**AI BASED DIABETES PREDICTION SYSTEM**

**SUBMITTED BY**

**AJAY S – 710021106001**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**ANNA UNIVERSITY REGIONAL CAMPUS COIMBATORE**

**ABSTRACT**

Diabetes is one of the most deadly and chronic diseases which cause an increase in blood sugar. If diabetes remains untreated and unidentified many difficulties may arise due to that. The tedious work is in identifying the process which results in visiting the clinic and consulting the doctor. But this tedious work has been solved with the rise in the approaches used by machine learning. This project gives a comprehensive prospect of work accomplished to develop a model that can predict the possibility of diabetes in patients with extreme accuracy. Therefore, various machine learning classification algorithms namely genetic algorithm, decision tree, random forest, Logistic regression, SVM and Naive Bayes are used for detecting diabetes. Further we have done the comparison among various performances of all the different algorithms. The performances are categories of various measures like Precision, Accuracy, F-Measure and Recall. The project helps in identifying the algorithm to classify the risk of diabetes. Different techniques were applied to the algorithms for improving the robustness. Additionally, the findings suggest that the best performance of disease risk classification is done with the help of a genetic algorithm.

**PROBLEM STATEMENT**

The problem we aim to address is the early prediction of diabetes using Artificial Intelligence (AI) to improve healthcare outcomes. Diabetes is a widespread chronic disease with severe health implications, and early detection is crucial for effective management and prevention of complications.

The AI-based diabetes prediction system is a cutting-edge application of artificial intelligence and machine learning techniques aimed at early detection and management of diabetes. This system leverages data from various sources, including medical records, wearable devices, and lifestyle information, to provide accurate predictions of an individual's risk of developing diabetes. By identifying at-risk individuals and offering personalized recommendations, the system can help in preventing or managing diabetes effectively. This abstract outlines the key components and design thinking behind this innovative solution.

**Details about the dataset:**

The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

* **Pregnancies**: Number of times pregnant
* **Glucose**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
* **BloodPressure**: Diastolic 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)
* **DiabetesPedigreeFunction**: Diabetes pedigree function
* **Age**: Age (years)
* **Outcome**: Class variable (0 or 1)

**1) Exploratory Data Analysis**

In [1]:

*#Installation of required libraries*

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\_score, roc\_auc\_score, roc\_curve, classification\_report

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 lightgbm import LGBMClassifier

from sklearn.model\_selection import KFold

import warnings

warnings.simplefilter(action = "ignore")

In [2]:

*#Reading the dataset*

df **=** pd**.**read\_csv("../input/pima-indians-diabetes-database/diabetes.csv")

In [3]:

*# The first 5 observation units of the data set were accessed.*

df**.**head()

Out[3]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **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]:

*# The size of the data set was examined. It consists of 768 observation units and 9 variables.*

df**.**shape

Out[4]:

(768, 9)

In [5]:

*#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

In [6]:

*# Descriptive statistics of the data set accessed.*

df**.**describe([0.10,0.25,0.50,0.75,0.90,0.95,0.99])**.**T

Out[6]:

|  | **count** | **mean** | **std** | **min** | **10%** | **25%** | **50%** | **75%** | **90%** | **95%** | **99%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pregnancies** | 768.0 | 3.845052 | 3.369578 | 0.000 | 0.000 | 1.00000 | 3.0000 | 6.00000 | 9.0000 | 10.00000 | 13.00000 | 17.00 |
| **Glucose** | 768.0 | 120.894531 | 31.972618 | 0.000 | 85.000 | 99.00000 | 117.0000 | 140.25000 | 167.0000 | 181.00000 | 196.00000 | 199.00 |
| **BloodPressure** | 768.0 | 69.105469 | 19.355807 | 0.000 | 54.000 | 62.00000 | 72.0000 | 80.00000 | 88.0000 | 90.00000 | 106.00000 | 122.00 |
| **SkinThickness** | 768.0 | 20.536458 | 15.952218 | 0.000 | 0.000 | 0.00000 | 23.0000 | 32.00000 | 40.0000 | 44.00000 | 51.33000 | 99.00 |
| **Insulin** | 768.0 | 79.799479 | 115.244002 | 0.000 | 0.000 | 0.00000 | 30.5000 | 127.25000 | 210.0000 | 293.00000 | 519.90000 | 846.00 |
| **BMI** | 768.0 | 31.992578 | 7.884160 | 0.000 | 23.600 | 27.30000 | 32.0000 | 36.60000 | 41.5000 | 44.39500 | 50.75900 | 67.10 |
| **DiabetesPedigreeFunction** | 768.0 | 0.471876 | 0.331329 | 0.078 | 0.165 | 0.24375 | 0.3725 | 0.62625 | 0.8786 | 1.13285 | 1.69833 | 2.42 |
| **Age** | 768.0 | 33.240885 | 11.760232 | 21.000 | 22.000 | 24.00000 | 29.0000 | 41.00000 | 51.0000 | 58.00000 | 67.00000 | 81.00 |
| **Outcome** | 768.0 | 0.348958 | 0.476951 | 0.000 | 0.000 | 0.00000 | 0.0000 | 1.00000 | 1.0000 | 1.00000 | 1.00000 | 1.00 |

In [7]:

*# The distribution of the Outcome variable was examined.*

df["Outcome"]**.**value\_counts()**\***100**/**len(df)

Out[7]:

0 65.104167

1 34.895833

Name: Outcome, dtype: float64

In [8]:

*# The classes of the outcome variable were examined.*

df**.**Outcome**.**value\_counts()

Out[8]:

