PHASE 3

AI–POWERED DIABETES PREDICTION SYSTEM

LOADING AND PREPROCESSING THE DATASET:

Loading and preprocessing a dataset are essential steps in preparing data for machine learning. Once you have loaded your dataset, you may need to perform various types of analysis and preprocessing depending on the nature of your data and the problem you are trying to solve. Here's a general workflow for loading, preprocessing, and analyzing a dataset:

1. **Loading the Dataset:**

   - Load your dataset using appropriate libraries (e.g., `pandas` for CSV data, `json` for JSON data, or database connectors for SQL databases).

   - Inspect the first few rows of the dataset to understand its structure.

```python
import pandas as pd

# Load the dataset

df = pd.read_csv('your_dataset.csv')

# View the first few rows

print(df.head())
```

2. **Data Cleaning:**

   - Check for missing values and decide how to handle them (e.g., impute missing values or remove rows/columns).

   - Handle outliers or anomalies in the data.

   - Ensure consistency in data types (e.g., converting dates to a standardized format).

   - Remove duplicates if necessary.

3. **Data Exploration:**

- Calculate basic statistics, such as mean, median, standard deviation, and quartiles for numerical features.

- Visualize the data using histograms, scatter plots, and other charts to identify patterns and relationships between variables.

```python
# Basic statistics
print(df.describe())
# Data visualization
import matplotlib.pyplot as plt
df['feature'].hist()
plt.xlabel('Feature')
plt.ylabel('Frequency')
plt.title('Histogram of a Feature')
plt.show()
```

4. **Handling Categorical Data:**

- Encode categorical variables using techniques like one-hot encoding or label encoding.

- Check for class imbalances in target variables if you're working on a classification problem.

```python
# One-hot encoding for categorical variables
df_encoded = pd.get_dummies(df, columns=['categorical_feature'])
```

5. **Feature Scaling and Normalization:**

- Scale or normalize numerical features if required, especially when using algorithms sensitive to feature scales (e.g., gradient descent-based algorithms).

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['numeric_feature'] = scaler.fit_transform(df[['numeric_feature']])
```

6. **Feature Engineering:**

- Create new features or transform existing ones based on domain knowledge or insights gained during data exploration.

7. **Data Splitting:**

- Split the data into training and testing sets for model evaluation.

```python
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)
```

8. **Further Analysis:**

- Depending on your problem, you may need to perform additional analysis, such as time series decomposition, text preprocessing, or image augmentation.

9. **Data Visualization and Insights:**

- Use data visualization techniques to gain insights into your dataset and relationships between features.

10. **Iterative Process:**

- Data preprocessing is often an iterative process. You may need to revisit previous steps as you build and evaluate machine learning models.

Diabetes Prediction

Exploratory Data Analysis

In [1]:
```
# Importing the packages
import numpy as np
import pandas as pd
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error, r2_s
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import KFold
import warnings
warnings.simplefilter(action = "ignore")
sns.set()
plt.style.use('ggplot')
%matplotlib inline
```

In [2]:
```
# Reading the dataset
df = pd.read_csv('data/diabetes.csv')
```

In [3]:
```
# Printing the first 5 rows of the dataframe.
df.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [4]:
```
#Feature 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 consist of several medical predictor (independent) variables and one target (dependent) variable, Outcome. Independent variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

In [5]:
```
# Descriptive statistics of the data set
df.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
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 [6]:
```
# Print the size of the data set. It consists of 768 observation units and 9 variables
print("Dataset shape:", df.shape)
```

```
Dataset shape: (768, 9)
```

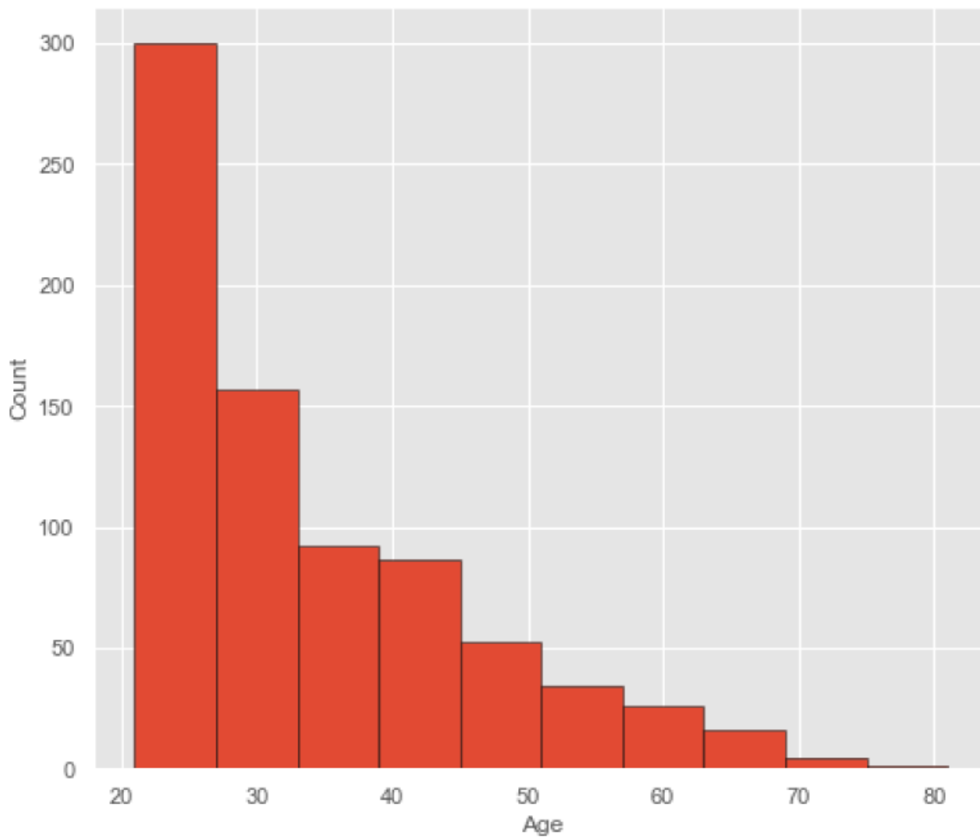
In [7]:
```
# Print the distribution of the Outcome variable.
df["Outcome"].value_counts()*100/len(df)
```

Out[7]:
```
0    65.104167
1    34.895833
Name: Outcome, dtype: float64
```

In [8]:
```
# Print the classes of the outcome variable.
df.Outcome.value_counts()
```

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

In [9]:
```
# Plot the histogram of the Age variable
plt.figure(figsize=(8,7))
plt.xlabel('Age', fontsize=12)
plt.ylabel('Count', fontsize=12)
df["Age"].hist(edgecolor = "black");
```

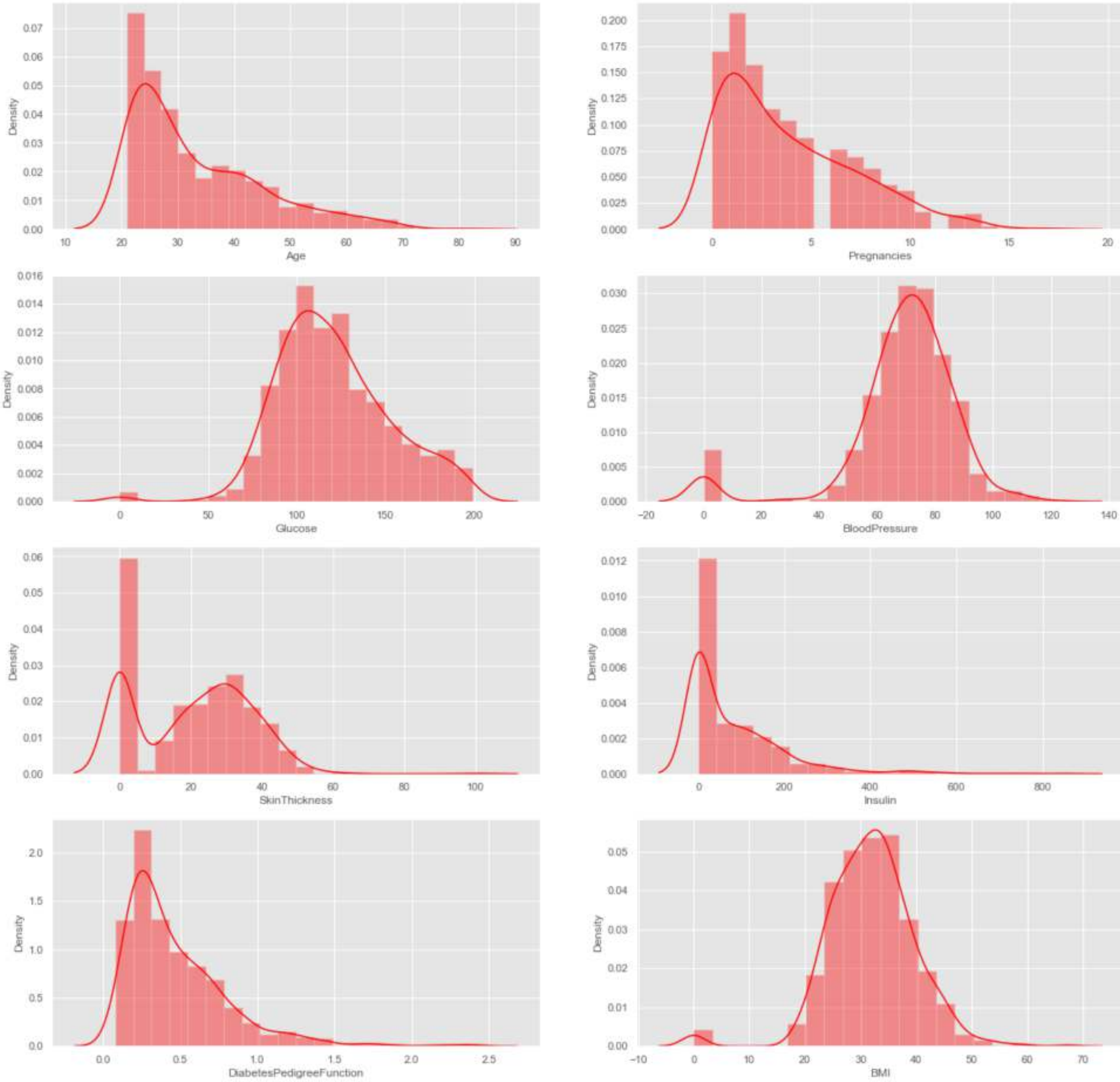


In [10]:
```
print("Max Age: " + str(df["Age"].max()) +','+ " Min Age: " + str(df["Age"].min()))
```

```
Max Age: 81, Min Age: 21
```

In [11]:
```
# Plot histogram and density graphs of all variables
fig, ax = plt.subplots(4,2, figsize=(20,20))
sns.distplot(df.Age, bins = 20, ax=ax[0,0], color="red")
sns.distplot(df.Pregnancies, bins = 20, ax=ax[0,1], color="red")
sns.distplot(df.Glucose, bins = 20, ax=ax[1,0], color="red")
sns.distplot(df.BloodPressure, bins = 20, ax=ax[1,1], color="red")
sns.distplot(df.SkinThickness, bins = 20, ax=ax[2,0], color="red")
sns.distplot(df.Insulin, bins = 20, ax=ax[2,1], color="red")
sns.distplot(df.DiabetesPedigreeFunction, bins = 20, ax=ax[3,0], color="red")
sns.distplot(df.BMI, bins = 20, ax=ax[3,1], color="red")
```

Out[11]:
```
<AxesSubplot:xlabel='BMI', ylabel='Density'>
```



In [12]:
```
df.groupby("Outcome").agg({"Pregnancies":"mean"})
```

Out[12]:

Outcome	Pregnancies
0	3.298000
1	4.865672

In [13]:
```
df.groupby("Outcome").agg({"Age":"mean"})
```

Out[13]:

Outcome	Age
0	31.190000
1	37.067164

In [14]:
```
df.groupby("Outcome").agg({"Age":"max"})
```

Out[14]:

	Age
Outcome	
0	81
1	70

In [15]:
```
df.groupby("Outcome").agg({"Insulin": "mean"})
```

Out[15]:

	Insulin
Outcome	
0	68.792000
1	100.335821

In [16]:
```
df.groupby("Outcome").agg({"Insulin": "max"})
```

Out[16]:

	Insulin
Outcome	
0	744
1	846

In [17]:
```
df.groupby("Outcome").agg({"Glucose": "mean"})
```

Out[17]:

	Glucose
Outcome	
0	109.980000
1	141.257463

In [18]:
```
df.groupby("Outcome").agg({"Glucose": "max"})
```

Out[18]:

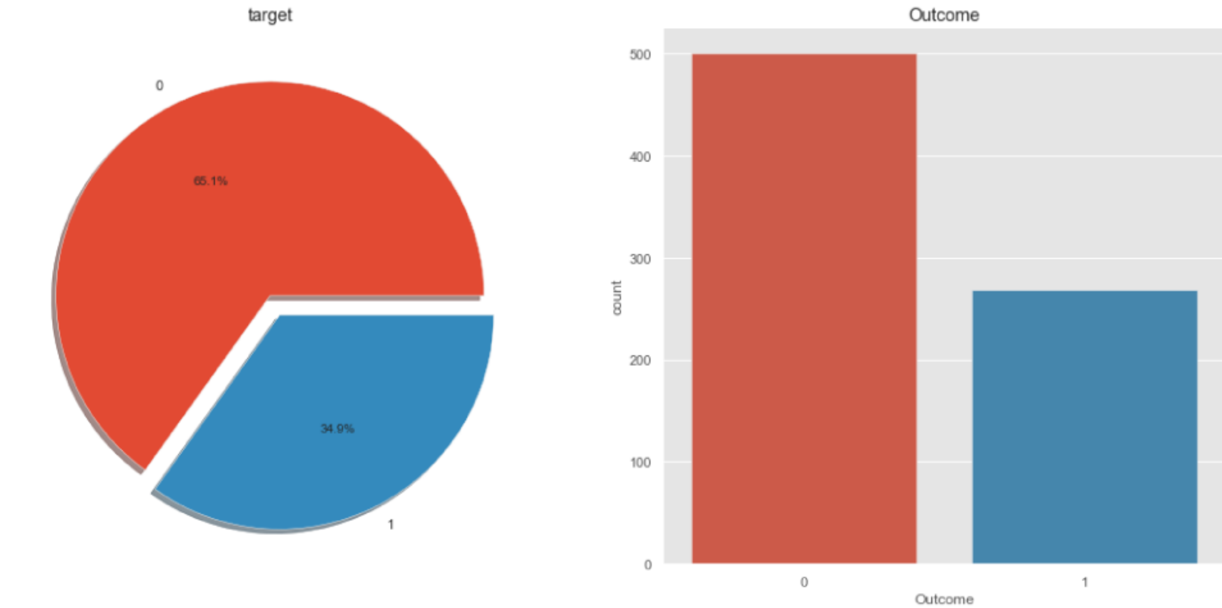
	Glucose
Outcome	
0	197
1	199

In [19]:
```
df.groupby("Outcome").agg({"BMI": "mean"})
```

Out[19]:

	BMI
Outcome	
0	30.304200
1	35.142537

In [20]:
```
# Visualize the distribution of the outcome variable in the data -> 0 - Healthy, 1 - D
f,ax=plt.subplots(1,2,figsize=(18,8))
df['Outcome'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadc
ax[0].set_title('target')
ax[0].set_ylabel('')
sns.countplot('Outcome',data=df,ax=ax[1])
ax[1].set_title('Outcome')
plt.show()
```

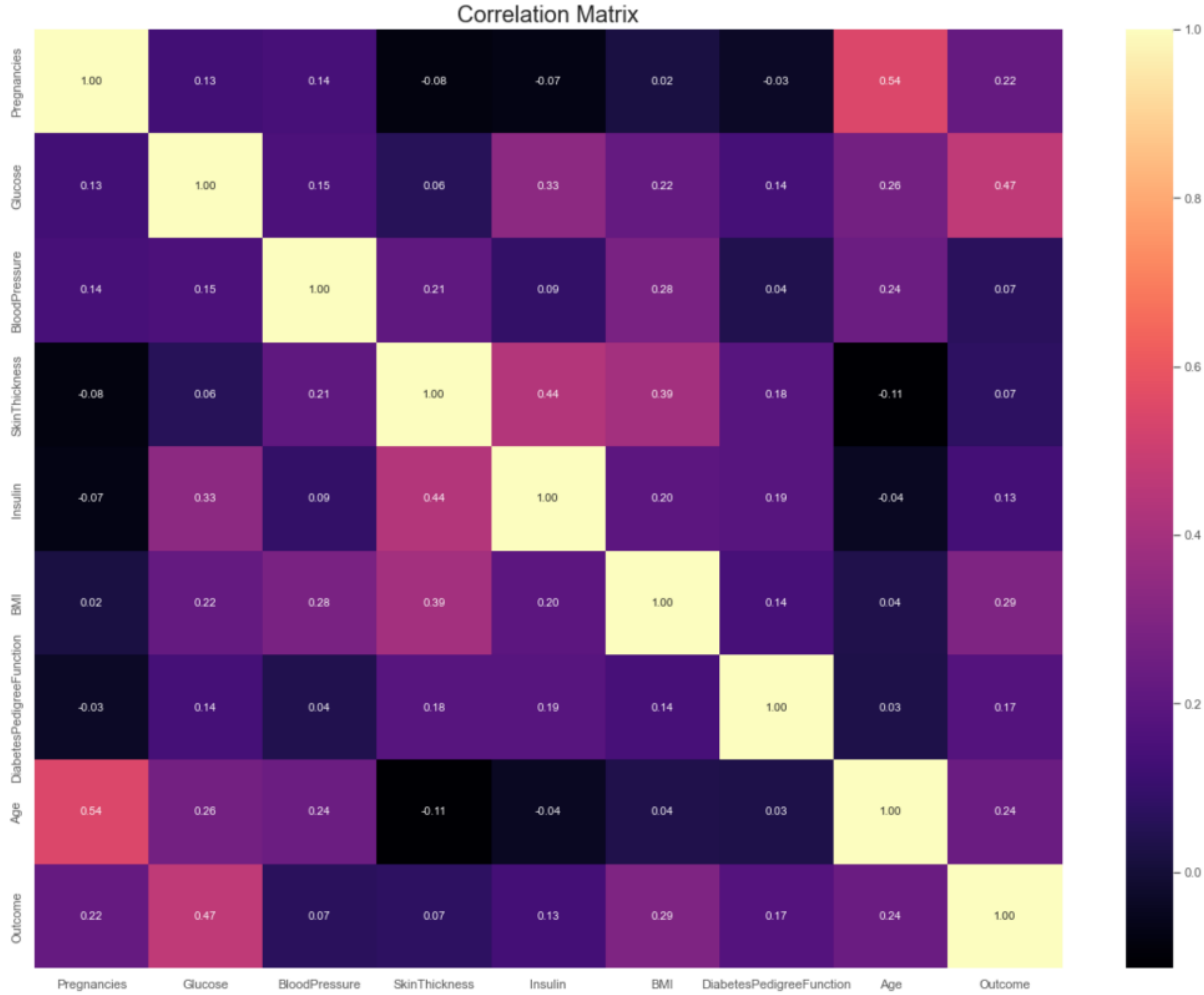


In [21]:
```
# corr() is used to find the pairwise correlation of all columns in the dataframe
df.corr()
```

Out[21]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695

In [22]:
```
# Correlation matrix of the data set
f, ax = plt.subplots(figsize= [20,15])
sns.heatmap(df.corr(), annot=True, fmt=".2f", ax=ax, cmap ='magma' )
ax.set_title("Correlation Matrix", fontsize=20)
#plt.savefig("corr.png", dpi=400)
plt.show()
```



Data Preprocessing

Missing Observation Analysis

We saw on df.head() that some features contain 0, it doesn't make sense here and this indicates missing value. Below we replace 0 value by NaN:

In [23]:
```
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[['Glucose','Blooc
```

In [24]:
```
df.head()
```

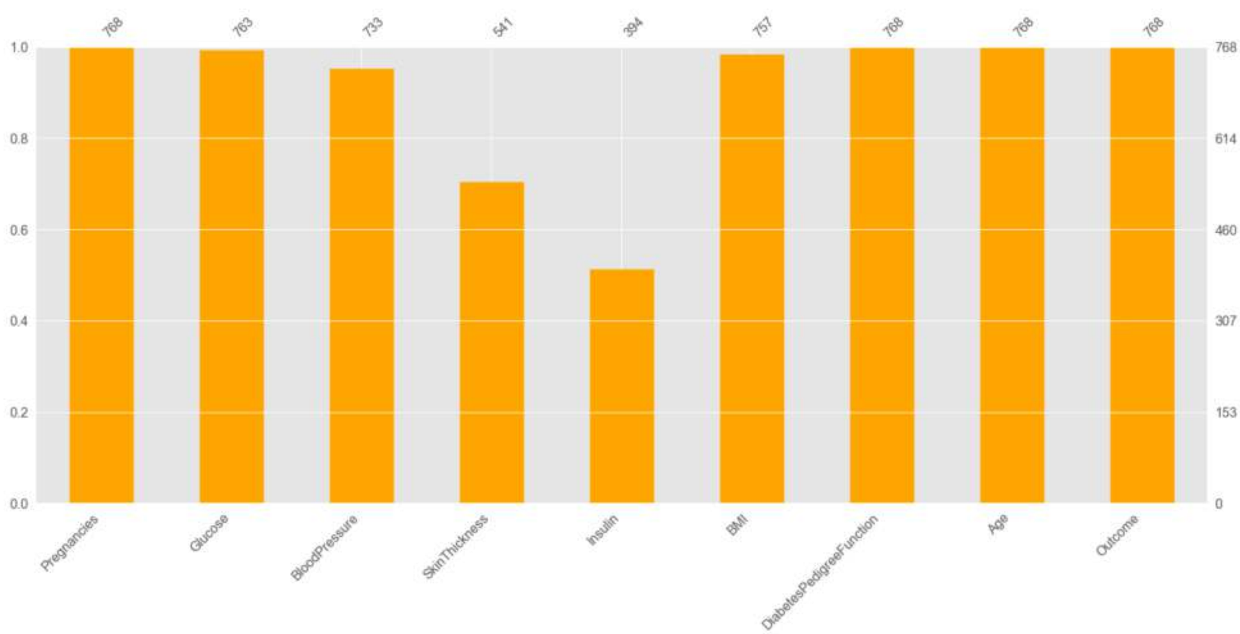
Out[24]:

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

In [25]:
```python
# Now, we can look at where are missing values
df.isnull().sum()
```

Out[25]:
```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

In [26]:
```python
# Visualizing the missing observations using the missingno library
import missingno as msno
msno.bar(df, color="orange");
```



In [27]:
```python
# The missing values will be filled with the median values of each variable
def median_target(var):
    temp = df[df[var].notnull()]
    temp = temp[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset_index()
    return temp
```

In [28]:
```python
# The values to be given for incomplete observations are given the median value of pec
columns = df.columns
```

```
columns = columns.drop("Outcome")
for i in columns:
    median_target(i)
    df.loc[(df['Outcome'] == 0 ) & (df[i].isnull()), i] = median_target(i)[i][0]
    df.loc[(df['Outcome'] == 1 ) & (df[i].isnull()), i] = median_target(i)[i][1]
```

In [29]:
```
df.head()
```

Out[29]:

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

In [30]:
```
# Number of missing values
df.isnull().sum()
```

Out[30]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Pair plot for clean data

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.

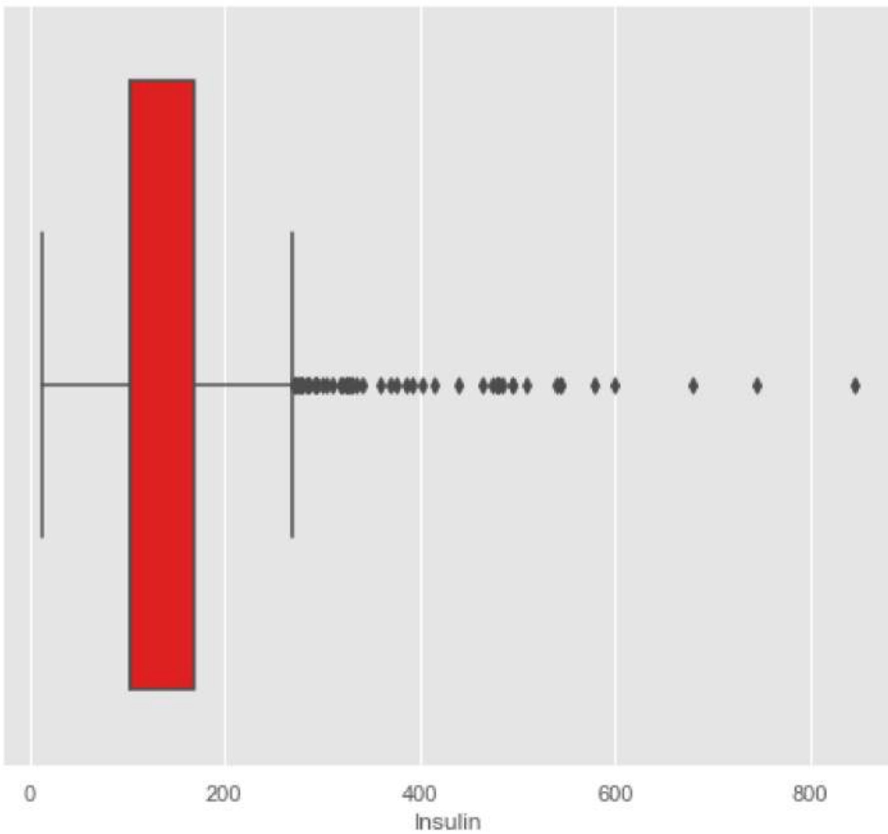
In [31]:
```
p=sns.pairplot(df, hue = 'Outcome')
```

Outlier Observation Analysis

In [32]:
```
for feature in df:

    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3-Q1
    lower = Q1- 1.5*IQR
    upper = Q3 + 1.5*IQR

    if df[(df[feature] > upper)].any(axis=None):
        print(feature,"yes")
    else:
        print(feature, "no")
```

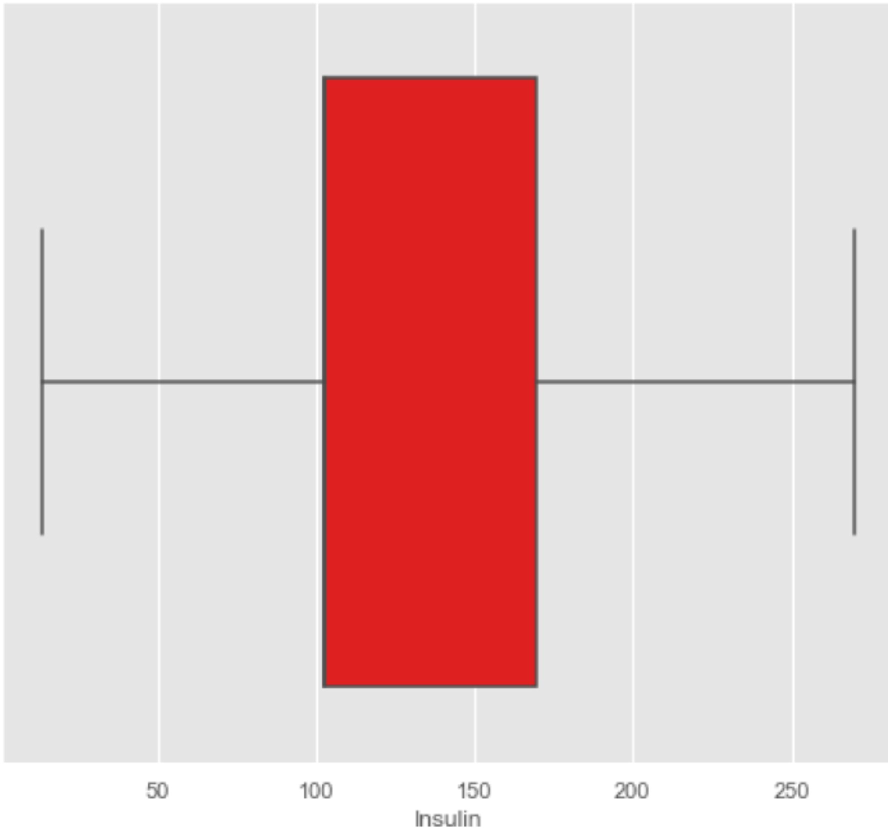
```
Pregnancies yes
Glucose no
BloodPressure yes
SkinThickness yes
Insulin yes
BMI yes
DiabetesPedigreeFunction yes
Age yes
Outcome no
```

In [33]:
```python
# Outlier observation of Insulin
import seaborn as sns
plt.figure(figsize=(8,7))
sns.boxplot(x = df["Insulin"], color="red");
```



In [34]:
```python
# Conducting a stand alone observation review for the Insulin variable
# Suppressing contradictory values
Q1 = df.Insulin.quantile(0.25)
Q3 = df.Insulin.quantile(0.75)
IQR = Q3-Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR
df.loc[df["Insulin"] > upper,"Insulin"] = upper
```

In [35]:
```python
import seaborn as sns
plt.figure(figsize=(8,7))
sns.boxplot(x = df["Insulin"], color="red");
```



Local Outlier Factor (LOF)

In [36]:
```
# Determining the outliers between all variables with the LOF method
from sklearn.neighbors import LocalOutlierFactor
lof =LocalOutlierFactor(n_neighbors= 10)
lof.fit_predict(df)
```

Out[36]:
```
array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1, -1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
       -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
       -1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1, -1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1])
```

In [37]:
```
df_scores = lof.negative_outlier_factor_
np.sort(df_scores)[0:30]
```

Out[37]:
```
array([-3.05893469, -2.37289269, -2.15297995, -2.09708735, -2.0772561 ,
       -1.95255968, -1.86384019, -1.74003158, -1.72703492, -1.71674689,
       -1.70343883, -1.6688722 , -1.64296768, -1.64190437, -1.61620872,
       -1.61369917, -1.60057603, -1.5988774 , -1.59608032, -1.57027568,
       -1.55876022, -1.55674614, -1.51852389, -1.50843907, -1.50280943,
       -1.50160698, -1.48391514, -1.4752983 , -1.4713427 , -1.47006248])
```

In [38]:
```
# Choosing the threshold value according to lof scores
threshold = np.sort(df_scores)[7]
threshold
```

Out[38]:
```
-1.740031580305444
```

In [39]:
```
# Deleting those that are higher than the threshold
outlier = df_scores > threshold
df = df[outlier]
```

In [40]:
```
# Examining the size of the data.
df.shape
```

Out[40]:
```
(760, 9)
```

Feature Engineering

Creating new variables is important for models. But we need to create a logical new variable. For this data set, some new variables were created according to BMI, Insulin and glucose variables.

In [41]:
```
# According to BMI, some ranges were determined and categorical variables were assigne
NewBMI = pd.Series(["Underweight", "Normal", "Overweight", "Obesity 1", "Obesity 2", '
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]
```

In [42]:
```
df.head()
```

Out[42]:

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

In [43]:
```
# 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"
```

In [44]:
```
# The operation performed was added to the dataframe.
df = df.assign(NewInsulinScore=df.apply(set_insulin, axis=1))

df.head()
```

Out[44]:

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

In [45]:

```
# Some intervals were determined according to the glucose variable and these were assi
NewGlucose = pd.Series(["Low", "Normal", "Overweight", "Secret", "High"], dtype = "cat
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]
```

In [46]:

```
df.head()
```

Out[46]:

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

One Hot Encoding

Categorical variables in the data set should be converted into numerical values. For this reason, these transformation processes are performed with Label Encoding and One Hot Encoding method.

In [47]:

```
# Here, by making One Hot Encoding transformation, categorical variables were converte
df = pd.get_dummies(df, columns =["NewBMI","NewInsulinScore", "NewGlucose"], drop_firs
```

In [48]:

```
df.head()
```

Out[48]:

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

In [49]:
```
categorical_df = df[['NewBMI_Obesity 1','NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'NewBM
                     'NewInsulinScore_Normal','NewGlucose_Low','NewGlucose_Normal', 'N
```

In [50]:
```
categorical_df.head()
```

Out[50]:

	NewBMI_Obesity 1	NewBMI_Obesity 2	NewBMI_Obesity 3	NewBMI_Overweight	NewBMI_Underweight
0	1	0	0	0	0
1	0	0	0	1	0
2	0	0	0	0	0
3	0	0	0	1	0
4	0	0	1	0	0

In [51]:
```
y = df["Outcome"]
X = df.drop(["Outcome",'NewBMI_Obesity 1','NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'New
                     'NewInsulinScore_Normal','NewGlucose_Low','NewGlucose_Normal', 'N
cols = X.columns
index = X.index
```

In [52]:
```
X.head()
```

Out[52]:

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

In [53]:
```
# The variables in the data set are an effective factor in increasing the performance
# There are multiple standardization methods. These are methods such as "Normalize", '
from sklearn.preprocessing import RobustScaler
transformer = RobustScaler().fit(X)
```

```
X = transformer.transform(X)
X = pd.DataFrame(X, columns = cols, index = index)
```

In [54]:
```
X.head()
```

Out[54]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctior
0	0.6	0.775	0.000	1.000000	1.000000	0.177778	0.66970
1	-0.4	-0.800	-0.375	0.142857	0.000000	-0.600000	-0.04951
2	1.0	1.650	-0.500	0.571429	1.000000	-0.966667	0.78697
3	-0.4	-0.700	-0.375	-0.714286	-0.126866	-0.433333	-0.52899
4	-0.6	0.500	-2.000	1.000000	0.977612	1.233333	4.99804

In [55]:
```
X = pd.concat([X,categorical_df], axis = 1)
```

In [56]:
```
X.head()
```

Out[56]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctior
0	0.6	0.775	0.000	1.000000	1.000000	0.177778	0.66970
1	-0.4	-0.800	-0.375	0.142857	0.000000	-0.600000	-0.04951
2	1.0	1.650	-0.500	0.571429	1.000000	-0.966667	0.78697
3	-0.4	-0.700	-0.375	-0.714286	-0.126866	-0.433333	-0.52899
4	-0.6	0.500	-2.000	1.000000	0.977612	1.233333	4.99804

In [57]:
```
y.head()
```

Out[57]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

In [58]:
```
# splitting data into training and test set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_sta
```

In [59]:
```
# scaling data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

LR

In [60]:
```
# fitting data to model

from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

Out[60]:
```
LogisticRegression()
```

In [61]:
```
# model predictions

y_pred = log_reg.predict(X_test)
```

In [62]:
```
# accuracy score

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

print(accuracy_score(y_train, log_reg.predict(X_train)))

log_reg_acc = accuracy_score(y_test, log_reg.predict(X_test))
print(log_reg_acc)
```

```
0.8402255639097744
0.881578947368421
```

In [63]:
```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[134  13]
 [ 14  67]]
```

In [64]:
```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.91      0.91       147
           1       0.84      0.83      0.83        81

    accuracy                           0.88       228
   macro avg       0.87      0.87      0.87       228
weighted avg       0.88      0.88      0.88       228
```

KNN

In [65]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

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

In [66]:
```
# model predictions

y_pred = knn.predict(X_test)
```

In [67]:
```
# accuracy score

print(accuracy_score(y_train, knn.predict(X_train)))

knn_acc = accuracy_score(y_test, knn.predict(X_test))
print(knn_acc)
```

```
0.8665413533834586
0.8333333333333334
```

In [68]:
```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[131  16]
 [ 22  59]]
```

In [69]:
```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.89      0.87       147
           1       0.79      0.73      0.76        81

    accuracy                           0.83       228
   macro avg       0.82      0.81      0.81       228
weighted avg       0.83      0.83      0.83       228
```

SVM

In [70]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

svc = SVC(probability=True)
parameters = {
    'gamma' : [0.0001, 0.001, 0.01, 0.1],
    'C' : [0.01, 0.05, 0.5, 0.1, 1, 10, 15, 20]
}

grid_search = GridSearchCV(svc, parameters)
grid_search.fit(X_train, y_train)
```

Out[70]:
```
GridSearchCV(estimator=SVC(probability=True),
             param_grid={'C': [0.01, 0.05, 0.5, 0.1, 1, 10, 15, 20],
                         'gamma': [0.0001, 0.001, 0.01, 0.1]})
```

In [71]:
```
# best parameters

grid_search.best_params_
```

Out[71]:
```
{'C': 1, 'gamma': 0.1}
```

In [72]:
```
# best score

grid_search.best_score_
```

Out[72]:
```
0.8665843766531477
```

In [73]:
```
svc = SVC(C = 1, gamma = 0.1, probability=True)
svc.fit(X_train, y_train)
```

Out[73]:
```
SVC(C=1, gamma=0.1, probability=True)
```

In [74]:
```
# model predictions

y_pred = svc.predict(X_test)
```

In [75]:
```
# accuracy score

print(accuracy_score(y_train, svc.predict(X_train)))

svc_acc = accuracy_score(y_test, svc.predict(X_test))
print(svc_acc)
```

```
0.8947368421052632
0.8421052631578947
```

In [76]:
```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[134  13]
 [ 23  58]]
```

In [77]:
```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.91      0.88       147
           1       0.82      0.72      0.76        81

    accuracy                           0.84       228
   macro avg       0.84      0.81      0.82       228
weighted avg       0.84      0.84      0.84       228
```

DT

In [78]:
```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of decision tree
```

```
dtc_acc = accuracy_score(y_test, dtc.predict(X_test))

print(f"Training Accuracy of Decision Tree Classifier is {accuracy_score(y_train, dtc.
print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, dtc.predict(X_test)
```

```
Training Accuracy of Decision Tree Classifier is 1.0
Test Accuracy of Decision Tree Classifier is 0.8245614035087719

Confusion Matrix :-
[[121  26]
 [ 14  67]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.90      0.82      0.86       147
           1       0.72      0.83      0.77        81

    accuracy                           0.82       228
   macro avg       0.81      0.83      0.81       228
weighted avg       0.83      0.82      0.83       228
```

In [79]:

```
# hyper parameter tuning of decision tree

from sklearn.model_selection import GridSearchCV
grid_param = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 10],
    'splitter' : ['best', 'random'],
    'min_samples_leaf' : [1, 2, 3, 5, 7],
    'min_samples_split' : [1, 2, 3, 5, 7],
    'max_features' : ['auto', 'sqrt', 'log2']
}

grid_search_dtc = GridSearchCV(dtc, grid_param, cv = 50, n_jobs = -1, verbose = 1)
grid_search_dtc.fit(X_train, y_train)
```

```
Fitting 50 folds for each of 1200 candidates, totalling 60000 fits
```

Out[79]:

```
GridSearchCV(cv=50, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 7, 10],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 2, 3, 5, 7],
                         'min_samples_split': [1, 2, 3, 5, 7],
                         'splitter': ['best', 'random']},
             verbose=1)
```

In [80]:

```
# best parameters and best score

print(grid_search_dtc.best_params_)
print(grid_search_dtc.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 7,
'min_samples_split': 5, 'splitter': 'best'}
0.8585454545454545
```

In [81]:
```
# best estimator

dtc = grid_search_dtc.best_estimator_

# accuracy score, confusion matrix and classification report of decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(X_test))

print(f"Training Accuracy of Decision Tree Classifier is {accuracy_score(y_train, dtc.
print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, dtc.predict(X_test)
```

```
Training Accuracy of Decision Tree Classifier is 0.8665413533834586
Test Accuracy of Decision Tree Classifier is 0.8947368421052632

Confusion Matrix :-
[[132  15]
 [  9  72]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.94      0.90      0.92       147
           1       0.83      0.89      0.86        81

    accuracy                           0.89       228
   macro avg       0.88      0.89      0.89       228
weighted avg       0.90      0.89      0.90       228
```

RF

In [103…
```
from sklearn.ensemble import RandomForestClassifier

rand_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 15, max_features
rand_clf.fit(X_train, y_train)
```

Out[103]:
```
RandomForestClassifier(criterion='entropy', max_depth=15, min_samples_leaf=2,
                       min_samples_split=3, n_estimators=130)
```

In [104…
```
y_pred = rand_clf.predict(X_test)
```

In [105…
```
# accuracy score

print(accuracy_score(y_train, rand_clf.predict(X_train)))

ran_clf_acc = accuracy_score(y_test, y_pred)
print(ran_clf_acc)
```

```
0.9830827067669173
0.9254385964912281
```

In [106…
```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[138   9]
 [  8  73]]
```

In [107…

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.94      0.94       147
           1       0.89      0.90      0.90        81

    accuracy                           0.93       228
   macro avg       0.92      0.92      0.92       228
weighted avg       0.93      0.93      0.93       228
```

GBDT

In [115…

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier()

parameters = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.001, 0.1, 1, 10],
    'n_estimators': [100, 150, 180, 200]
}

grid_search_gbc = GridSearchCV(gbc, parameters, cv = 10, n_jobs = -1, verbose = 1)
grid_search_gbc.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 32 candidates, totalling 320 fits
```

Out[115]:

```
GridSearchCV(cv=10, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'learning_rate': [0.001, 0.1, 1, 10],
                         'loss': ['deviance', 'exponential'],
                         'n_estimators': [100, 150, 180, 200]},
             verbose=1)
```

In [116…

```
# best parameters

grid_search_gbc.best_params_
```

Out[116]:

```
{'learning_rate': 0.1, 'loss': 'deviance', 'n_estimators': 180}
```

In [117…

```
# best score

grid_search_gbc.best_score_
```

Out[117]:

```
0.8834381551362684
```

In [118…

```
gbc = GradientBoostingClassifier(learning_rate = 0.1, loss = 'deviance', n_estimators
gbc.fit(X_train, y_train)
```

Out[118]:

```
GradientBoostingClassifier(n_estimators=180)
```

In [119…
```
y_pred = gbc.predict(X_test)
```

In [120…
```
# accuracy score

print(accuracy_score(y_train, gbc.predict(X_train)))

gbc_acc = accuracy_score(y_test, y_pred)
print(gbc_acc)
```

```
1.0
0.8903508771929824
```

In [121…
```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[136  11]
 [ 14  67]]
```

In [122…
```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       147
           1       0.86      0.83      0.84        81

    accuracy                           0.89       228
   macro avg       0.88      0.88      0.88       228
weighted avg       0.89      0.89      0.89       228
```

XGBoost

In [123…
```
from xgboost import XGBClassifier

xgb = XGBClassifier(objective = 'binary:logistic', learning_rate = 0.01, max_depth = 1

xgb.fit(X_train, y_train)
```

Out[123]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.01, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=10, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=180,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

In [124…
```
y_pred = xgb.predict(X_test)
```

In [125…
```
# accuracy score

print(accuracy_score(y_train, xgb.predict(X_train)))
```

```
xgb_acc = accuracy_score(y_test, y_pred)
print(xgb_acc)
```

```
0.9849624060150376
0.8771929824561403
```

In [126…

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[132  15]
 [ 13  68]]
```

In [127…

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.90      0.90       147
           1       0.82      0.84      0.83        81

    accuracy                           0.88       228
   macro avg       0.86      0.87      0.87       228
weighted avg       0.88      0.88      0.88       228
```

Model Comparison

In [132…

```
models = pd.DataFrame({
    'Model': ['Logistic Regression', 'KNN', 'SVM', 'Decision Tree Classifier', 'Random
    'Score': [100*round(log_reg_acc,4), 100*round(knn_acc,4), 100*round(svc_acc,4), 10
              100*round(gbc_acc,4), 100*round(xgb_acc,4)]
})
models.sort_values(by = 'Score', ascending = False)
```

Out[132]:

	Model	Score
4	Random Forest Classifier	92.54
3	Decision Tree Classifier	89.47
5	Gradient Boosting Classifier	89.04
0	Logistic Regression	88.16
6	XgBoost	87.72
2	SVM	84.21
1	KNN	83.33

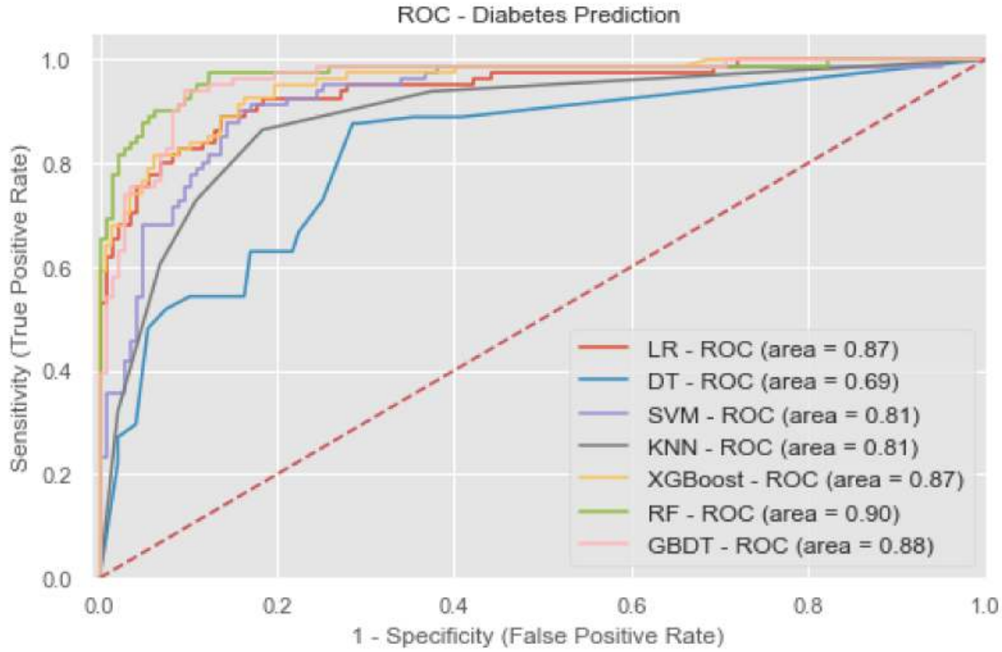
In [134…

```
import pickle
model = rand_clf
pickle.dump(model, open("models/diabetes.pkl",'wb'))
```

