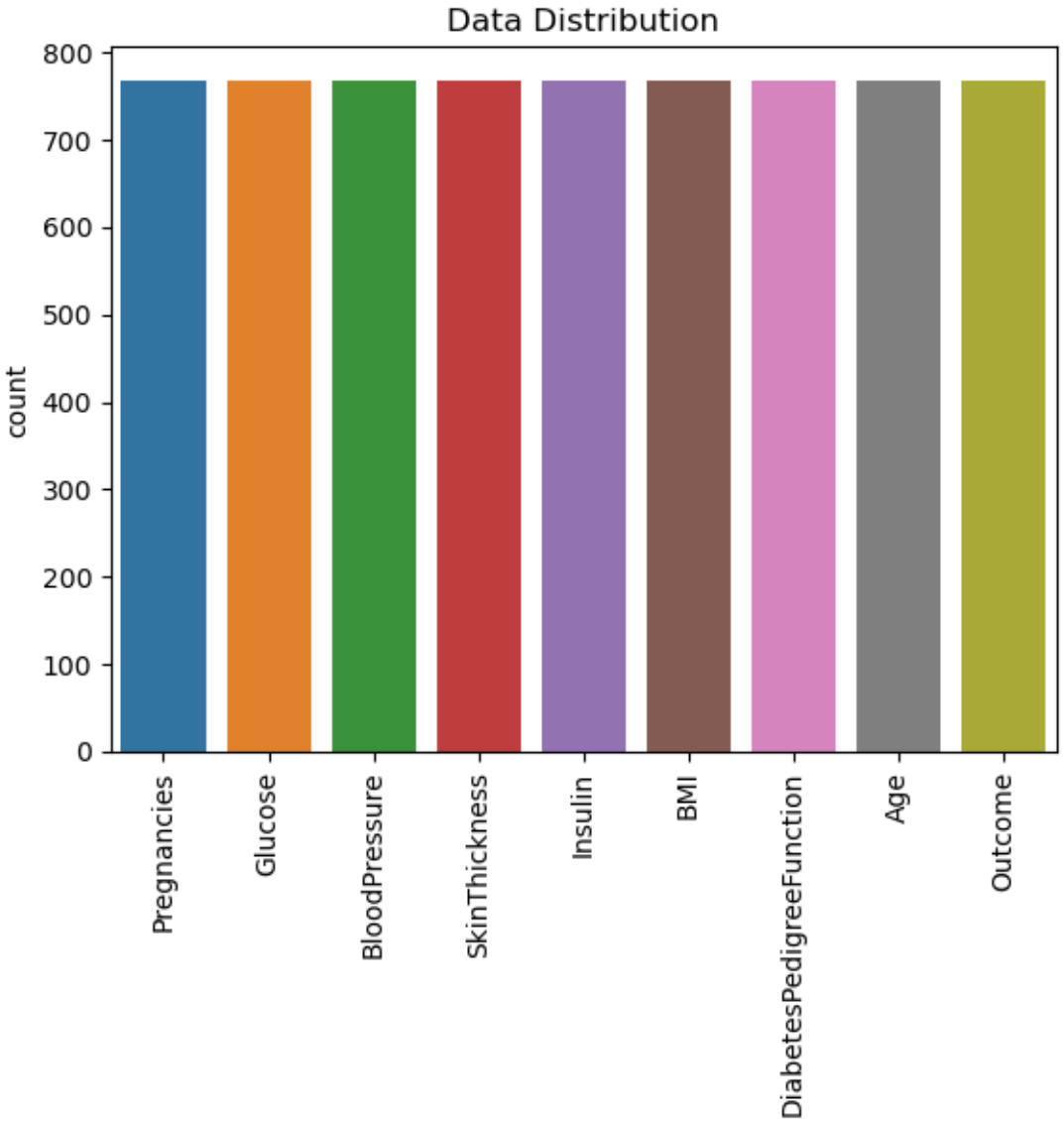
7.0.3 Create a count (frequency) plot describing the data types and the count of variables.

[32]:
```
#Looking up the Data types in our Dataset

df.dtypes
```

[32]:
```
Pregnancies                 int64
Glucose                   float64
BloodPressure               int64
SkinThickness               int64
Insulin                   float64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

Theere are 7 integers and 2 float data types

[33]:
```
##Visualizing data completeness
sns.countplot(data=df)
plt.title('Data Distribution')
plt.xticks(rotation=90)
plt.show()
```



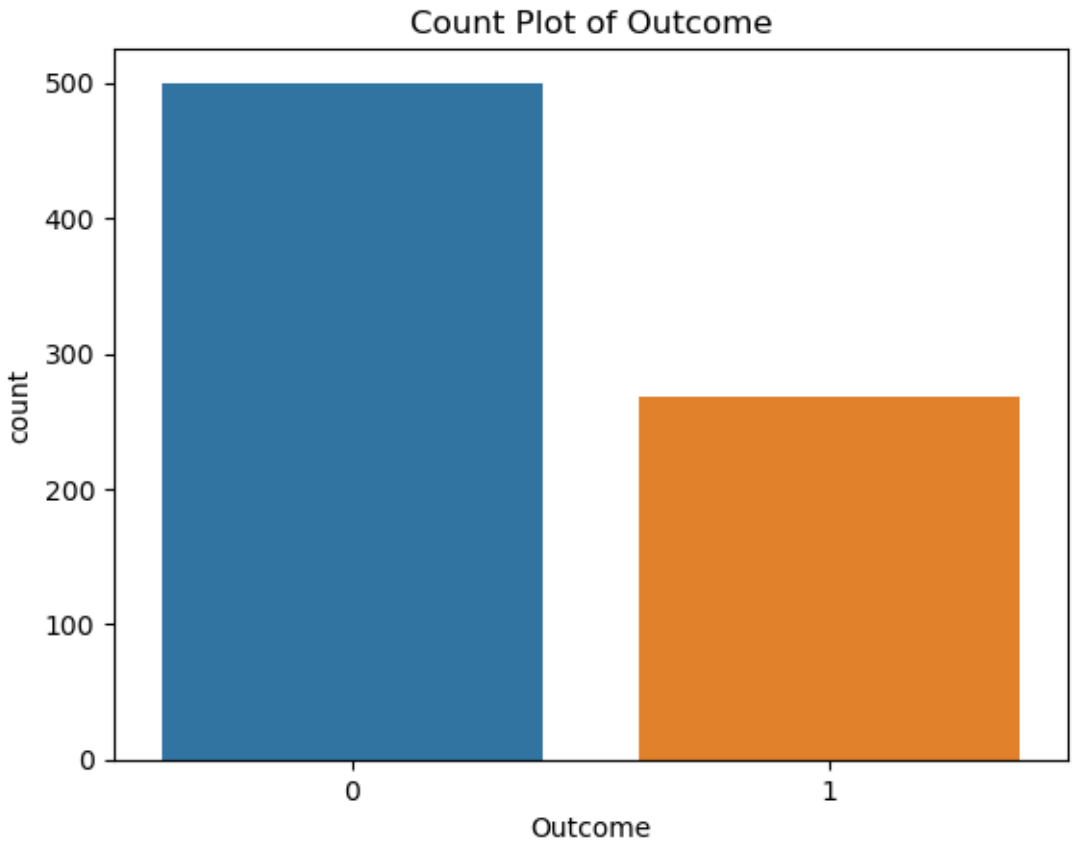
All the variables are equal in distribution

[34]:
```
#Value count of Outcome variable
df['Outcome'].value_counts()
```

[34]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

Patient with diabetes are 268 in total.

[35]:
```
#Visualizing the Target varaible to understand Diabetes distribution among␣
 ↪patient.
sns.countplot(x='Outcome', data=df)
plt.title('Count Plot of Outcome')
plt.show()
```



268 patient have dabetes which are represented as 1, and 500 patients are without diabetes which are 0's.

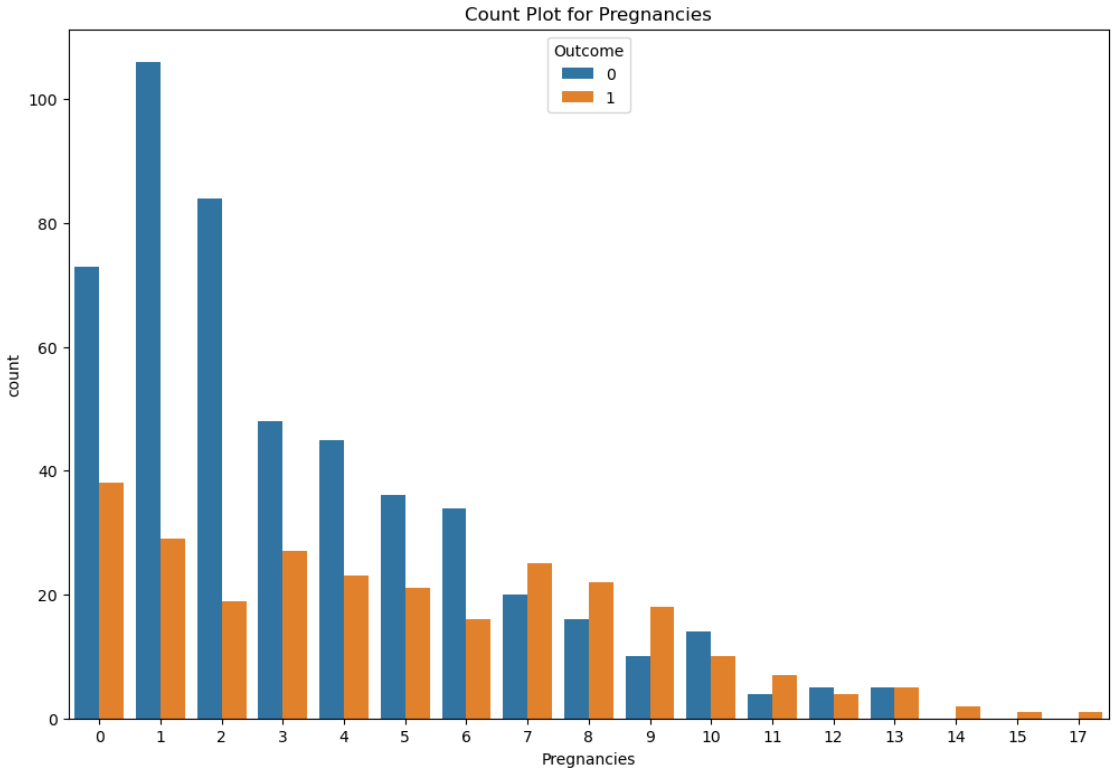
[36]:
```
#Value count of Pregnancies variable
Pregnancies_frequency=df['Pregnancies'].value_counts()
Pregnancies_frequency
```

[36]:
```
1     135
0     111
```

```
2      103
3       75
4       68
5       57
6       50
7       45
8       38
9       28
10      24
11      11
13      10
12       9
14       2
15       1
17       1
Name: Pregnancies, dtype: int64
```

There were 135 patients who experienced their first pregnancy exclusively, which represents the highest number in this category.

[37]:
```
#Visualizing the Pregnancies datatype.
plt.figure(figsize=(12,8))
sns.countplot(data=df,x='Pregnancies', hue='Outcome')
plt.title('Count Plot for Pregnancies')
plt.show()
```



Diabetic individuals are found in all pregnancy and non preganant frequency categories, including women experiencing their first pregnancy who also had diabetes.

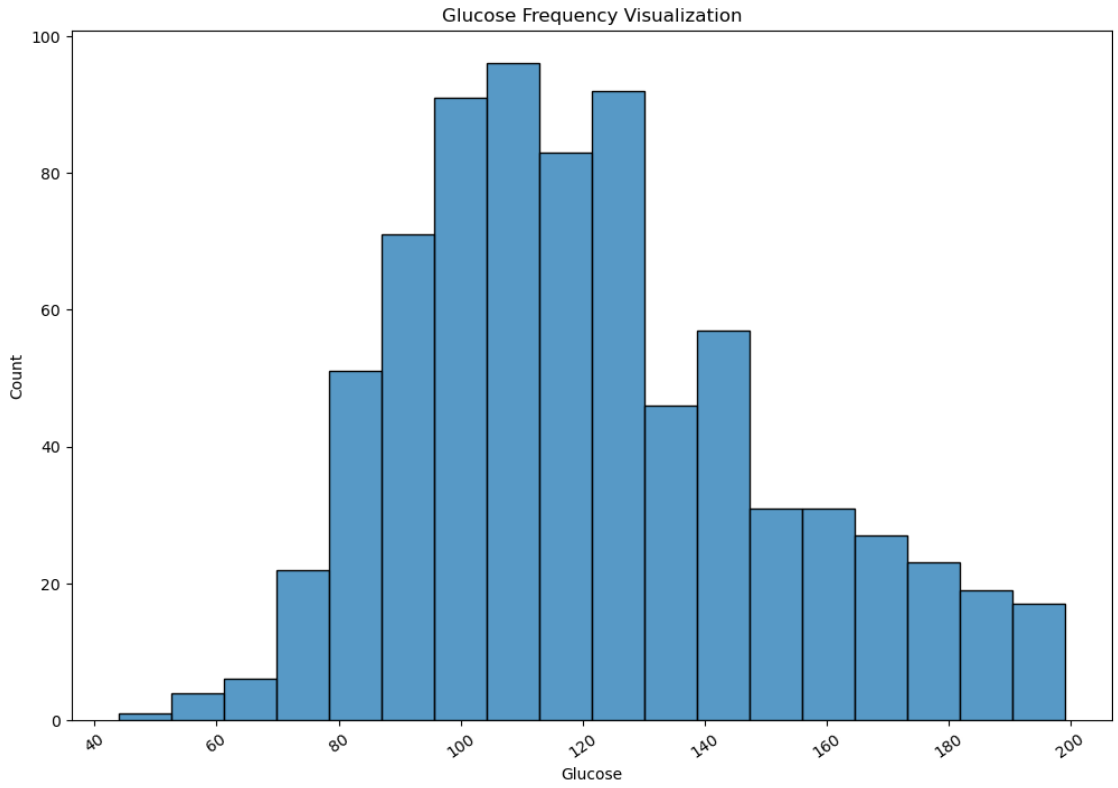
[38]:
```
#Value count of Glucose variable
Glucose_frequency=df['Glucose'].value_counts()
Glucose_frequency
```

[38]:
```
99.0     17
100.0    17
111.0    14
129.0    14
125.0    14
         ..
191.0     1
177.0     1
44.0      1
62.0      1
190.0     1
Name: Glucose, Length: 136, dtype: int64
```

[ ]:
```

```

17 Patients had 99 and 100 Plasma glucose concentration in an oral glucose tolerance test respectively.

[39]:
```
#Visualizing Glucose datatype.

plt.figure(figsize=(12,8))

sns.histplot(data=df, x='Glucose')
plt.title('Glucose Frequency Visualization')

plt.xticks(rotation=35)

plt.show()
```



Glucose in plasma frequency visualization

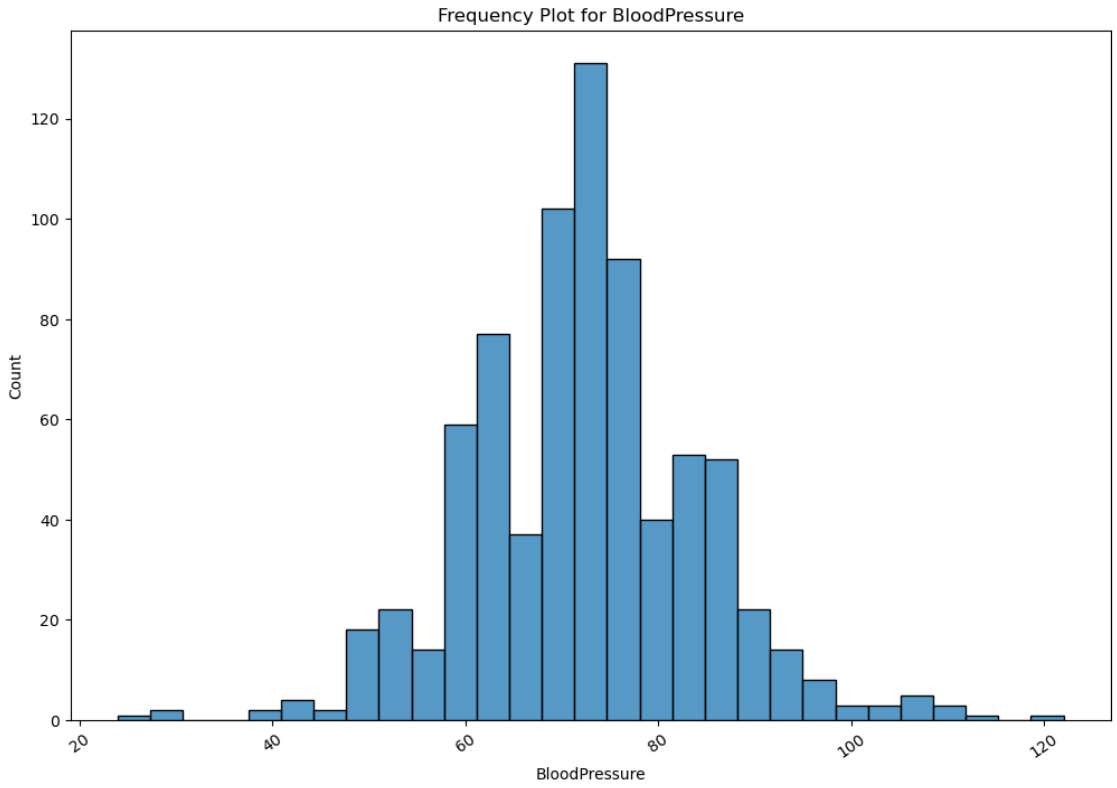
[40]:
```
#Value count of BloodPressure variable
BP_Count=df['BloodPressure'].value_counts().head()
BP_Count
```

[40]:
```
72    79
70    57
74    52
78    45
68    45
Name: BloodPressure, dtype: int64
```

The 79 patients have 72mmHg Diastolic blood pressure (mm Hg).

[41]:
```
#Visualizing the BloodPressure datatype.

plt.figure(figsize=(12,8))

sns.histplot(data=df,x='BloodPressure')

plt.title('Frequency Plot for BloodPressure')
```

```
plt.xticks(rotation=35)

plt.show()
```



Frequency visualizationn of Diastolic blood pressure (mm Hg)

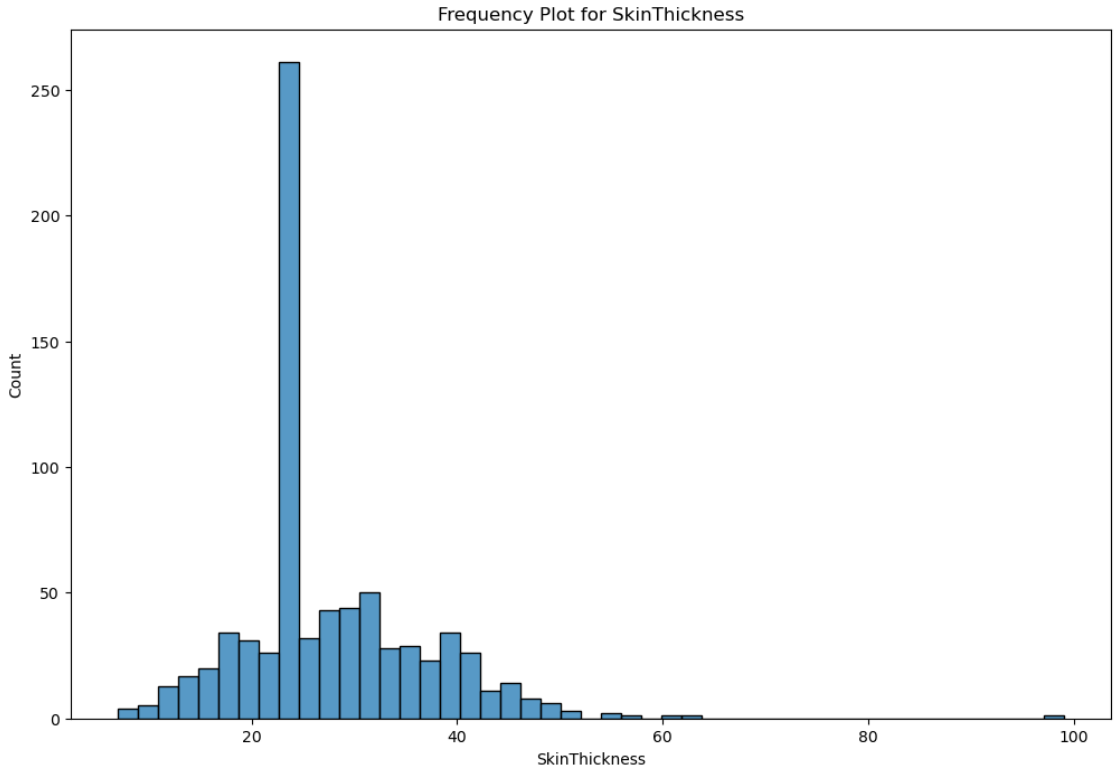
[42]:
```
#Value count of SkinThickness variable
ST_Count=df['SkinThickness'].value_counts().head()
ST_Count
```

[42]:
```
23    249
32     31
30     27
27     23
28     20
Name: SkinThickness, dtype: int64
```

249 patients have 23mm Triceps skinfold thickness (mm)

[43]:
```
#Visualizing the SkinThickness datatype.

plt.figure(figsize=(12,8))
```

```
sns.histplot(data=df,x='SkinThickness')

plt.title('Frequency Plot for SkinThickness')

plt.show()
```



Frequency visualization of SkinThickness

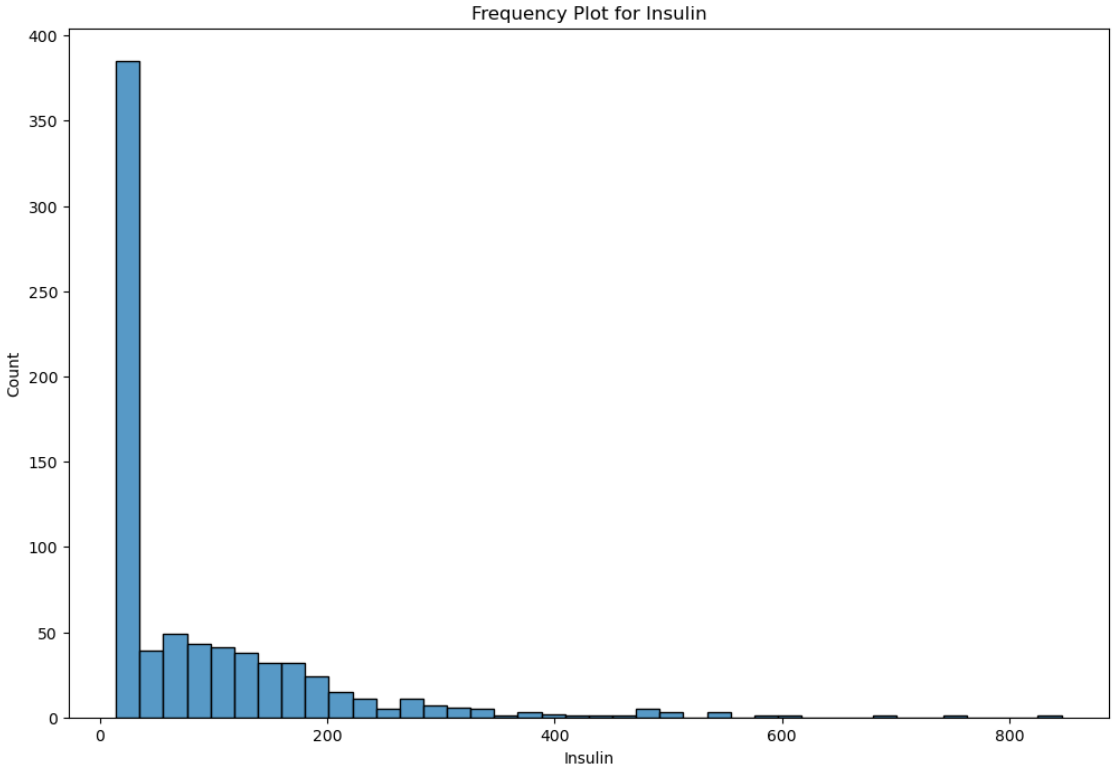
[44]:
```
#Value count of Insulin variable
Insulin_Count=df['Insulin'].value_counts()
Insulin_Count
```

[44]:
```
30.5      374
105.0      11
130.0       9
140.0       9
120.0       8
          ...
73.0        1
171.0       1
255.0       1
52.0        1
```

```
112.0      1
Name: Insulin, Length: 186, dtype: int64
```

374 patients have 30.5 insulin score which is the most occurence.

[45]:
```
#Visualizing the 'Insulin datatype.

plt.figure(figsize=(12,8))

sns.histplot(data=df,x='Insulin')

plt.title('Frequency Plot for Insulin')

plt.show()
```



Frequency visualization of Insulin.

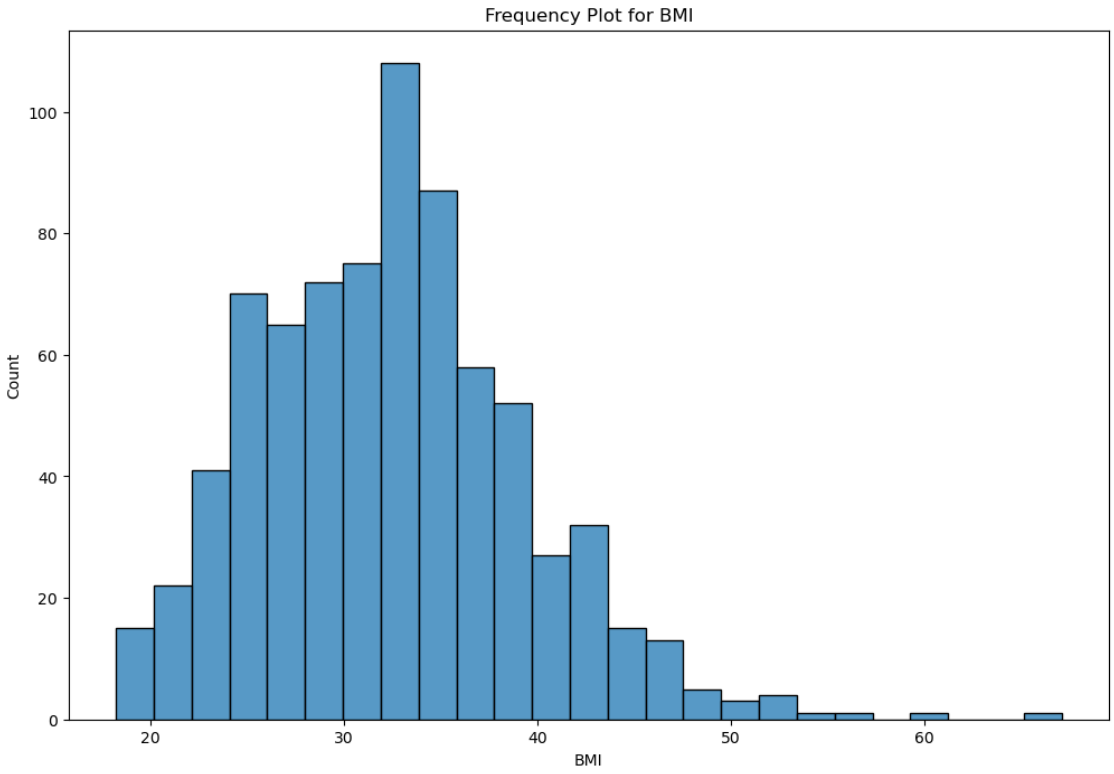
[46]:
```
#Value count of BMI variable
BMI_Count=df['BMI'].value_counts()
BMI_Count
```

[46]:
```
32.0    24
31.2    12
```

```
31.6    12
32.4    10
33.3    10
        ..
36.7     1
41.8     1
42.6     1
42.8     1
46.3     1
Name: BMI, Length: 247, dtype: int64
```

Total number of 24 patient had 32 BMI score which is most prevalent.

[47]:
```
#Visualizing the BMI datatype.

plt.figure(figsize=(12,8))

sns.histplot(data=df,x='BMI')

plt.title('Frequency Plot for BMI')

plt.show()
```



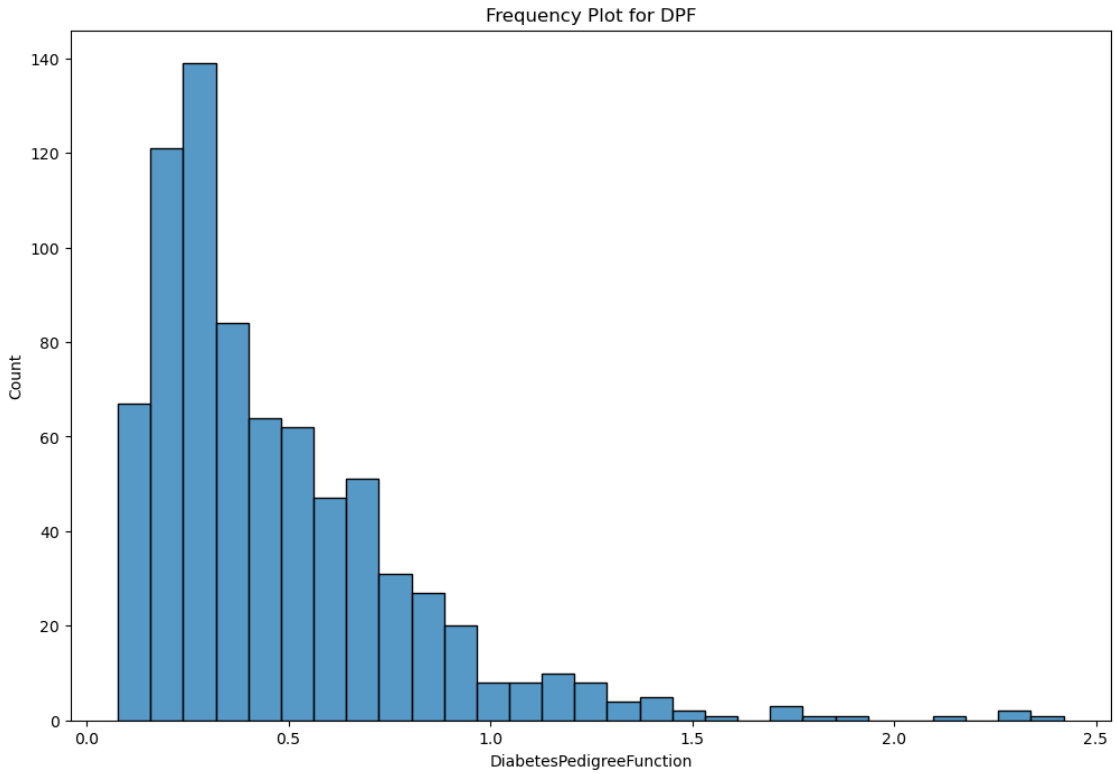
Frequency plot for BMI using histogram.

[48]:
```
#Value count of Diabetes Pedigree Function variable
DPF_Count=df['DiabetesPedigreeFunction'].value_counts()
DPF_Count
```

[48]:
```
0.258    6
0.254    6
0.268    5
0.207    5
0.261    5
        ..
1.353    1
0.655    1
0.092    1
0.926    1
0.171    1
Name: DiabetesPedigreeFunction, Length: 517, dtype: int64
```

6 patients has a Diabetes Pedigree Function score of 0.258

[49]:
```
#Visualizing the Diabetes Pedigree Function datatype.

plt.figure(figsize=(12,8))

sns.histplot(data=df,x='DiabetesPedigreeFunction')

plt.title('Frequency Plot for DPF')

plt.show()
```

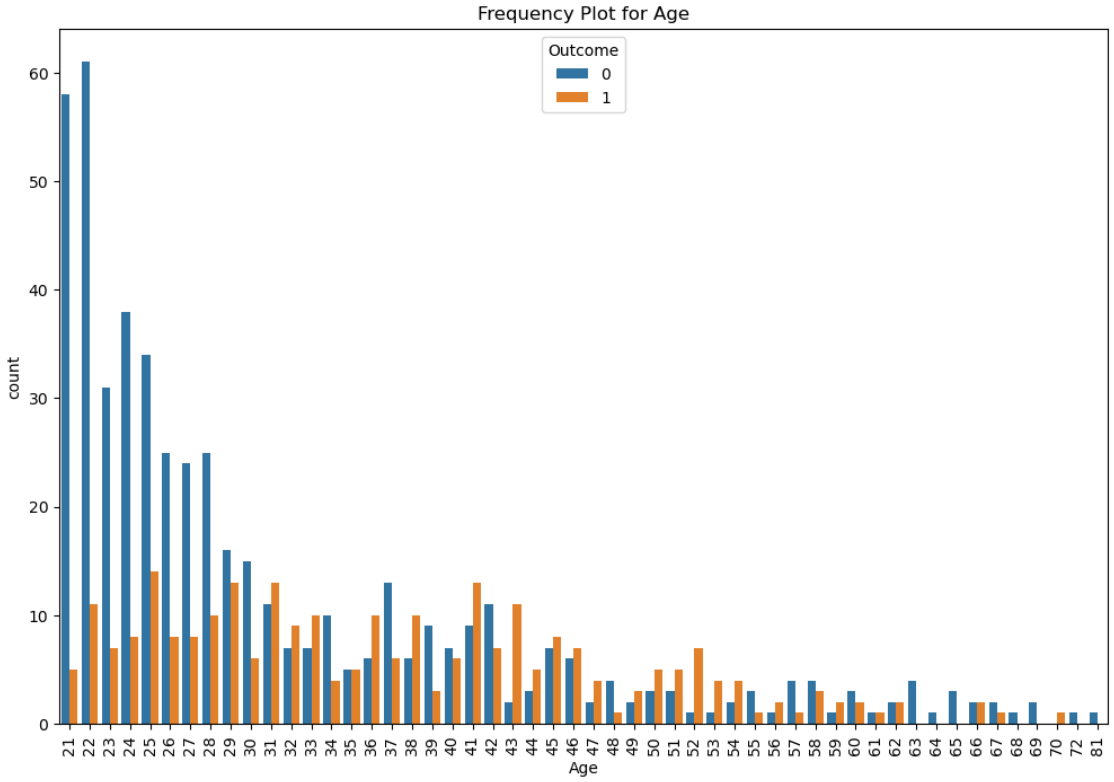


Frequency plot forDiabetes Pedigree Function using histogram.

[50]:
```
#Value count of Age variable
Age_Count=df['Age'].value_counts().head()
Age_Count
```

[50]:
```
22    72
21    63
25    48
24    46
23    38
Name: Age, dtype: int64
```

[51]:
```
#count plot Age Variable
plt.figure(figsize=(12,8))

sns.countplot(x=df['Age'], hue='Outcome', data=df)

plt.title('Frequency Plot for Age')

plt.xticks(rotation=90)
plt.show()
```

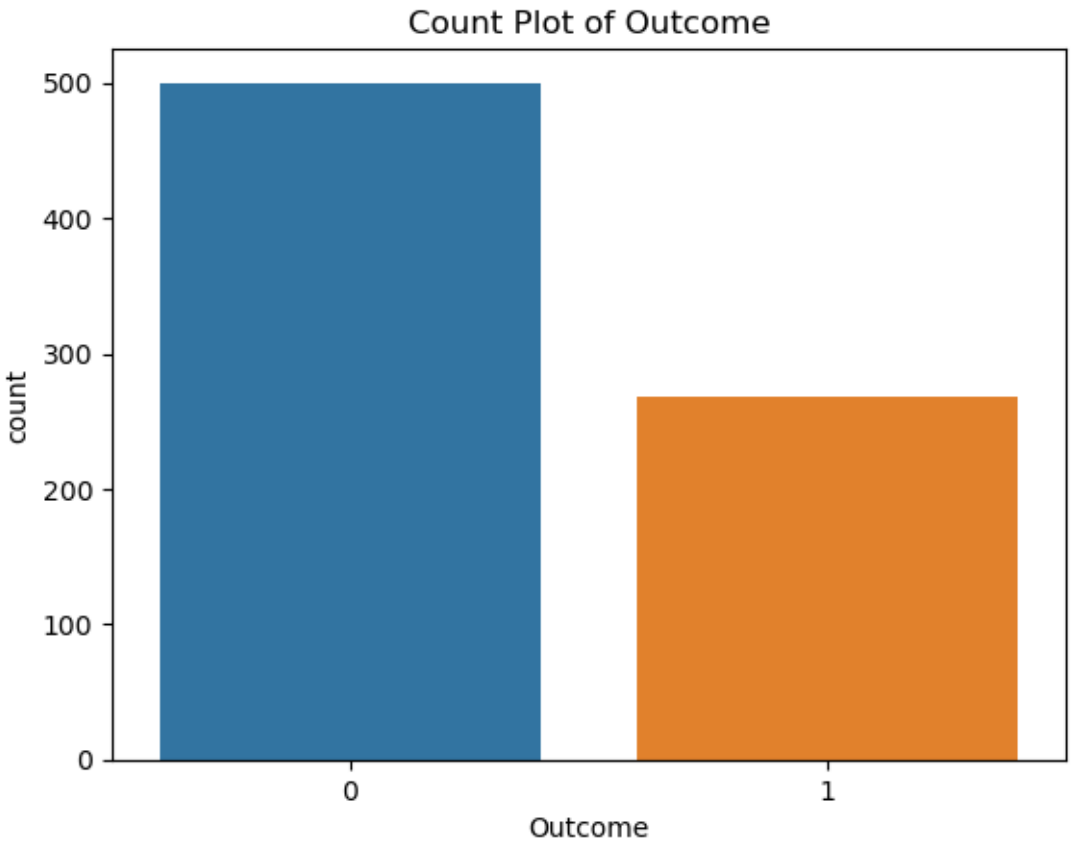


1.Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

[52]:
```
#Value count of Outcome variable
Target_count=df['Outcome'].value_counts()
Target_count
```

[52]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

[53]:
```
#Visualizing the Target varaible to understand Diabetes distribution among␣
 ↪patient.
sns.countplot(x='Outcome', data=df)
plt.title('Count Plot of Outcome')
plt.show()
```



Examining our dataset, it's evident that we are dealing with an imbalanced dataset. There are more occurrences of zeros compared to ones in the target variable. Consequently, we will employ ensemble models, known for their resilience in managing outliers.

The presence of outliers in our dataset holds significant importance; otherwise, we might have considered options like outlier trimming, oversampling, or undersampling to achieve a balanced dataset.

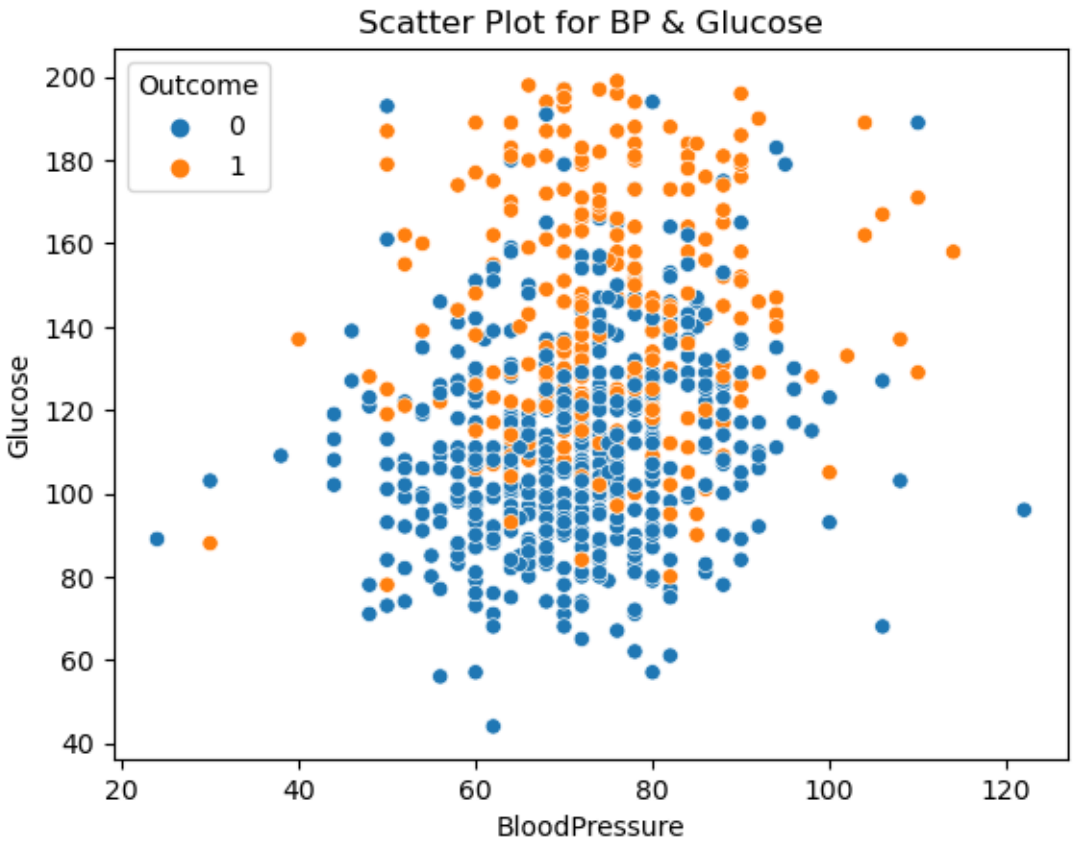
In this analysis, we will apply several machine learning algorithms, including Logistic Regression, K-Nearest Neighbors (KNN), Random Forest Classifier, Decision Tree, and Support Vector Machine.

APPLYING BIVARIATE ANALYSIS USING SCATTER PLOT

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

[54]:
```
#Bivariate Analysis for BloodPressure and Glucose

sns.scatterplot(x=df['BloodPressure'],y=df['Glucose'], hue='Outcome', data=df)

