SUMMER PROJECT
DIABETES PREDICTION USING MACHINE LEARNING

R. SREVARDHANI
Information Technology
2019506095

MODULE:

DATA VISUALIZATION, FEATURE ENGINEERING AND MODEL BUILDING USING RANDOM FOREST.

Link to view the implementation of my module:

- https://colab.research.google.com/drive/1s3WGDzQ1Vji5JWUMN4OlLILSqRVb9BIh?usp=sharing
- https://colab.research.google.com/drive/1jympg4ZgpguTSKkmFd4LNkE7mo0P7O5H?usp=sharing

There are a total of 768 records and 9 features in the dataset. Each feature can be either of integer or float datatype. Some features like Glucose, Blood pressure , Insulin, BMI have zero values which represent missing data. There are zero NaN values in the dataset. In the outcome column, 1 represents diabetes positive (500) and 0 represents diabetes negative(268).

ALGORITHM:

Steps involved in my module:

1. Import the necessary package and load the dataset.
2. Plot different graphs- Data Visualization with suitable codes.
3. Pre-process the dataset- filling the missing values with median values.
4. Feature Engineering : Outlier observation analysis by IQR and Z SCORE method.
5. Add logical new categorical variables to the dataset.
6. Convert categorical variable to numeric variable by Label Encoding and One Hot Encoding.
7. Build a Random Forest classifier.
8. Evaluate the accuracy of the model.

DATA VISUALIZATION:

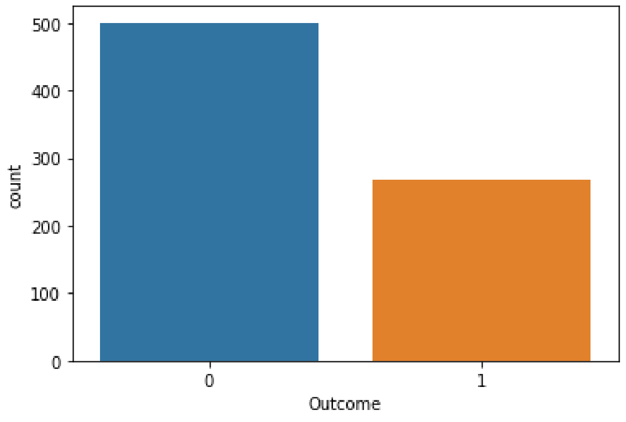
DATA STRUCTURE USED: Pandas DataFrame
With the help of data visualization, we can see how the data looks like and what kind of correlation is held by the attributes of data. It is the fastest way to see if the features correspond to the output. Visualization of data is an imperative aspect that helps to understand data and also explain the data to others.

COUNTPLOT:

Seaborn library has a function countplot() for creating countplot using sns.countplot() function._One easy way to visualize the counts of observations for outcome type is to create a count plot.

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc63f756090>
```



OBSERVATION:

We can observe that the count of 0 outcome that is count of diabetes negative is more than diabetes positive which is denoted as 1.

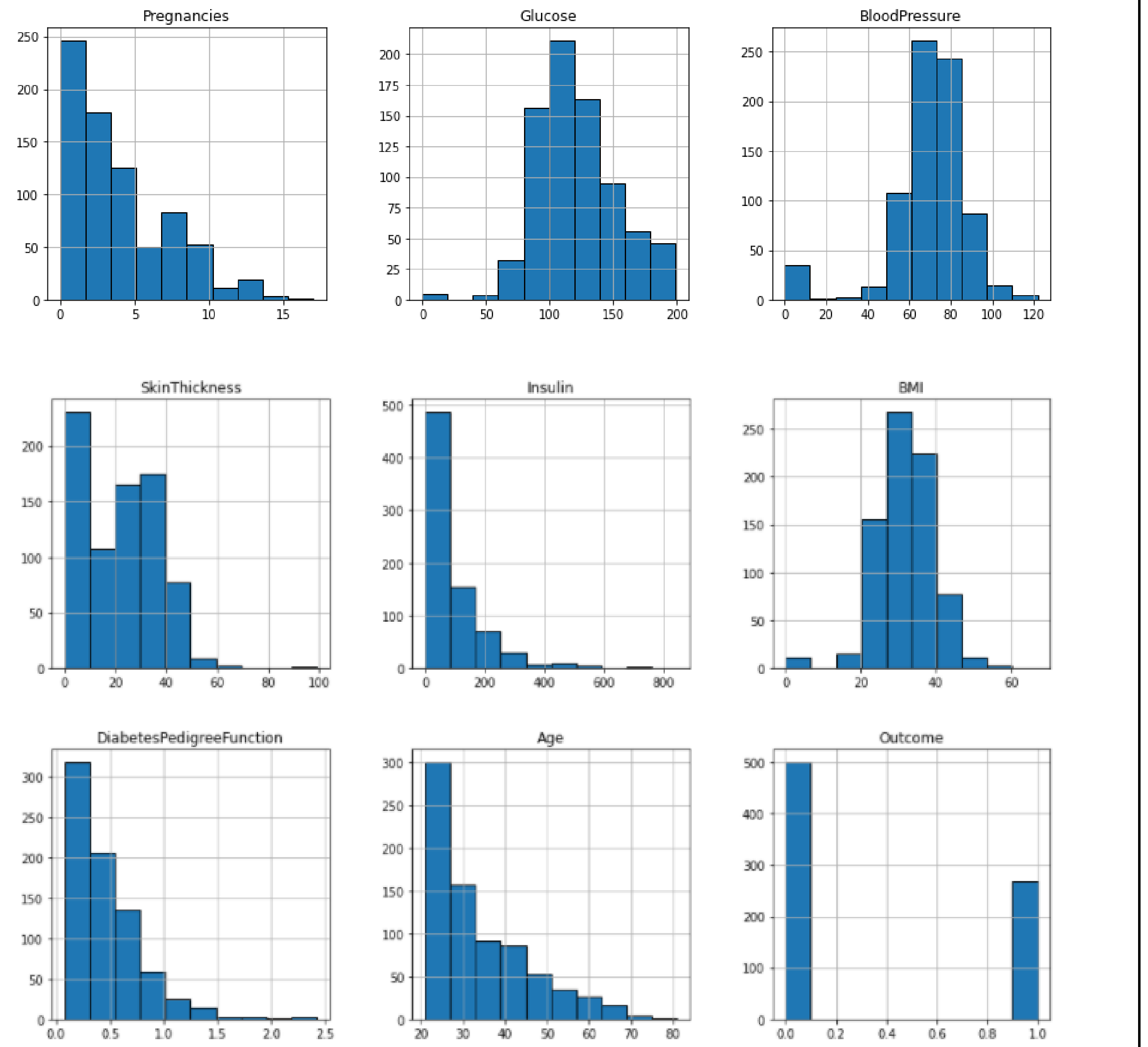
UNIVARIATE PLOTS: Understanding Attributes Independently

The simplest type of visualization is single-variable or "univariate" visualization. With the help of univariate visualization, we can understand each attribute of our dataset independently.

HISTOGRAMS:

Histograms group the data in bins and is the fastest way to get idea about the distribution of each attribute in dataset. Characteristics–

- It provides us a count of the number of observations in each bin created for visualization.
- From the shape of the bin, we can easily observe the distribution i.e. weather it is Gaussian, skewed or exponential.
- Histograms also help us to see possible outliers.
- hist() function on **Pandas DataFrame** are used to generate histograms and **matplotlib** are used for plotting them.



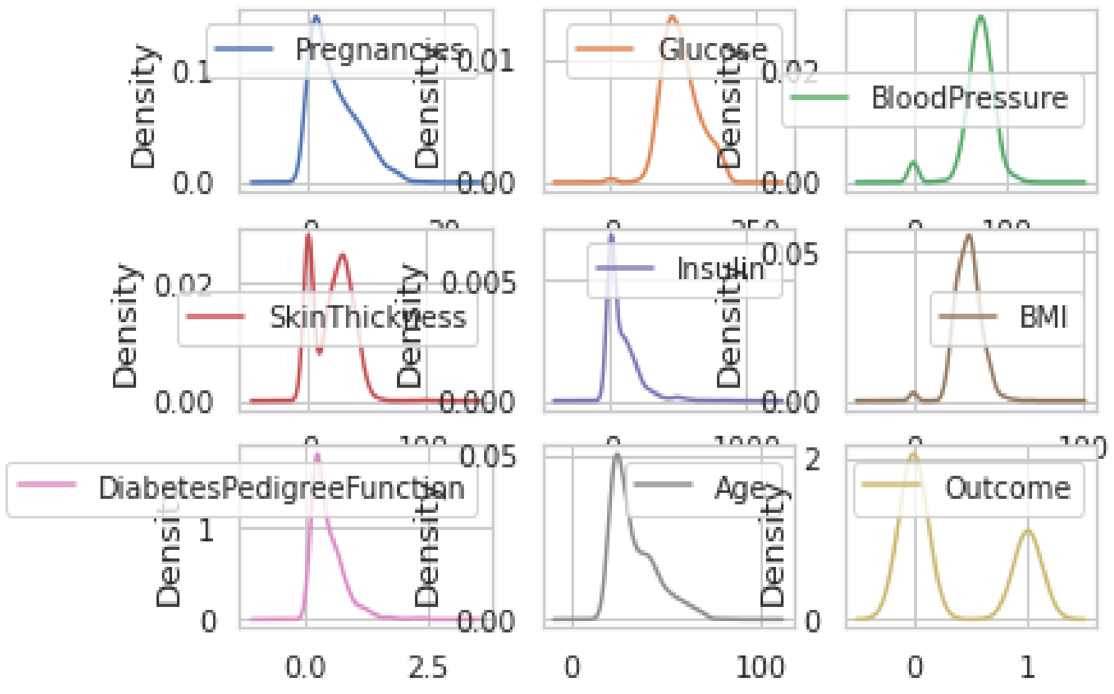
OBSERVATION:

The output shows that the histogram for each attribute in the dataset is created.
From this, we can observe that perhaps Age, Insulin and DiabetesPedigreeFunction attributes may have exponential distribution while BMI and Glucose have Gaussian distribution.

DENSITY PLOTS:

Another quick and easy technique for getting each attributes distribution is Density plots. It is also like histogram but having a smooth curve drawn through the top of each bin. We can call them as abstracted histograms and **matplotlib** library is used for plotting them.

```
df.plot(kind='density', subplots=True, layout=(3,3), sharex=False)
plt.show()
```



<u>OBSERVATION</u>:

The difference between Density plots and Histograms can be easily understood.

<u>BOX AND WHISKER PLOTS:</u>

Box and Whisker plots, also called boxplots in short, is another useful technique to review the distribution of each attribute's distribution and **Seaborn library** is used for plotting.

The following are the characteristics of this technique –

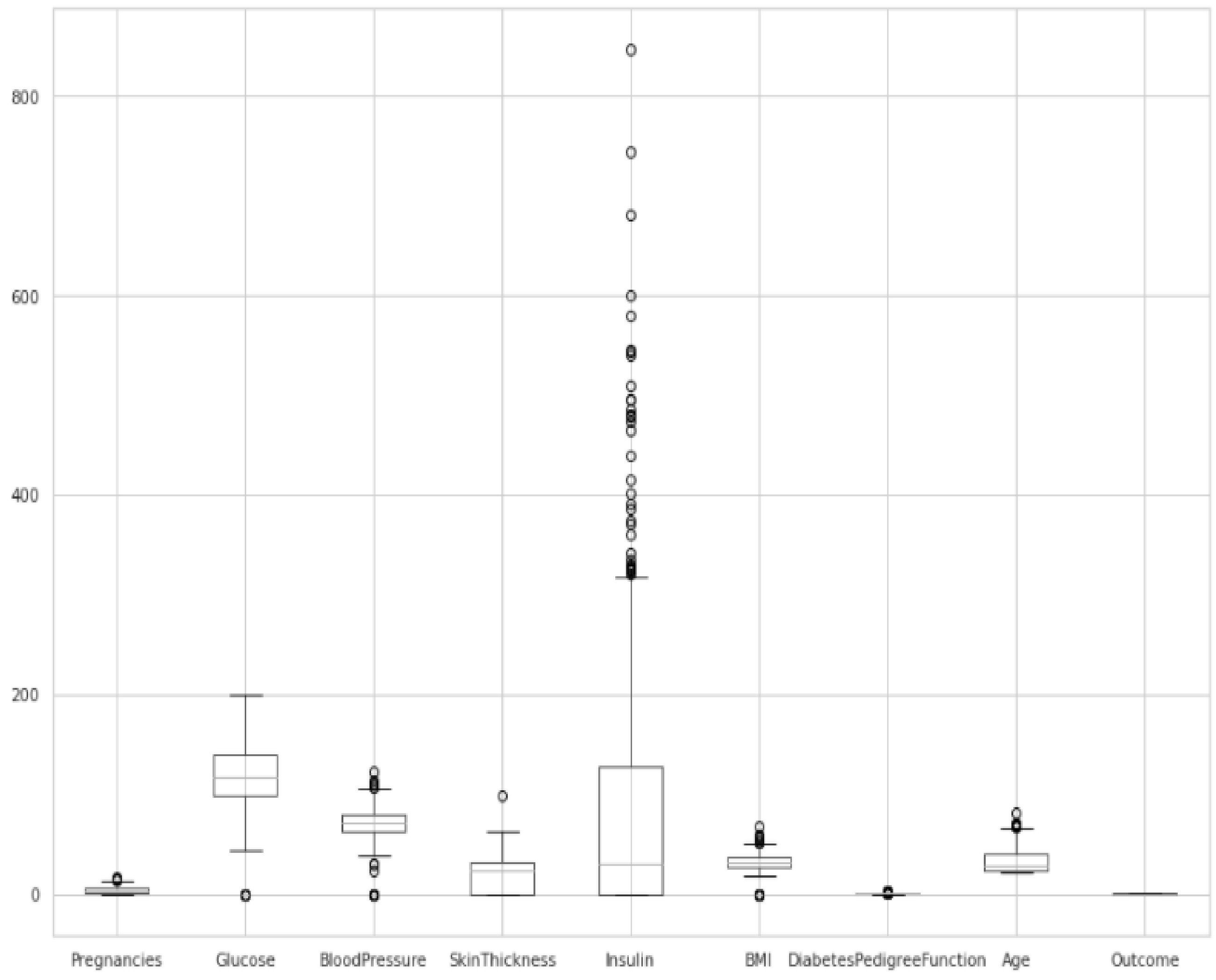
- It is univariate in nature and summarizes the distribution of each attribute.
- It draws a line for the middle value i.e. for median.
- It draws a box around the 25% and 75%.
- It also draws whiskers which will give us an idea about the spread of the data.
- The dots outside the whiskers signifies the outlier values. Outlier values would be 1.5 times greater than the size of the spread of the middle data.

<u>OBSERVATION</u>:

We can observe the outlier values.

```
sns.set(style="whitegrid")
df.boxplot(figsize=(15,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc63ddb8e10>
```

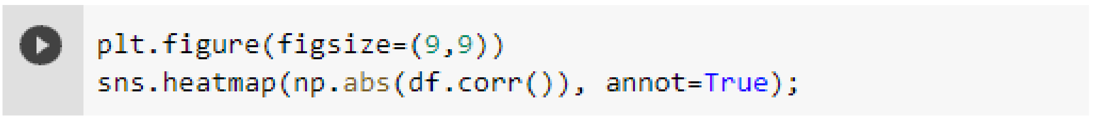


MULTIVARIATE PLOTS: Interaction Among Multiple Variables

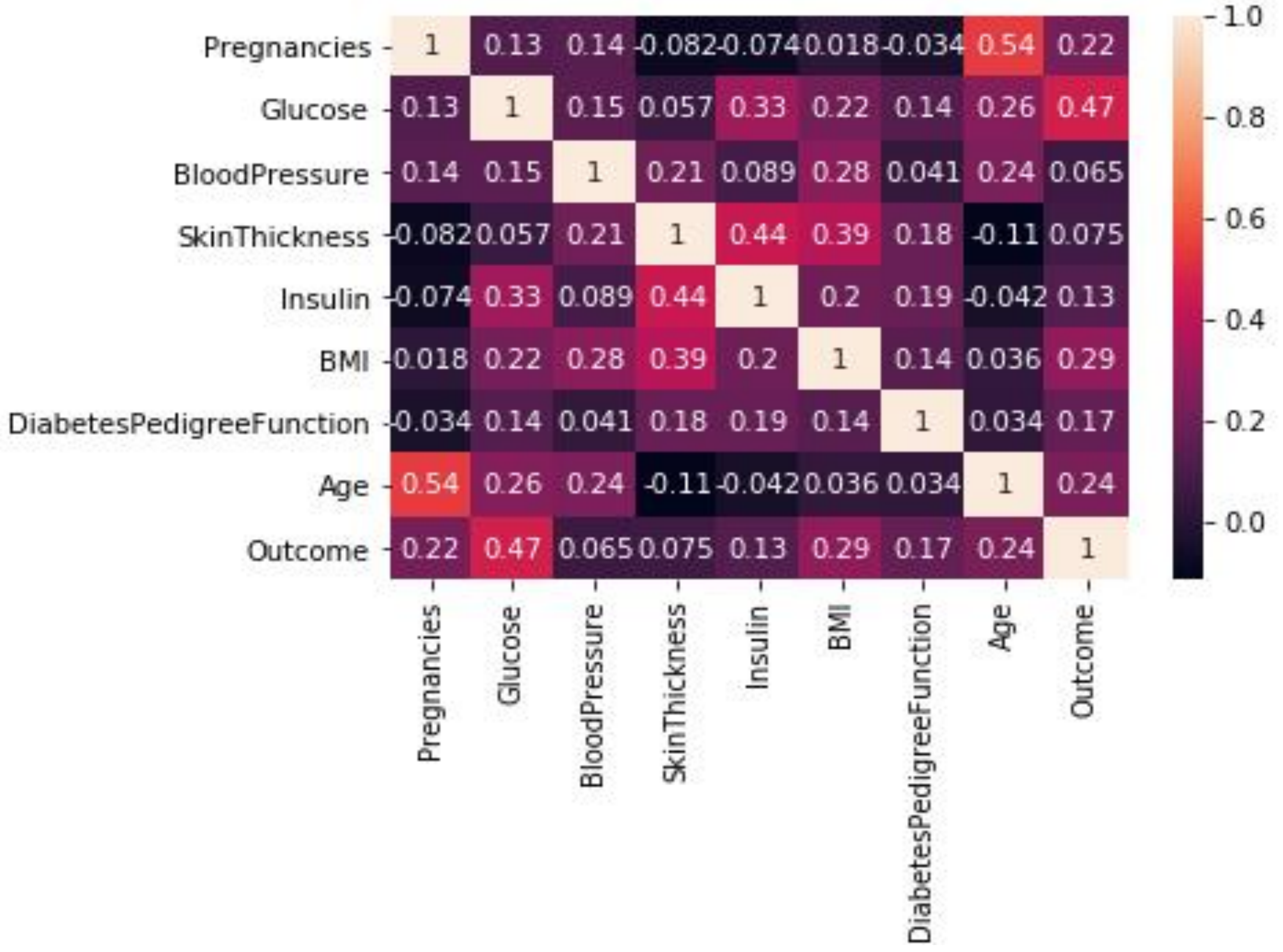
Another type of visualization is multi-variable or “multivariate” visualization. With the help of multivariate visualization, we can understand interaction between multiple attributes of our dataset.

CORRELATION MATRIX PLOT:

Correlation is an indication about the changes between two variables. We can plot correlation matrix to show which variable is having a high or low correlation in respect to another variable. Correlation describes the strength and direction of the linear association between two quantitative variables. It ranges from -1 to +1, Positive value indicate positive association and negative value indicate negative association. **Seaborn library** is used for plotting.



```
plt.figure(figsize=(9,9))
sns.heatmap(np.abs(df.corr()), annot=True);
```



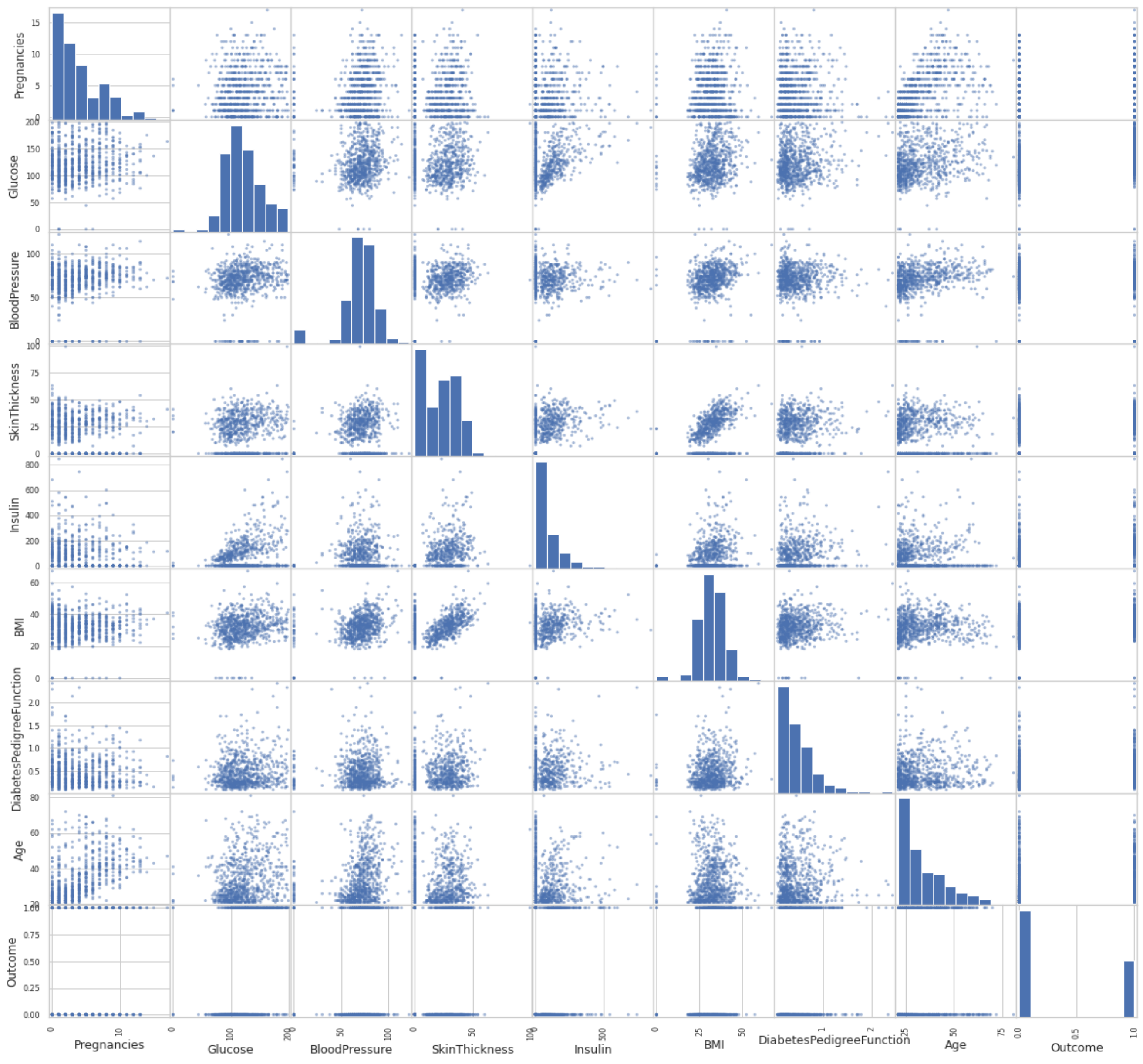
<u>OBSERVATION</u>:

Glucose, BMI, Age, Pregnancies are the top 4 features that are related to the outcome. We can also see that it is symmetrical i.e. the bottom left is same as the top right. It is also observed that each variable is positively correlated with each other.

<u>SCATTER MATRIX PLOT</u>:

Scatter plots shows how much one variable is affected by another or the relationship between them with the help of dots in two dimensions. Scatter plots are very much like line graphs in the concept that they use horizontal and vertical axes to plot data points.

Python script will generate and plot Scatter matrix for the Pima Indian Diabetes dataset. It can be generated with the help of scatter_matrix() function on Pandas DataFrame and plotted with the help of pyplot.



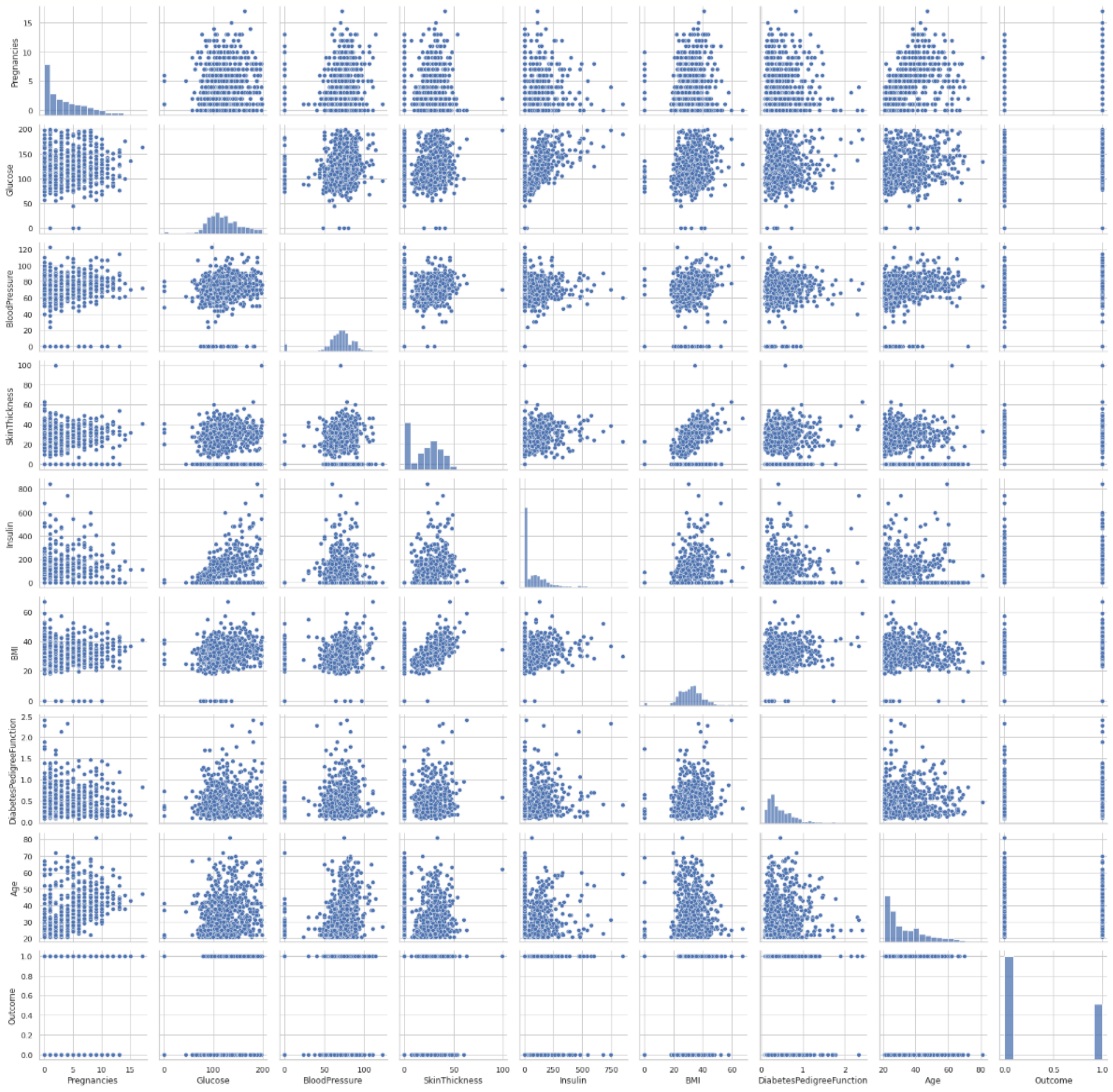
OBSERVATION:

We can observe that there is a positive linear relation between the points. That is, the value of y increases with the value of x.

PAIR PLOT:

Pairplot visualizes given data to find the relationship between them. Pairplot is a module of seaborn library which provides a high-level interface for drawing attractive and informative statistical graphics. The pairs plot builds on two basic figures, the histogram and the scatter plot. The histogram on the diagonal allows us to see the distribution of a single variable while the scatter plots on the upper and lower triangles show the relationship between two variables. The **matplotlib** library is used for plotting them.

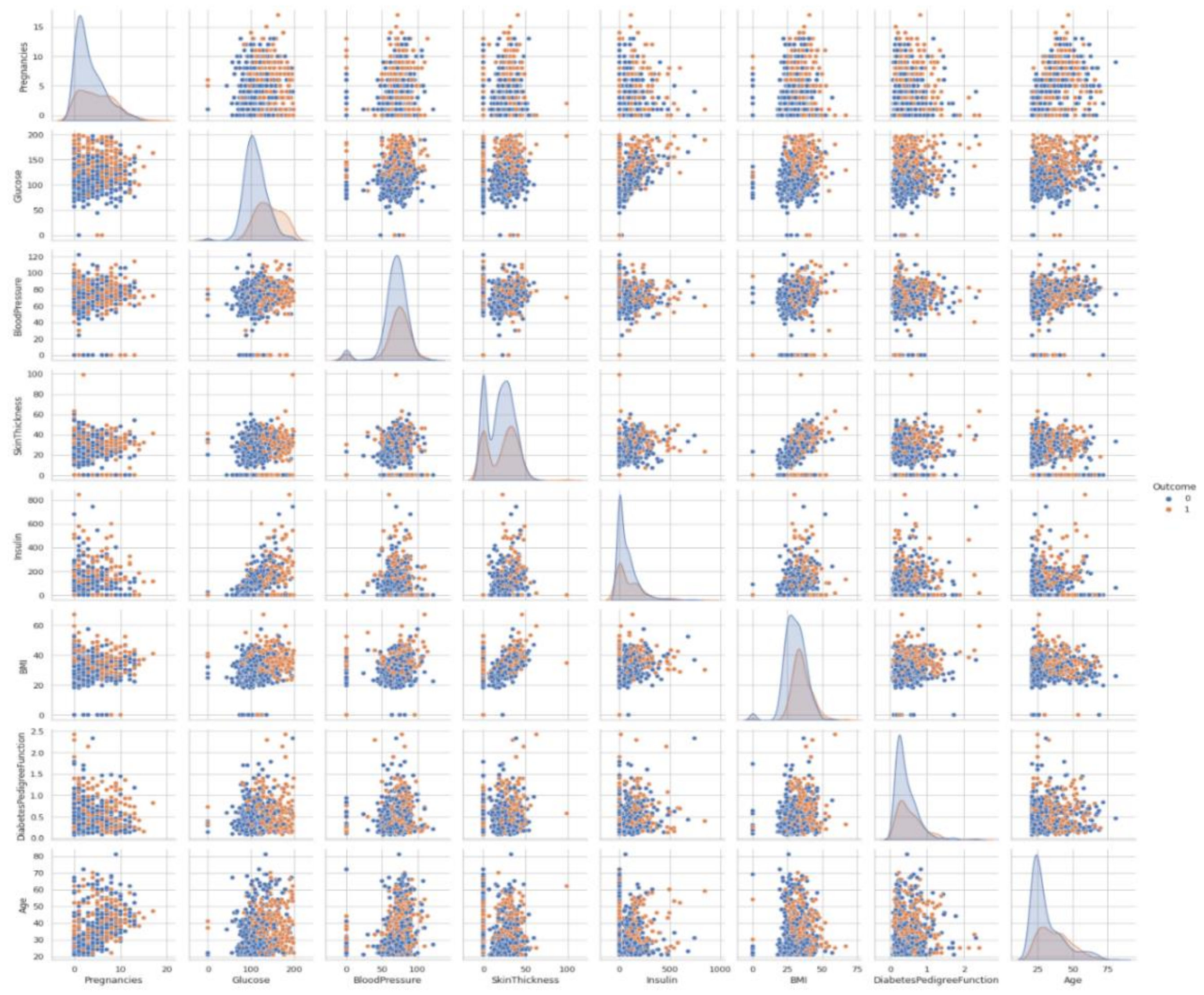
sns.pairplot(data = df)
plt.show()



<u>OBSERVATION</u>:

We can observe the variations in each plot. The plots are in matrix format where the row name represents x axis and column name represents the y axis. The main-diagonal subplots are the univariate histograms (distributions) for each attribute.

Hue helps us to get the difference in variable in data to map plot aspects to different colours.
sns.pairplot(data = df, hue = 'Outcome')
plt.show()



OBSERVATION:

From the above pairplots, we could observe that there is a positive correlation between various features:

- Glucose level increases with Age.
- Insulin level increases with Glucose level.
- Insulin level increases with Age.
- Slight increase in BloodPressure with Age.
- Slight increase in BloodPressure with Insulin.
- Skin thickness increases with insulin.
- Skin thickness increases with BMI.

MISSING OBSERVATION ANALYSIS: We saw that some features contain 0, it doesn't make sense here and this indicates missing value, we replace 0 value by NaN, so that we can look at where are missing values. In order to proceed with my module- feature engineering, missing values are filled median values of each variable that is the median value of people who are not sick and the median values of people who are sick.

FEATURE ENGINEERING:

DATA STRUCTURE USED: Numpy Pandas
Process of extracting useful features from a raw dataset using mathematics, statistics and domain knowledge. Algorithms require features with some specific characteristic to work properly.

Feature engineering mainly have two goals:

- Preparing the proper input dataset, compatible with the machine learning algorithm requirements.
- Improving the performance of machine learning models.

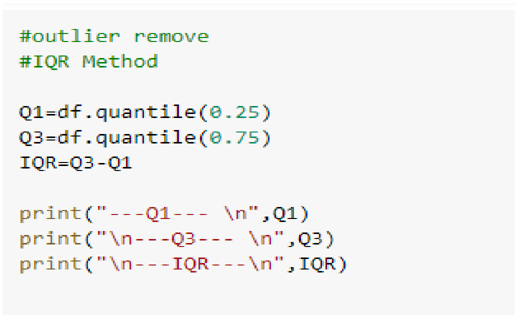
It allows avoiding overfitting the model despite providing many input features. The ultimate goal is to get the best results from the algorithms.

OUTLIER OBSERVATION ANALYSIS:

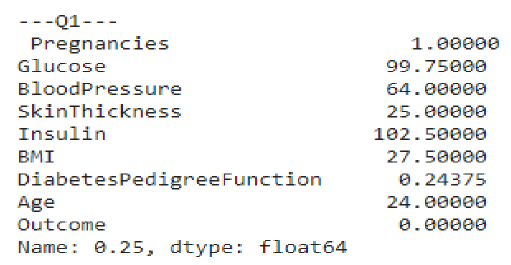
Outliers are datapoints that are far from other data points or unusual values in the dataset. These show some abnormality and deviate significantly from the normal data. Tool used for finding the outliers: BOXPLOTS- Tool for exploratory data analysis and easily make comparisons between distributions.

IQR METHOD:

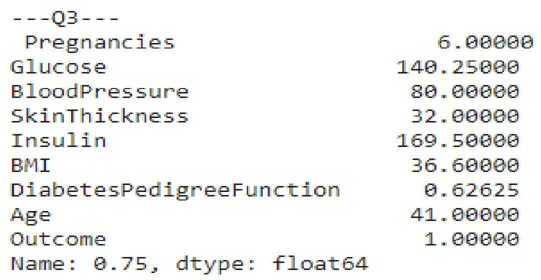
Box plot use the IQR method to display data and outliers(shape of the data) but in order to be get a list of identified outlier, we will need to use the mathematical formula and retrieve the outlier data.
The **interquartile range (IQR),** also called the **middle 50%,** is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles, IQR = $Q3 - Q1$; these quartiles can be clearly seen on a box plot on the data. **Seaborn and numpy** are the libraries used.



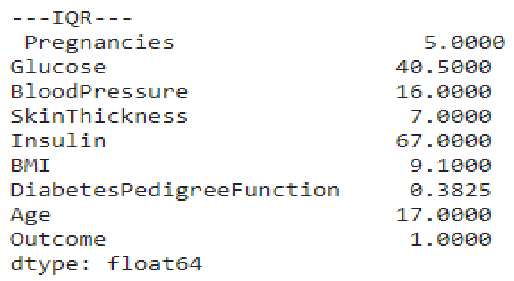
```
#outlier remove
#IQR Method

Q1=df.quantile(0.25)
Q3=df.quantile(0.75)
IQR=Q3-Q1

print("---Q1--- \n",Q1)
print("\n---Q3--- \n",Q3)
print("\n---IQR---\n",IQR)
```



```
---Q1---
 Pregnancies                   1.00000
Glucose                      99.75000
BloodPressure                64.00000
SkinThickness                25.00000
Insulin                     102.50000
BMI                          27.50000
DiabetesPedigreeFunction      0.24375
Age                          24.00000
Outcome                       0.00000
Name: 0.25, dtype: float64
```



```
---Q3---
 Pregnancies                   6.00000
Glucose                     140.25000
BloodPressure                80.00000
SkinThickness                32.00000
Insulin                     169.50000
BMI                          36.60000
DiabetesPedigreeFunction      0.62625
Age                          41.00000
Outcome                       1.00000
Name: 0.75, dtype: float64
```



```
---IQR---
 Pregnancies                   5.0000
Glucose                     40.5000
BloodPressure               16.0000
SkinThickness                7.0000
Insulin                     67.0000
BMI                          9.1000
DiabetesPedigreeFunction     0.3825
Age                         17.0000
Outcome                      1.0000
dtype: float64
```

Printing the values of Q1, Q3 and IQR.

```
#outlier remove
df_out = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape, df_out.shape
```

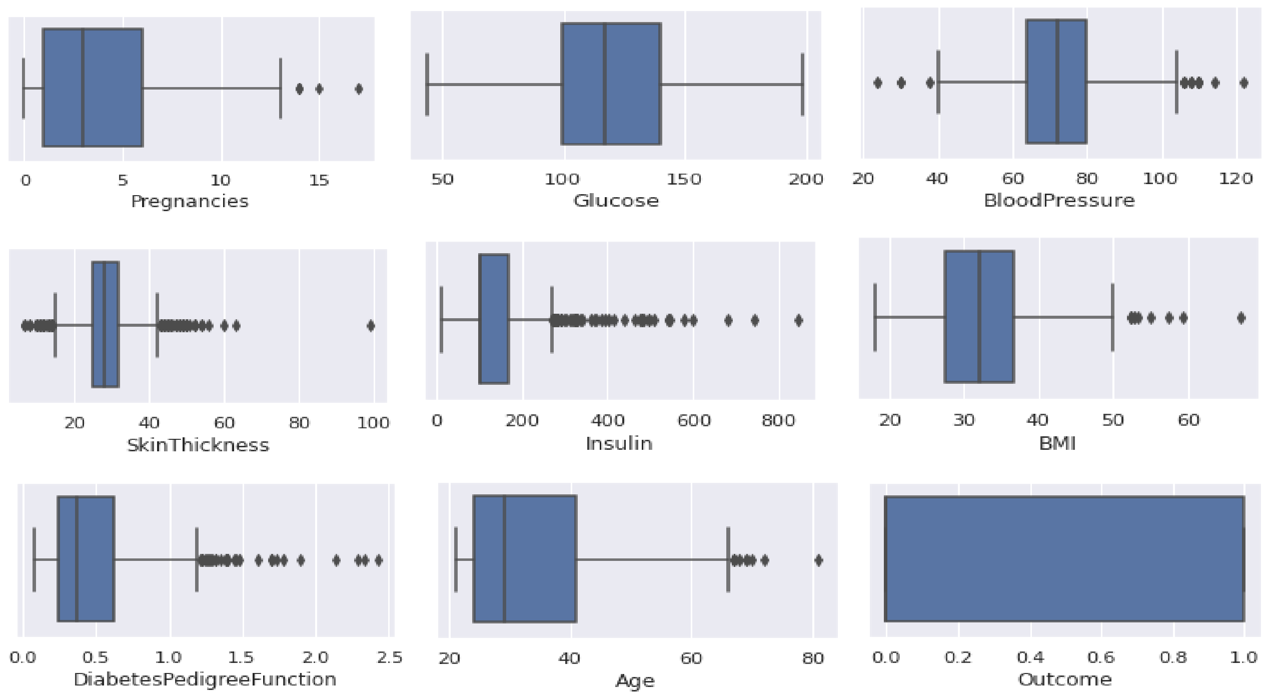
```
((768, 9), (590, 9))
```

OBSERVATION:

This removed the outliers from the dataset. After the removal of the outliers, the dataset has decreased by 178 records. All the outliers that are datapoints lesser than lower limit (Q1 - 1.5*IQR) and datapoints that are larger than the upper limit (Q3 + 1.5*IQR) are removed.

```
sns.set(style="whitegrid")

sns.set(rc={'figure.figsize':(4,2)})
sns.boxplot(x=df_out['Pregnancies'])
plt.show()
sns.boxplot(x=df_out['Glucose'])
plt.show()
sns.boxplot(x=df_out['BloodPressure'])
plt.show()
sns.boxplot(x=df_out['SkinThickness'])
plt.show()
sns.boxplot(x=df_out['Insulin'])
plt.show()
sns.boxplot(x=df_out['BMI'])
plt.show()
sns.boxplot(x=df_out['DiabetesPedigreeFunction'])
plt.show()
sns.boxplot(x=df_out['Age'])
plt.show()
sns.boxplot(x=df_out['Outcome'])
plt.show()
```

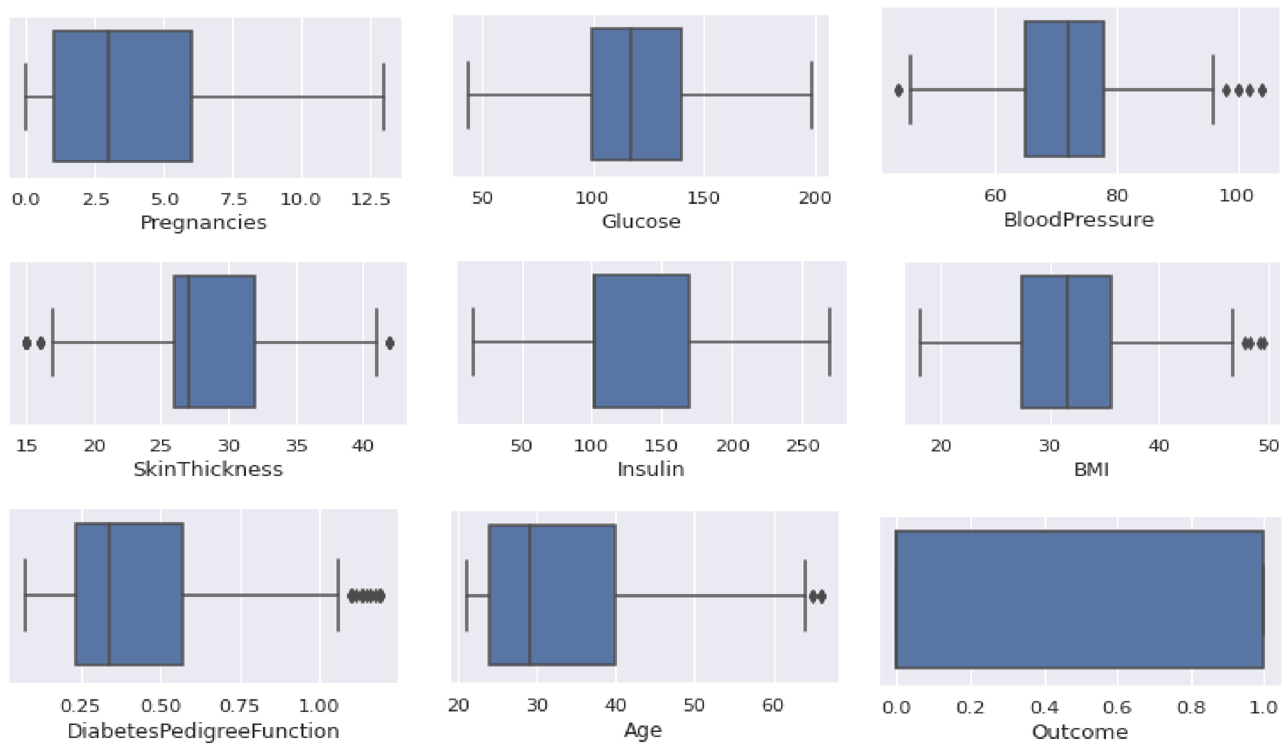
Box Plot for outlier visualization- Outliers are plotted as individual points.



OBSERVATION:

We could observe that there is no outliers for Glucose and Outcome features and there are outliers for all other features.

Box Plot after removing outliers



OBSERVATION:

Boxplots are plotted for each feature to visualize the outliers and also boxplots are plotted after the removal of the outliers for comparisons between them. We could observe that the outliers of each feature are removed to a large extent.

Z SCORE METHOD:

Z score, also called standard score is another method that can be used to remove outliers.. This score helps to understand if a data value is greater or smaller than mean and how far away it is from the mean. More specifically, Z score tells how many standard deviations away a data point is from the mean.
Z score = (x -mean) / std. deviation.

If the z score of a data point is more than 3, it indicates that the data point is quite different from the other data points. Such a data point can be an outlier. Z-score function defined in scipy library to detect the outliers.

```
z = np.abs(stats.zscore(df))
print(z)
```

```
[[0.63994726 0.86462486 0.03218035 ... 0.46849198 1.4259954  1.36589591]
 [0.84488505 1.20472661 0.52812374 ... 0.36506078 0.19067191 0.73212021]
 [1.23388019 2.01426457 0.69343821 ... 0.60439732 0.10558415 1.36589591]
 ...
 [0.3429808  0.02224005 0.03218035 ... 0.68519336 0.27575966 0.73212021]
 [0.84488505 0.14199419 1.02406713 ... 0.37110101 1.17073215 1.36589591]
 [0.84488505 0.94195182 0.19749482 ... 0.47378505 0.87137393 0.73212021]]
```

Looking the code and the output, it is difficult to say which data point is an outlier. So, we need to define a threshold to identify an outlier.

```
threshold = 3
print(np.where(z > 3 ))
```

```
(array([  4,   8,  13,  18,  43,  45,  57,  58,  88, 106, 111, 120, 120,
       123, 125, 125, 153, 159, 177, 177, 186, 220, 228, 228, 247, 286,
       298, 330, 370, 370, 371, 392, 395, 409, 415, 445, 445, 445, 453,
       455, 459, 486, 549, 579, 584, 593, 597, 621, 645, 655, 666, 673,
       684, 691, 695, 753]), array([6, 4, 4, 2, 2, 6, 3, 6, 0, 2, 4, 3, 5,
       4, 6, 4, 4, 0, 6, 4, 6, 6, 4, 6, 4, 4, 3, 5, 6, 7, 0, 7, 4, 2, 3,
       4, 6, 2, 6, 4, 4, 7, 5, 7, 2, 4, 4]))
```

The first array contains the list of row numbers and second array respective column numbers, which mean z[4][6] have a Z-score higher than 3. So, the data point 4th record on column 6 is an outlier.

```
df_remove = df [(z < 3).all(axis = 1)]
df_remove
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
5	5	116.0	74.0	27.0	102.5	25.6	0.201	30	0

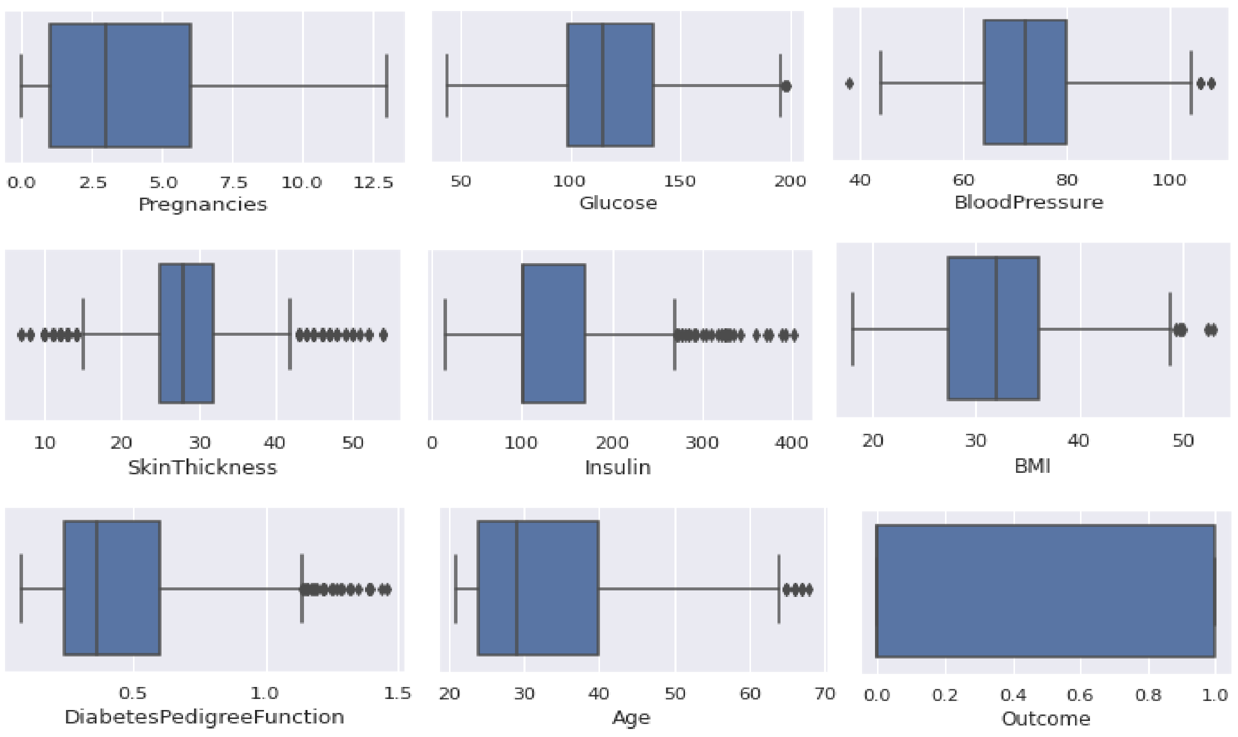
```
df_remove.shape, df.shape
```

```
((719, 9), (768, 9))
```

<u>OBSERVATION</u>:

All the outliers are removed, for example, 4th record is deleted as it had outlier and thus we got the clean data. After the removal of the outliers, the dataset has decreased by 49 records.

Box Plot after removing outliers using Z score method:



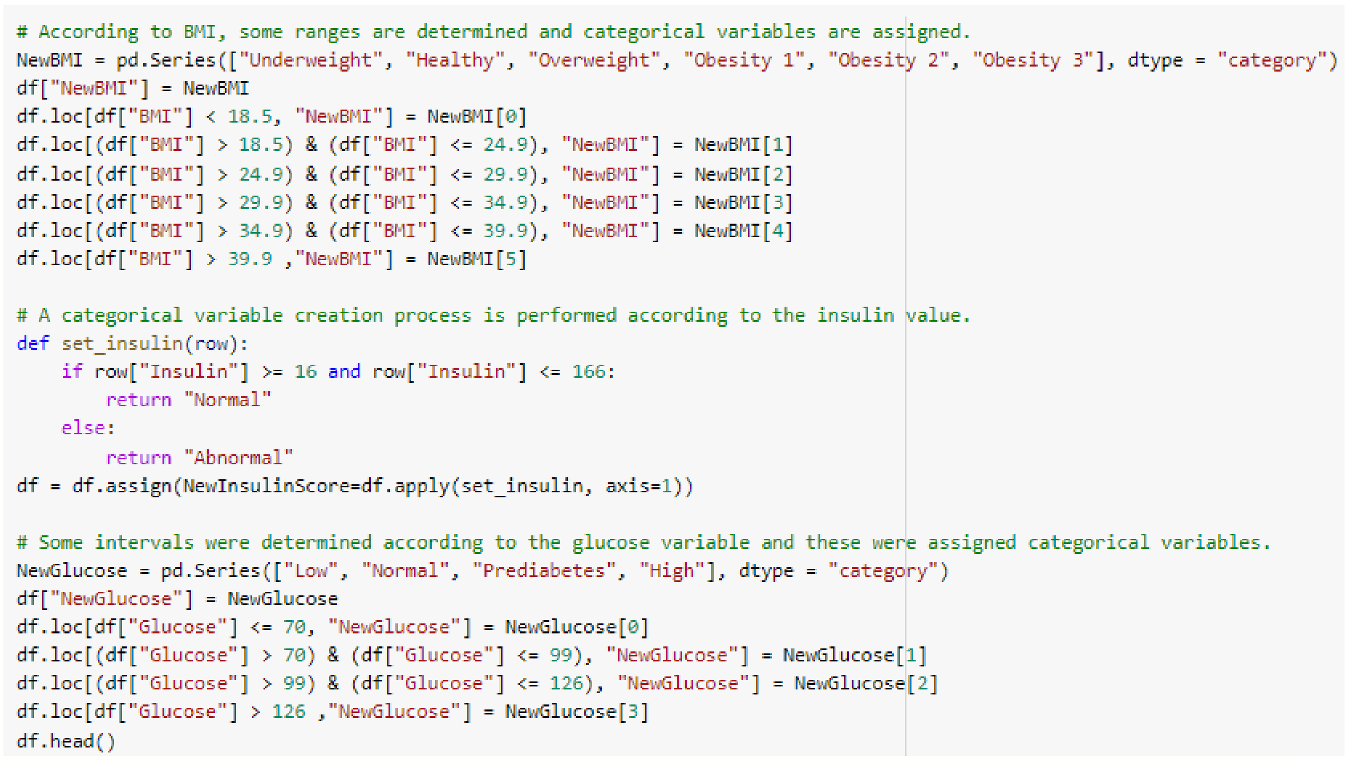
<u>OBSERVATION:</u>

We can observe that outliers are removed to some extent only when compared to IQR method.

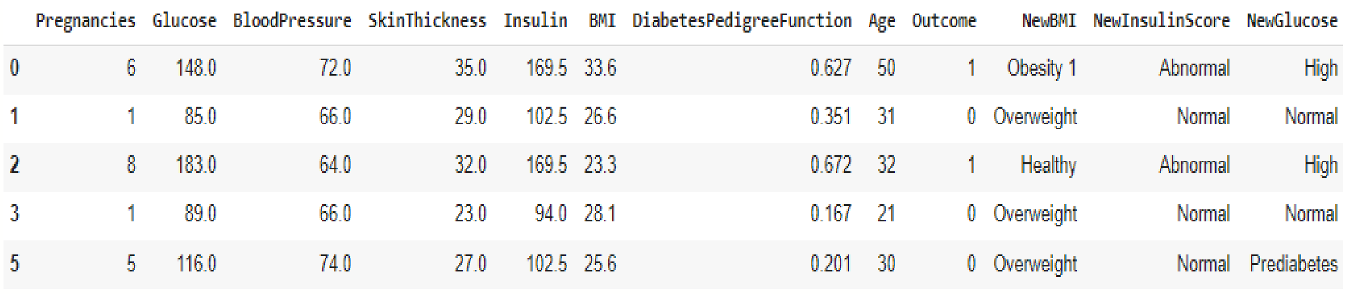
I have implemented two methods – IQR method and Z SCORE method to detect and remove the outliers from the dataset. The former method decreases the dataset to 590 records, whereas the later method decreases to719 records. Thus, Z score method deletes only 49 records from the dataset. On the other hand, Box Plots shows better results for IQR method than the Z score method. So, the method for the analysis of outliers will be selected according to accuracy level.

Creating new variables is important for models. But we need to create a logical new variable. For this data set, some new variables are created according to BMI, Insulin and Glucose variables. Max and Min values of each variable are found and some ranges are determined and categorical variables are assigned to these variables. These categorical variable is ordinal- Ordinal categorical attributes have some sense or notion of order amongst its values.

- According to BMI, 6 different ranges -Underweight, Healthy, Overweight, Obesity 1, Obesity 2, Obesity 3 are determined and categorical variable NewBMI is assigned.
- A categorial variable creation process is performed according to the insulin value. NewInsulinScore has 2 range- Normal and Abnormal and it is an ordinal categorical variable.
- Four intervals- Low, Normal, Prediabetes, High were determined according to the glucose variable and these were assigned to an ordinal categorical variable, NewGlucose.



```python
# According to BMI, some ranges are determined and categorical variables are assigned.
NewBMI = pd.Series(["Underweight", "Healthy", "Overweight", "Obesity 1", "Obesity 2", "Obesity 3"], dtype = "category")
df["NewBMI"] = NewBMI
df.loc[df["BMI"] < 18.5, "NewBMI"] = NewBMI[0]
df.loc[(df["BMI"] > 18.5) & (df["BMI"] <= 24.9), "NewBMI"] = NewBMI[1]
df.loc[(df["BMI"] > 24.9) & (df["BMI"] <= 29.9), "NewBMI"] = NewBMI[2]
df.loc[(df["BMI"] > 29.9) & (df["BMI"] <= 34.9), "NewBMI"] = NewBMI[3]
df.loc[(df["BMI"] > 34.9) & (df["BMI"] <= 39.9), "NewBMI"] = NewBMI[4]
df.loc[df["BMI"] > 39.9 ,"NewBMI"] = NewBMI[5]

# A categorical variable creation process is performed according to the insulin value.
def set_insulin(row):
    if row["Insulin"] >= 16 and row["Insulin"] <= 166:
        return "Normal"
    else:
        return "Abnormal"
df = df.assign(NewInsulinScore=df.apply(set_insulin, axis=1))

# Some intervals were determined according to the glucose variable and these were assigned categorical variables.
NewGlucose = pd.Series(["Low", "Normal", "Prediabetes", "High"], dtype = "category")
df["NewGlucose"] = NewGlucose
df.loc[df["Glucose"] <= 70, "NewGlucose"] = NewGlucose[0]
df.loc[(df["Glucose"] > 70) & (df["Glucose"] <= 99), "NewGlucose"] = NewGlucose[1]
df.loc[(df["Glucose"] > 99) & (df["Glucose"] <= 126), "NewGlucose"] = NewGlucose[2]
df.loc[df["Glucose"] > 126 ,"NewGlucose"] = NewGlucose[3]
df.head()
```

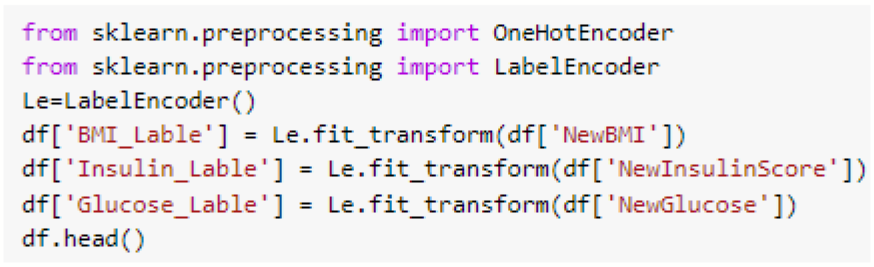


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	NewBMI	NewInsulinScore	NewGlucose
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1	Obesity 1	Abnormal	High
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0	Overweight	Normal	Normal
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1	Healthy	Abnormal	High
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	Overweight	Normal	Normal
5	5	116.0	74.0	27.0	102.5	25.6	0.201	30	0	Overweight	Normal	Prediabetes

We can now observe three categorical variables that are created. Categorical variables in the data set should be converted into numerical values. For this reason, these transformation processes are performed with 2 methos: Label Encoding and One Hot Encoding method.

LABEL ENCODING:

Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form. Here, each category is assigned a value from 1 to n, where n is the number of labels for the categorical feature.



```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
Le=LabelEncoder()
df['BMI_Lable'] = Le.fit_transform(df['NewBMI'])
df['Insulin_Lable'] = Le.fit_transform(df['NewInsulinScore'])
df['Glucose_Lable'] = Le.fit_transform(df['NewGlucose'])
df.head()
```

OBSERVATION:

➢ The categorical variable NewBMI is encoded by label encoding method. The new numeric BMI_Label feature has been encoded with:

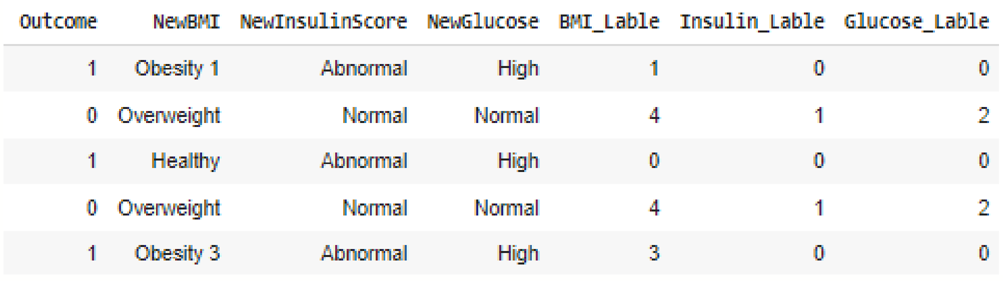
0 → Healthy,
1 → Obesity_1,
2 → Obesity_2,
3 → Obesity_3,
4 → Overweight and
5 → Underweight.

➢ We can observe that the categorical variable NewInsulinScore is encoded by label encoding method. The new numeric Insulin_Label feature has been encoded with:

0 → Abnormal, and
1 → Normal.

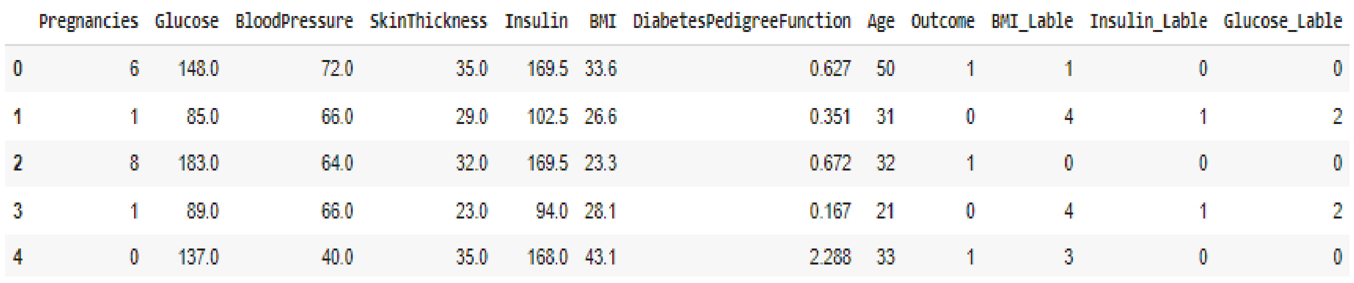
➢ The categorical variable NewGlucose is encoded by label encoding method. The new numeric Glucose_Label feature has been encoded with:

0 → High,
1 → Low,
2 → Normal, and
3 → Prediabetes.



Outcome	NewBMI	NewInsulinScore	NewGlucose	BMI_Lable	Insulin_Lable	Glucose_Lable
1	Obesity 1	Abnormal	High	1	0	0
0	Overweight	Normal	Normal	4	1	2
1	Healthy	Abnormal	High	0	0	0
0	Overweight	Normal	Normal	4	1	2
1	Obesity 3	Abnormal	High	3	0	0

Now, categorical variables can be dropped.



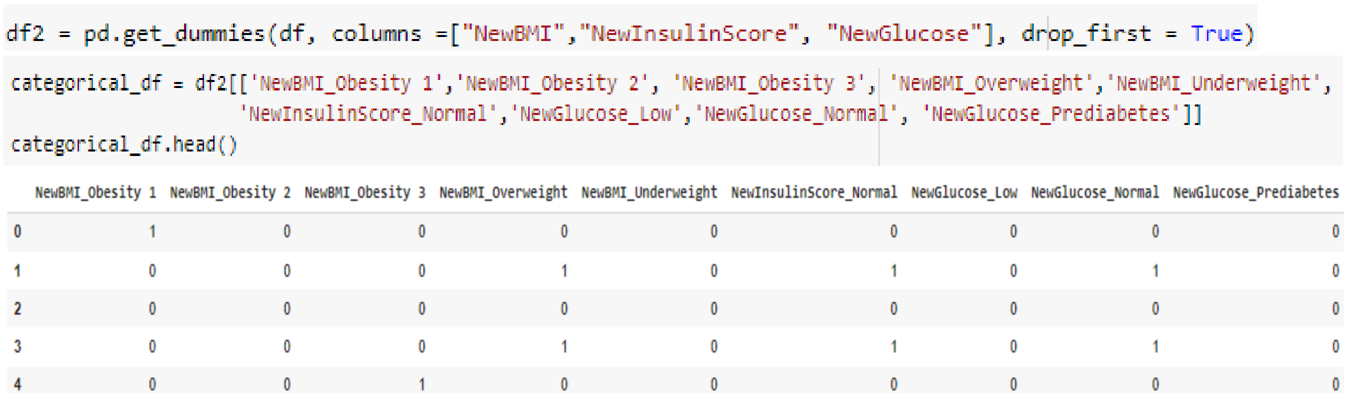
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	BMI_Lable	Insulin_Lable	Glucose_Lable
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1	1	0	0
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0	4	1	2
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1	0	0	0
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	4	1	2
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	3	0	0

There is another common alternative approach called ‘One-Hot Encoding’. In this strategy, each category value is converted into a new column and assigned a 1 or 0 (notation for true/false) value to the column.

ONE-HOT ENCODING:

One-hot encoding is one of the most common encoding methods in machine learning. This method spreads the values in a column to multiple flag columns and assigns 0 or 1 to them. These binary values express the relationship between grouped and encoded column. This method changes our categorical data, which is challenging to understand for algorithms, to a numerical format and enables us to group our categorical data without losing any information.

If we have N distinct values in the column, it is enough to map them to N-1 binary columns, because the missing value can be deducted from other columns. If all the columns in our hand are equal to 0, the missing value must be equal to 1. This is the reason why it is called as one-hot encoding.



```
df2 = pd.get_dummies(df, columns =["NewBMI","NewInsulinScore", "NewGlucose"], drop_first = True)
```

```
categorical_df = df2[['NewBMI_Obesity 1','NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'NewBMI_Overweight','NewBMI_Underweight',
                     'NewInsulinScore_Normal','NewGlucose_Low','NewGlucose_Normal', 'NewGlucose_Prediabetes']]
categorical_df.head()
```

	NewBMI_Obesity 1	NewBMI_Obesity 2	NewBMI_Obesity 3	NewBMI_Overweight	NewBMI_Underweight	NewInsulinScore_Normal	NewGlucose_Low	NewGlucose_Normal	NewGlucose_Prediabetes
0	1	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	1	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	1	0	1	0
4	0	0	1	0	0	0	0	0	0

OBSERVATION:

- The categorical variable, NewBMI have 6(k) categories which are converted into 5(k-1) new columns: NewBMI_Obesity 1, NewBMI_Obesity 2, NewBMI_Obesity 3, NewBMI_Overweight and are assigned with 1or 0 (notation for true/false) values to the columns.

- The categorical variable, NewInsulinScore have 2 categories which are converted into 1 new column: NewInsulinScore_Normaland are assigned with 1or 0 values to the column.

- The categorical variable, NewGlucose have 4 categories which are converted into 3 new columns: NewGlucose_Low, NewGlucose_Normal, NewGlucose_Prediabetes and are assigned with 1or 0 values to the columns.

All categorical variables in the dataset are converted into numeric values by encoding.

By trying different combinations of ZSCORE AND IQR method (outlier detection and removal) and LABEL ENCODING and ONE HOT ENCODING (conversion of categorical variable to numeric variable), it was found that the ZSCORE method and LABEL ENCODING method gives a higher accuracy for all the models.

It was observed that ZSCORE METHOD deleted only 49 records whereas IQR METHOD removed 178 records from the dataset. Because IQR method decreased the dataset to 590 records, accuracy also got reduced. Whereas Zscore method results in higher accuracy as dataset has 719 records.

With respect to convertion of categorical variable to numeric variable, the categorical variables in the data are NewBMI, NewInsulinScore and NewGlucose which are Ordinal categorical attributes that is attributes have some sense or notion of order amongst its values. For ordinal categorical variables, Label encoding is the most suitable method and it also gives higher accuracy for the model.

<u>**RANDOM FOREST:**</u>

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

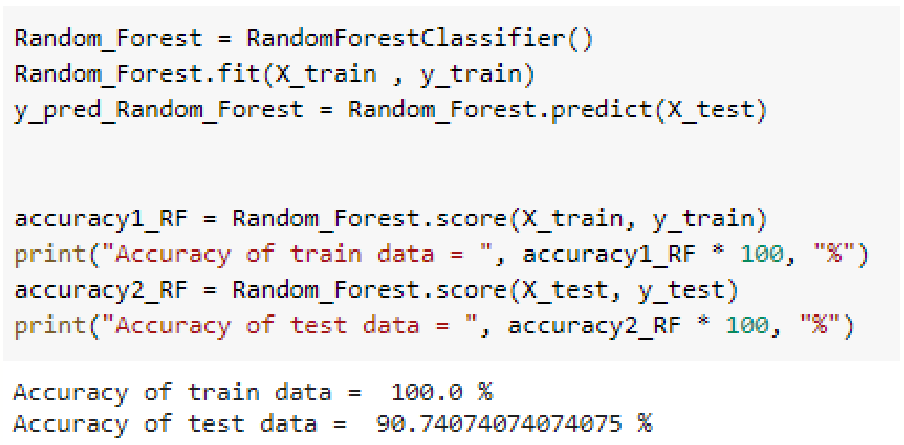
In this technique, a set of decision trees are grown and each tree votes for the most popular class, then the votes of different trees are integrated and a class is predicted for each sample. This approach is designed to increase the accuracy of the decision tree, more trees are produced to vote for class prediction. This approach is an ensemble classifier composed of some decision trees and the final result is the mean of individual trees results.

The Working process can be explained in the below steps:

- Select random K data points from the training set.
- Build the decision trees associated with the selected data points (Subsets).
- Choose the number N for decision trees that you want to build.
- Repeat Step 1 & 2
- For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

<u>ADVANTAGES OF RANDOM FOREST:</u>

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue



```
Random_Forest = RandomForestClassifier()
Random_Forest.fit(X_train , y_train)
y_pred_Random_Forest = Random_Forest.predict(X_test)


accuracy1_RF = Random_Forest.score(X_train, y_train)
print("Accuracy of train data = ", accuracy1_RF * 100, "%")
accuracy2_RF = Random_Forest.score(X_test, y_test)
print("Accuracy of test data = ", accuracy2_RF * 100, "%")
```

```
Accuracy of train data =  100.0 %
Accuracy of test data =  90.74074074074075 %
```

After the completion of data preprocessing, feature engineering and label encoding, the accuracy of the random forest algorithm has been found and this has been increased compared with the implementation of original dataset.

RESULT:

Thus, I have implemented my module, DATA VISUALIZATION using different plots, FEATURE ENGINEERING by outlier detection and removal, adding new variables and encoding the categorical variables and model building using RANDOM FOREST. Thus, finally we got a dataset which gives better results.