0 500

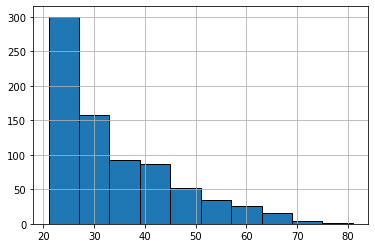
1 268

Name: Outcome, dtype: int64

In [9]:

*# The histagram of the Age variable was reached.*

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]:

*# Histogram and density graphs of all variables were accessed.*

fig, ax **=** plt**.**subplots(4,2, figsize**=**(16,16))

sns**.**distplot(df**.**Age, bins **=** 20, ax**=**ax[0,0])

sns**.**distplot(df**.**Pregnancies, bins **=** 20, ax**=**ax[0,1])

sns**.**distplot(df**.**Glucose, bins **=** 20, ax**=**ax[1,0])

sns**.**distplot(df**.**BloodPressure, bins **=** 20, ax**=**ax[1,1])

sns**.**distplot(df**.**SkinThickness, bins **=** 20, ax**=**ax[2,0])

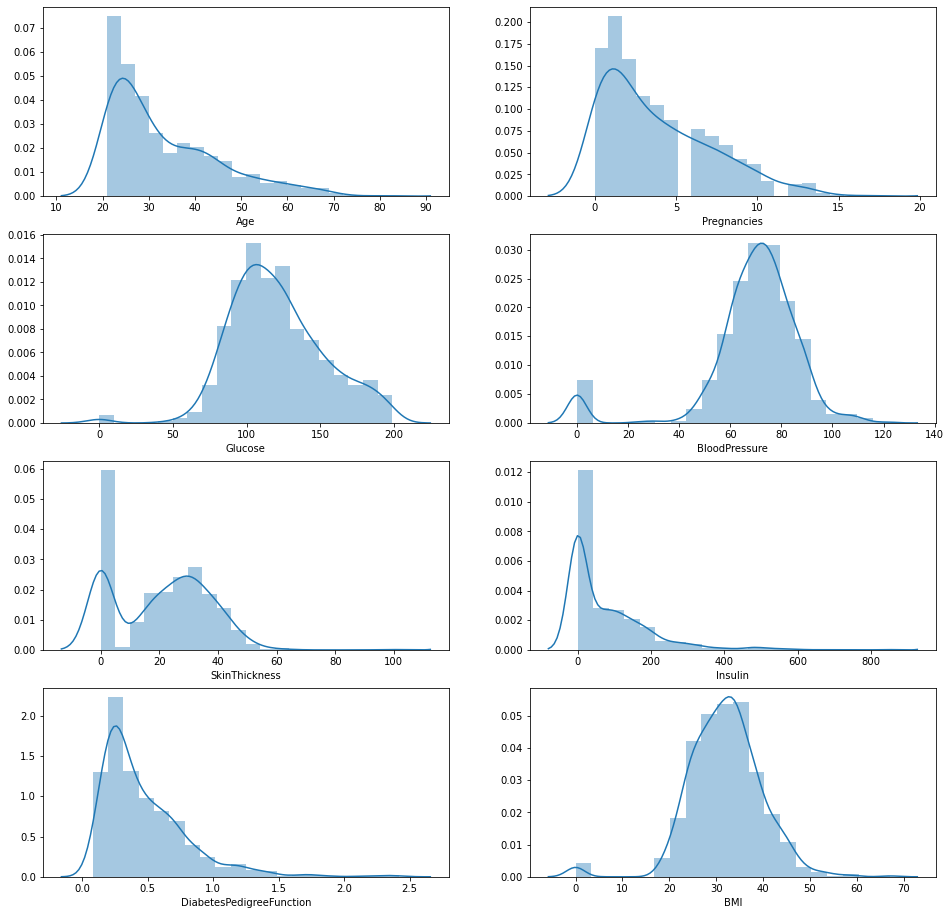
sns**.**distplot(df**.**Insulin, bins **=** 20, ax**=**ax[2,1])

sns**.**distplot(df**.**DiabetesPedigreeFunction, bins **=** 20, ax**=**ax[3,0])

sns**.**distplot(df**.**BMI, bins **=** 20, ax**=**ax[3,1])

Out[11]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f77b83d5950>



In [12]:

df**.**groupby("Outcome")**.**agg({"Pregnancies":"mean"})

Out[12]:

|  | **Pregnancies** |
| --- | --- |
| **Outcome** |  |
| **0** | 3.298000 |
| **1** | 4.865672 |

In [13]:

df**.**groupby("Outcome")**.**agg({"Age":"mean"})

Out[13]:

|  | **Age** |
| --- | --- |
| **Outcome** |  |
| **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]:

*# The distribution of the outcome variable in the data was examined and visualized.*

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],shadow**=True**)

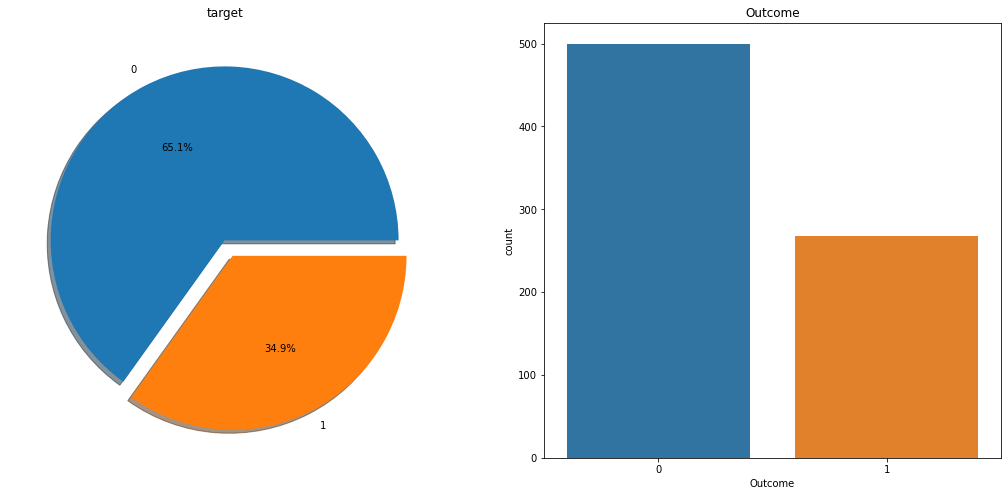
ax[0]**.**set\_title('target')

ax[0]**.**set\_ylabel('')

sns**.**countplot('Outcome',data**=**df,ax**=**ax[1])

ax[1]**.**set\_title('Outcome')

plt**.**show()



In [21]:

*# Access to the correlation of the data set was provided. What kind of relationship is examined between the variables.*

*# If the correlation value is> 0, there is a positive correlation. While the value of one variable increases, the value of the other variable also increases.*

*# Correlation = 0 means no correlation.*

*# If the correlation is <0, there is a negative correlation. While one variable increases, the other variable decreases.*

*# When the correlations are examined, there are 2 variables that act as a positive correlation to the Salary dependent variable.*

*# These variables are Glucose. As these increase, Outcome variable increases.*

df**.**corr()

Out[21]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pregnancies** | 1.000000 | 0.129459 | 0.141282 | -0.081672 | -0.073535 | 0.017683 | -0.033523 | 0.544341 | 0.221898 |
| **Glucose** | 0.129459 | 1.000000 | 0.152590 | 0.057328 | 0.331357 | 0.221071 | 0.137337 | 0.263514 | 0.466581 |
| **BloodPressure** | 0.141282 | 0.152590 | 1.000000 | 0.207371 | 0.088933 | 0.281805 | 0.041265 | 0.239528 | 0.065068 |
| **SkinThickness** | -0.081672 | 0.057328 | 0.207371 | 1.000000 | 0.436783 | 0.392573 | 0.183928 | -0.113970 | 0.074752 |
| **Insulin** | -0.073535 | 0.331357 | 0.088933 | 0.436783 | 1.000000 | 0.197859 | 0.185071 | -0.042163 | 0.130548 |
| **BMI** | 0.017683 | 0.221071 | 0.281805 | 0.392573 | 0.197859 | 1.000000 | 0.140647 | 0.036242 | 0.292695 |
| **DiabetesPedigreeFunction** | -0.033523 | 0.137337 | 0.041265 | 0.183928 | 0.185071 | 0.140647 | 1.000000 | 0.033561 | 0.173844 |
| **Age** | 0.544341 | 0.263514 | 0.239528 | -0.113970 | -0.042163 | 0.036242 | 0.033561 | 1.000000 | 0.238356 |
| **Outcome** | 0.221898 | 0.466581 | 0.065068 | 0.074752 | 0.130548 | 0.292695 | 0.173844 | 0.238356 | 1.000000 |

In [22]:

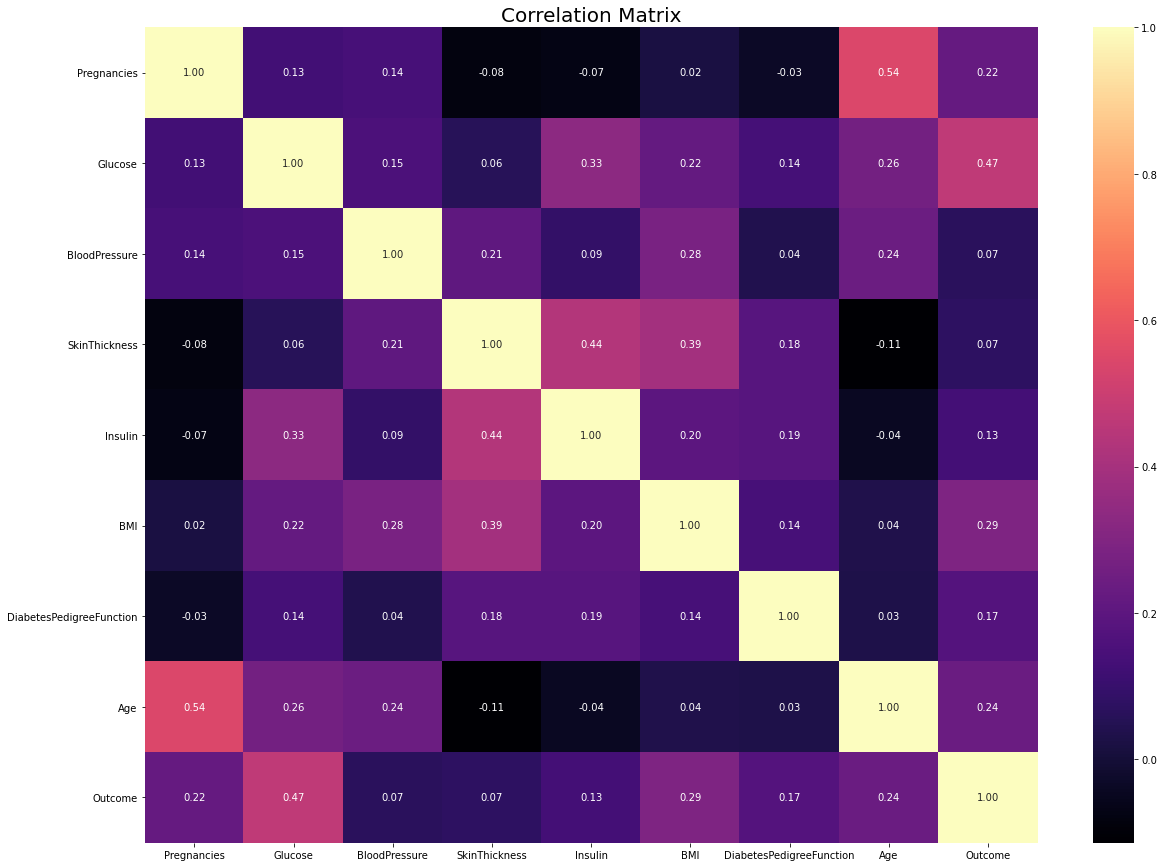
*# Correlation matrix graph of the data set*

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**.**show()



**2) Data Preprocessing**

**2.1) 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','BloodPressure','SkinThickness','Insulin','BMI']]**.**replace(0,np**.**NaN)

In [24]:

df**.**head()

Out[24]:

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

In [25]:

*# 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

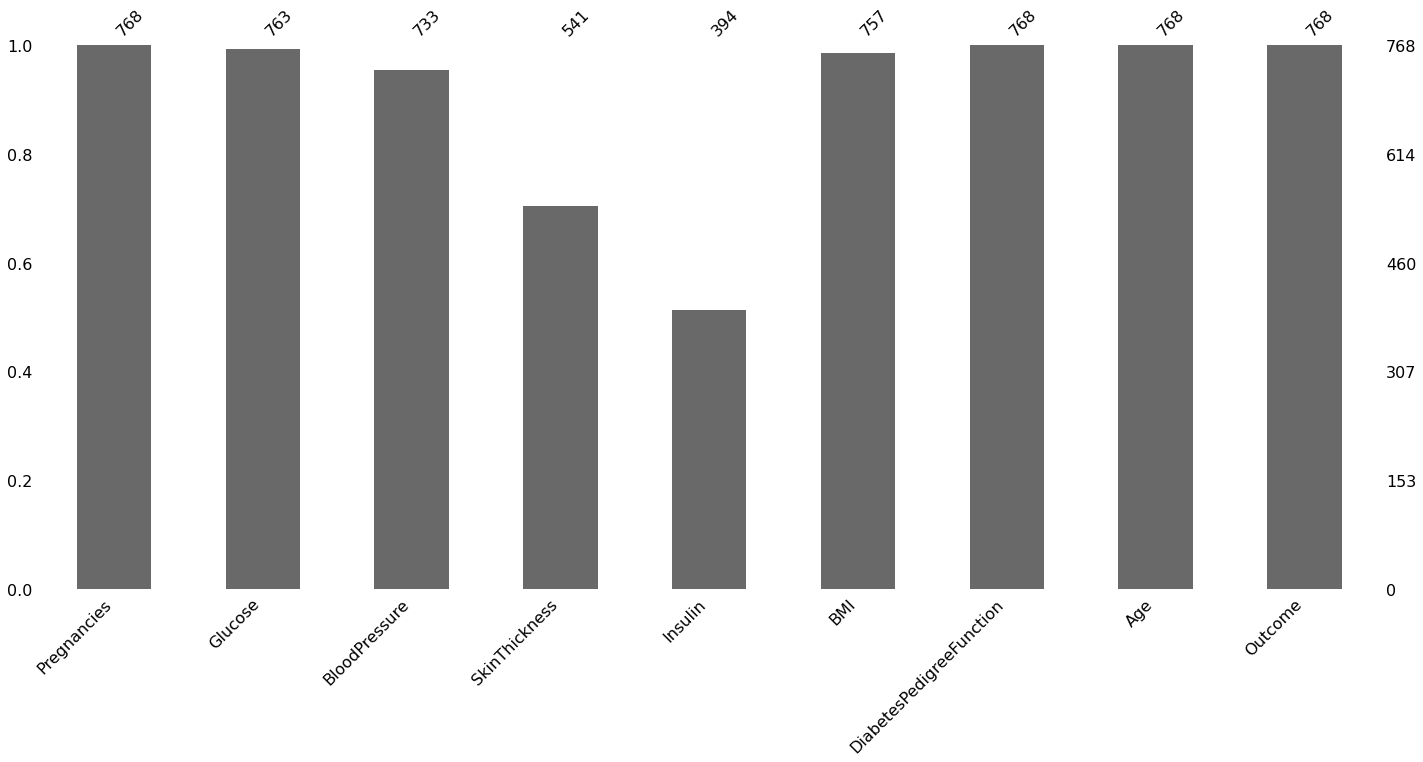
In [26]:

*# Have been visualized using the missingno library for the visualization of missing observations.*

*# Plotting*

**import** missingno **as** msno

msno**.**bar(df);



In [27]:

*# 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]:

*# The values to be given for incomplete observations are given the median value of people who are not sick and the median values of people who are sick.*

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** | **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 |
| **4** | 0 | 137.0 | 40.0 | 35.0 |  | 168.0 | 43.1 | 2.288 | 33 | 1 |

In [30]:

*# Missing values were filled.*

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

**2.2) Outlier Observation Analysis**

In [31]:

*# In the data set, there were asked whether there were any outlier observations compared to the 25% and 75% quarters.*

*# It was found to be an outlier observation.*

**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

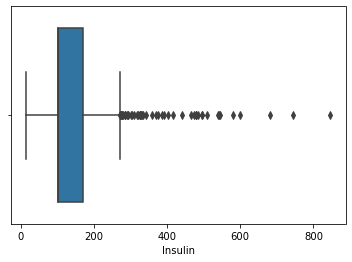
Outcome no

In [32]:

*# The process of visualizing the Insulin variable with boxplot method was done. We find the outlier observations on the chart.*

**import** seaborn **as** sns

sns**.**boxplot(x **=** df["Insulin"]);



In [33]:

*#We conduct a stand alone observation review for the Insulin variable*

*#We suppress contradictory values*

Q1 **=** df**.**Insulin**.**quantile(0.25)

Q3 **=** df**.**Insulin**.**quantile(0.75)

IQR **=** Q3**-**Q1

lower **=** Q1 **-** 1.5**\***IQR

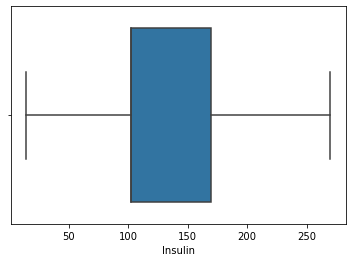
upper **=** Q3 **+** 1.5**\***IQR

df**.**loc[df["Insulin"] **>** upper,"Insulin"] **=** upper

In [34]:

**import** seaborn **as** sns

sns**.**boxplot(x **=** df["Insulin"]);



**2.3) Local Outlier Factor (LOF)**

In [35]:

*# We determine outliers between all variables with the LOF method*

from sklearn.neighbors import LocalOutlierFactor

lof =LocalOutlierFactor(n\_neighbors= 10)

lof.fit\_predict(df)

Out[35]:

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 [36]:

df\_scores **=** lof**.**negative\_outlier\_factor\_

np**.**sort(df\_scores)[0:30]

Out[36]:

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 [37]:

*#We choose the threshold value according to lof scores*

threshold **=** np**.**sort(df\_scores)[7]

threshold

Out[37]:

-1.740031580305444

In [38]:

*#We delete those that are higher than the threshold*

outlier **=** df\_scores **>** threshold

df **=** df[outlier]

In [39]:

*# The size of the data set was examined.*

df**.**shape

Out[39]:

(760, 9)

**3)Feature Engineering**

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

In [40]:

*# According to BMI, some ranges were determined and categorical variables were assigned.*

NewBMI **=** pd**.**Series(["Underweight", "Normal", "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]

In [41]:

df**.**head()

Out[41]:

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

In [42]:

*# 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 [43]:

*# The operation performed was added to the dataframe.*

df **=** df**.**assign(NewInsulinScore**=**df**.**apply(set\_insulin, axis**=**1))

df**.**head()

Out[43]:

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

In [44]:

*# Some intervals were determined according to the glucose variable and these were assigned categorical variables.*

NewGlucose **=** pd**.**Series(["Low", "Normal", "Overweight", "Secret", "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]

In [45]:

df**.**head()

Out[45]:

|  | **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 | Secret |
| **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 | Normal | Abnormal | Secret |
| **3** | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 | 0 | Overweight | Normal | Normal |
| **4** | 0 | 137.0 | 40.0 | 35.0 | 168.0 | 43.1 | 2.288 | 33 | 1 | Obesity 3 | Abnormal | Secret |

**4) 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 [46]:

*# Here, by making One Hot Encoding transformation, categorical variables were converted into numerical values. It is also protected from the Dummy variable trap.*

df **=** pd**.**get\_dummies(df, columns **=**["NewBMI","NewInsulinScore", "NewGlucose"], drop\_first **=** **True**)

In [47]:

df**.**head()

Out[47]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** | **NewBMI\_Obesity 1** | **NewBMI\_Obesity 2** | **NewBMI\_Obesity 3** | **NewBMI\_Overweight** | **NewBMI\_Underweight** | **NewInsulinScore\_Normal** | **NewGlucose\_Low** | **NewGlucose\_Normal** | **NewGlucose\_Overweight** | **NewGlucose\_Secret** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 6 | 148.0 | 72.0 | 35.0 | 169.5 | 33.6 | 0.627 | 50 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **1** | 1 | 85.0 | 66.0 | 29.0 | 102.5 | 26.6 | 0.351 | 31 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 8 | 183.0 | 64.0 | 32.0 | 169.5 | 23.3 | 0.672 | 32 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **3** | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **4** | 0 | 137.0 | 40.0 | 35.0 | 168.0 | 43.1 | 2.288 | 33 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

In [48]:

categorical\_df **=** df[['NewBMI\_Obesity 1','NewBMI\_Obesity 2', 'NewBMI\_Obesity 3', 'NewBMI\_Overweight','NewBMI\_Underweight',

'NewInsulinScore\_Normal','NewGlucose\_Low','NewGlucose\_Normal', 'NewGlucose\_Overweight', 'NewGlucose\_Secret']]

In [49]:

categorical\_df**.**head()

Out[49]:

|  | **NewBMI\_Obesity 1** | **NewBMI\_Obesity 2** | **NewBMI\_Obesity 3** | **NewBMI\_Overweight** | **NewBMI\_Underweight** | **NewInsulinScore\_Normal** | **NewGlucose\_Low** | **NewGlucose\_Normal** | **NewGlucose\_Overweight** | **NewGlucose\_Secret** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **1** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **3** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **4** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

In [50]:

y **=** df["Outcome"]

X **=** df**.**drop(["Outcome",'NewBMI\_Obesity 1','NewBMI\_Obesity 2', 'NewBMI\_Obesity 3', 'NewBMI\_Overweight','NewBMI\_Underweight',

'NewInsulinScore\_Normal','NewGlucose\_Low','NewGlucose\_Normal', 'NewGlucose\_Overweight', 'NewGlucose\_Secret'], axis **=** 1)

cols **=** X**.**columns

index **=** X**.**index

In [51]:

X**.**head()

Out[51]:

|  | **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 [52]:

*# The variables in the data set are an effective factor in increasing the performance of the models by standardization.*

*# There are multiple standardization methods. These are methods such as" Normalize"," MinMax"," Robust" and "Scale".*

**from** sklearn.preprocessing **import** RobustScaler

transformer **=** RobustScaler()**.**fit(X)

X **=** transformer**.**transform(X)

X **=** pd**.**DataFrame(X, columns **=** cols, index **=** index)

In [53]:

X**.**head()

Out[53]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.6 | 0.775 | 0.000 | 1.000000 | 1.000000 | 0.177778 | 0.669707 | 1.235294 |
| **1** | -0.4 | -0.800 | -0.375 | 0.142857 | 0.000000 | -0.600000 | -0.049511 | 0.117647 |
| **2** | 1.0 | 1.650 | -0.500 | 0.571429 | 1.000000 | -0.966667 | 0.786971 | 0.176471 |
| **3** | -0.4 | -0.700 | -0.375 | -0.714286 | -0.126866 | -0.433333 | -0.528990 | -0.470588 |
| **4** | -0.6 | 0.500 | -2.000 | 1.000000 | 0.977612 | 1.233333 | 4.998046 | 0.235294 |

In [54]:

X **=** pd**.**concat([X,categorical\_df], axis **=** 1)

In [55]:

X**.**head()

Out[55]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **NewBMI\_Obesity 1** | **NewBMI\_Obesity 2** | **NewBMI\_Obesity 3** | **NewBMI\_Overweight** | **NewBMI\_Underweight** | **NewInsulinScore\_Normal** | **NewGlucose\_Low** | **NewGlucose\_Normal** | **NewGlucose\_Overweight** | **NewGlucose\_Secret** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.6 | 0.775 | 0.000 | 1.000000 | 1.000000 | 0.177778 | 0.669707 | 1.235294 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **1** | -0.4 | -0.800 | -0.375 | 0.142857 | 0.000000 | -0.600000 | -0.049511 | 0.117647 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 1.0 | 1.650 | -0.500 | 0.571429 | 1.000000 | -0.966667 | 0.786971 | 0.176471 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **3** | -0.4 | -0.700 | -0.375 | -0.714286 | -0.126866 | -0.433333 | -0.528990 | -0.470588 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **4** | -0.6 | 0.500 | -2.000 | 1.000000 | 0.977612 | 1.233333 | 4.998046 | 0.235294 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

In [56]:

y**.**head()

Out[56]:

0 1

1 0

2 1

3 0

4 1

Name: Outcome, dtype: int64

**5) Base Models**

In [57]:

*# Validation scores of all base models*

models = []

models.append(('LR', LogisticRegression(random\_state = 12345)))

models.append(('KNN', KNeighborsClassifier()))

models.append(('CART', DecisionTreeClassifier(random\_state = 12345)))

models.append(('RF', RandomForestClassifier(random\_state = 12345)))

models.append(('SVM', SVC(gamma='auto', random\_state = 12345)))

models.append(('XGB', GradientBoostingClassifier(random\_state = 12345)))

models.append(("LightGBM", LGBMClassifier(random\_state = 12345)))

*# evaluate each model in turn*

results = []

names = []

In [58]:

**for** name, model **in** models:

kfold = KFold(n\_splits = 10, random\_state = 12345)

cv\_results = cross\_val\_score(model, X, y, cv = 10, scoring= "accuracy")

results.append(cv\_results)

names.append(name)

msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

print(msg)

*# boxplot algorithm comparison*

fig = plt.figure(figsize=(15,10))

fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results)

ax.set\_xticklabels(names)

plt.show()

LR: 0.848684 (0.036866)

KNN: 0.840789 (0.023866)

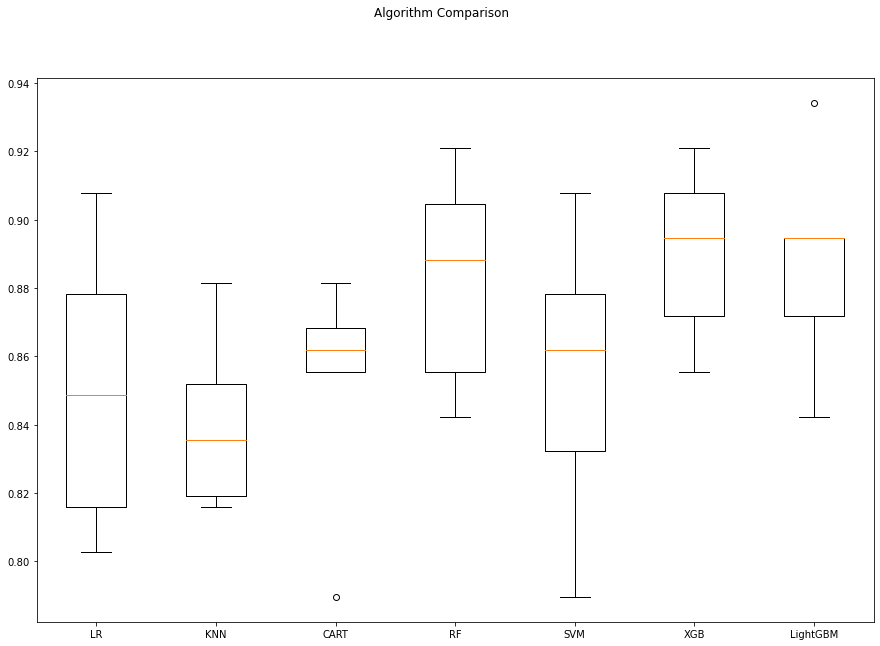
CART: 0.857895 (0.024826)

RF: 0.881579 (0.026316)

SVM: 0.853947 (0.036488)

XGB: 0.890789 (0.020427)

LightGBM: 0.885526 (0.024298)



**6) Model Tuning**

**1) Random Forests Tuning**

In [59]:

rf\_params **=** {"n\_estimators" :[100,200,500,1000],

"max\_features": [3,5,7],

"min\_samples\_split": [2,5,10,30],

"max\_depth": [3,5,8,**None**]}

In [60]:

rf\_model **=** RandomForestClassifier(random\_state **=** 12345)

In [61]:

gs\_cv **=** GridSearchCV(rf\_model,

rf\_params,

cv **=** 10,

n\_jobs **=** **-**1,

verbose **=** 2)**.**fit(X, y)

Fitting 10 folds for each of 192 candidates, totalling 1920 fits

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n\_jobs=-1)]: Done 33 tasks | elapsed: 10.4s

[Parallel(n\_jobs=-1)]: Done 154 tasks | elapsed: 48.2s

[Parallel(n\_jobs=-1)]: Done 357 tasks | elapsed: 1.9min

[Parallel(n\_jobs=-1)]: Done 640 tasks | elapsed: 3.5min

[Parallel(n\_jobs=-1)]: Done 1005 tasks | elapsed: 5.7min

[Parallel(n\_jobs=-1)]: Done 1450 tasks | elapsed: 8.4min

[Parallel(n\_jobs=-1)]: Done 1920 out of 1920 | elapsed: 11.4min finished

In [62]:

gs\_cv**.**best\_params\_

Out[62]:

{'max\_depth': 8,

'max\_features': 7,

'min\_samples\_split': 2,

'n\_estimators': 500}

**1.1) Final Model Installation**

In [63]:

rf\_tuned **=** RandomForestClassifier(**\*\***gs\_cv**.**best\_params\_)

In [64]:

rf\_tuned **=** rf\_tuned**.**fit(X,y)

In [65]:

cross\_val\_score(rf\_tuned, X, y, cv **=** 10)**.**mean()

Out[65]:

0.8921052631578947

In [66]:

feature\_imp **=** pd**.**Series(rf\_tuned**.**feature\_importances\_,

index**=**X**.**columns)**.**sort\_values(ascending**=False**)

sns**.**barplot(x**=**feature\_imp, y**=**feature\_imp**.**index)

plt**.**xlabel('Significance Score Of Variables')

plt**.**ylabel('Variables')

plt**.**title("Variable Severity Levels")

plt**.**show()

**2) LightGBM Tuning**

In [67]:

lgbm **=** LGBMClassifier(random\_state **=** 12345)

In [68]:

lgbm\_params **=** {"learning\_rate": [0.01, 0.03, 0.05, 0.1, 0.5],

"n\_estimators": [500, 1000, 1500],

"max\_depth":[3,5,8]}

In [69]:

gs\_cv **=** GridSearchCV(lgbm,

lgbm\_params,

cv **=** 10,

n\_jobs **=** **-**1,

verbose **=** 2)**.**fit(X, y)

Fitting 10 folds for each of 45 candidates, totalling 450 fits

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n\_jobs=-1)]: Done 33 tasks | elapsed: 3.7s

[Parallel(n\_jobs=-1)]: Done 154 tasks | elapsed: 25.3s

[Parallel(n\_jobs=-1)]: Done 357 tasks | elapsed: 57.4s

[Parallel(n\_jobs=-1)]: Done 450 out of 450 | elapsed: 1.1min finished

In [70]:

gs\_cv**.**best\_params\_

Out[70]:

{'learning\_rate': 0.01, 'max\_depth': 3, 'n\_estimators': 1000}

**2.1) Final Model Installation**

In [71]:

lgbm\_tuned **=** LGBMClassifier(**\*\***gs\_cv**.**best\_params\_)**.**fit(X,y)

In [72]:

cross\_val\_score(lgbm\_tuned, X, y, cv **=** 10)**.**mean()

Out[72]:

0.8960526315789474

In [73]:

feature\_imp **=** pd**.**Series(lgbm\_tuned**.**feature\_importances\_,

index**=**X**.**columns)**.**sort\_values(ascending**=False**)

sns**.**barplot(x**=**feature\_imp, y**=**feature\_imp**.**index)

plt**.**xlabel('Significance Score Of Variables')

plt**.**ylabel('Variables')

plt**.**title("Variable Severity Levels")

plt**.**show()

**3) XGBoost Tuning**

In [74]:

xgb **=** GradientBoostingClassifier(random\_state **=** 12345)

In [75]:

xgb\_params **=** {

"learning\_rate": [0.01, 0.1, 0.2, 1],

"min\_samples\_split": np**.**linspace(0.1, 0.5, 10),

"max\_depth":[3,5,8],

"subsample":[0.5, 0.9, 1.0],

"n\_estimators": [100,1000]}

In [76]:

xgb\_cv\_model **=** GridSearchCV(xgb,xgb\_params, cv **=** 10, n\_jobs **=** **-**1, verbose **=** 2)**.**fit(X, y)

Fitting 10 folds for each of 720 candidates, totalling 7200 fits

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n\_jobs=-1)]: Done 33 tasks | elapsed: 4.0s

[Parallel(n\_jobs=-1)]: Done 154 tasks | elapsed: 42.3s

[Parallel(n\_jobs=-1)]: Done 357 tasks | elapsed: 1.9min

[Parallel(n\_jobs=-1)]: Done 640 tasks | elapsed: 3.1min

[Parallel(n\_jobs=-1)]: Done 1005 tasks | elapsed: 5.4min

[Parallel(n\_jobs=-1)]: Done 1450 tasks | elapsed: 8.4min

[Parallel(n\_jobs=-1)]: Done 1977 tasks | elapsed: 11.6min

[Parallel(n\_jobs=-1)]: Done 2584 tasks | elapsed: 14.8min

[Parallel(n\_jobs=-1)]: Done 3273 tasks | elapsed: 19.3min

[Parallel(n\_jobs=-1)]: Done 4042 tasks | elapsed: 23.6min

[Parallel(n\_jobs=-1)]: Done 4893 tasks | elapsed: 28.7min

[Parallel(n\_jobs=-1)]: Done 5824 tasks | elapsed: 34.5min

[Parallel(n\_jobs=-1)]: Done 6837 tasks | elapsed: 40.9min

[Parallel(n\_jobs=-1)]: Done 7200 out of 7200 | elapsed: 43.2min finished

In [77]:

xgb\_cv\_model**.**best\_params\_

Out[77]:

{'learning\_rate': 0.1,

'max\_depth': 5,

'min\_samples\_split': 0.1,

'n\_estimators': 100,

'subsample': 1.0}

**3.1) Final Model Installation**

In [78]:

xgb\_tuned **=**

GradientBoostingClassifier(**\*\***xgb\_cv\_model**.**best\_params\_)**.**fit(X,y)

In [79]:

cross\_val\_score(xgb\_tuned, X, y, cv **=** 10)**.**mean()

Out[79]:

0.9013157894736843

In [80]:

feature\_imp **=** pd**.**Series(xgb\_tuned**.**feature\_importances\_,

index**=**X**.**columns)**.**sort\_values(ascending**=False**)

sns**.**barplot(x**=**feature\_imp, y**=**feature\_imp**.**index)

plt**.**xlabel('Significance Score Of Variables')

plt**.**ylabel('Variables')

plt**.**title("Variable Severity Levels")

plt**.**show()

**7) Comparison of Final Models**

In [81]:

models **=** []

models**.**append(('RF', RandomForestClassifier(random\_state **=** 12345, max\_depth **=** 8, max\_features **=** 7, min\_samples\_split **=** 2, n\_estimators **=** 500)))

models**.**append(('XGB', GradientBoostingClassifier(random\_state **=** 12345, learning\_rate **=** 0.1, max\_depth **=** 5, min\_samples\_split **=** 0.1, n\_estimators **=** 100, subsample **=** 1.0)))

models**.**append(("LightGBM", LGBMClassifier(random\_state **=** 12345, learning\_rate **=** 0.01, max\_depth **=** 3, n\_estimators **=** 1000)))

*# evaluate each model in turn*

results **=** []

names **=** []

In [82]:

**for** name, model **in** models:

kfold **=** KFold(n\_splits **=** 10, random\_state **=** 12345)

cv\_results **=** cross\_val\_score(model, X, y, cv **=** 10, scoring**=** "accuracy")

results**.**append(cv\_results)

names**.**append(name)

msg **=** "%s: %f (%f)" **%** (name, cv\_results**.**mean(), cv\_results**.**std())

print(msg)

*# boxplot algorithm comparison*

fig **=** plt**.**figure(figsize**=**(15,10))

fig**.**suptitle('Algorithm Comparison')

ax **=** fig**.**add\_subplot(111)

plt**.**boxplot(results)

ax**.**set\_xticklabels(names)

plt**.**show()

RF: 0.897368 (0.034211)

XGB: 0.901316 (0.028373)

LightGBM: 0.896053 (0.033000)

**8) Reporting**

The aim of this project was to create classification models for the diabetes data set and to predict whether a person is sick by establishing models and to obtain maximum validation scores in the established models. The work done is as follows:

1. Diabetes Data Set read.
2. With Exploratory Data Analysis; The data set's structural data were checked.

The types of variables in the dataset were examined. Size information of the dataset was accessed. The 0 values in the data set are missing values. Primarily these 0 values were replaced with NaN values. Descriptive statistics of the data set were examined.

1. Data Preprocessing section;

df for: The NaN values missing observations were filled with the median values of whether each variable was sick or not. The outliers were determined by LOF and dropped. The X variables were standardized with the rubost method..

1. During Model Building;

Logistic Regression, KNN, SVM, CART, Random Forests, XGBoost, LightGBM like using machine learning models Cross Validation Score were calculated. Later Random Forests, XGBoost, LightGBM hyperparameter optimizations optimized to increase Cross Validation value.

1. Result;

The model created as a result of XGBoost hyperparameter optimization became the model with the lowest Cross Validation Score value. (0.90)

**CONCLUSION**

The AI-based diabetes prediction system is a proactive solution to the problem of late diabetes diagnosis. By applying a design thinking approach, we have developed a system that leverages advanced AI techniques to predict diabetes risk early, enabling timely interventions and personalized healthcare. This system has the potential to improve the lives of individuals at risk of diabetes and enhance the efficiency of healthcare delivery. Early diagnosis is the key to better managing diabetes and reducing its long-term complications, and our AI system is a step forward in achieving this goal.