plt.title('Scatter Plot for BP & Glucose')
```

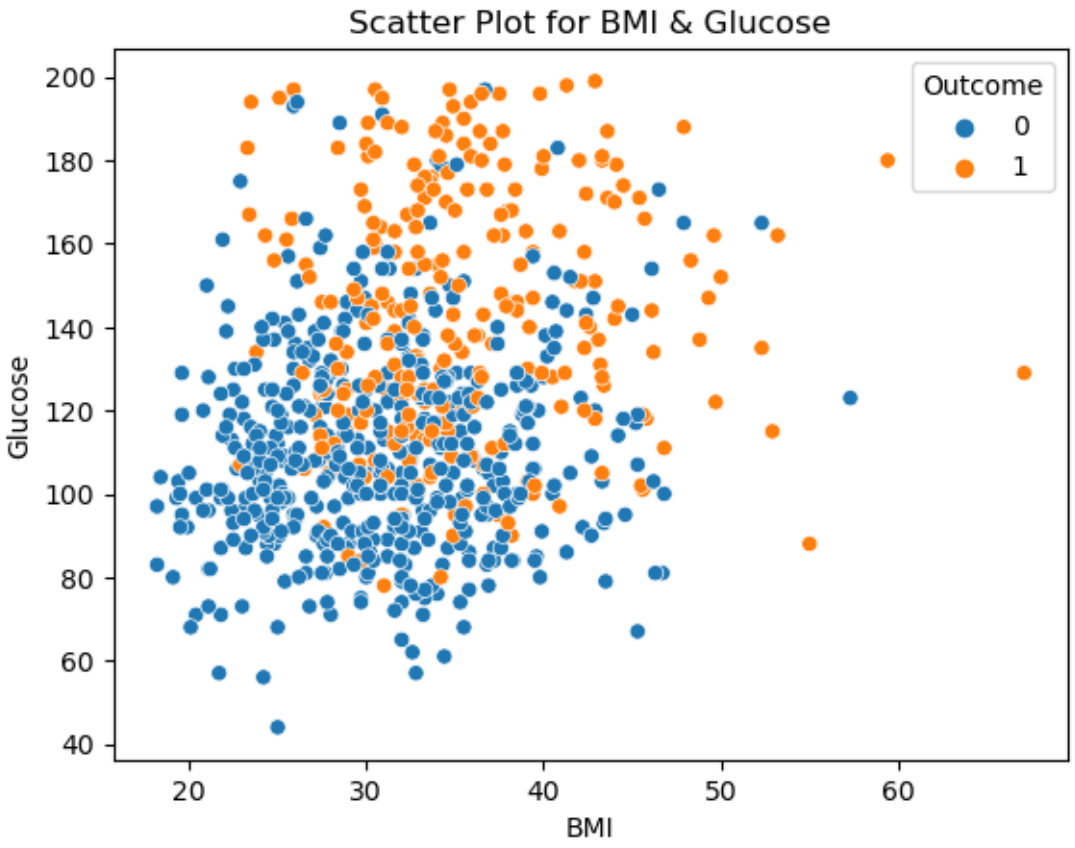
```
plt.show()
```



As glucose levels and blood pressure rise, we observe a corresponding increase in the prevalence of diabetes. The presence of elevated plasma glucose plays a pivotal role in the onset of diabetes. However, this is a weak correlation.

Looking at the plot, we can infer that the Glucose score from 140 to 200 and Diastolic blood pressure from 60 to 90(mm Hg) had diabetes prevalence among patients.

[55]:
```
#Bivariate Analysis for BMI and Glucose

sns.scatterplot(x=df['BMI'],y=df['Glucose'], hue='Outcome', data=df)

plt.title('Scatter Plot for BMI & Glucose')

plt.show()
```



We observe the rise of diabetes as Glucose score increases across BMI.

[56]:
```
#Correlation Analysis for BloodPressure and Glucose

Correlation_Matrix=df.corr()
Correlation_Matrix
```

[56]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	\
Pregnancies	1.000000	0.127964	0.208615	0.032568	
Glucose	0.127964	1.000000	0.218623	0.172361	
BloodPressure	0.208615	0.218623	1.000000	0.147809	
SkinThickness	0.032568	0.172361	0.147809	1.000000	
Insulin	-0.055697	0.357081	-0.028721	0.238188	
BMI	0.021546	0.231469	0.281132	0.546951	
DiabetesPedigreeFunction	-0.033523	0.137106	-0.002378	0.142977	
Age	0.544341	0.266600	0.324915	0.054514	
Outcome	0.221898	0.492908	0.165723	0.189065	

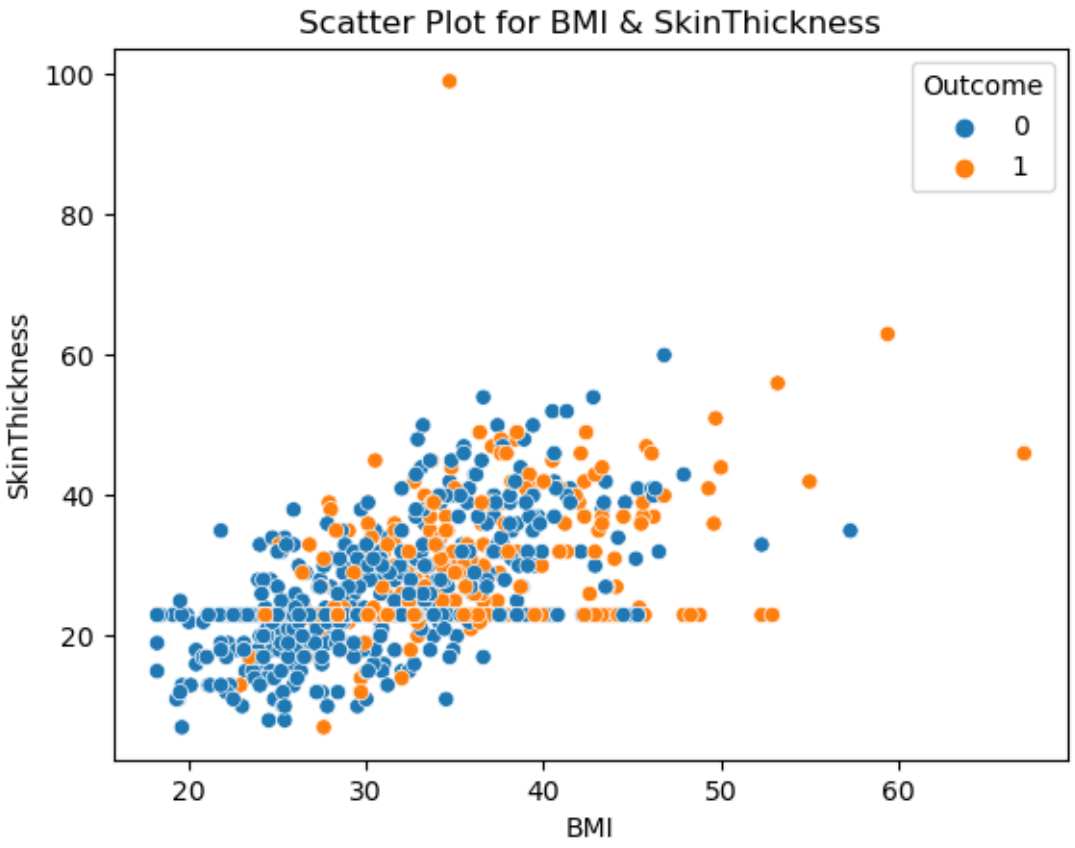
	Insulin	BMI	DiabetesPedigreeFunction	\
Pregnancies	-0.055697	0.021546	-0.033523	

```
Glucose                    0.357081  0.231469                 0.137106
BloodPressure             -0.028721  0.281132                -0.002378
SkinThickness              0.238188  0.546951                 0.142977
Insulin                    1.000000  0.189022                 0.178029
BMI                        0.189022  1.000000                 0.153506
DiabetesPedigreeFunction   0.178029  0.153506                 1.000000
Age                       -0.015413  0.025744                 0.033561
Outcome                    0.148457  0.312249                 0.173844

                                Age   Outcome
Pregnancies                0.544341  0.221898
Glucose                    0.266600  0.492908
BloodPressure              0.324915  0.165723
SkinThickness              0.054514  0.189065
Insulin                   -0.015413  0.148457
BMI                        0.025744  0.312249
DiabetesPedigreeFunction   0.033561  0.173844
Age                        1.000000  0.238356
Outcome                    0.238356  1.000000
```

BMI and SkinThickness, Age and Pregnancies have a moderate positive correlation.

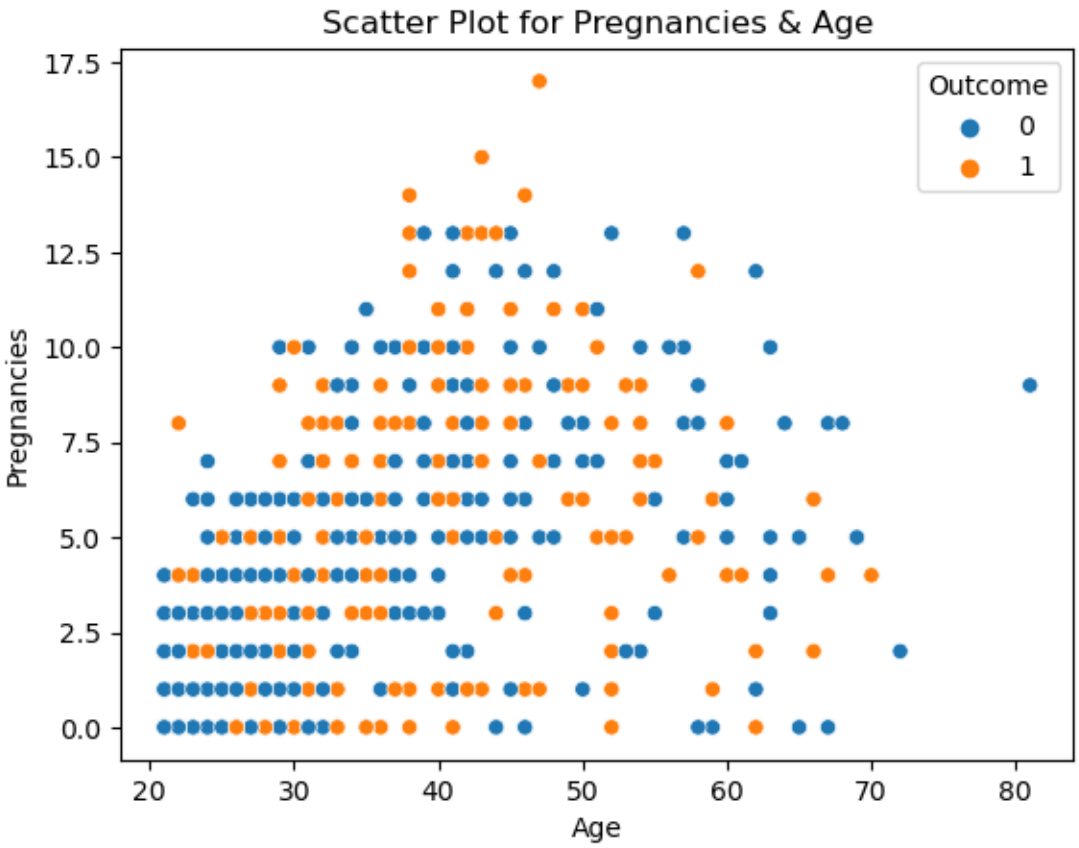
It's important to note that the strength of a relationship, as determined by the correlation coefficient, only measures linear associations. Non-linear relationships may exist but would not be captured by the correlation coefficient. Additionally, correlation does not imply causation, so even if a relationship is strong, it does not necessarily mean that one variable causes the other.

[57]:
```
#Visualizing Correlation analysis for BMI and SkinThickness

sns.scatterplot(x=df['BMI'],y=df['SkinThickness'], hue='Outcome', data=df)

plt.title('Scatter Plot for BMI & SkinThickness')

plt.show()
```



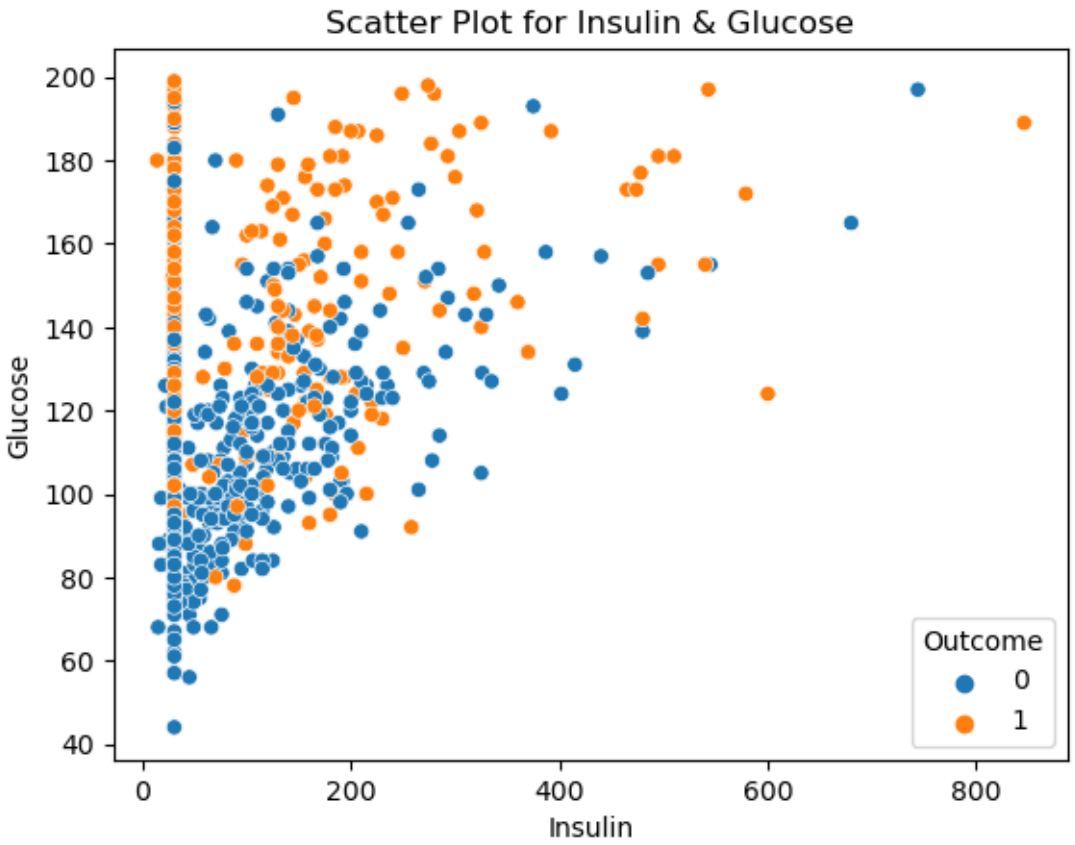
From the above plot, we can notice a moderate linear relationship between BMI and SkinThickness. As SkinThickness and BMI increase we observe the increase of the prevalence of diabetes.

[58]:
```
#Visualizing Correlation analysis for Age and Pregnancies