In [130…

```
from sklearn import metrics
plt.figure(figsize=(8,5))
```

```
models = [
{
    'label': 'LR',
    'model': log_reg,
},
{
    'label': 'DT',
    'model': dtc,
},
{
    'label': 'SVM',
    'model': svc,
},
{
    'label': 'KNN',
    'model': knn,
},
{
    'label': 'XGBoost',
    'model': xgb,
},
{
    'label': 'RF',
    'model': rand_clf,
},
{
    'label': 'GBDT',
    'model': gbc,
}
]
for m in models:
    model = m['model']
    model.fit(X_train, y_train)
    y_pred=model.predict(X_test)
    fpr1, tpr1, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_test)[:,1
    auc = metrics.roc_auc_score(y_test,model.predict(X_test))
    plt.plot(fpr1, tpr1, label='%s - ROC (area = %0.2f)' % (m['label'], auc))

plt.plot([0, 1], [0, 1],'r--')
plt.xlim([-0.01, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1 - Specificity (False Positive Rate)', fontsize=12)
plt.ylabel('Sensitivity (True Positive Rate)', fontsize=12)
plt.title('ROC - Diabetes Prediction', fontsize=12)
plt.legend(loc="lower right", fontsize=12)
plt.savefig("outputs/roc_diabetes.jpeg", format='jpeg', dpi=400, bbox_inches='tight')
plt.show()
```

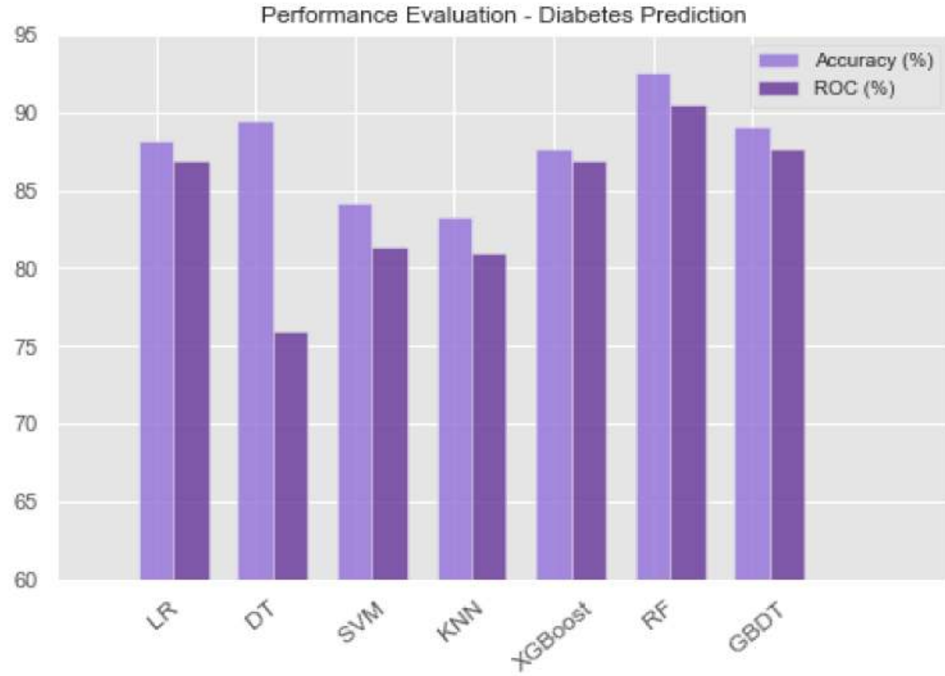


In [131…

```
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
models = [
{
    'label': 'LR',
    'model': log_reg,
},
{
    'label': 'DT',
    'model': dtc,
},
{
    'label': 'SVM',
    'model': svc,
},
{
    'label': 'KNN',
    'model': knn,
},
{
    'label': 'XGBoost',
    'model': xgb,
},
{
    'label': 'RF',
    'model': rand_clf,
},
{
    'label': 'GBDT',
    'model': gbc,
}
]

means_roc = []
means_accuracy = [100*round(log_reg_acc,4), 100*round(dtc_acc,4), 100*round(svc_acc,4)
                  100*round(ran_clf_acc,4), 100*round(gbc_acc,4)]
```

```
for m in models:
    model = m['model']
    model.fit(X_train, y_train)
    y_pred=model.predict(X_test)
    fpr1, tpr1, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_test)[:,1
    auc = metrics.roc_auc_score(y_test,model.predict(X_test))
    auc = 100*round(auc,4)
    means_roc.append(auc)

print(means_accuracy)
print(means_roc)

# data to plot
n_groups = 7
means_accuracy = tuple(means_accuracy)
means_roc = tuple(means_roc)

# create plot
fig, ax = plt.subplots(figsize=(8,5))
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, means_accuracy, bar_width,
alpha=opacity,
color='mediumpurple',
label='Accuracy (%)')

rects2 = plt.bar(index + bar_width, means_roc, bar_width,
alpha=opacity,
color='rebeccapurple',
label='ROC (%)')

plt.xlim([-1, 8])
plt.ylim([60, 95])

plt.title('Performance Evaluation - Diabetes Prediction', fontsize=12)
plt.xticks(index, ('   LR', '   DT', '   SVM', '   KNN', 'XGBoost' , '   RF', '   GBDT
plt.legend(loc="upper right", fontsize=10)
plt.savefig("outputs/PE_diabetes.jpeg", format='jpeg', dpi=400, bbox_inches='tight')
plt.show()
```

```
[88.16000000000001, 89.47, 84.21, 83.33, 87.72, 92.54, 89.03999999999999]
[86.94, 75.88000000000001, 81.38, 80.97999999999999, 86.87, 90.49000000000001, 87.62]
```


Performance Evaluation - Diabetes Prediction
95
90
85
80
75
70
65
60
Accuracy (%)
ROC (%)
LR
DT
SVM
KNN
XGBoost
RF
GBDT

In [ ]: