#The dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases

```
# Summary of the Dataset
This dataset seems to contain information related to diabetes, with a
focus on various healthrelated
variables that could be associated with diabetes risk. The variables
include:
• Pregnancies: The number of pregnancies an individual has had.
• Glucose: Glucose levels in the blood.
• BloodPressure: Blood pressure readings.
• SkinThickness: Thickness of a skinfold at a certain location on the
body.
• Insulin: Levels of insulin in the blood.
• BMI (Body Mass Index): A measure of body fat based on height and
weight.
• DiabetesPedigreeFunction: A function that scores the likelihood of
diabetes based on
family history.
• Age: Age of the individuals.
• Outcome: A binary variable indicating the presence (1) or absence (0)
of a diabetes
outcome.
```

Objective:

The objective of the dataset is to diagnostically predict whether a patient has diabetes or not based on certain diagnostic measurements included in the dataset and analyze various approach to boost performance and accuracy.

In [1]:
```
#Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
#Load the dataset
diabetes = pd.read_csv('diabetes.csv')
```

In [3]:
```python
diabetes
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
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	
...	...	...	...	...	...	...	
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	

768 rows × 9 columns

#Dataset Outline: The dataset has total 768 observations and 8 feature columns and a targe variable'Outcome'.

In [4]:
```python
# checking the outline of the dataset
diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [5]:
```
#Checking for null values
diabetes.isnull().sum()
```

Out[5]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [6]:
```
#Checking the duplicate
diabetes.duplicated().sum()
```