sns.scatterplot(x=df['Age'],y=df['Pregnancies'], hue='Outcome', data=df)

plt.title('Scatter Plot for Pregnancies & Age')

plt.show()
```



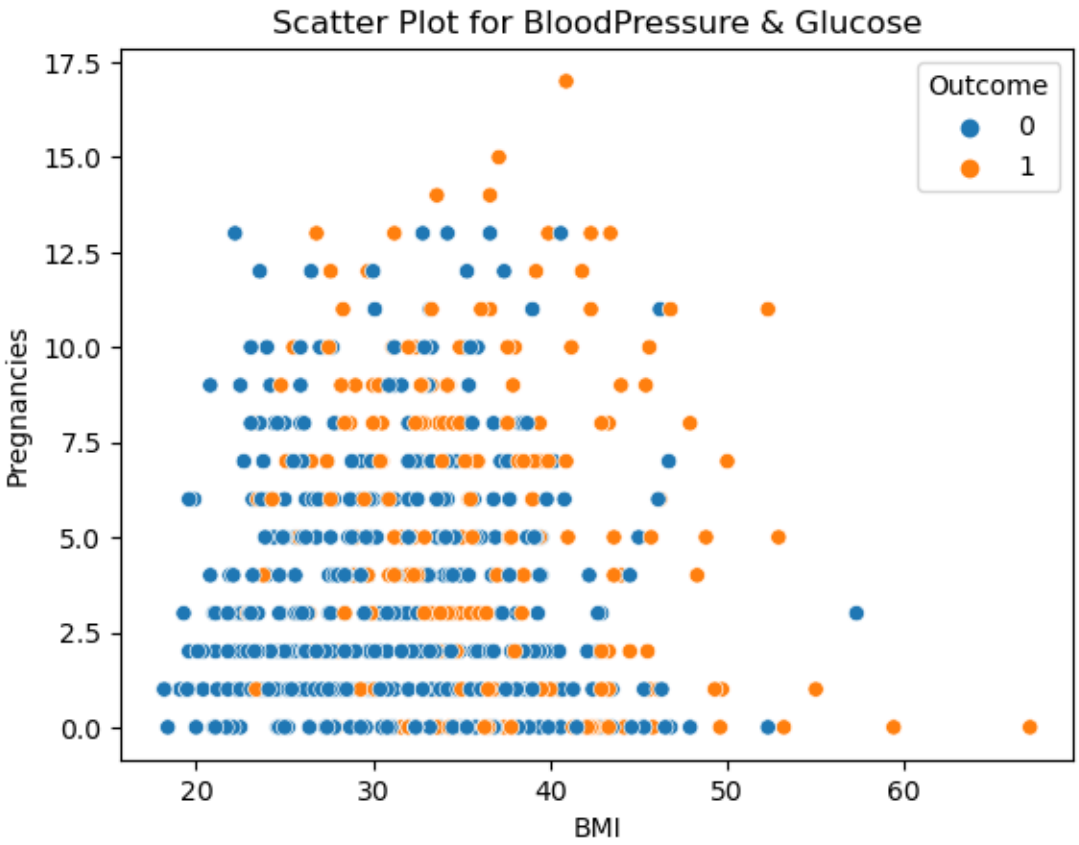
We observe the rise in Diabetes, in number of times pregnant from 0.0 to 17.5, with Age concentration from 30 to 55.

[59]:
```
#Visualizing Correlation analysis for Insulin and Glucose

sns.scatterplot(x=df['Insulin'],y=df['Glucose'], hue='Outcome', data=df)

plt.title('Scatter Plot for Insulin & Glucose')

plt.show()
```



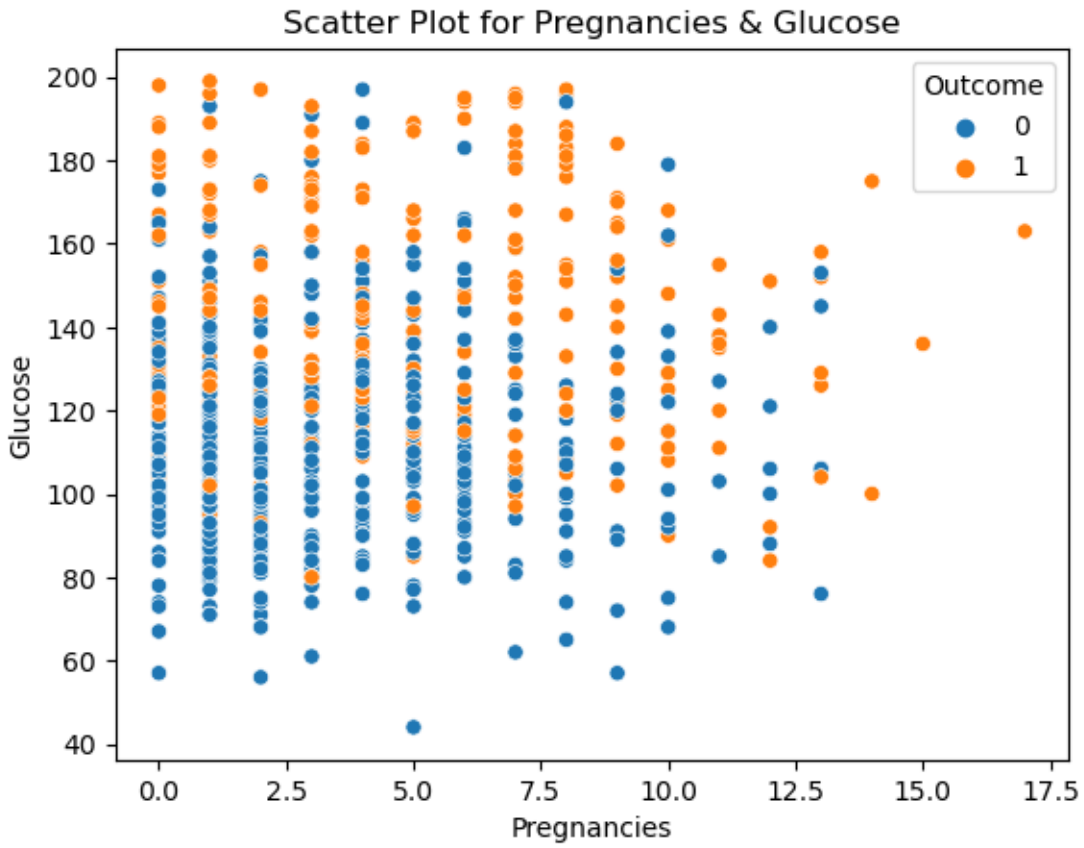
With increase in Glucose and Insulin we observe increase in Diabetes.

[60]:
```
#Visualizing Correlation analysis for BloodPressure and Glucose

sns.scatterplot(x=df['BMI'],y=df['Pregnancies'], hue='Outcome', data=df)

plt.title('Scatter Plot for BloodPressure & Glucose')

plt.show()
```



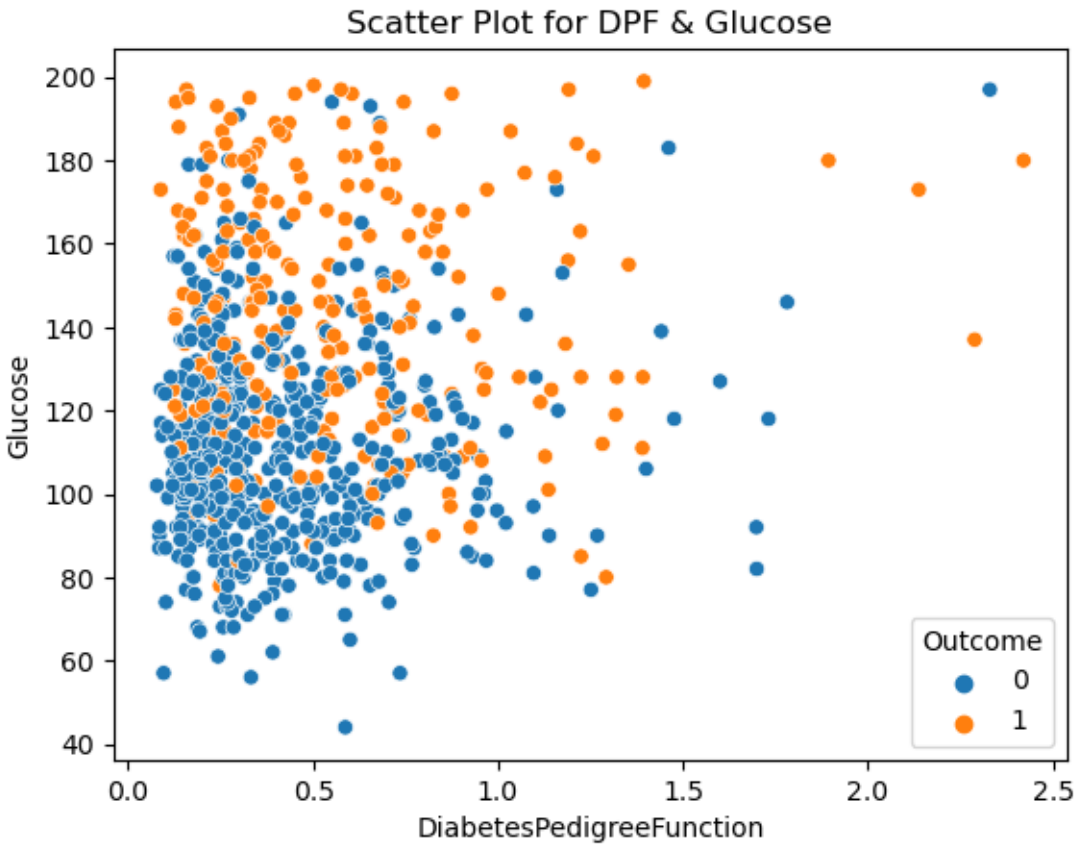
We can infer that women who had never been pregnant also had diabetes predominantly more from BMI above 40. women with BMI from 30 to 50 that have been pregnant more than once and above have diabetes.

[61]:
```
#Visualizing Correlation analysis for Pregnancies and Glucose

sns.scatterplot(x=df['Pregnancies'],y=df['Glucose'], hue='Outcome', data=df)

plt.title('Scatter Plot for Pregnancies & Glucose')

plt.show()
```



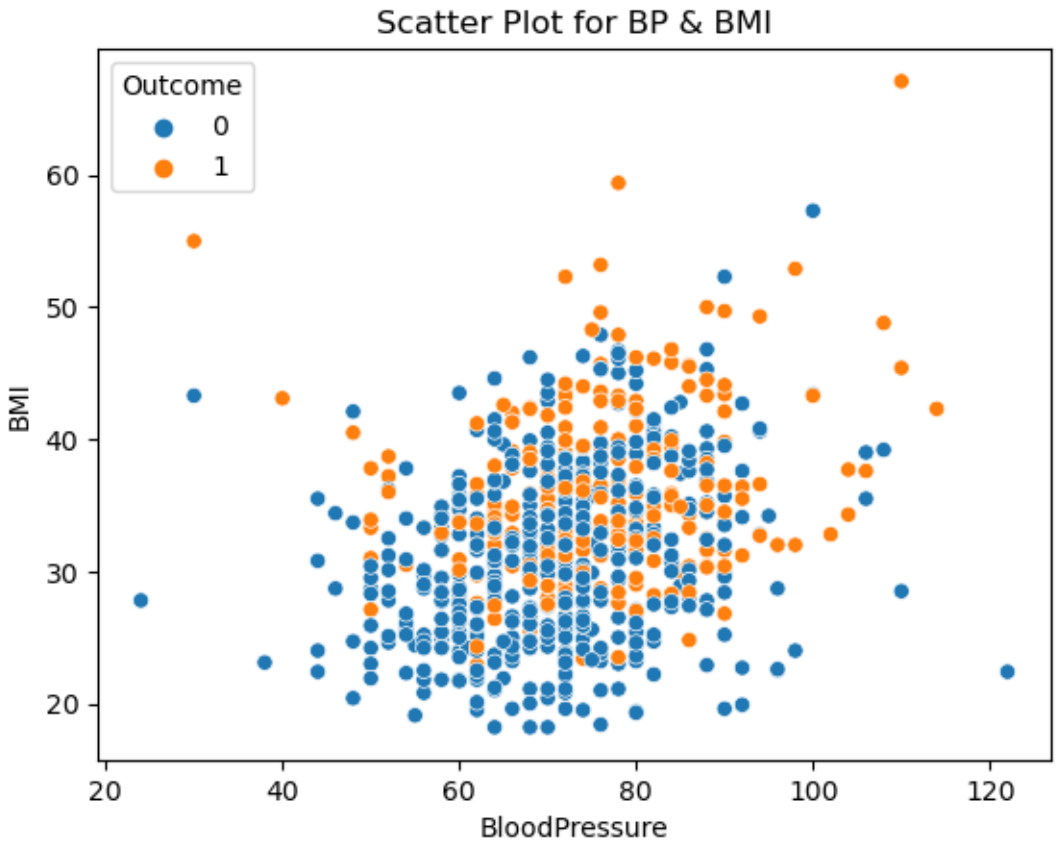
Individuals with plasma glucose levels, whether pregnant or not ranging from 0 to 11, have diabetes when their glucose score is 120 or higher.

[62]:
```
#Visualizing Correlation analysis for DiabetesPedigreeFunction and Glucose

