Name: Doha Saad Dajam

ID : 445816588

Diabetes Prediction

What is Diabetes?

Diabetes is a chronic disease that occurs when the pancreas is no longer able to make insulin, or when the body cannot make good use of the insulin it produces.

About this project :-

- The objective of this project is to classify whether someone has diabetes or not.
- Dataset consists of several Medical Variables(Independent) and one Outcome Variable(Dependent)
- The independent variables in this data set are :-'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age'
- The outcome variable value is either 1 or 0 indicating whether a person has diabetes(1) or not(0).

About the Dataset

- Pregnancies :- Number of times a woman has been pregnant
- Glucose :- Plasma Glucose concentration of 2 hours in an oral glucose tolerance test
- BloodPressure :- Diastollic Blood Pressure (mm hg)
- SkinThickness :- Triceps skin fold thickness(mm)
- Insulin :- 2 hour serum insulin(mu U/ml)
- BMI :- Body Mass Index ((weight in kg/height in m)^2)
- Age :- Age(years)
- DiabetesPedigreeFunction :-scores likelihood of diabetes based on family history)
- Outcome :- 0(doesn't have diabetes) or 1 (has diabetes)

Research Problem :-

Understanding the underlying factors and mechanisms that contribute to the development and progression of diabetes, specifically focusing on the impaired insulin production or utilization by the pancreas and the body, in order to develop effective prevention strategies and treatment approaches for individuals affected by diabetes.

1. Import Required Libraries

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt #to plot charts
import seaborn as sns #used for data visualization
import warnings #avoid warning flash
warnings.filterwarnings('ignore')
```

2. Loading the dataset

In [5]:
```
df=pd.read_csv("diabete.csv")
```

3. Exploratory Data Analysis

a. Understanding the dataset

- Head of the dataset
- Shape of the data set
- Types of columns
- Information about data set
- Summary of the data set

In [3]:
```
df.head() #get familier with dataset, display the top 5 data records
```

Out[3]:

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

In [4]:
```
df.shape #getting to know about rows and columns we're dealing with - 768 rows
```

Out[4]:
```
(768, 9)
```

```
In [5]: df.columns #learning about the columns
```

```
Out[5]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
              dtype='object')
```

```
In [6]: df.dtypes #knowledge of data type helps for computation
```

```
Out[6]: Pregnancies                   int64
        Glucose                       int64
        BloodPressure                 int64
        SkinThickness                 int64
        Insulin                       int64
        BMI                         float64
        DiabetesPedigreeFunction    float64
        Age                           int64
        Outcome                       int64
        dtype: object
```

```
In [7]: df.info() #Print a concise summary of a DataFrame. This method prints informati
```

```
<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 [8]:
```
df.describe() #helps us to understand how data has been spread across the table
# count :- the number of NoN-empty rows in a feature.
# mean :- mean value of that feature.
# std :- Standard Deviation Value of that feature.
# min :- minimum value of that feature.
# max :- maximum value of that feature.
# 25%, 50%, and 75% are the percentile/quartile of each features.
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesI
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	

b. Data Cleaning

- Dropping duplicate values
- Checking NULL values
- Checking for 0 value and replacing it :- It isn't medically possible for some data record to have 0 value such as Blood Pressure or Glucose levels. Hence we replace them with the mean value of that particular column.

In [9]:
```
df=df.drop_duplicates()
```

In [10]:
```
#check for missing values, count them and print the sum for every column
df.isnull().sum()
```

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

In [11]:
```
#checking for 0 values in 5 columns , Age & DiabetesPedigreeFunction do not hav
print(df[df['BloodPressure']==0].shape[0])
print(df[df['Glucose']==0].shape[0])
print(df[df['SkinThickness']==0].shape[0])
print(df[df['Insulin']==0].shape[0])
print(df[df['BMI']==0].shape[0])
```

```
35
5
227
374
11
```

outliers :-

Some of the columns have a skewed distribution, so the mean is more affected by outliers than the median. Glucose and Blood Pressure have normal distributions hence we replace 0 values in those columns by mean value. SkinThickness, Insulin,BMI have skewed distributions hence median is a better choice as it is less affected by outliers.

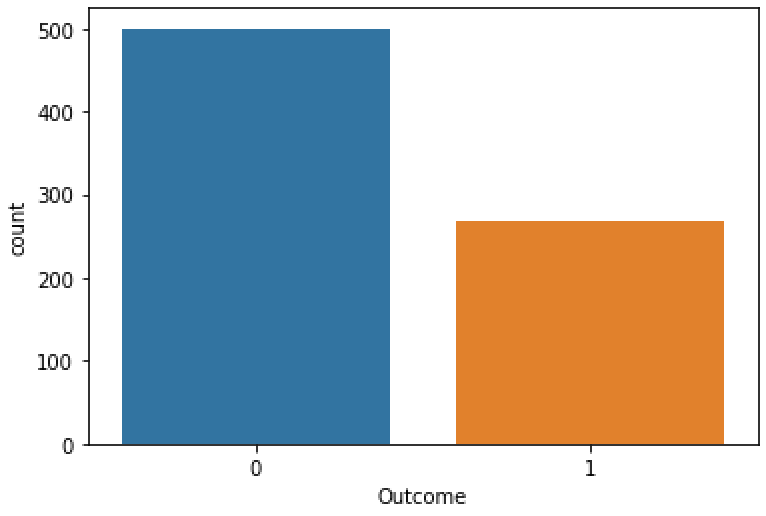
In [12]:
```
#replacing 0 values with median of that column
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())#normal distributior
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())#r
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].median())
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].median())#skewed distributi
df['BMI']=df['BMI'].replace(0,df['BMI'].median())#skewed distribution
```

4. Data Visualization

- Count Plot :- to see if the dataset is balanced or not
- Histograms :- to see if data is normally distributed or skewed
- Box Plot :- to analyse the distribution and see the outliers
- Scatter plots :- to understand relationship between any two variables
- Pair plot :- to create scatter plot between all the variables

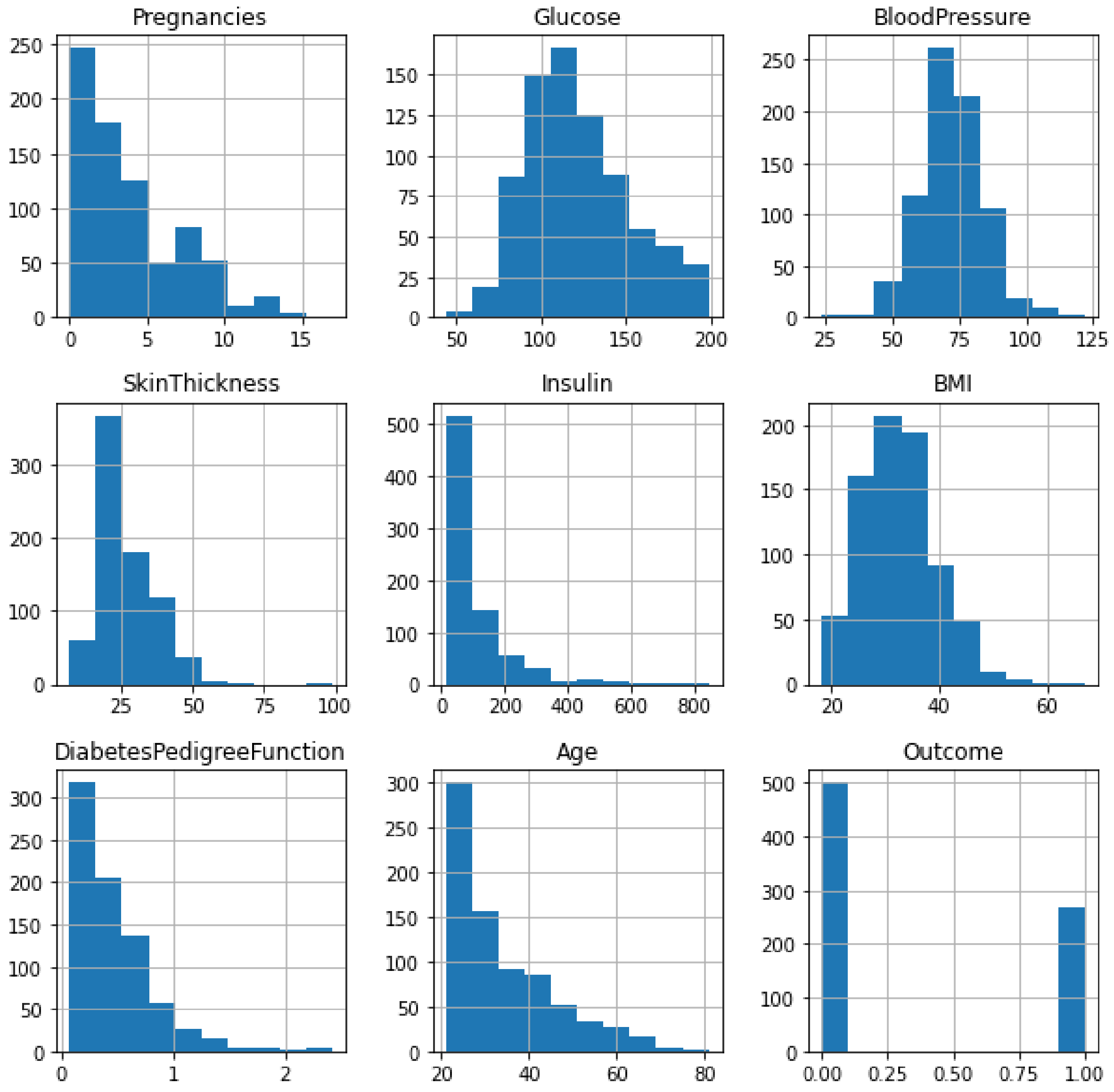
```
In [13]: sns.countplot('Outcome',data=df)
```

```
Out[13]: <AxesSubplot:xlabel='Outcome', ylabel='count'>
```



We observe that number of people who do not have diabetes is far more than people who do which indicates that our data is imbalanced.

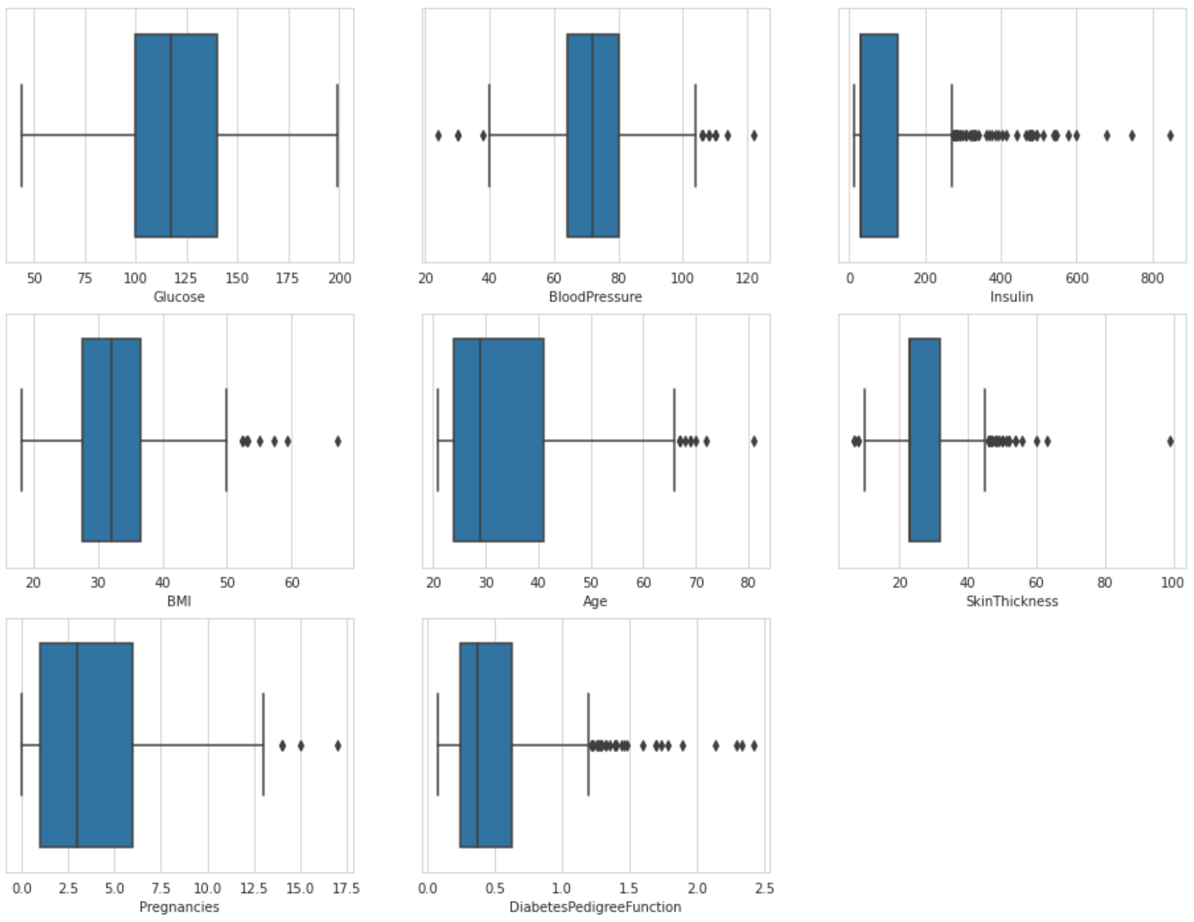
```
In [14]: #histogram for each  feature
         df.hist(bins=10,figsize=(10,10))
         plt.show()
```



We observe that only glucose and Blood Pressure are normally distributed rest others are skewed and have outliers

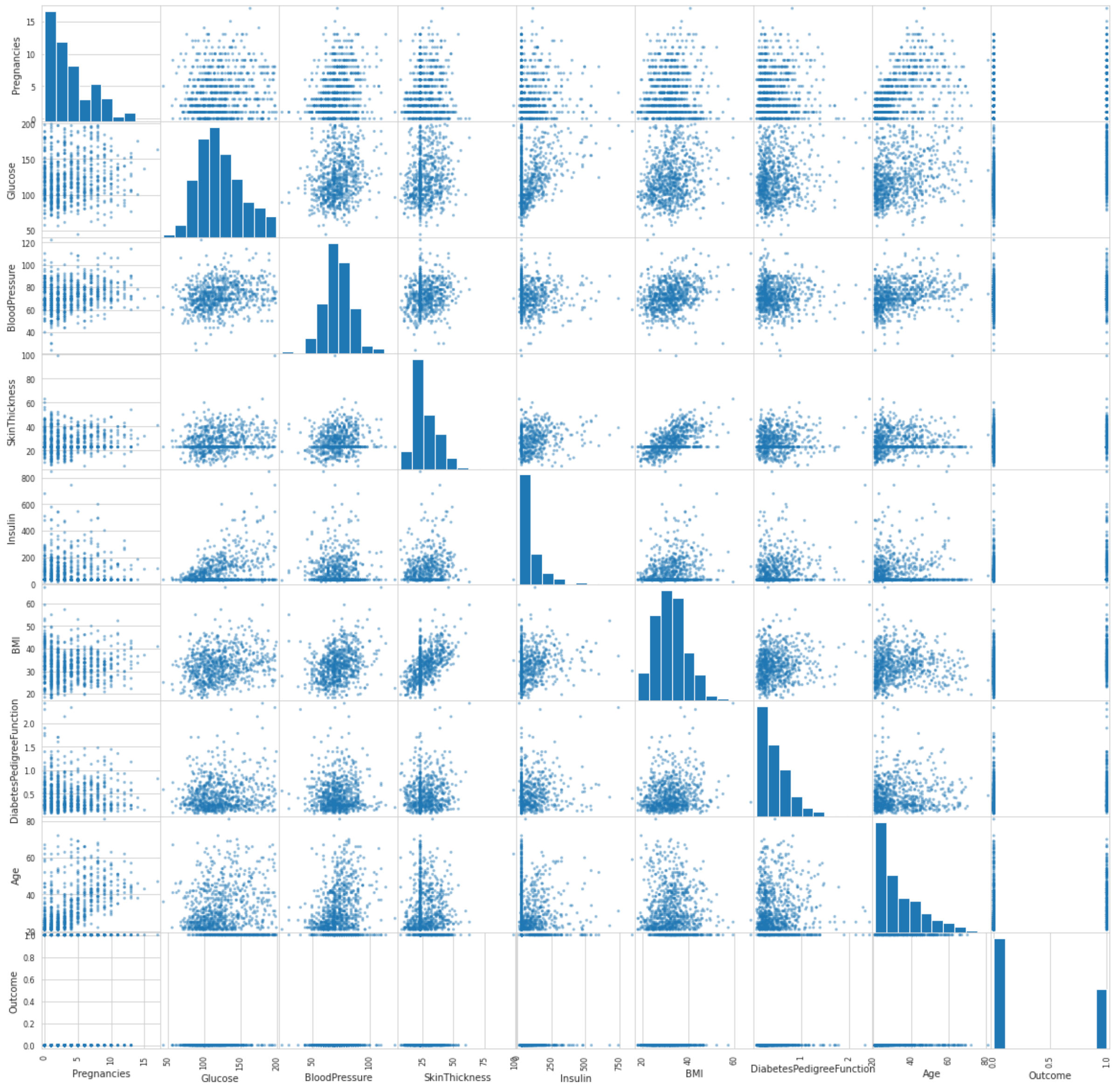
In [15]:
```
plt.figure(figsize=(16,12))
sns.set_style(style='whitegrid')
plt.subplot(3,3,1)
sns.boxplot(x='Glucose',data=df)
plt.subplot(3,3,2)
sns.boxplot(x='BloodPressure',data=df)
plt.subplot(3,3,3)
sns.boxplot(x='Insulin',data=df)
plt.subplot(3,3,4)
sns.boxplot(x='BMI',data=df)
plt.subplot(3,3,5)
sns.boxplot(x='Age',data=df)
plt.subplot(3,3,6)
sns.boxplot(x='SkinThickness',data=df)
plt.subplot(3,3,7)
sns.boxplot(x='Pregnancies',data=df)
plt.subplot(3,3,8)
sns.boxplot(x='DiabetesPedigreeFunction',data=df)
```

Out[15]:
```
<AxesSubplot:xlabel='DiabetesPedigreeFunction'>
```



Outliers are unusual values in your dataset, and they can distort statistical analyses and violate their assumptions. Hence it is of utmost importance to deal with them. In this case removing outliers can cause data loss so we have to deal with it using various scaling and transformation techniques.

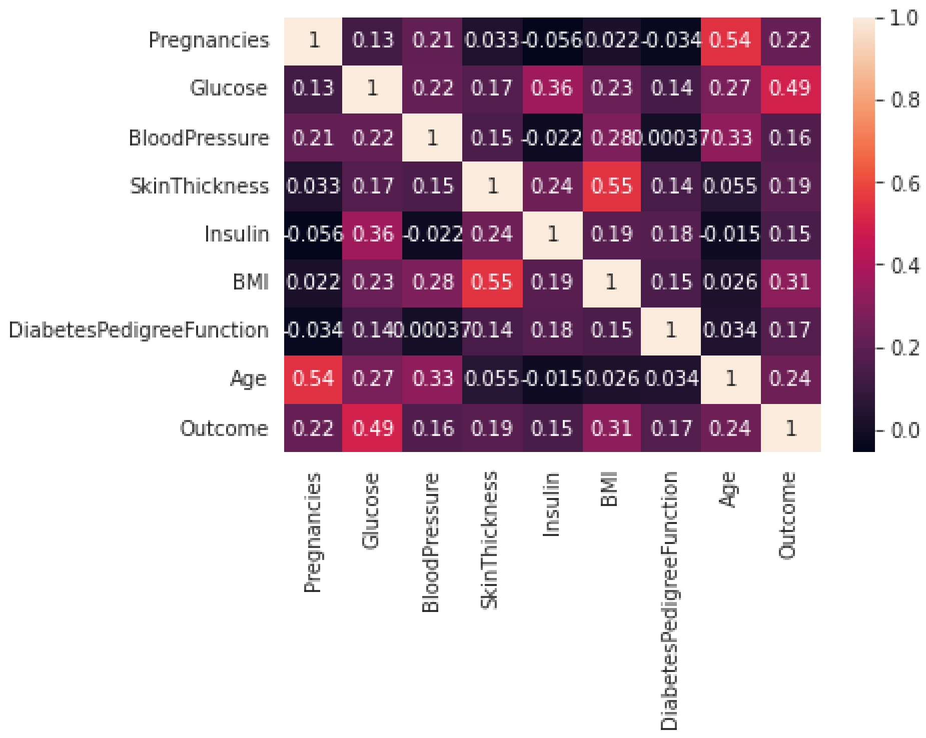
In [16]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(df,figsize=(20,20));
```



5. Feature Selection

In [17]:
```python
corrmat=df.corr()
sns.heatmap(corrmat, annot=True)
```

Out[17]:
```
<AxesSubplot:>
```



In [18]:
```python
df_selected=df.drop(['BloodPressure','Insulin','DiabetesPedigreeFunction'],axis
```

6. Handling Outliers

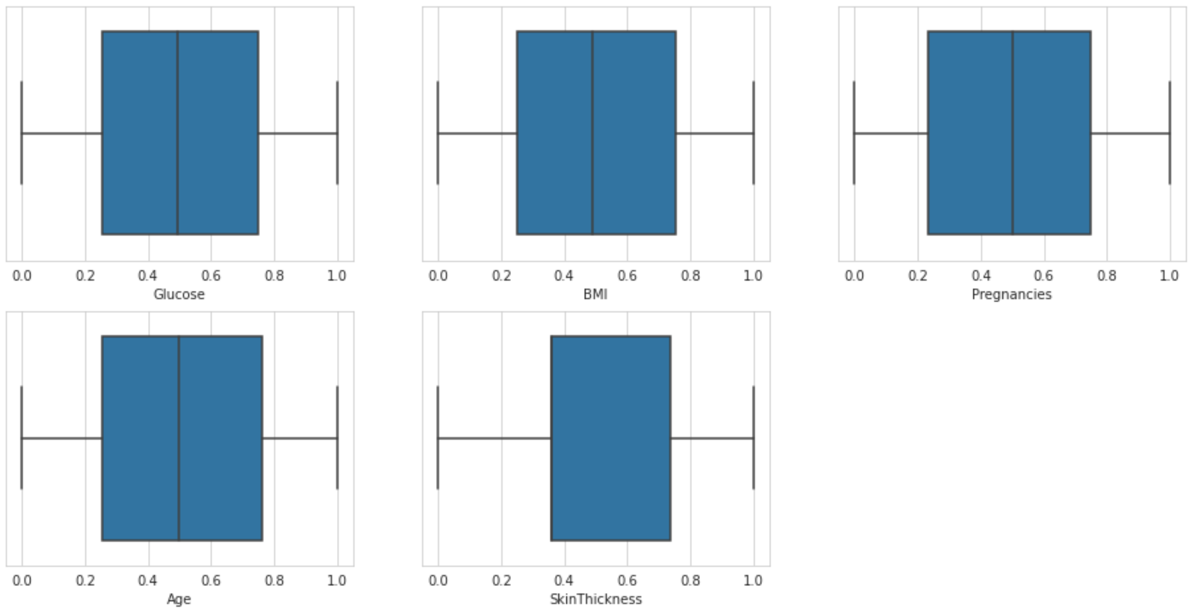
In [19]:
```python
from sklearn.preprocessing import QuantileTransformer
x=df_selected
quantile  = QuantileTransformer()
X = quantile.fit_transform(x)
df_new=quantile.transform(X)
df_new=pd.DataFrame(X)
df_new.columns =['Pregnancies', 'Glucose','SkinThickness','BMI','Age','Outcome'
df_new.head()
```

Out[19]:

	Pregnancies	Glucose	SkinThickness	BMI	Age	Outcome
0	0.747718	0.810300	0.801825	0.591265	0.889831	1.0
1	0.232725	0.091265	0.644720	0.213168	0.558670	0.0
2	0.863755	0.956975	0.357888	0.077575	0.585398	1.0
3	0.232725	0.124511	0.357888	0.284224	0.000000	0.0
4	0.000000	0.721643	0.801825	0.926988	0.606258	1.0

In [20]:
```
plt.figure(figsize=(16,12))
sns.set_style(style='whitegrid')
plt.subplot(3,3,1)
sns.boxplot(x=df_new['Glucose'],data=df_new)
plt.subplot(3,3,2)
sns.boxplot(x=df_new['BMI'],data=df_new)
plt.subplot(3,3,3)
sns.boxplot(x=df_new['Pregnancies'],data=df_new)
plt.subplot(3,3,4)
sns.boxplot(x=df_new['Age'],data=df_new)
plt.subplot(3,3,5)
sns.boxplot(x=df_new['SkinThickness'],data=df_new)
```

Out[20]:
```
<AxesSubplot:xlabel='SkinThickness'>
```



5. Split the Data Frame into X and y

In [21]:
```
target_name='Outcome'
y= df_new[target_name]#given predictions - training data
X=df_new.drop(target_name,axis=1)#dropping the Outcome column and keeping all c
```

In [22]:
```
X.head() # contains only independent features
```

Out[22]:

	Pregnancies	Glucose	SkinThickness	BMI	Age
0	0.747718	0.810300	0.801825	0.591265	0.889831
1	0.232725	0.091265	0.644720	0.213168	0.558670
2	0.863755	0.956975	0.357888	0.077575	0.585398
3	0.232725	0.124511	0.357888	0.284224	0.000000
4	0.000000	0.721643	0.801825	0.926988	0.606258

In [23]:
```
y.head() #contains dependent feature
```
Out[23]:
```
0    1.0
1    0.0
2    1.0
3    0.0
4    1.0
Name: Outcome, dtype: float64
```

7. TRAIN TEST SPLIT

In [24]:
```
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_sta
```
In [25]:
```
X_train.shape,y_train.shape
```
Out[25]:
```
((614, 5), (614,))
```
In [26]:
```
X_test.shape,y_test.shape
```
Out[26]:
```
((154, 5), (154,))
```

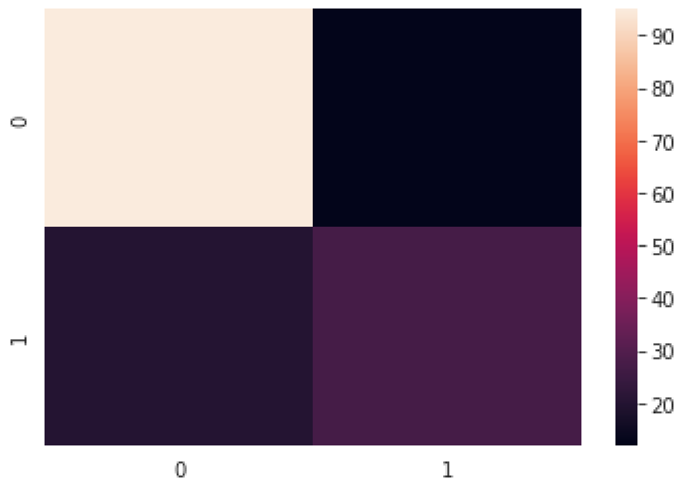
9. Classification Algorithms

Logistic Regression:-

In [55]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score,accuracy_sc
```
In [56]:
```
reg = LogisticRegression()
reg.fit(X_train,y_train)
```
Out[56]:
```
LogisticRegression()
```
In [57]:
```
lr_pred=reg.predict(X_test)
```

```
In [58]: print("Classification Report is:\n",classification_report(y_test,lr_pred))
         print("\n F1:\n",f1_score(y_test,lr_pred))
         print("\n Precision score is:\n",precision_score(y_test,lr_pred))
         print("\n Recall score is:\n",recall_score(y_test,lr_pred))
         print("\n Confusion Matrix:\n")
         sns.heatmap(confusion_matrix(y_test,lr_pred))
```

```
Classification Report is:
               precision    recall  f1-score   support

         0.0       0.83      0.89      0.86       107
         1.0       0.69      0.57      0.63        47

    accuracy                           0.79       154
   macro avg       0.76      0.73      0.74       154
weighted avg       0.79      0.79      0.79       154


 F1:
 0.627906976744186

 Precision score is:
 0.6923076923076923

 Recall score is:
 0.574468085106383

 Confusion Matrix:

Out[58]: <AxesSubplot:>
```



conclusion

a logistic regression model was developed to predict diabetes using key medical attributes. The model achieved an overall accuracy of 79% on the test dataset, demonstrating its ability to reasonably classify diabetes status.

The precision and recall scores indicate that while the model predicts diabetes (class 1) reasonably well with 69% precision, it fails to recall/capture all true diabetes cases, achieving only 57% recall. This suggests some diabetes cases are being misclassified as non-diabetes.

The F1 score of 0.63 provides a balanced measure of model performance, taking into account both precision and recall. It confirms that while the logistic regression model performs fairly well, there is still scope for improvement in correctly identifying all diabetes cases.

Feature analysis can help understand the predictive value of different attributes to diabetes. A larger representative dataset may also help improve model generalizability. Advanced techniques like neural networks could potentially utilize attribute interactions better for more accurate predictions.

this study demonstrates the feasibility of applying machine learning for diabetes prediction. However, further refinement of models is needed to achieve clinically acceptable performance