Out[6]: 0

In [7]:
```
#Analyzing the summary of the dataset
diabetes.describe()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diab
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	

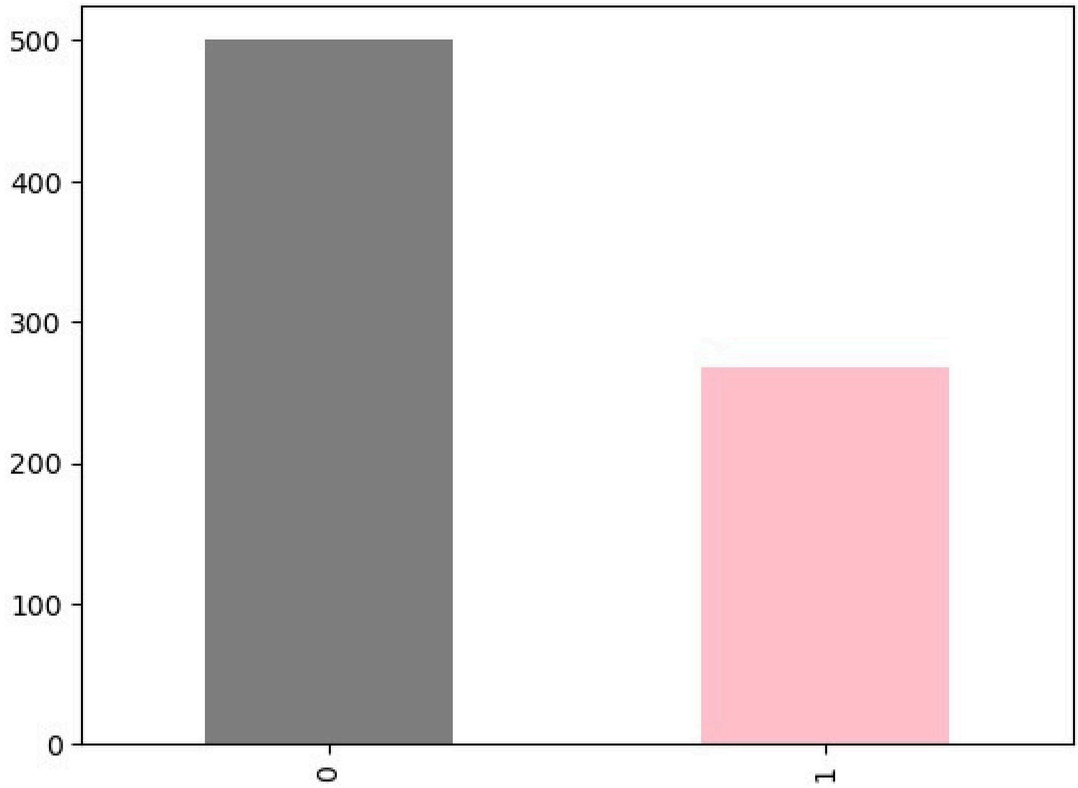
In [8]:
```
diabetes['Outcome'].value_counts()
```

Out[8]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

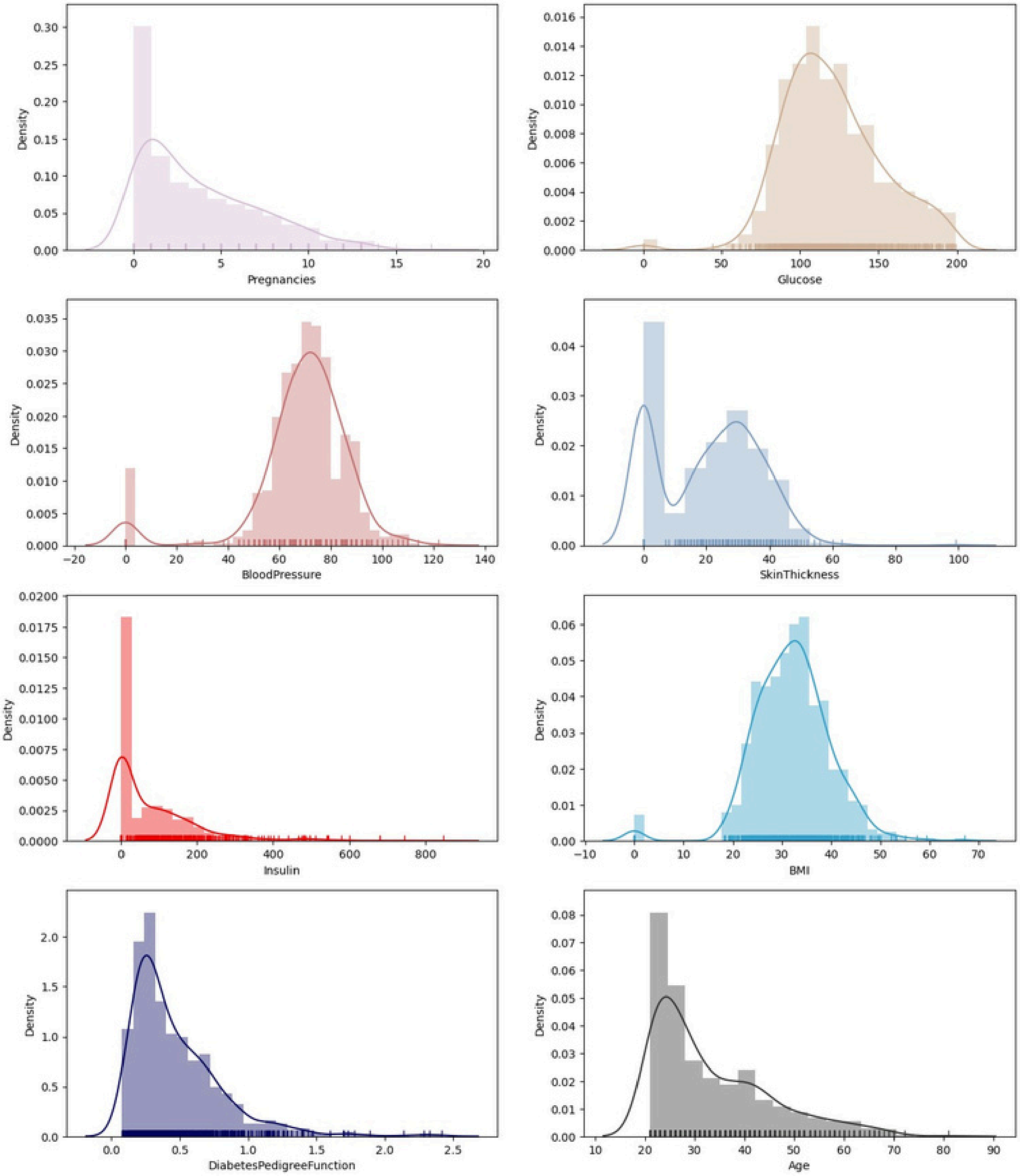
Exploratory Data Analysis

In [9]:
```
#Visualizing bar graph of the outcome
# 1 means diabetes patient and 0 means no diabetes
diabetes['Outcome'].value_counts().plot(kind='bar',color=['grey','pink'])
```

Out[9]: <AxesSubplot:>

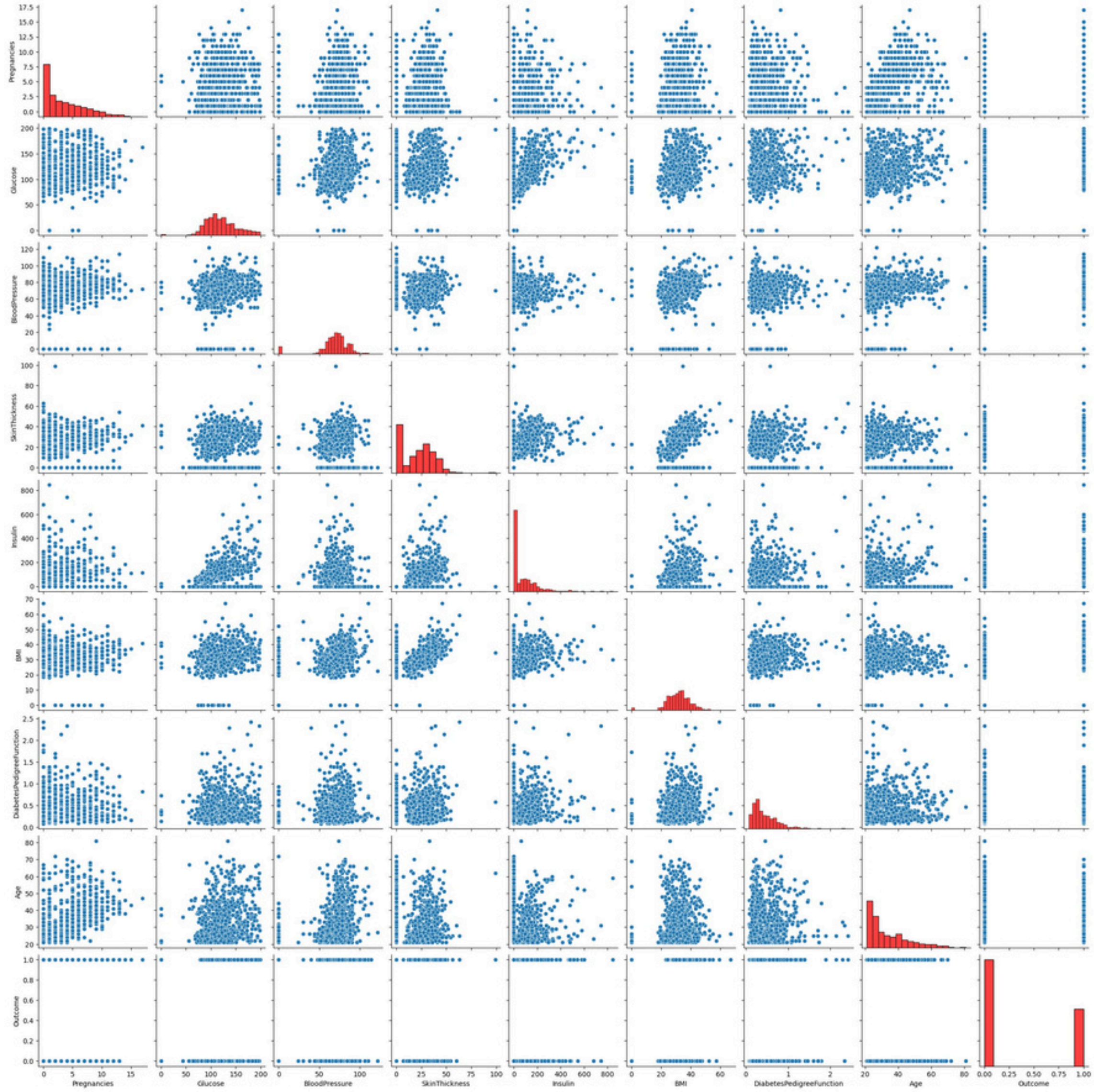


In [10]:
```
fig, axs = plt.subplots(4, 2, figsize=(15,18))
axs                                  =                                  axs.flatten()
sns.distplot(diabetes['Pregnancies'],rug=True,color='#D8BFD8',ax=axs[0])
sns.distplot(diabetes['Glucose'],rug=True,color='#CDAF95',ax=axs[1])
sns.distplot(diabetes['BloodPressure'],rug=True,color='#C67171',ax=axs[2])
sns.distplot(diabetes['SkinThickness'],rug=True,color='#7D9EC0',ax=axs[3])
sns.distplot(diabetes['Insulin'],rug=True,color='#EE0000',ax=axs[4])
sns.distplot(diabetes['BMI'],color='#33A1C9',rug=True,ax=axs[5])
sns.distplot(diabetes['DiabetesPedigreeFunction'],color='#03045e',rug=True,
sns.distplot(diabetes['Age'],rug=True,color='#333533',ax=axs[7]) plt.show()
```

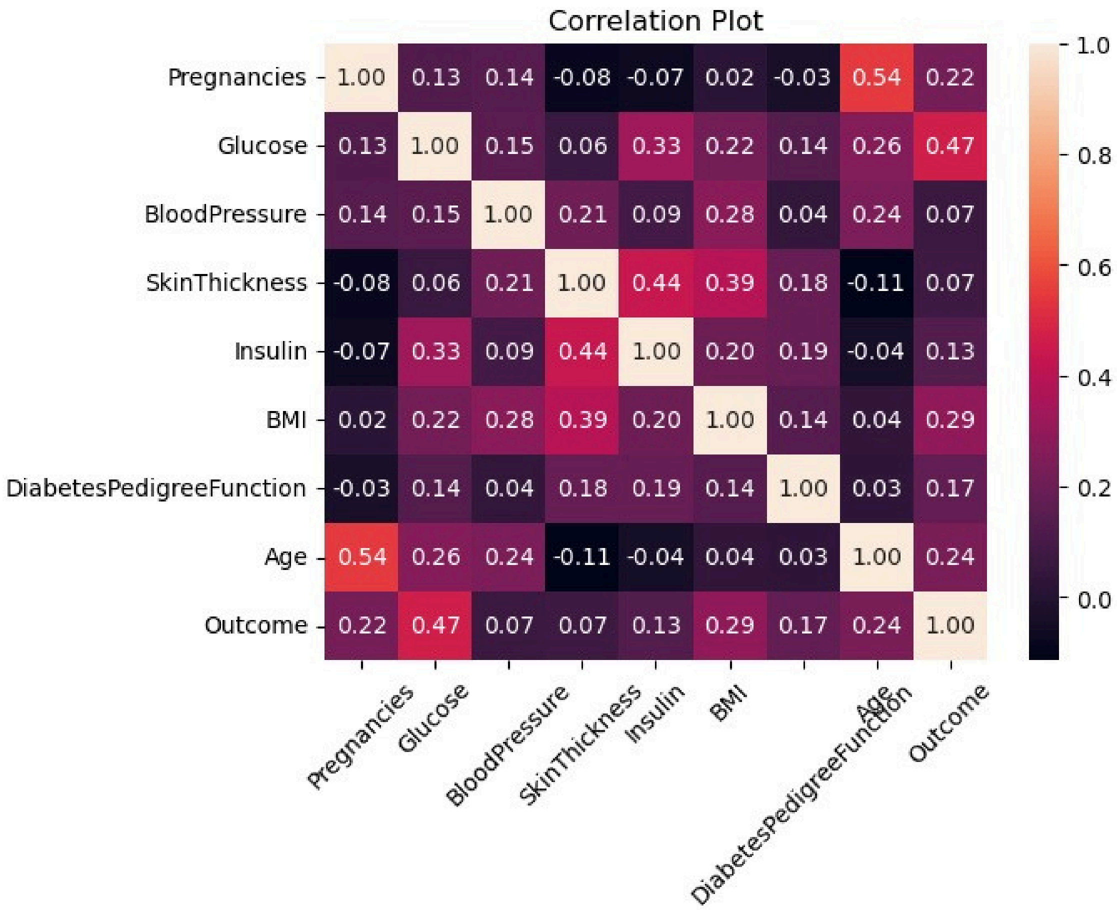


In [11]:
```
sns.pairplot(diabetes ,diag_kws={'color':'red'})
```

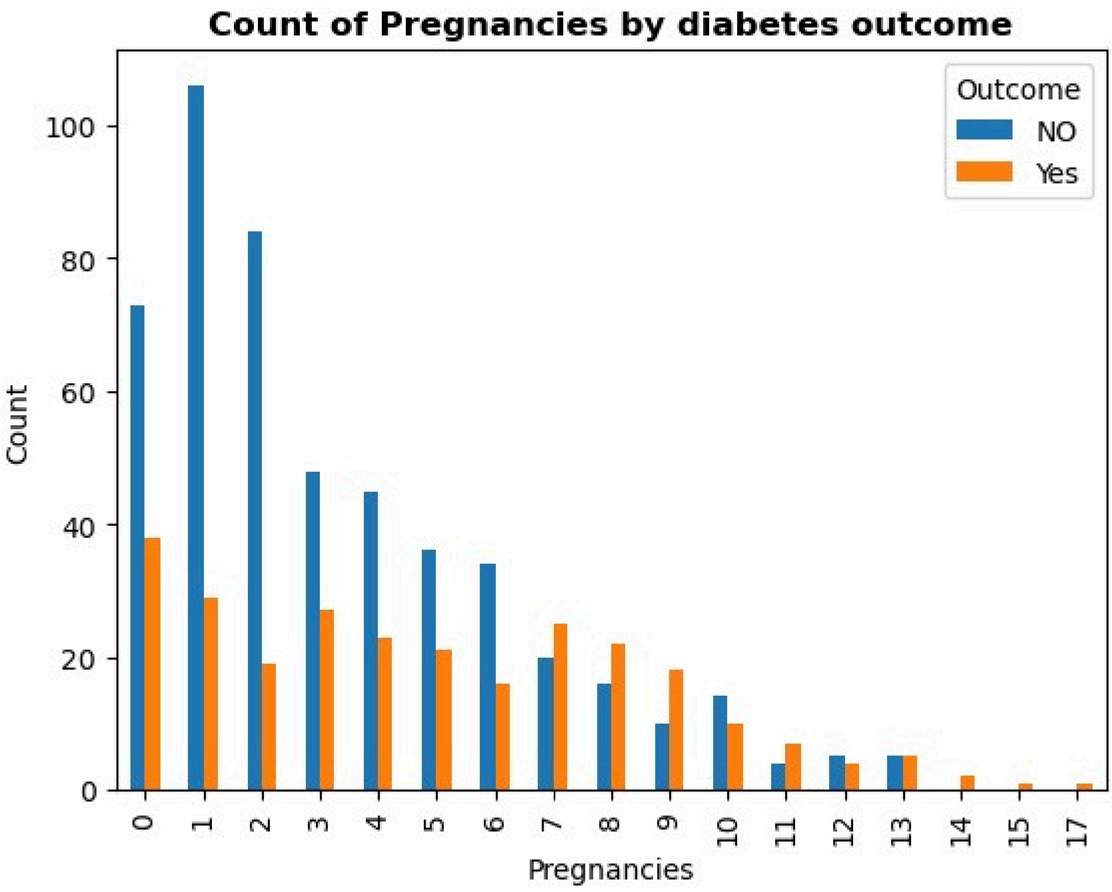
Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x230d995acd0>
```



```
In [12]: #Visualizing Heatmap
sns.heatmap(diabetes.corr(),annot=True,fmt='.2f')
plt.title('Correlation Plot')
plt.xticks(rotation=45)
plt.show()
```