sns.scatterplot(x=df['DiabetesPedigreeFunction'],y=df['Glucose'],␣
 ↪hue='Outcome', data=df)

plt.title('Scatter Plot for DPF & Glucose')

plt.show()
```



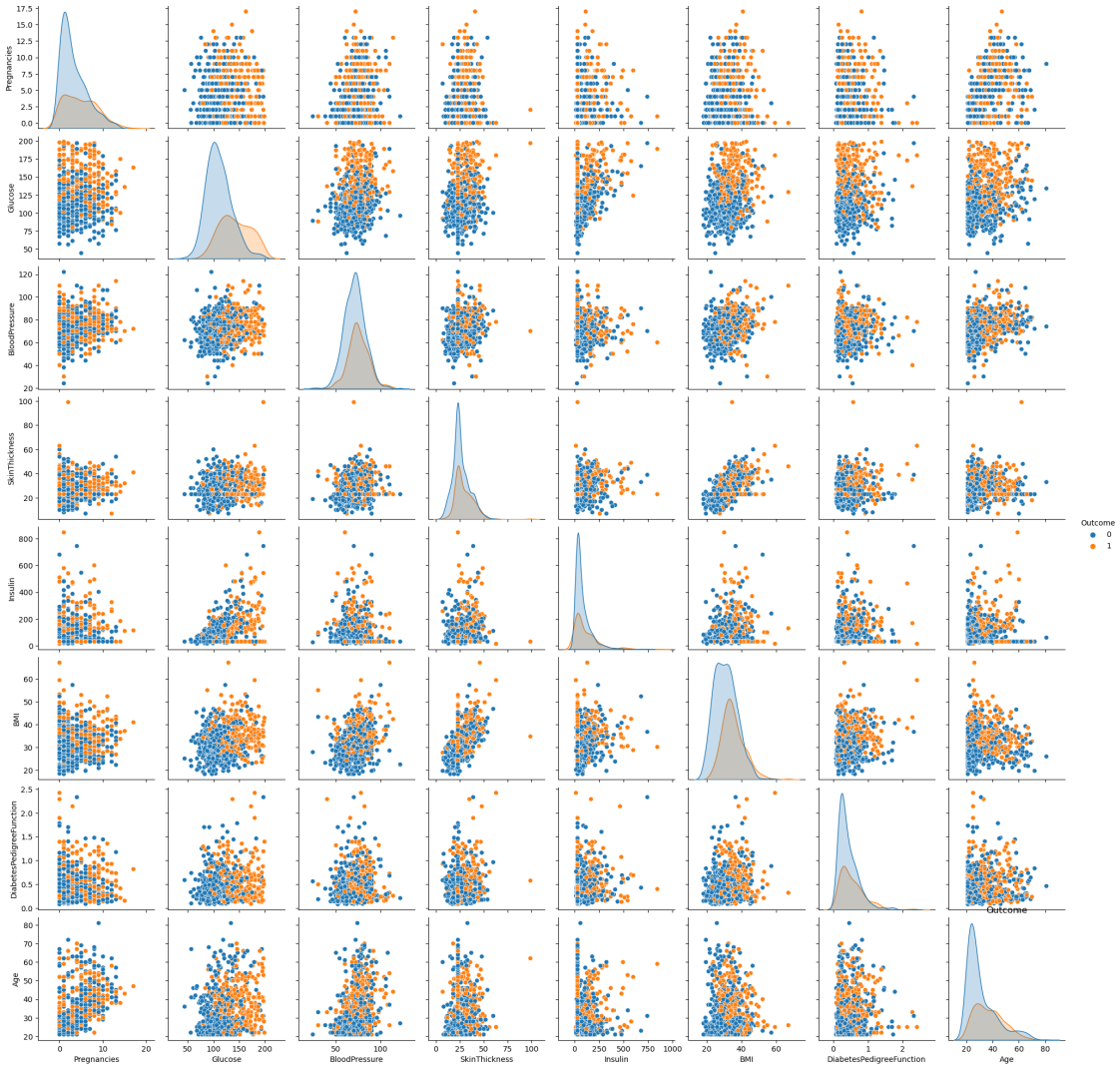
From 0.0 to 1.4 patients with this score of Diabetes Pedigree Function have diabetes as it realtes to glucose score that ranges from 120 and above.

[63]:
```
#Visualizing Correlation analysis for  and Glucose

sns.scatterplot(x=df['BloodPressure'],y=df['BMI'], hue='Outcome', data=df)

plt.title('Scatter Plot for BP & BMI')

plt.show()
```



From 30 to 50 BMI and from 60 t0 110mmg Blood Pressure, we observe the occurence of diabetes.

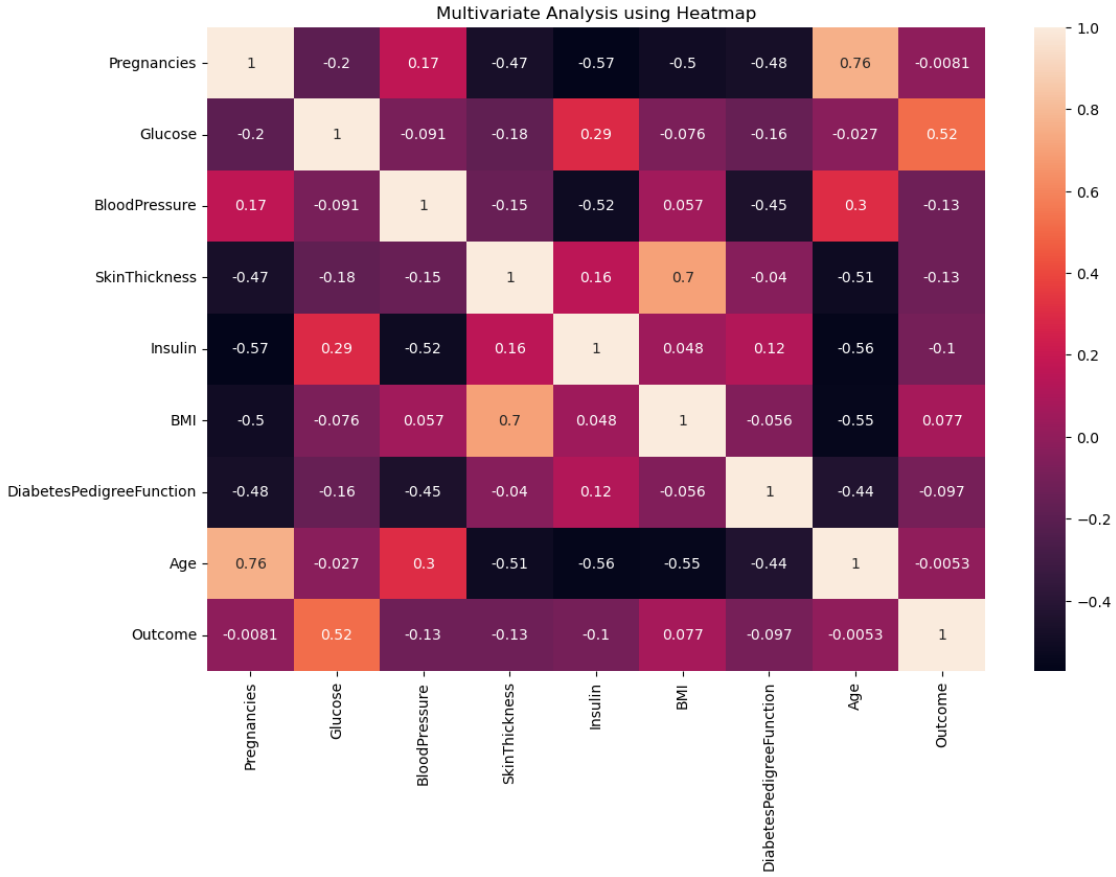
[64]:
```
#Using Seaborn to visualise all features

sns.pairplot(df, hue="Outcome")

plt.title("Outcome")

plt.show()
```



Pairplot visualising the features in the dataset

7.0.4 3.Perform correlation analysis. Visually explore it using a heat map.

[65]:
```
#Multivariate Analysis using Heatmap

plt.figure(figsize=(12,8))

sns.heatmap(Correlation_Matrix.corr(),annot=True)

plt.title('Multivariate Analysis using Heatmap')

plt.show()
```



The heatmap, through its color variations, validates that there is a correlation among Age and the variables of Pregnancies, BMI, and SkinThickness when compared to the remaining variables in our dataset.

Data Modeling: 1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process. 2. Apply an appropriate classification algorithm to build a model. 3. Compare various models with the results from KNN algorithm. 4. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

7.0.5 DATA MODELING

[66]:
```
#create dependent and IDV

X= df.iloc[:,:-1].values
y= df.iloc [:,-1].values
```

[67]:
```
#Checking our IDV output after splitting
X
```

[67]:
```
array([[  6.   , 148.   ,  72.   , …,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , …,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , …,  23.3  ,   0.672,  32.   ],
       …,
       [  5.   , 121.   ,  72.   , …,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , …,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , …,  30.4  ,   0.315,  23.   ]])
```

[68]:
```
# Data Split using Scikit learn library

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)
```

We shall split the datset in an 80% to 20% train, test ratio.

[69]:
```
#checking the split ratio for train set
X_train.shape
```

[69]:
```
(614, 8)
```

[70]:
```
#checking the split ratio for test set
X_test.shape
```

[70]:
```
(154, 8)
```

7.0.6 APPLY LOGISTIC REGRESSION

Logistic regression is often used to handle imbalanced datasets because it is a binary classification algorithm that can be adapted to address the class imbalance problem effectively.

[71]:
```
#Apply Logistic Regression

from sklearn.linear_model import LogisticRegression
log_reg= LogisticRegression()
```

[72]:
```
#Model Training
log_reg.fit(X_train,y_train)
```

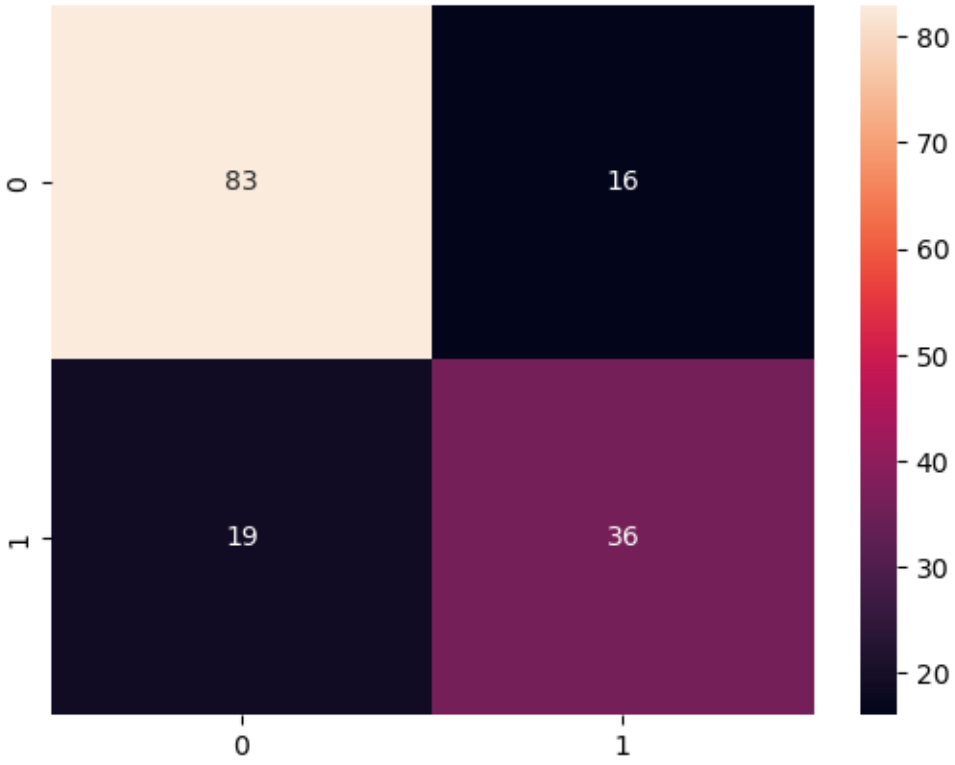
[72]:
```
LogisticRegression()
```

[73]:
```
#Predicting the Test Data
y_pred = log_reg.predict(X_test)
```

[74]:
```
#Evaluating the model
from sklearn.metrics import␣
 ↪confusion_matrix,accuracy_score,classification_report
```

[75]:
```
#Creating confusion Matrix
CM=confusion_matrix(y_test,y_pred)
```

[76]:
```
#visualizing confusion matrix

sns.heatmap(CM, annot=True)
plt.title
plt.show()
```



83 is the True Negative, because the data is imbalanced. The model predicted the True positive score as 36.

[149]:
```
#Accuracy Score

print('LR Accuracy Score', accuracy_score(y_test, y_pred))
```

```
LR Accuracy Score 0.6558441558441559
```

The accuracy score of our LR model is 77%, this is satisfactory considering the outliers in our dataset. However, we shall be looking at other models to make to compare which have better accuracy.

[78]:
```
#Print classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.84      0.83        99
           1       0.69      0.65      0.67        55

    accuracy                           0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154
```

The precision, recall and f1-score for those patient with diabetes as predicted by our model is low. However, the accuracy remains 77%

7.0.7 ROC_AUC_SCORE

[79]:
```
from sklearn.metrics import roc_auc_score,roc_curve

prob=log_reg.predict_proba(X)
prob
```

[79]:
```
array([[0.21786763, 0.78213237],
       [0.94168349, 0.05831651],
       [0.19382289, 0.80617711],
       …,
       [0.85730635, 0.14269365],
       [0.6072614 , 0.3927386 ],
       [0.92771299, 0.07228701]])
```

The probabilistic outcome predicted both the positive and negative classes which are the zeros and ones..

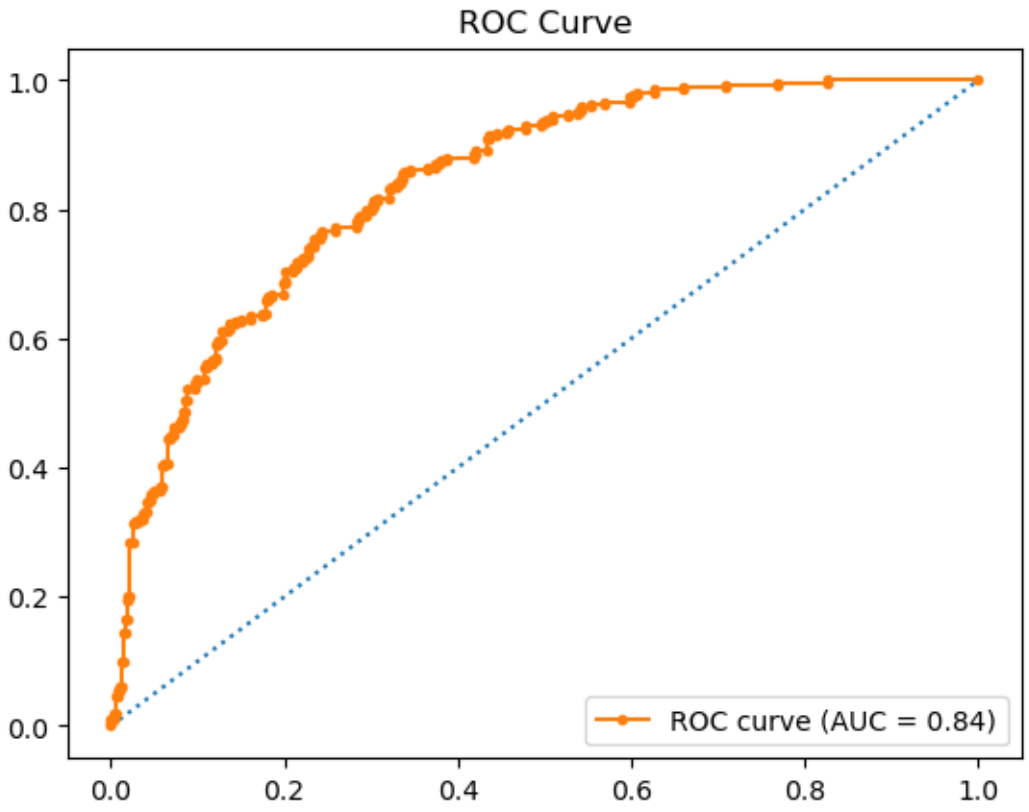
[80]:
```
#select the probability for positive outcome only for patients with diabetes
prob=prob[:,1]
```

prob = prob[:, 1]: This line extracts the second column (index 1) of the prob array, which contains the predicted probabilities for the positive class. As a result, prob will now be a 1D array containing the predicted probabilities for the positive class.

[81]:
```
#calculate roc_auc score

auc= roc_auc_score(y,prob)

print('AUC Score is',auc)
```

```
AUC Score is 0.8408805970149253
```

[82]:
```
#Create ROC Curve

fpr,tpr,thresholds=roc_curve(y,prob)

# consider all possible values for random FPR and TPR
plt.plot([0,1],[0,1], ls='dotted')

#Plotting the ROC
plt.plot(fpr,tpr,marker='.', label=f'ROC curve (AUC = {auc:.2f})')

plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



It's a valuable tool for understanding how well your model distinguishes between the two classes.

The ROC curve is a plot of True Positive Rate (Sensitivity) on the y-axis and False Positive Rate on the x-axis. True Positive Rate (TPR) is the proportion of actual positive cases correctly predicted as positive by the model (sensitivity). False Positive Rate (FPR) is the proportion of actual negative cases incorrectly predicted as positive by the model (1-specificity).

We can infer that our model evaluation of binary classification which is 0.84, is reasonably close to

the perfect score of 1. However, we shall be considering other classification algalgorithm.

7.0.8 APPLYING DECISION TREEE TO THE MODEL

[83]:
```
#Import important library

from sklearn.tree import DecisionTreeClassifier
decision_tree= DecisionTreeClassifier()
```

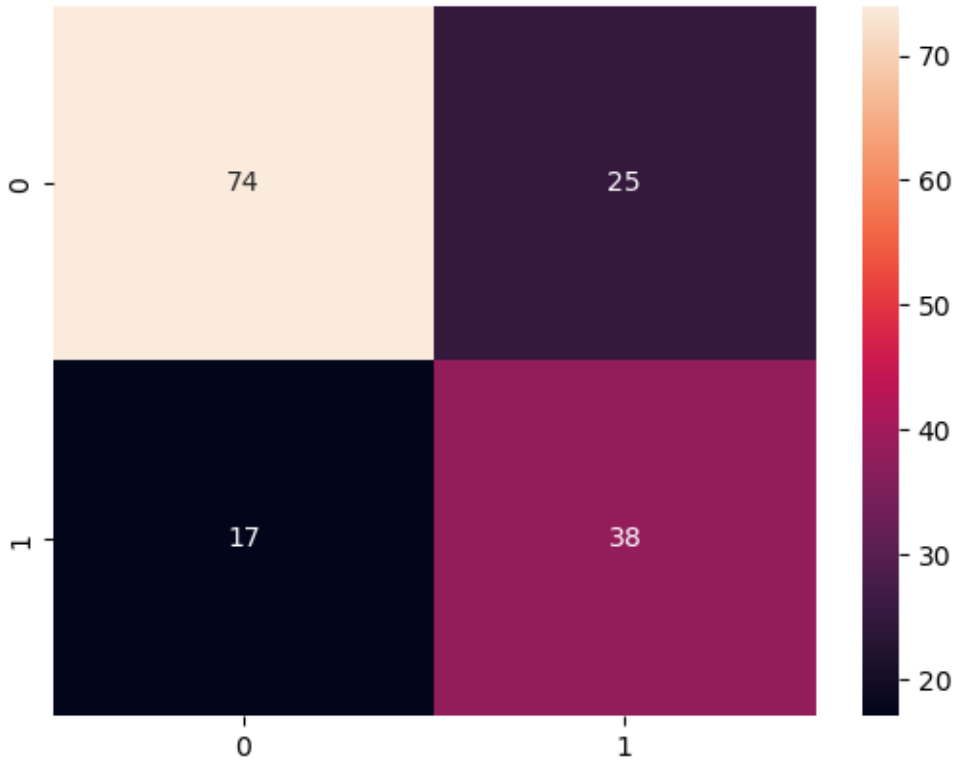
[84]:
```
#Training the model

decision_tree.fit(X_train,y_train)
```

[84]:
```
DecisionTreeClassifier()
```

[85]:
```
#Making prediction
y_pred=decision_tree.predict(X_test)
```

[86]:
```
#Creating confusion Matrix to understand model classification
cm_DT=confusion_matrix(y_test,y_pred)
```

[87]:
```
#visualizing confusion matrix

sns.heatmap(cm_DT, annot=True)
plt.title
plt.show()
```



74 is the True Negative score, the model predicted the True positive score as 38. The true positive prediction classification is relatively higher than the LogisticRegression model.

[88]:
```
#Accuracy score for Decision Tree Model

print('Acurracy Score=',accuracy_score(y_test, y_pred))
```

```
Acurracy Score= 0.7272727272727273
```

the model accurately predicted the outcome or class label for approximately 73.38% of the total instances in the dataset.

[89]:
```
#Print classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.75      0.78        99
           1       0.60      0.69      0.64        55

    accuracy                           0.73       154
   macro avg       0.71      0.72      0.71       154
weighted avg       0.74      0.73      0.73       154
```

These metrics collectively give insights into the model's ability to correctly classify instances into the two classes. The overall accuracy of the model is 0.73, indicating that it correctly predicts 73% of all instances in the dataset. For class 1, precision is 0.61, indicating that when the model predicts an instance as class 1, it is correct 61% of the time. For class 1, recall is 0.73, meaning the model correctly identifies 73% of all actual class 1 instances. The F1-score is 0.66, providing a balance between precision and recall.

7.0.9 APPLYING RANDOM FOREST ALGORITHM

[90]:
```
#Importing the necessary Library
from sklearn.ensemble import RandomForestClassifier

Random_Forest = RandomForestClassifier(criterion='gini',
                                       max_depth=7,
                                       n_estimators=200,
                                       random_state=5)
```

These hyperparameters collectively influence the Random Forest's ability to make accurate predictions. The choice of criterion, max_depth, and n_estimators affects the model's complexity, bias-variance trade-off, and generalization ability.

[91]:
```
#Training the model
Random_Forest.fit(X_train,y_train)
```

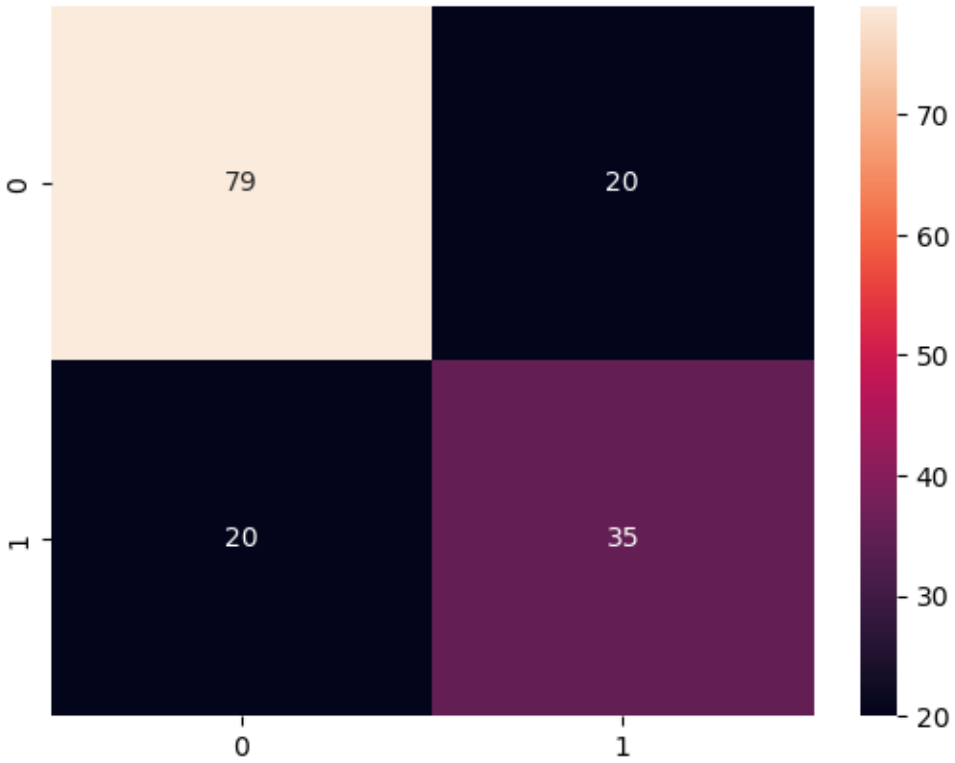
[91]:
```
RandomForestClassifier(max_depth=7, n_estimators=200, random_state=5)
```

[92]:
```
#Predicting Outcome
y_pred = Random_Forest.predict(X_test)
```

[93]:
```
#Creating confusion Matrix
RF_cm=confusion_matrix(y_test,y_pred)
RF_cm
```

[93]:
```
array([[79, 20],
       [20, 35]], dtype=int64)
```

[94]:
```
#Creatung a heatmap to visualize confusion matrix
sns.heatmap(RF_cm, annot=True)
plt.show()
```



TP score is 35 and TN score is 79.

[95]:
```
#Accuracy Score

print('Acurracy Score=',accuracy_score(y_test, y_pred))
```

```
Acurracy Score= 0.7402597402597403
```

[96]:
```
#Print classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.80      0.80        99
           1       0.64      0.64      0.64        55

    accuracy                           0.74       154
   macro avg       0.72      0.72      0.72       154
weighted avg       0.74      0.74      0.74       154
```

These metrics collectively give insights into the model's ability to correctly classify instances into the two classes. The true positive for zeros and ones.

7.0.10 SUPPORT VECTOR MACHINE MODEL

[97]:
```
#Importing important Library
from sklearn.svm import SVC

svc= SVC(probability=True)
```

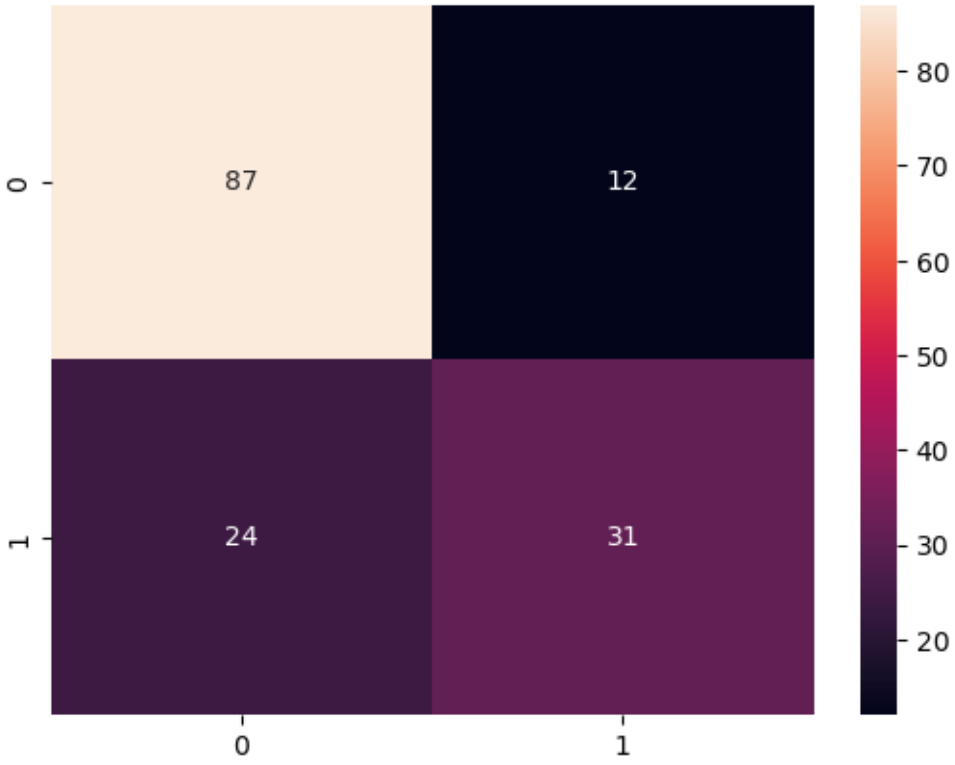
[98]:
```
#Training the model
svc.fit(X_train,y_train)
```

[98]:
```
SVC(probability=True)
```

[99]:
```
#Evaluating our model
y_pred = svc.predict(X_test)
```

[100]:
```
#Creating confusion Matrix
Svm_cm= confusion_matrix(y_test,y_pred)
```

[101]:
```
#Creatung a heatmap of the confusion matrix
sns.heatmap(Svm_cm, annot=True)
plt.show()
```



TPR is 31 and TNR is 87.

[102]:
```
#print accuracy
print('Acurracy Score=',accuracy_score(y_test, y_pred))
```

```
Acurracy Score= 0.7662337662337663
```

[103]:
```
#Print classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.88      0.83        99
           1       0.72      0.56      0.63        55

    accuracy                           0.77       154
   macro avg       0.75      0.72      0.73       154
weighted avg       0.76      0.77      0.76       154
```

The SVM accuracy score of 77% is relatively same as the accuracy score of Logistic Regression

7.0.11 ROC_AUC_SCORE FOR SVM

[111]:
```
from sklearn.metrics import roc_auc_score,roc_curve
class_probabilities = svc.predict_proba(X_test)
```

[112]:
```
#select the probability for positive outcome only for patients with diabetes
class_probabilities=class_probabilities[:,1]
```

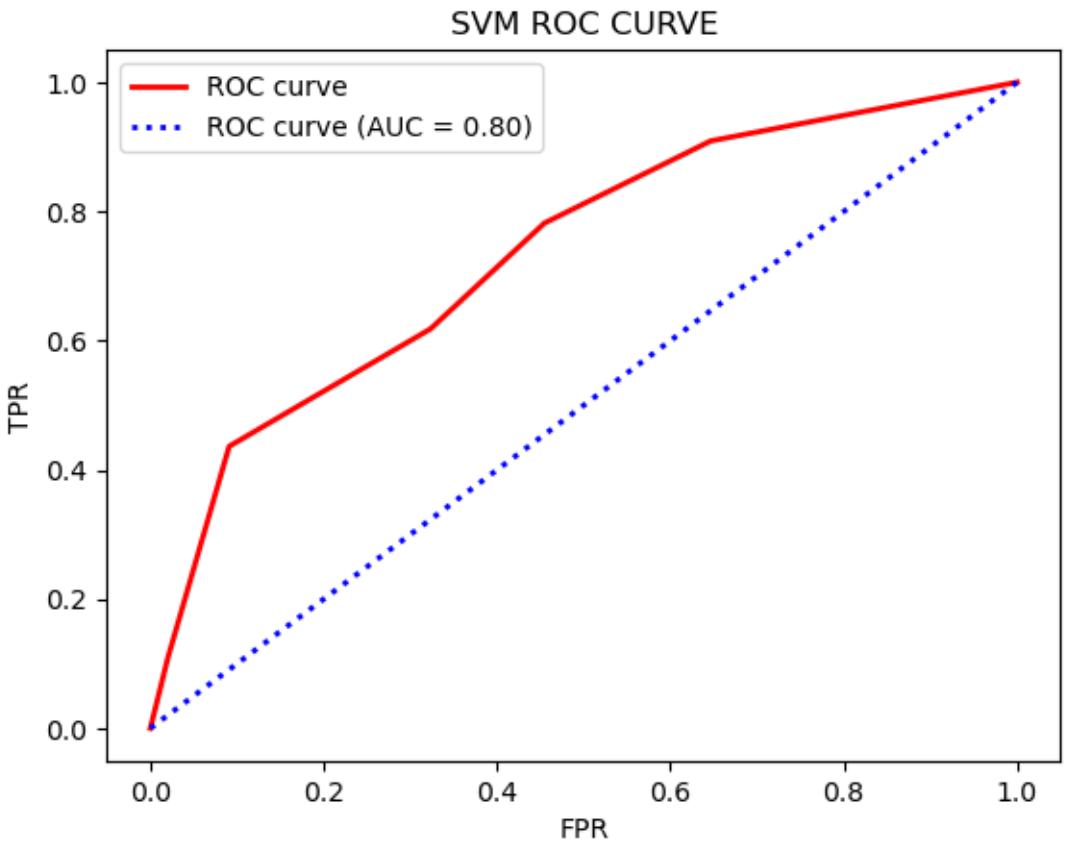
[113]:
```
#calculate roc_auc score

#auc= roc_auc_score(`y_test,class_probabilities)

#print('AUC Score is',auc)
```

[115]:
```
#calculate roc_auc score

svm_auc= roc_auc_score(y_test,class_probabilities)

print('AUC Score is',auc)
```

```
AUC Score is 0.8034894398530762
```

[116]:
```
#Compute the FPR and TPR

fpr,tpr,_=roc_curve(y_test, class_probabilities,␣
 ↪drop_intermediate=False)#drop_intermediate, implies the diff cut off␣
 ↪probabilities for which we gwt diff TPR and FPR
```

[150]:
```
#adding the ROC
plt.plot(fpr,tpr,color='r',lw=2,label='ROC curve')

#consider all possible values for random FPR and TPR
plt.plot([0,1],[0,1], color='b',lw=2,linestyle='dotted', label=f'ROC curve (AUC␣
  ↪= {svm_auc:.2f})')#Threshold



##title and label
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('SVM ROC CURVE')
plt.legend()

plt.show()
```



The ROC curve shows how the model's sensitivity and specificity were classifeid. This curve does not meet with the top-left corner, We can infer that our model evaluation of binary classification is 0.80, and is reasonably close to the perfect score of 1.

7.0.12 APPLYING KNN ALGORITHM

[131]:
```
#Importing the library
from sklearn.neighbors import KNeighborsClassifier

Knn_Classifier=KNeighborsClassifier(n_neighbors=5)
```

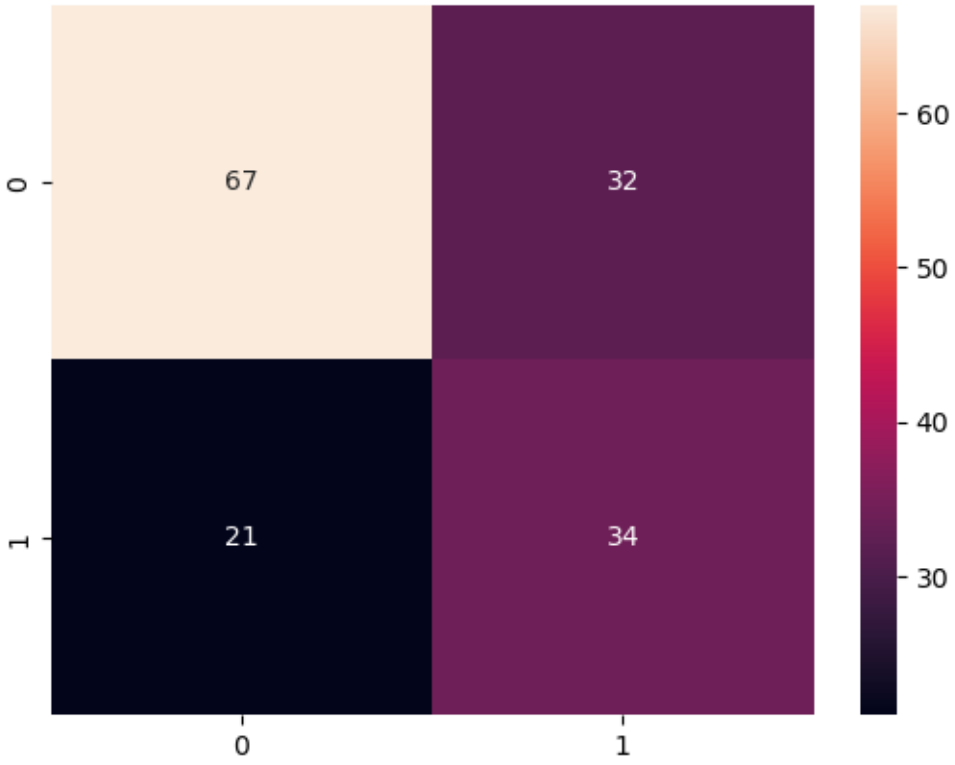
[132]:
```
#Model Training
Knn_Classifier.fit(X_train,y_train)
```

[132]:
```
KNeighborsClassifier()
```

[134]:
```
#Evaluatimg the model
y_pred = Knn_Classifier.predict(X_test)
```

[135]:
```
#Creating confusion Matrix
Knn_cm= confusion_matrix(y_test,y_pred)
```

[136]:
```
#Creatung a heatmap of the confusion matrix
sns.heatmap(Knn_cm, annot=True)
plt.show()
```



With a True Positive Rate (TPR) of 34%, a True Negative Rate (TNR) of 67%, a False Positive

Rate (FPR) of 32%, and a False Negative Rate (FNR) of 21%, it's evident that the KNN algorithm, when applied to your model, is experiencing a relatively high rate of misclassification

[137]:
```
#Model Evaluation
Knn_accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy:{Knn_accuracy}')
```

```
Accuracy:0.6558441558441559
```

[138]:
```
#Print classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.68      0.72        99
           1       0.52      0.62      0.56        55

    accuracy                           0.66       154
   macro avg       0.64      0.65      0.64       154
weighted avg       0.67      0.66      0.66       154
```

"Among the various algorithms applied to our model, including Logistic Regression, Random Forest Classifier, Support Vector Machine, and Decision Tree, the KNN algorithm exhibited limitations comparing the accuracy score of other algorithm.

7.0.13 ROC_AUC_SCORE OF KNN

[139]:
```
from sklearn.metrics import roc_auc_score,roc_curve
probabilities = Knn_Classifier.predict_proba(X_test)
```

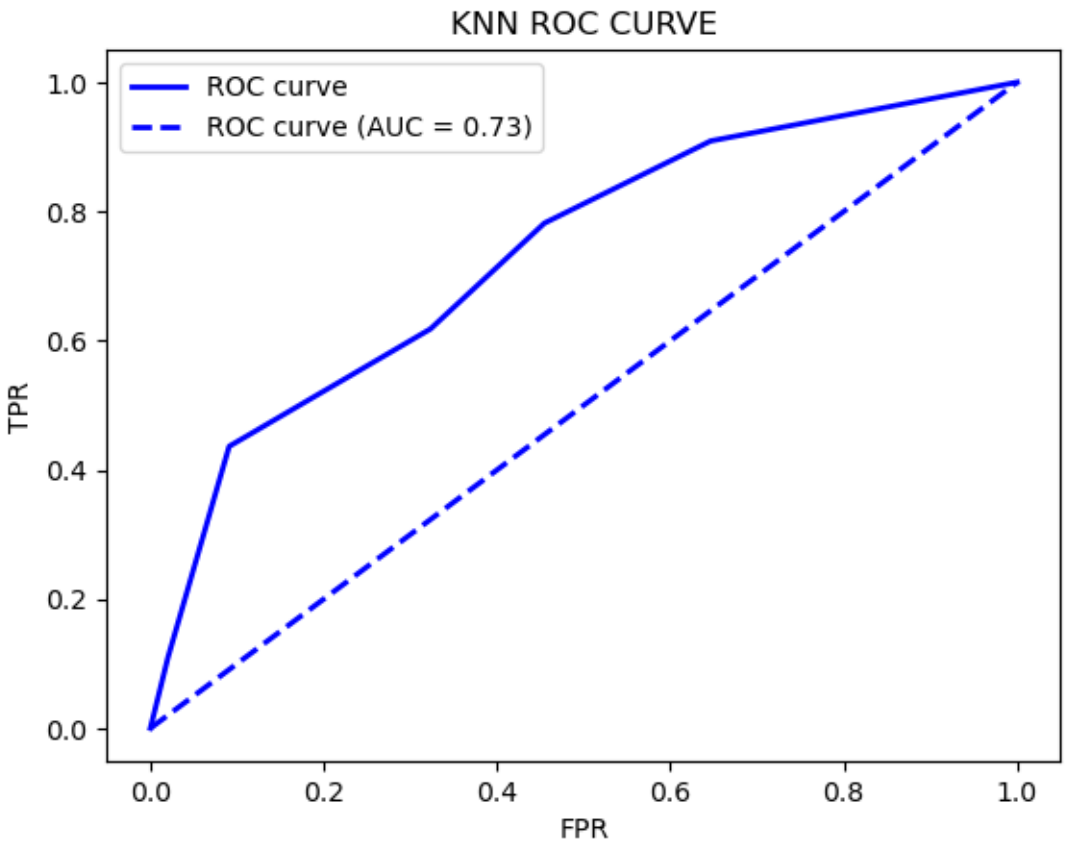
[140]:
```
#select the probability for positive outcome only for patients with diabetes
probabilities=probabilities[:,1]
```

[142]:
```
#calculate roc_auc score

Knn_auc= roc_auc_score(y_test,probabilities)

print('AUC Score is',Knn_auc)
```

```
AUC Score is 0.7345270890725437
```

[147]:
```
#Compute the FPR and TPR

fpr,tpr,_=roc_curve(y_test, probabilities,␣
 ↪drop_intermediate=False)#drop_intermediate, implies the diff cut off␣
 ↪probabilities for which we gwt diff TPR and FPR
```

[151]:
```
#Adding the ROC
plt.plot(fpr,tpr,color='b',lw=2,label='ROC curve')

# consider all possible values for random FPR and TPR
plt.plot([0,1],[0,1], color='b',lw=2,linestyle='--', label=f'ROC curve (AUC =␣
 ↪{Knn_auc:.2f})')#Threshold



#Title and label
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('KNN ROC CURVE')
plt.legend()

plt.show()
```



An AUC score of 0.73 suggests that the model has moderate discriminatory power. It performs better than random chance but may not be highly accurate in distinguishing between positive and negative instances.

[ ]:

[ ]: