AI BASED DIABETES PREDICTION SYSTEM

TEAM NAME: Proj_224748_Team1

TOPIC: PREDICTING DIABETES USING MACHINE LEARNING

BY:

B. HEMALATHA

R. SATHYABAMA

G. GURU PRIYA

C. ROOPAVATHY

J. ABINAYA

Diabetes Prediction

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)

INDEX :-

5. Feature Selection

6. Handling Outliers

7. Split the Data Frame into X and y

8. Train test split

1. Import Required Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

2. Loading the dataset

In [2]:

```
df=pd.read_csv("../input/pima-indians-diabetes-database/diabetes.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]:

SL. NO.	PREGNANCIES	GLUCOSE	BLOOD PRESSURE	SKIN THICKNESS	INSULIN	BMI	DIABETES PEDIGREE FUNCTION	AGE	OUTCOME
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [4]:

```
df.shape
```

Out[4]:

```
(768, 9)
```

In [5]:

```
df.columns
```

Out[5]:

```
Index(['Pregnancies', 'Glucose', 'Blood Pressure', 'Skin Thickness',
'Insulin' ,'BMI', 'Diabetes Pedigree Function', 'Age', 'Outcome'],
 dtype='object')
```

In [6]:

```
df.dtypes
```

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()
```

```
<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()
# 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	BLOOD PRESSURE	SKIN THICKNESS	INSULIN	BMI	DIABETES PEDIGREE FUNCTION	AGE	OUTCOME
COUNT	768.000000	768.000000	768.000000	768. 000000	768. 000000	768.000000	768.000000	768.000000	768.000000
MEAN	3.845052	120.894531	69.105496	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
STD	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
MIN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
MAX	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

CONCLUSION :- We observe that min value of some columns is 0 which cannot be possible medically.Hence in the data cleaning process we'll have to replace them with median/mean value depending on the distribution. Also in the max column we can see insulin levels as high as 846! We have to treat outliers.

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]:

```
#dropping duplicate values
df=df.drop_duplicates()
```

In [10]:

```
#check for missing values
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]:

```
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
```

In [12]:

```
#replacing 0 values with median of that column
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].m
ean())
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness']
.median())
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].median())
df['BMI']=df['BMI'].replace(0,df['BMI'].median())
```

4. Data Visualization

Here we are going to plot :-

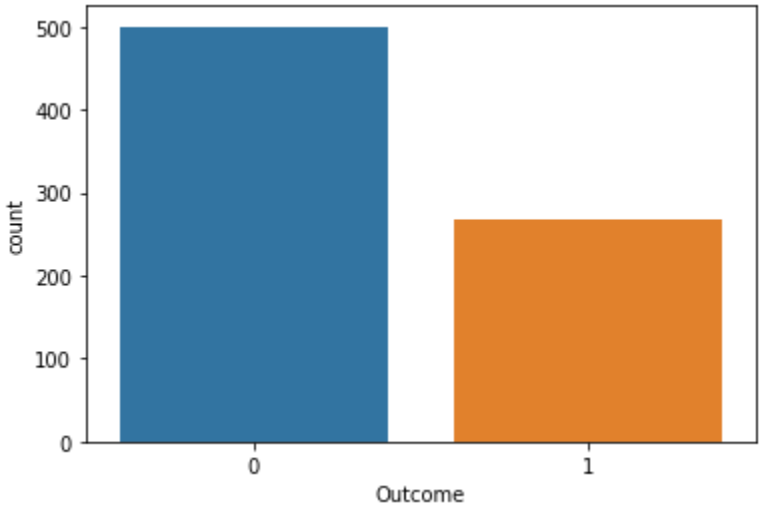
- Count Plot :- to see if the dataset is balanced or not
- Histograms :- to see if data is normally distributed or skewed
- Box Plot :- to analyse the distribution and see the outliers
- Scatter plots :- to understand relationship between any two variables
- Pair plot :- to create scatter plot between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

```
sns.countplot('Outcome',data=df)
```

In[13]

Out[13]:

<AxesSubplot:xlabel='Outcome', ylabel='count'>

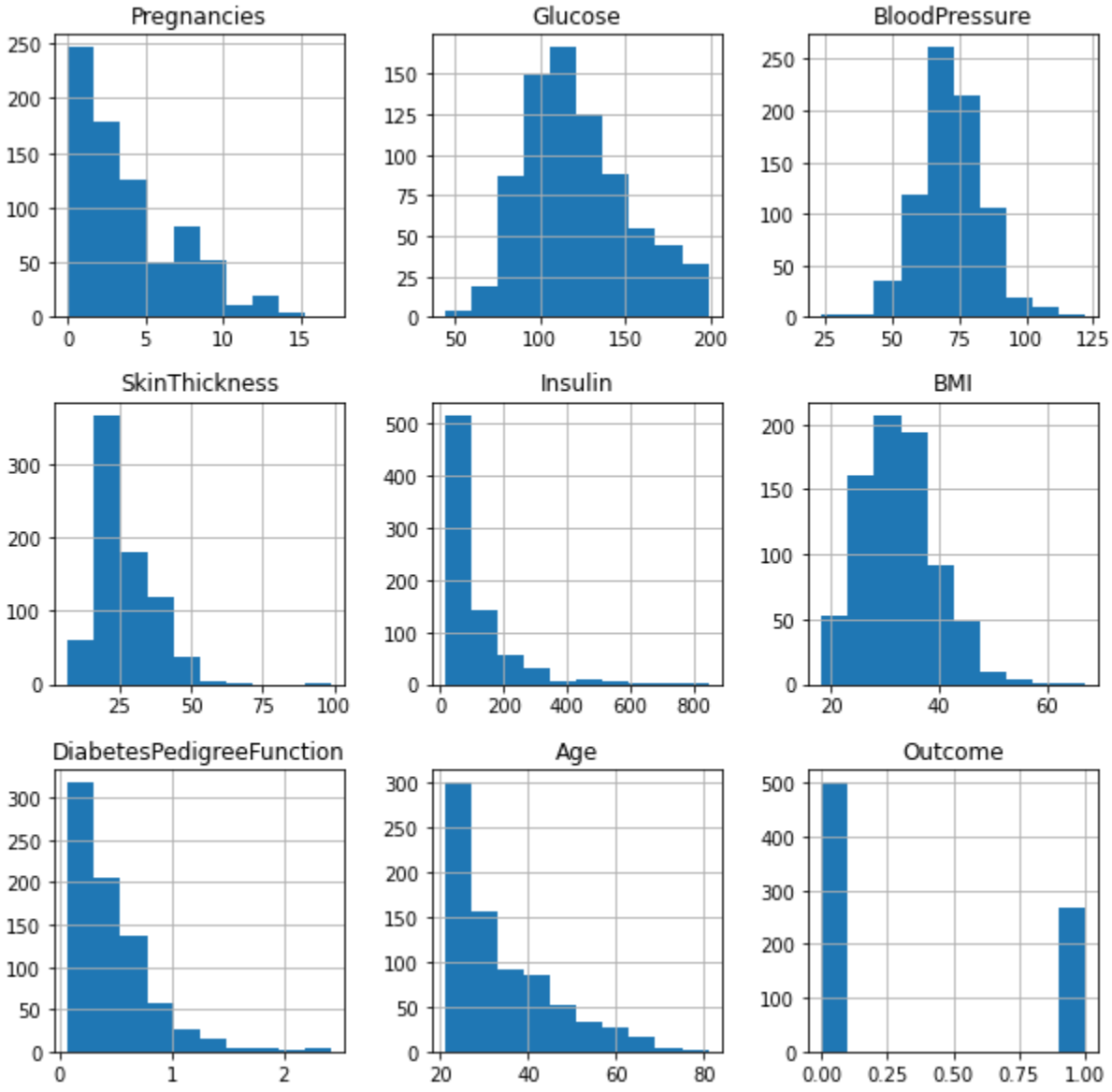


Conclusion :- We observe that number of people who do not have diabetes is far more than people who do which indicates that our data is imbalanced.

#histogram for each feature *In[14]*

```
df.hist(bins=10,figsize=(10,10))
```

plt.show()

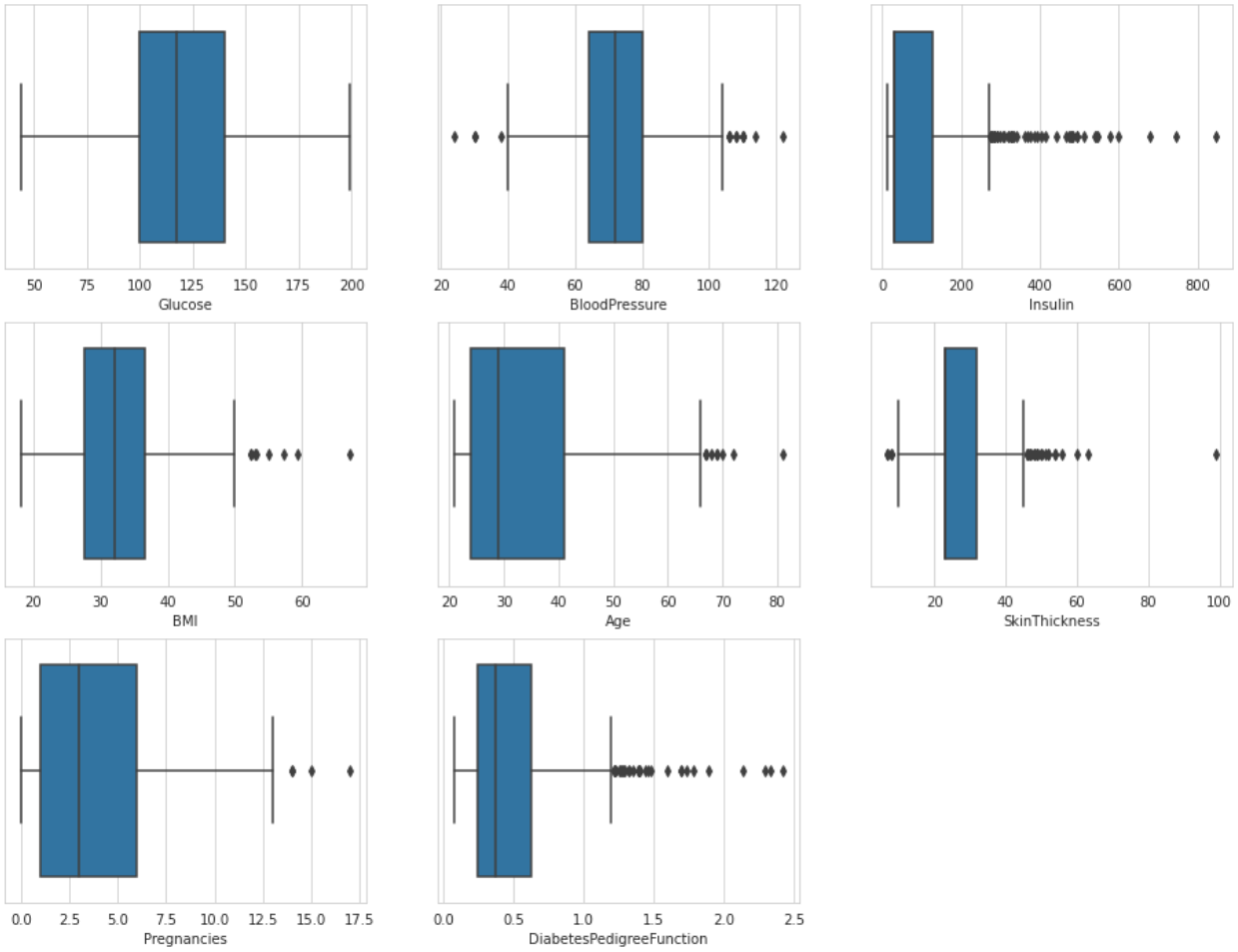


Conclusion :- We observe that only glucose and Blood Pressure are normally distributed rest others are skewed and have outliers

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)
```

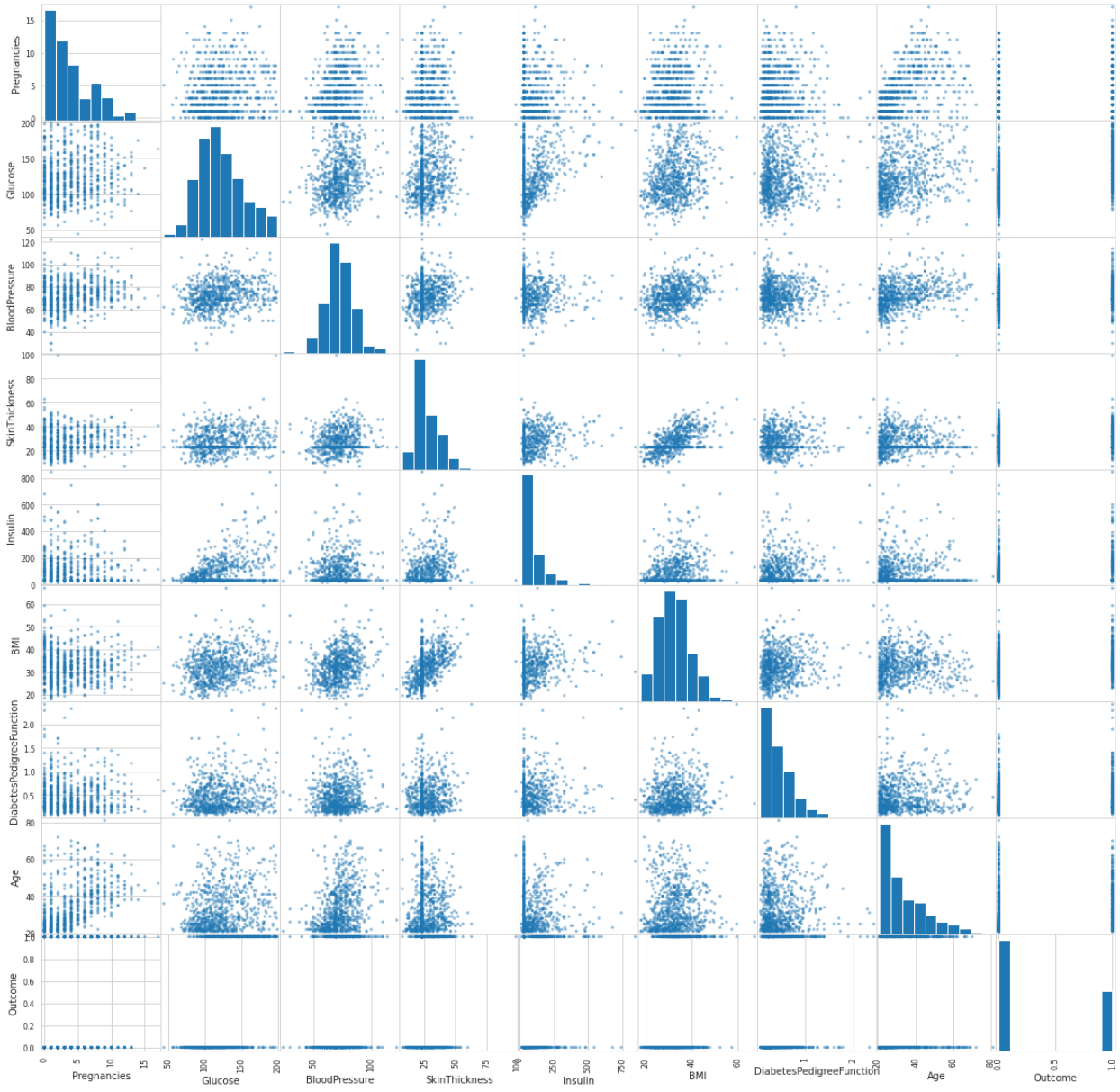
<Axes Subplot:xlabel='DiabetesPedigreeFunction'> Out[15]:



In [16]:

```
from pandas.plotting import scatter_matrix
scatter_matrix(df,figsize=(20,20));
```

 # we can come to various conclusion looking at these plots for example if you observe 5th plot in pregnancies with insulin, you can conclude that wo men with higher number of pregnancies have lower insulin



5. Feature Selection

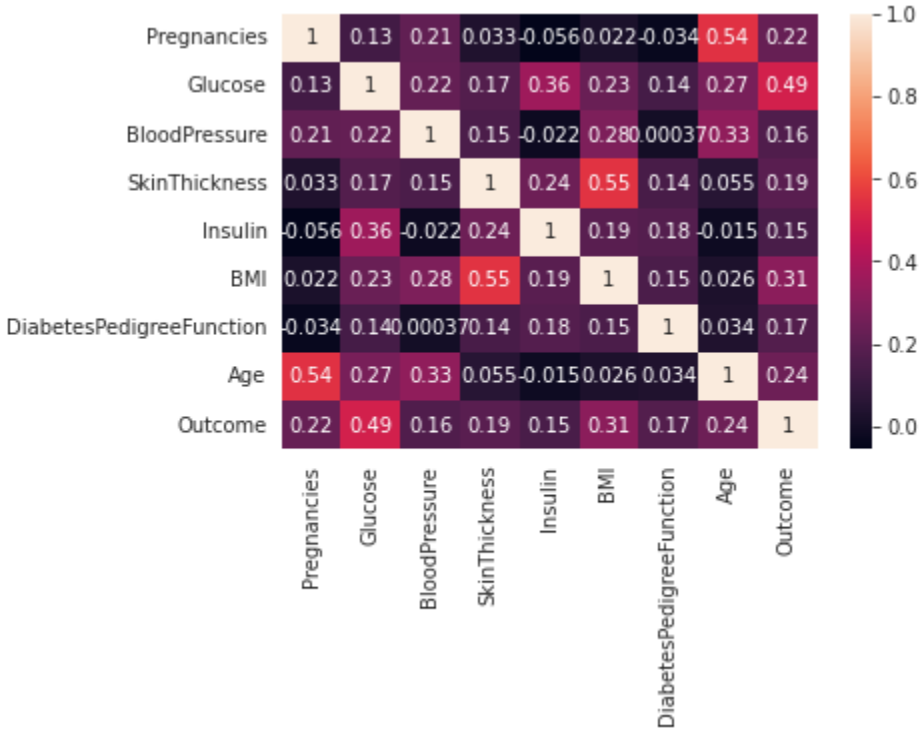
A heat map is a two-dimensional representation of information with the help of colors. Heat maps can help the user visualize simple or complex information.

In [17]:

```
corrmat=df.corr()
sns.heatmap(corrmat, annot=True)
```

Out[17]:

<AxesSubplot:>



CONCLUSION :- Observe the last row 'Outcome' and note its correlation scores with different features. We can observe that Glucose, BMI and Age are the most correlated with Outcome. BloodPressure, Insulin, DiabetesPedigreeFunction are the least correlated, hence they don't contribute much to the model so we can drop them

In [18]:

```
df_selected=df.drop(['BloodPressure','Insulin',
'DiabetesPedigreeFunction'],axis='columns')
```

6. Handling Outliers

An outlier is a data point in a data set that is distant from all other observations.

How can we Identify an outlier?

- Using Box plots
- Using Scatter plot
- Using Z score

I've used Box Plots above in data visualization step to detect outliers.

How am I treating the outliers ?

Quantile Transformer :- This method transforms the features to follow a uniform or a normal distribution. Therefore, for a given feature, this transformation tends to spread out the most frequent values. It also reduces the impact of (marginal) outliers: this is therefore a robust preprocessing scheme.

In[19]

```
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()
```

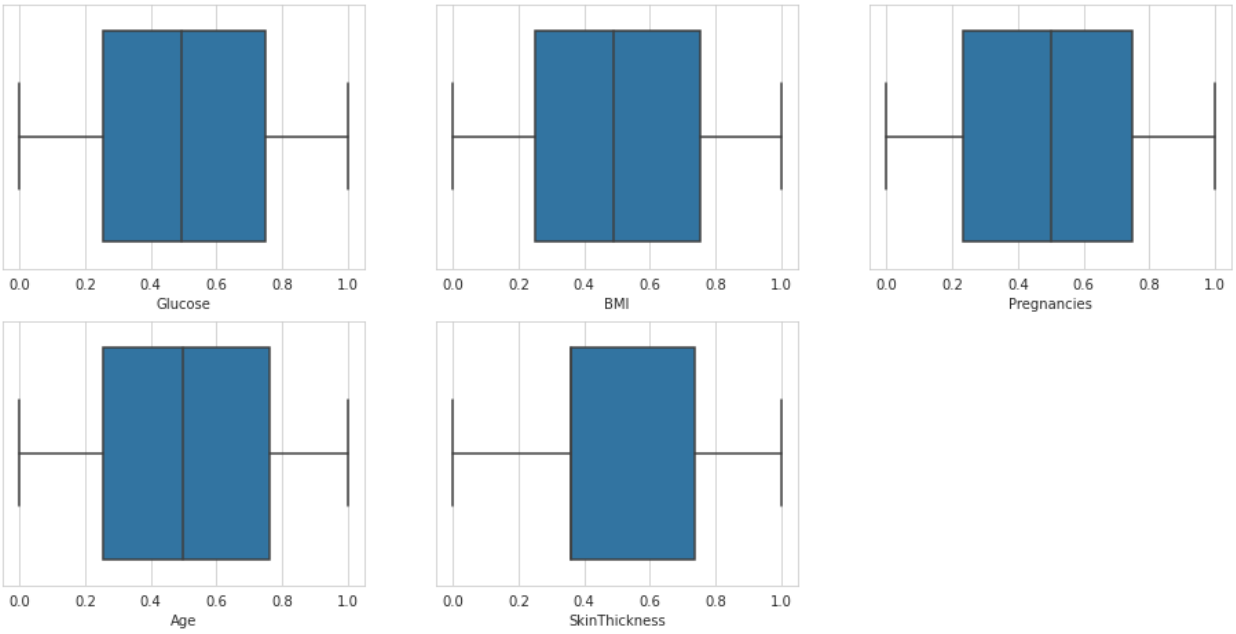
Out[19]:

	PTREGNANCIES	GLUCOSE	SKIN THICKNESS	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)
```

<Axes Subplot:xlabel='SkinThickness'> Out[20]:



7. 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)
```

In [22]:

```
X.head() # contains only independent features
```

Out[22]:

	PREGNANCIES	GLUCOSE	SKIN THICKNESS	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
```

8. Train Test Split

- The train-test split is a technique for evaluating the performance of a machine learning algorithm.
- Train Dataset: Used to fit the machine learning model.
- Test Dataset: Used to evaluate the fit machine learning model.
- Common split percentages include:

Train: 80%, Test: 20%

Train: 67%, Test: 33%

Train: 50%, Test: 50% . I've used 80% train and 20% test

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_state=0)#splitting data in 80% train, 20%test
```

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,))
```

Conclusion:

In the quest to predict diabetes, we have embarked on a critical journey that begins with loading and preprocessing the dataset. We have traversed through essential steps, starting with importing the necessary libraries to facilitate data manipulation and analysis.

Data preprocessing emerged as a pivotal aspect of this process. It involves cleaning, transforming, and refining the dataset to ensure that it aligns with the requirements of machine learning algorithms.

With these foundational steps completed, our dataset is now primed for the subsequent stages of predicting diabetes.