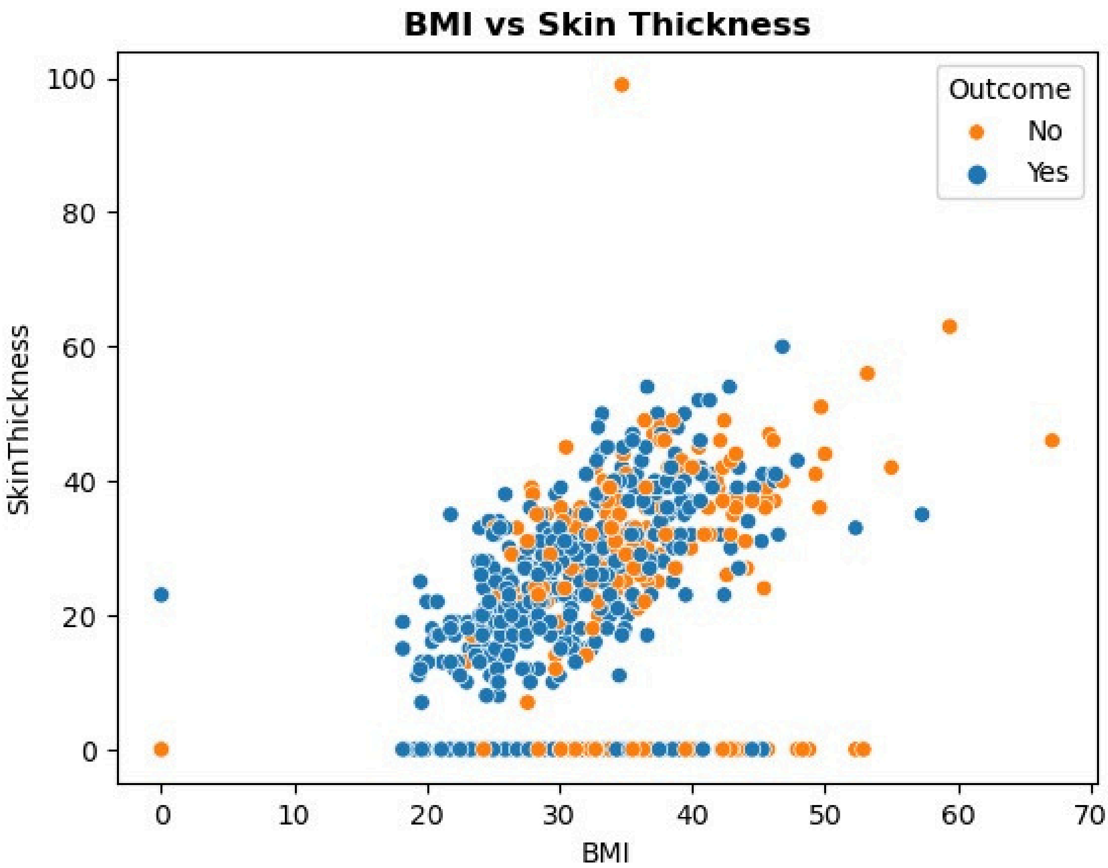


In [13]:
```
diabetes.groupby(['Pregnancies', 'Outcome']).size().unstack(level=1).plot(k
plt.ylabel('Count')
plt.legend(title ='Outcome', labels=['NO','Yes'])
plt.title('Count of Pregnancies by diabetes outcome', weight='bold')
plt.show()
```



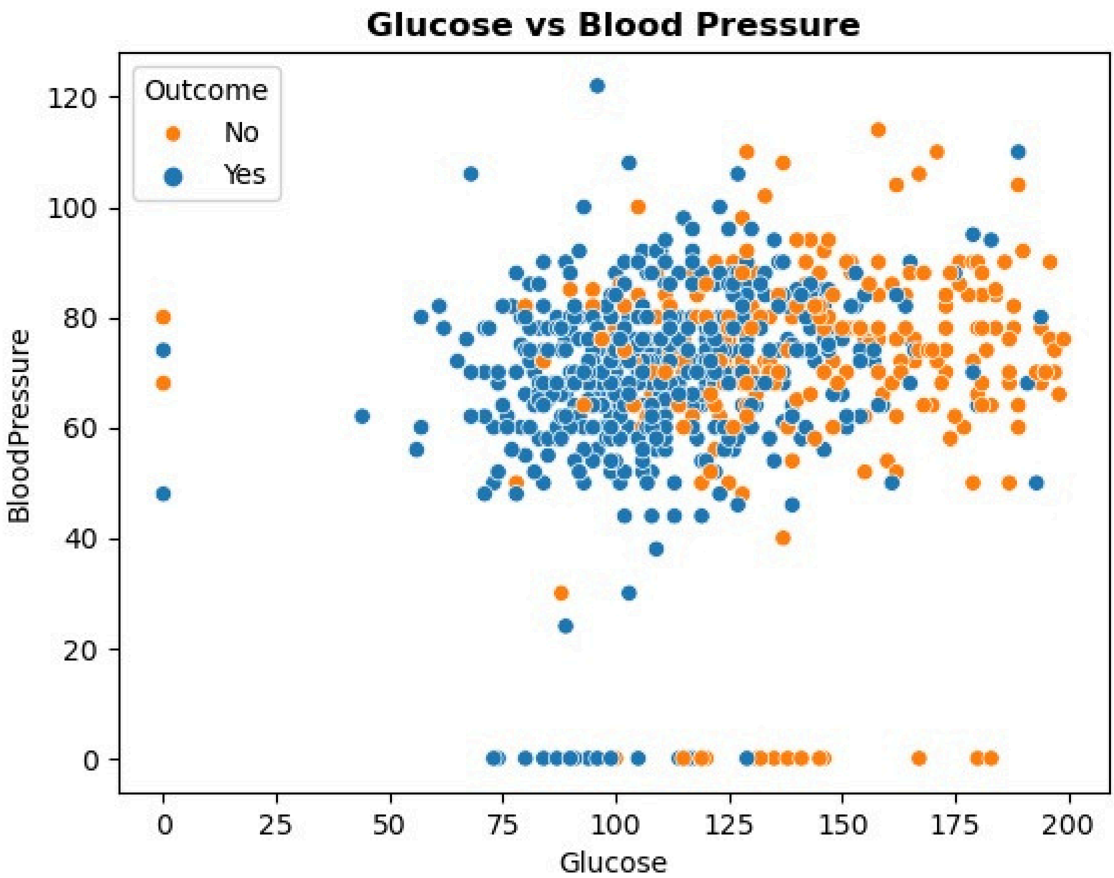
In [14]:
```
#Insight:
# The number of having diabetes is less when the number of pregnancies is l
# The possibility of having diabetes increases as the number of pregnancies
```

In [15]:
```
sns.scatterplot(data=diabetes, x='BMI', y='SkinThickness', hue='Outcome')
plt.legend(title='Outcome', labels=['No', 'Yes'])
plt.title('BMI vs Skin Thickness', weight='bold')
plt.show()
```



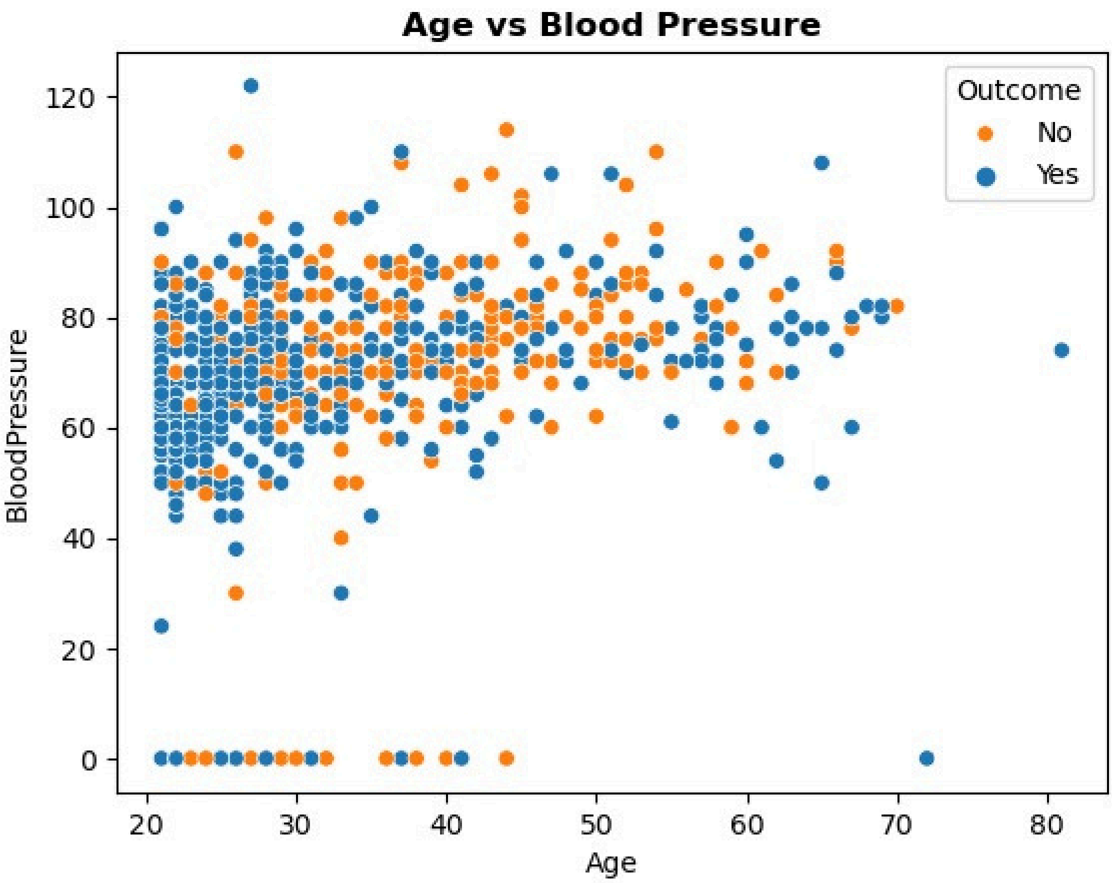
In [16]:
```
#Insights:
# From the scatterplot we can conclude that people with BMI<30 and skin thi
```

In [17]:
```
sns.scatterplot(data=diabetes,x='Glucose', y='BloodPressure', hue='Outcome'
plt.legend(title='Outcome', labels=['No', 'Yes'])
plt.title('Glucose vs Blood Pressure', weight='bold')
plt.show()
```



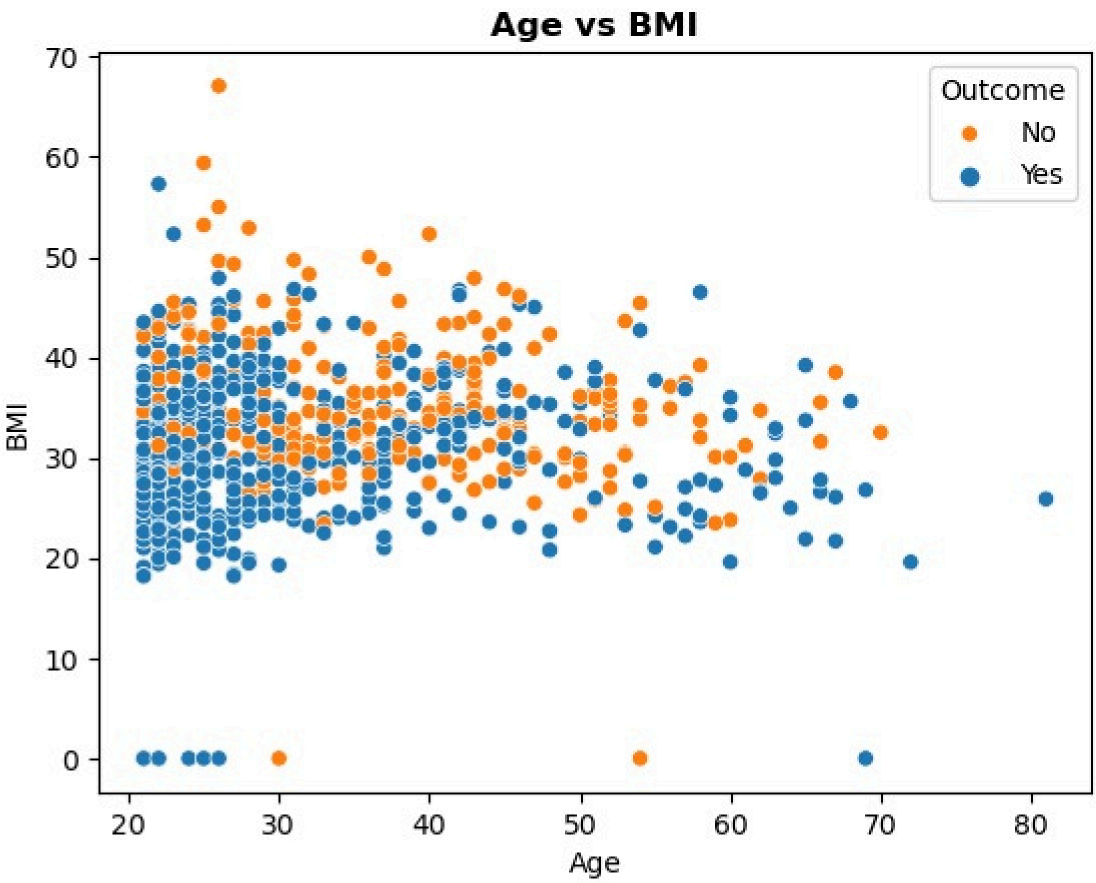
In [18]:
```
#Insights:
# From the scatterplot we can conclude that people with Glucose<100 and Blo
```

In [19]:
```
sns.scatterplot(data=diabetes,x='Age', y='BloodPressure', hue='Outcome')
plt.legend(title='Outcome', labels=['No', 'Yes'])
plt.title('Age vs Blood Pressure', weight='bold')
plt.show()
```



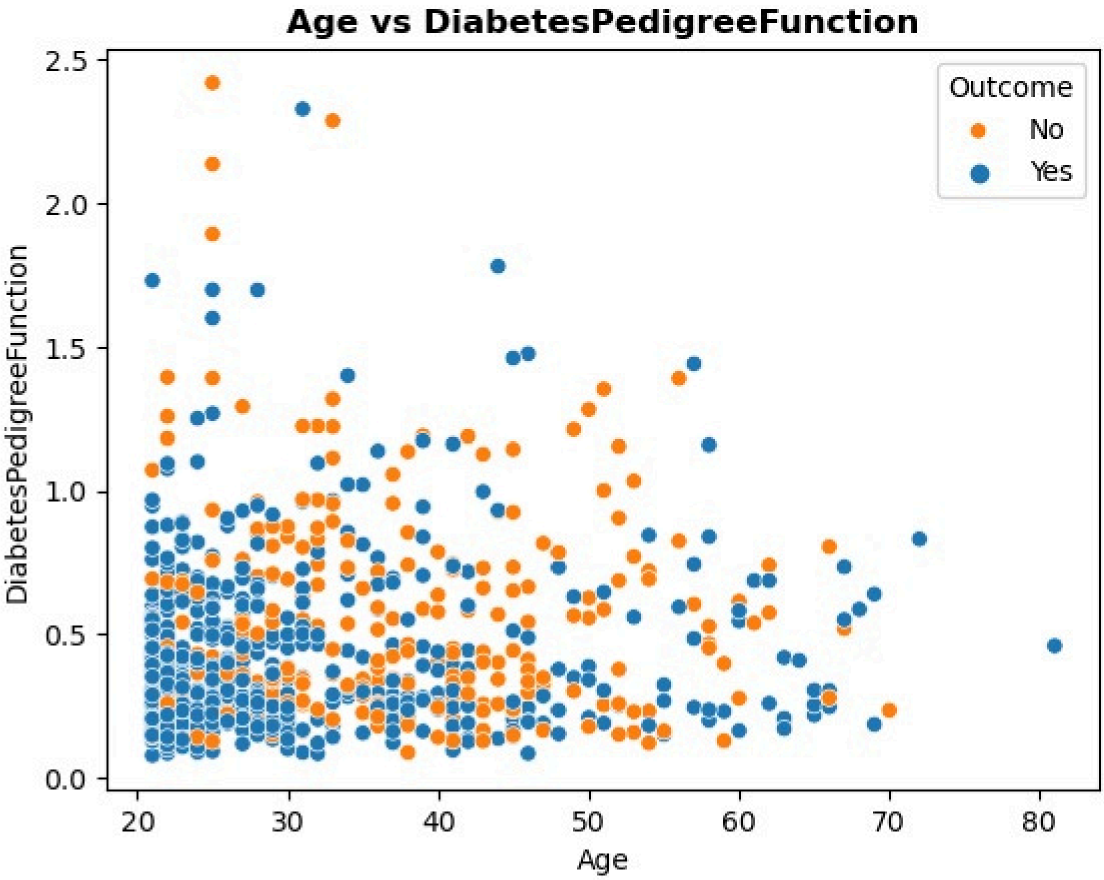
In [20]:
```
#Insights:
# From the scatterplot we can see age between 20 to 30 are more likely to b
```

In [21]:
```
sns.scatterplot(data=diabetes,x='Age', y='BMI', hue='Outcome')
plt.legend(title='Outcome', labels=['No', 'Yes'])
plt.title('Age vs BMI', weight='bold')
plt.show()
```



In [22]:
```
#Insights:
# From the scatterplot, we can see that people between 20 to 30 age and hav
```

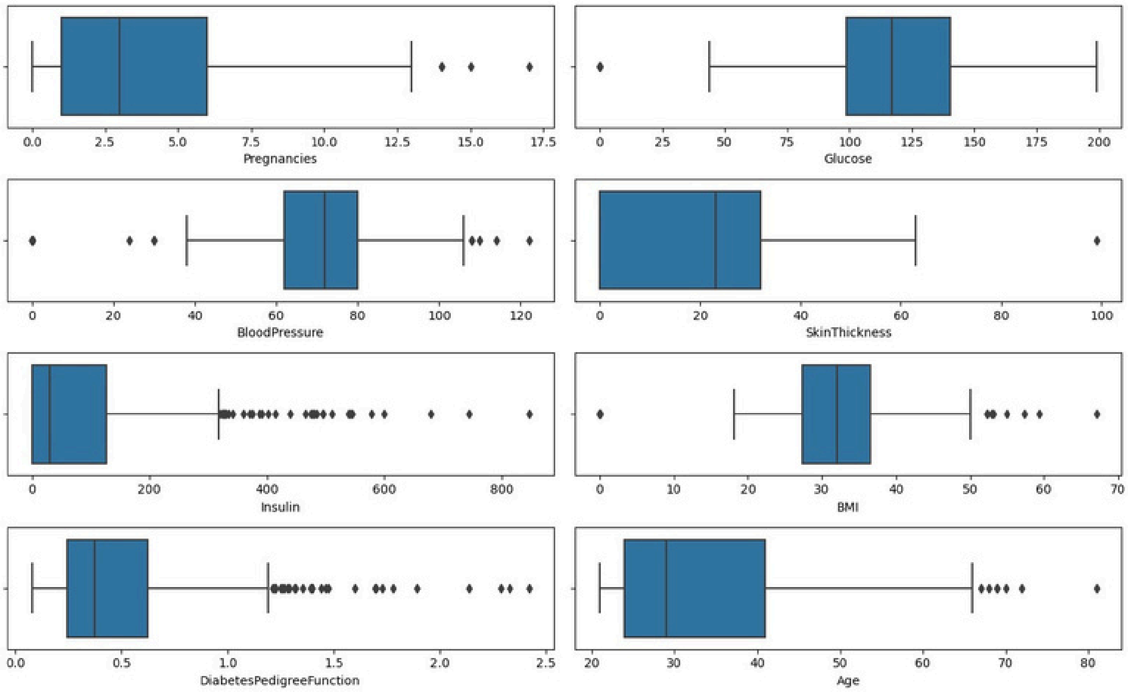
In [23]:
```
sns.scatterplot(data=diabetes,x='Age', y='DiabetesPedigreeFunction', hue='O
plt.legend(title='Outcome', labels=['No', 'Yes'])
plt.title('Age vs DiabetesPedigreeFunction', weight='bold')
plt.show()
```



In [24]:
```
# Insights:
# From the scatterplot we can see people between age 20 to 30 and having Di
```

In [25]:
```
#Checking the outliers
cols = np.array(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness'
fig, ax=plt.subplots(4,2,figsize=(13,8))

for row in range(4):
    for col in range(2):
        sns.boxplot(x=diabetes[cols[row,col]],ax=ax[row,col])
        plt.tight_layout()
plt.show()
```



In [26]:
```
#Removing the outliers from each features
```

In [27]:
```
diabetes["Pregnancies"] = diabetes["Pregnancies"].apply(lambda x: diabetes.
```

In [28]:
```
diabetes["BloodPressure"] = diabetes["BloodPressure"].apply(lambda x: diabe
diabetes["BloodPressure"] = diabetes["BloodPressure"].apply(lambda x: diabe
```

In [29]:
```
diabetes["Insulin"] = diabetes["Insulin"].apply(lambda x: diabetes.Insulin.
```

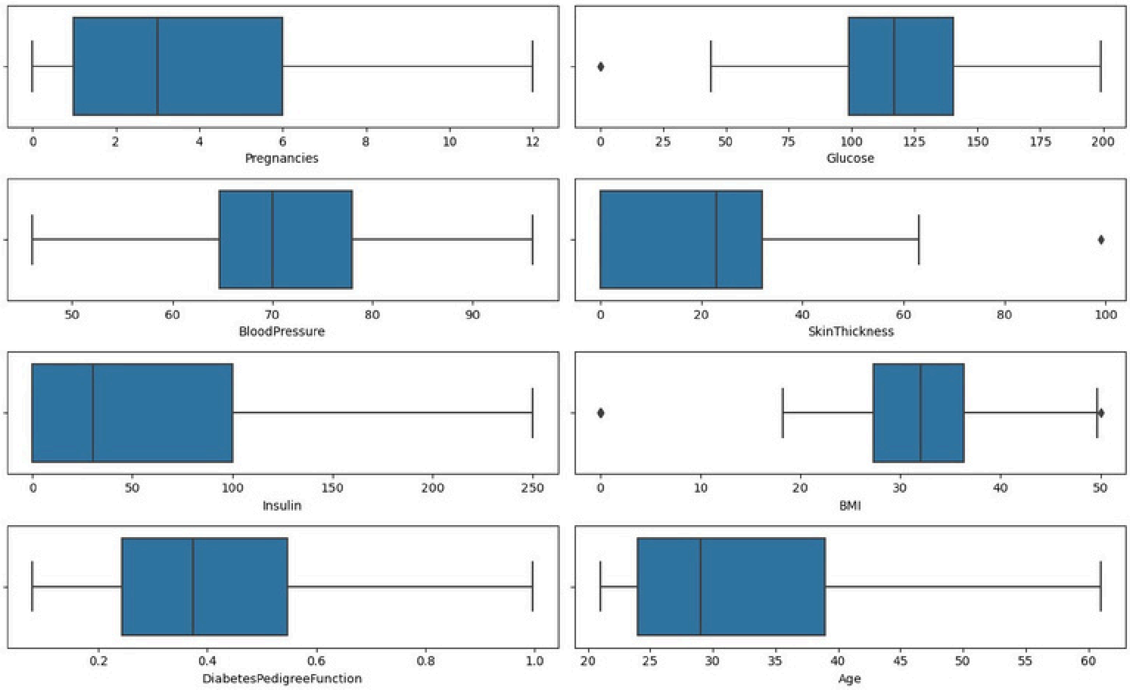
In [30]:
```
diabetes["BMI"] = diabetes["BMI"].apply(lambda x: diabetes.BMI.mean() if x>
```

In [31]:
```
diabetes["DiabetesPedigreeFunction"] = diabetes["DiabetesPedigreeFunction"]
```

In [32]:
```
diabetes["Age"] = diabetes["Age"].apply(lambda x: diabetes.Age.mean() if x>
```

In [33]:
```
#Cheking the features after removing outliers
cols = np.array(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness'
fig, ax=plt.subplots(4,2,figsize=(13,8))

for row in range(4):
    for col in range(2):
        sns.boxplot(x=diabetes[cols[row,col]],ax=ax[row,col])
        plt.tight_layout()
plt.show()
```



In [ ]:

In [34]:
```
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

In [35]:
```
#splitting the data into features and labels
X = diabetes.drop(["Outcome"], axis= "columns") # droping the label variabl
y = diabetes["Outcome"]
```

In [36]:
```
X.head()
```

Out[36]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6.0	148	72.000000	35	0.0	33.6	0.627
1	1.0	85	66.000000	29	0.0	26.6	0.351
2	8.0	183	64.000000	0	0.0	23.3	0.672
3	1.0	89	66.000000	23	94.0	28.1	0.167
4	0.0	137	68.205734	35	168.0	43.1	0.471

In [37]:
```
y.head()
```

Out[37]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

In [ ]:

In [38]:
```
#Splitting the dataset into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

Model Building

Logistic Regression

In [39]:
```
#Importing logistic regression from Sklearn module
from sklearn.linear_model import LogisticRegression
```

In [40]:
```
LR= LogisticRegression()
```

In [41]:
```
from sklearn.metrics import confusion_matrix, classification_report, accura
```

In [42]:
```
LR.fit(X_train, y_train)
```

Out[42]: LogisticRegression()

In [43]:
```
y_pred = LR.predict(X_test)
```

In [44]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7662337662337663
```

In [45]:
```
CM = confusion_matrix(y_test,y_pred)
print(CM)
```

```
[[90  9]
 [27 28]]
```

In [46]:
```
class_report = classification_report(y_test, y_pred)
print(class_report)
```

```
              precision    recall  f1-score   support

           0       0.77      0.91      0.83        99
           1       0.76      0.51      0.61        55

    accuracy                           0.77       154
   macro avg       0.76      0.71      0.72       154
weighted avg       0.76      0.77      0.75       154
```

Decision Tree

In [47]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [48]:
```
DT =DecisionTreeClassifier()
```

In [49]:
```
DT.fit(X_train, y_train)
```

Out[49]:
```
DecisionTreeClassifier()
```

In [50]:
```
y_pred = DT.predict(X_test)
```

In [51]:
```
DT.score(X_test, y_test)
```

Out[51]:
```
0.6883116883116883
```

In [52]:
```
accuracy_DT = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_DT)
```

```
Accuracy: 0.6883116883116883
```

In [53]:
```
CM_DT = confusion_matrix(y_test,y_pred)
print(CM_DT)
```

```
[[74 25]
 [23 32]]
```

In [54]:
```
class_report = classification_report(y_test, y_pred)
print(class_report)
```

```
              precision    recall  f1-score   support

           0       0.76      0.75      0.76        99
           1       0.56      0.58      0.57        55

    accuracy                           0.69       154
   macro avg       0.66      0.66      0.66       154
weighted avg       0.69      0.69      0.69       154
```

Random Forest Classifier

In [55]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [56]:
```
random_forest = RandomForestClassifier(n_estimators=10)
```

In [57]:
```
random_forest.fit(X_train, y_train)
```

Out[57]:
```
RandomForestClassifier(n_estimators=10)
```

In [58]:
```
y_pred = random_forest.predict(X_test)
```

In [59]:
```
random_forest.score(X_test, y_test)
```

Out[59]:
```
0.7142857142857143
```

In [60]:
```
accuracy_RF = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_RF)
```

```
Accuracy: 0.7142857142857143
```

In [61]:
```
CM_RF = confusion_matrix(y_test,y_pred)
print(CM_RF)
```

```
[[87 12]
 [32 23]]
```

In [62]:
```
class_report = classification_report(y_test, y_pred)
print(class_report)
```

```
              precision    recall  f1-score   support

           0       0.73      0.88      0.80        99
           1       0.66      0.42      0.51        55

    accuracy                           0.71       154
   macro avg       0.69      0.65      0.65       154
weighted avg       0.70      0.71      0.70       154
```

Gaussian Naive Bayes Classifier

In [64]:
```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
```

In [65]:
```
naive_bayes.fit(X_train, y_train)
```

Out[65]:
```
GaussianNB()
```

In [66]:
```
y_pred_NB = naive_bayes.predict(X_test)
```

In [67]:
```
naive_bayes.score(X_test, y_test)
```

Out[67]:
```
0.7467532467532467
```

In [68]:
```
accuracy_NB = accuracy_score(y_test, y_pred_NB)
print("Accuracy:", accuracy_NB)
```

```
Accuracy: 0.7467532467532467
```

In [69]:
```
CM_NB = confusion_matrix(y_test,y_pred_NB)
print(CM_NB)
```

```
[[86 13]
 [26 29]]
```

In [70]:
```
class_report = classification_report(y_test, y_pred_NB)
print(class_report)
```

```
              precision    recall  f1-score   support

           0       0.77      0.87      0.82        99
           1       0.69      0.53      0.60        55

    accuracy                           0.75       154
   macro avg       0.73      0.70      0.71       154
weighted avg       0.74      0.75      0.74       154
```

Interpretation and summary report

• Imported the necessary modules for the project and checked the outline of the data • Performed exploratory analysis to visualize the distribution of the different features. • Performed preprocessing steps to treat the missing values and Outliers. • Used 4 Models (Logistic Regression, Random Forest, Decision Tree and Naive Bayes classifier) to find the best model for predicting the Diabetes Outcome. • Evaluated the performance using Accuracy,Precision,Recall and F1 score. • Based on the performance evaluation, Logistic Regression performed well in predicting if someone has diabetes or not. • Logistic Regression has achieved the highest accuracy of 76%. This model also exhibited reasonable precision and recall, indicating its ability to correctly classify both positive and negative cases of diabetes. So, the Machine learning approach, specifically the Logistic regression, can be a valuable tool for predicting diabetes outcomes based on health- related variables.