AI BASED DIABETES PREDICTION SYSTEM

NAME : SABASTIN SELVIN JERALD J

CONTEXT

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

CONTENT

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- <u>Pregnancies</u>: Number of times pregnant

- <u>Glucose</u>: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

- <u>Blood Pressure:</u> Diastolic blood pressure (mm Hg)

- Skin Thickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- Diabetes Pedigree Function: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

PROBLEM STATEMENT

Develop an AI-powered diabetes prediction system that leverages machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes, providing early risk assessment and personalized preventive measures.

PROBLEM DEFINITION

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

SOURCES

(a) Original owners: National Institute of Diabetes and Digestive and Kidney Diseases

(b) Donor of database: Vincent Sigillito (vgs@aplcen.apl.jhu.edu) Research Center, RMI Group LeaderApplied Physics Laboratory The Johns Hopkins University Johns Hopkins Road Laurel, MD 20707 (301) 953-6231

(c) Date received: 9 May 1990

DATA SET

https://www.kaggle.com/datasets/mathchi/diabetes-data-set

DATA PRE PROCESSING

Gather a diverse and comprehensive dataset containing relevant features for diabetes prediction.- Preprocess the data, which may include tasks like normalization, handling missing values, and feature engineering.

ENSEMBLE LEARNING METHODS

Use techniques like Random Forests, Gradient Boosting, or AdaBoost to create multiple models using subsets of the data.- Each model in the ensemble focuses on different aspects and learns from different parts of the data.

DEEP LEARNING ALGORITHM

- Implement a deep learning model, perhaps a feedforward neural network or a recurrent neural network (RNN), to capture complex relationships within the data.

CODING

```
[9]: import numpy as np
     import pandas as pd
     # import statsmodels.api as sm
     import seaborn as sns
```

```
[7]: #Reading the dataset
     df = pd.read_csv("diabetes.csv")
```

```
[8]: df.head()
```

```
[8]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
     0            6      148             72             35        0  33.6
     1            1       85             66             29        0  26.6
     2            8      183             64              0        0  23.3
     3            1       89             66             23       94  28.1
     4            0      137             40             35      168  43.1

        DiabetesPedigreeFunction  Age  Outcome
     0                     0.627   50        1
     1                     0.351   31        0
     2                     0.672   32        1
     3                     0.167   21        0
     4                     2.288   33        1
```

```
[10]: # The size of the data set was examined. It consists of 768 observation units␣
       ↪and 9 variables.
      df.shape
```

```
[10]: (768, 9)
```

```
[11]: #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
```

[13]:
```
# Descriptive statistics of the data set accessed.
df.describe([0.10,0.25,0.50,0.75,0.90,0.95,0.99]).T
```

[13]:
```
                          count        mean         std     min     10%  \
Pregnancies               768.0    3.845052    3.369578   0.000   0.000
Glucose                   768.0  120.894531   31.972618   0.000  85.000
BloodPressure             768.0   69.105469   19.355807   0.000  54.000
SkinThickness             768.0   20.536458   15.952218   0.000   0.000
Insulin                   768.0   79.799479  115.244002   0.000   0.000
BMI                       768.0   31.992578    7.884160   0.000  23.600
DiabetesPedigreeFunction  768.0    0.471876    0.331329   0.078   0.165
Age                       768.0   33.240885   11.760232  21.000  22.000
Outcome                   768.0    0.348958    0.476951   0.000   0.000

                               25%       50%        75%       90%        95%  \
Pregnancies                1.00000    3.0000    6.00000    9.0000   10.00000
Glucose                   99.00000  117.0000  140.25000  167.0000  181.00000
BloodPressure             62.00000   72.0000   80.00000   88.0000   90.00000
SkinThickness              0.00000   23.0000   32.00000   40.0000   44.00000
Insulin                    0.00000   30.5000  127.25000  210.0000  293.00000
BMI                       27.30000   32.0000   36.60000   41.5000   44.39500
DiabetesPedigreeFunction   0.24375    0.3725    0.62625    0.8786    1.13285
Age                       24.00000   29.0000   41.00000   51.0000   58.00000
Outcome                    0.00000    0.0000    1.00000    1.0000    1.00000

                                99%     max
Pregnancies                13.00000   17.00
Glucose                   196.00000  199.00
BloodPressure             106.00000  122.00
SkinThickness              51.33000   99.00
Insulin                   519.90000  846.00
BMI                        50.75900   67.10
DiabetesPedigreeFunction    1.69833    2.42
Age                        67.00000   81.00
Outcome                     1.00000    1.00
```

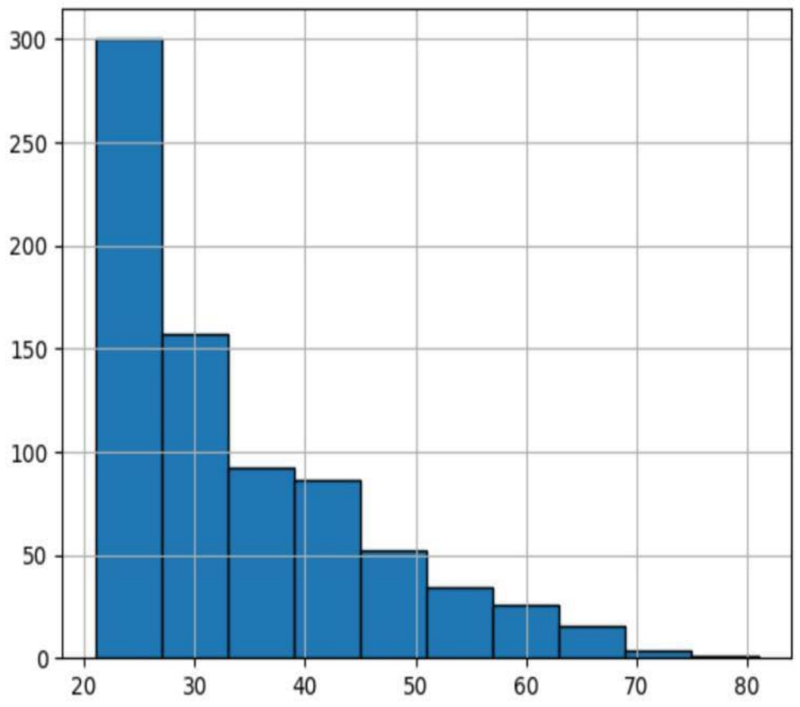
```
[14]: # The distribution of the Outcome variable was examined.
      df["Outcome"].value_counts()*100/len(df)
```

```
[14]: Outcome
      0    65.104167
      1    34.895833
      Name: count, dtype: float64
```

```
[15]: # The classes of the outcome variable were examined.
      df.Outcome.value_counts()
```

```
[15]: Outcome
      0    500
      1    268
      Name: count, dtype: int64
```

```
[16]: # The histogram of the Age variable was reached.
      df["Age"].hist(edgecolor = "black");
```



```
[18]: print("Max Age: " + str(df["Age"].max()) + " Min Age: " + str(df["Age"].min()))
```

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

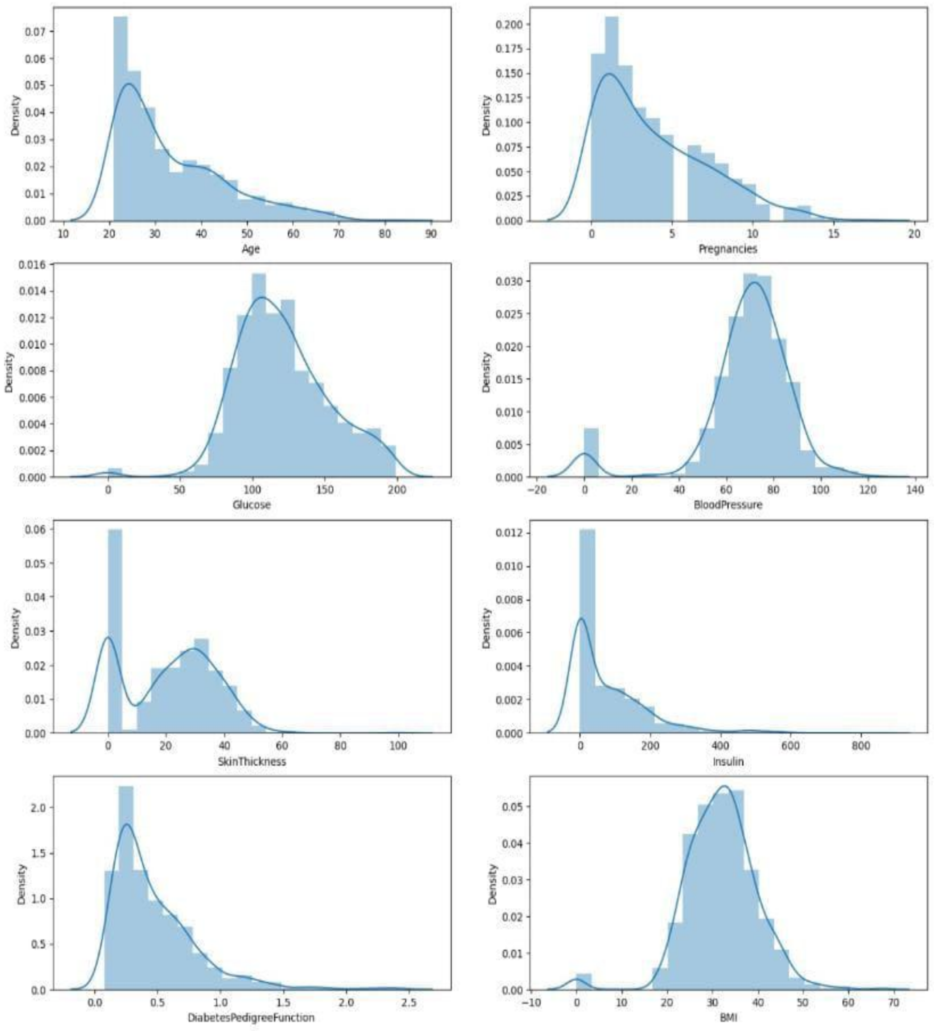
[24]:
```python
# Histogram and density graphs of all variables were accessed.
import matplotlib.pyplot as plt

# import statsmodels.api as sm
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])
```

```
C:\Users\AJITHKUMAR\AppData\Local\Temp\ipykernel_9092\1084607558.py:6:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.Age, bins = 20, ax=ax[0,0])
C:\Users\AJITHKUMAR\AppData\Local\Temp\ipykernel_9092\1084607558.py:7:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.Pregnancies, bins = 20, ax=ax[0,1])
C:\Users\AJITHKUMAR\AppData\Local\Temp\ipykernel_9092\1084607558.py:8:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
```

```
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.Glucose, bins = 20, ax=ax[1,0])
C:\Users\AJITHKUMAR\AppData\Local\Temp\ipykernel_9092\1084607558.py:9:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.BloodPressure, bins = 20, ax=ax[1,1])
C:\Users\AJITHKUMAR\AppData\Local\Temp\ipykernel_9092\1084607558.py:10:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.SkinThickness, bins = 20, ax=ax[2,0])
C:\Users\AJITHKUMAR\AppData\Local\Temp\ipykernel_9092\1084607558.py:11:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.Insulin, bins = 20, ax=ax[2,1])
C:\Users\AJITHKUMAR\AppData\Local\Temp\ipykernel_9092\1084607558.py:12:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
```

```
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.DiabetesPedigreeFunction, bins = 20, ax=ax[3,0])
C:\Users\AJITHKUMAR\AppData\Local\Temp\ipykernel_9092\1084607558.py:13:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

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

```
[24]: <Axes: xlabel='BMI', ylabel='Density'>
```



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

```
[25]:          Pregnancies
      Outcome
      0           3.298000
      1           4.865672
```

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

```
[26]:                Age
      Outcome
      0          31.190000
      1          37.067164
```

[27]:
```
df.groupby("Outcome").agg({"Age":"max"})
```

[27]:
```
         Age
Outcome
0         81
1         70
```

[28]:
```
df.groupby("Outcome").agg({"Insulin": "mean"})
```

[28]:
```
            Insulin
Outcome
0         68.792000
1        100.335821
```

[29]:
```
df.groupby("Outcome").agg({"Insulin": "max"})
```

[29]:
```
         Insulin
Outcome
0            744
1            846
```

[30]:
```
df.groupby("Outcome").agg({"Glucose": "mean"})
```

[30]:
```
            Glucose
Outcome
0        109.980000
1        141.257463
```

[31]:
```
df.groupby("Outcome").agg({"Glucose": "max"})
```

[31]:
```
         Glucose
Outcome
0            197
1            199
```

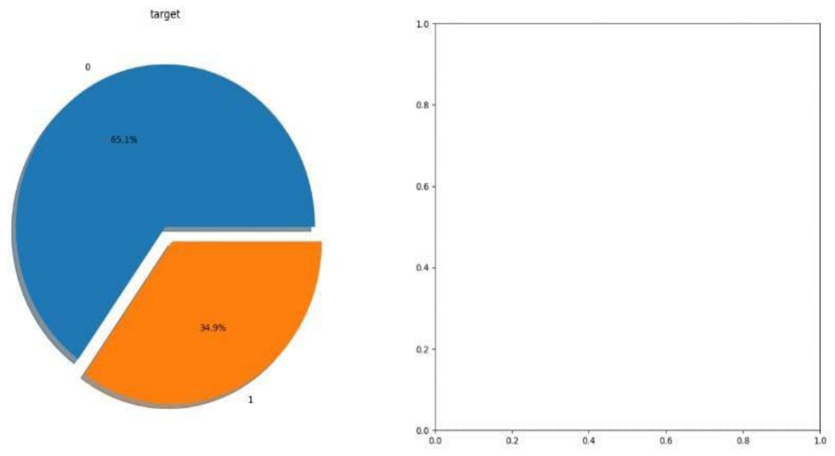
[32]:
```
df.groupby("Outcome").agg({"BMI": "mean"})
```

[32]:
```
               BMI
Outcome
0        30.304200
1        35.142537
```

[33]:
```
# 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()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[33], line 6
      4 ax[0].set_title('target')
      5 ax[0].set_ylabel('')
----> 6 sns.countplot('Outcome',data=df,ax=ax[1])
      7 ax[1].set_title('Outcome')
      8 plt.show()

TypeError: countplot() got multiple values for argument 'data'
```

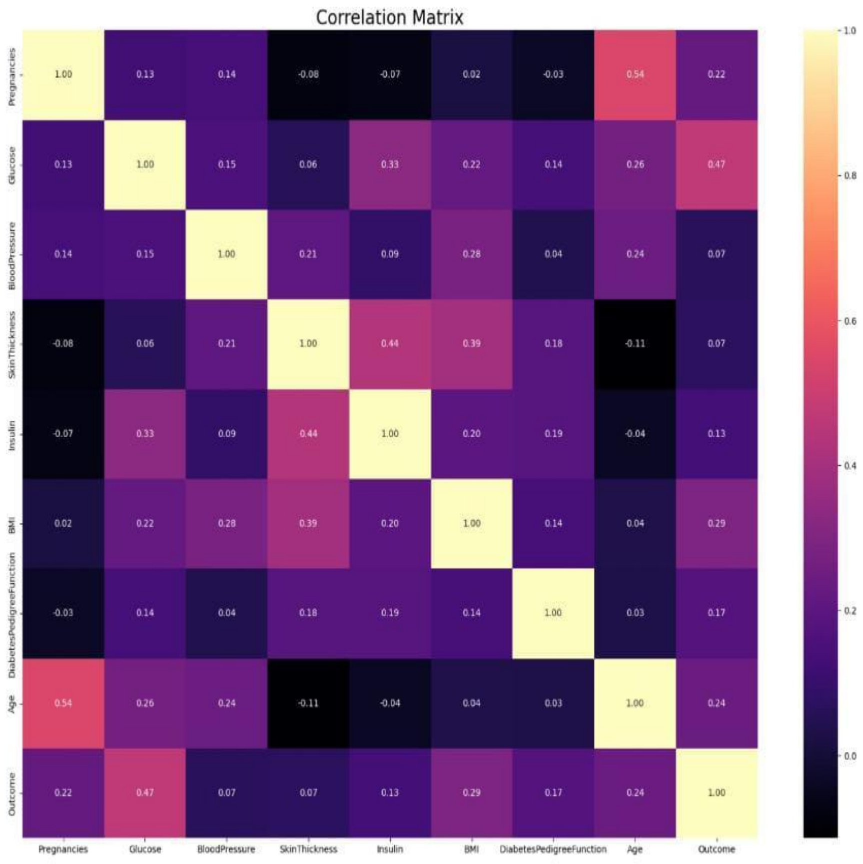


```
[34]: df.corr()
```

```
[34]:                           Pregnancies   Glucose  BloodPressure  SkinThickness  \
      Pregnancies                  1.000000  0.129459       0.141282      -0.081672
      Glucose                      0.129459  1.000000       0.152590       0.057328
      BloodPressure                0.141282  0.152590       1.000000       0.207371
      SkinThickness               -0.081672  0.057328       0.207371       1.000000
      Insulin                     -0.073535  0.331357       0.088933       0.436783
      BMI                          0.017683  0.221071       0.281805       0.392573
      DiabetesPedigreeFunction    -0.033523  0.137337       0.041265       0.183928
      Age                          0.544341  0.263514       0.239528      -0.113970
```

```
Outcome                    0.221898  0.466581       0.065068       0.074752

                            Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies               -0.073535  0.017683                 -0.033523
Glucose                    0.331357  0.221071                  0.137337
BloodPressure              0.088933  0.281805                  0.041265
SkinThickness              0.436783  0.392573                  0.183928
Insulin                    1.000000  0.197859                  0.185071
BMI                        0.197859  1.000000                  0.140647
DiabetesPedigreeFunction   0.185071  0.140647                  1.000000
Age                       -0.042163  0.036242                  0.033561
Outcome                    0.130548  0.292695                  0.173844

                                Age   Outcome
Pregnancies                0.544341  0.221898
Glucose                    0.263514  0.466581
BloodPressure              0.239528  0.065068
SkinThickness             -0.113970  0.074752
Insulin                   -0.042163  0.130548
BMI                        0.036242  0.292695
DiabetesPedigreeFunction   0.033561  0.173844
Age                        1.000000  0.238356
Outcome                    0.238356  1.000000
```

[35]:
```
# 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()
```



```
[36]: df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] =␣
      ↪df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.
      ↪NaN)
      df.head()
```

```
[36]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
     0             6    148.0           72.0           35.0      NaN  33.6
     1             1     85.0           66.0           29.0      NaN  26.6
     2             8    183.0           64.0            NaN      NaN  23.3
     3             1     89.0           66.0           23.0     94.0  28.1
     4             0    137.0           40.0           35.0    168.0  43.1

         DiabetesPedigreeFunction  Age  Outcome
     0                      0.627   50        1
     1                      0.351   31        0
     2                      0.672   32        1
     3                      0.167   21        0
```

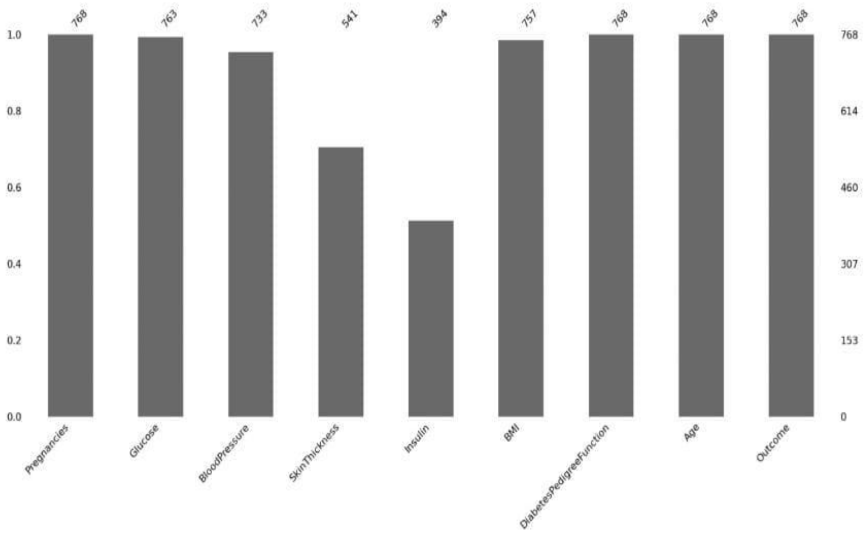
```
4                        2.288   33        1
```

[37]:
```
# Now, we can look at where are missing values
df.isnull().sum()
```

[37]:
```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

[45]:
```
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")
```

[47]:
```
# Have been visualized using the missingno library for the visualization of␣
 ↪missing observations.
# Plotting
import missingno as msno
msno.bar(df);
```



```
[48]: # 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
```

```
[49]: # 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]
```

```
[50]: df.head()
```

```
[50]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
      0            6    148.0           72.0           35.0    169.5  33.6
      1            1     85.0           66.0           29.0    102.5  26.6
      2            8    183.0           64.0           32.0    169.5  23.3
      3            1     89.0           66.0           23.0     94.0  28.1
      4            0    137.0           40.0           35.0    168.0  43.1

         DiabetesPedigreeFunction  Age  Outcome
      0                     0.627   50        1
```

```
1                      0.351   31         0
2                      0.672   32         1
3                      0.167   21         0
4                      2.288   33         1
```

[51]:
```
# Missing values were filled.
df.isnull().sum()
```

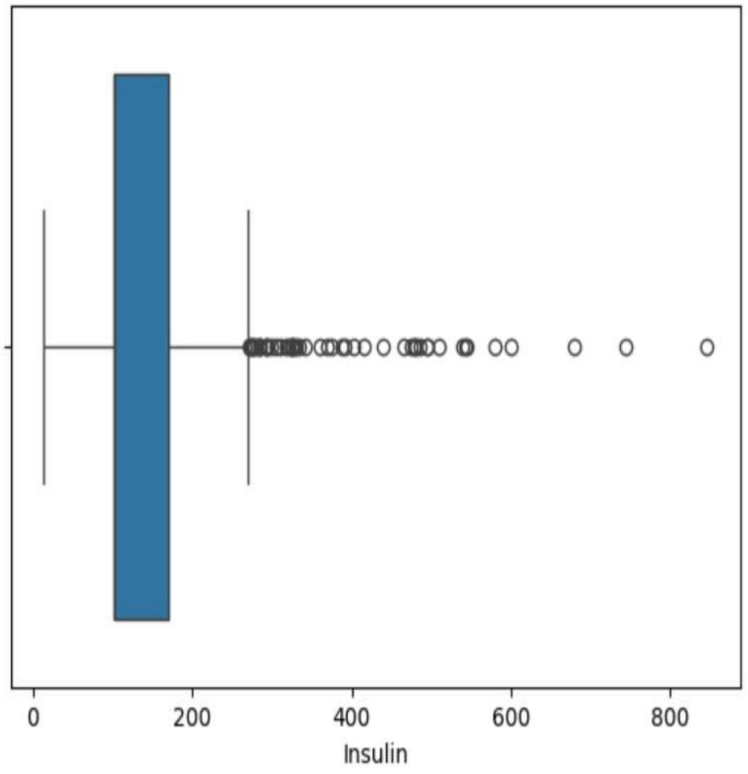
[51]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[52]:
```
# 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
```

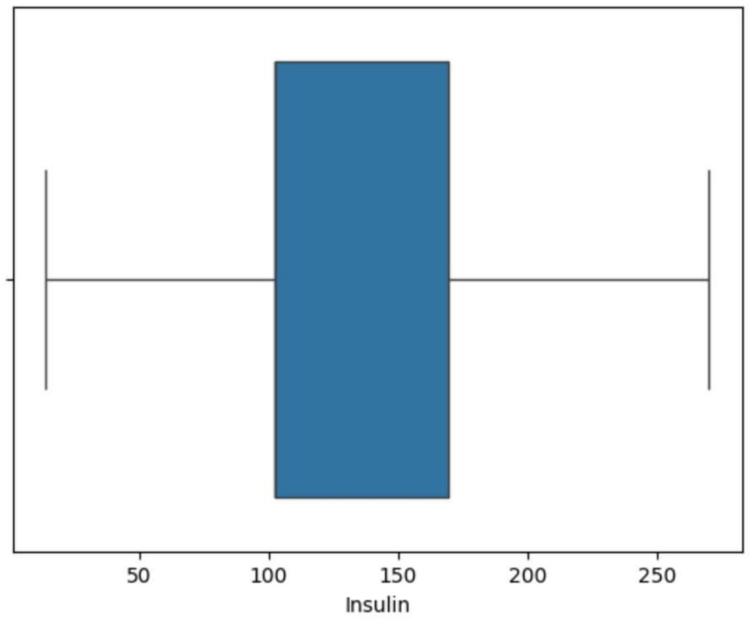
[53]:
```
# The process of visualizing the Insulin variable with boxplot method was done.␣
 ↪We find the outlier observations on the chart.
```

```
import seaborn as sns
sns.boxplot(x = df["Insulin"]);
```



```
[54]: #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
```

```
[55]: import seaborn as sns
      sns.boxplot(x = df["Insulin"]);
```



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

```
[56]: 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])
```

[57]:
```
df_scores = lof.negative_outlier_factor_
np.sort(df_scores)[0:30]
```

[57]:
```
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])
```

[58]:
```
#We choose the threshold value according to lof scores
threshold = np.sort(df_scores)[7]
threshold
```

[58]:
```
-1.740031580305444
```

[59]:
```
#We delete those that are higher than the threshold
outlier = df_scores > threshold
df = df[outlier]
# The size of the data set was examined.
df.shape
```

[59]:
```
(760, 9)
```

[60]:
```
# 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]
```

[61]:
```
df.head()
```

[61]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6    148.0           72.0           35.0    169.5  33.6
1            1     85.0           66.0           29.0    102.5  26.6
2            8    183.0           64.0           32.0    169.5  23.3
3            1     89.0           66.0           23.0     94.0  28.1
4            0    137.0           40.0           35.0    168.0  43.1

   DiabetesPedigreeFunction  Age  Outcome      NewBMI
0                     0.627   50        1   Obesity 1
1                     0.351   31        0  Overweight
2                     0.672   32        1      Normal
3                     0.167   21        0  Overweight
4                     2.288   33        1   Obesity 3
```

[62]:
```
# 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"
```

[63]:
```
# The operation performed was added to the dataframe.
df = df.assign(NewInsulinScore=df.apply(set_insulin, axis=1))
```

```
df.head()
```

```
[63]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
      0            6    148.0           72.0           35.0    169.5  33.6
      1            1     85.0           66.0           29.0    102.5  26.6
      2            8    183.0           64.0           32.0    169.5  23.3
      3            1     89.0           66.0           23.0     94.0  28.1
      4            0    137.0           40.0           35.0    168.0  43.1

         DiabetesPedigreeFunction  Age  Outcome      NewBMI NewInsulinScore
      0                     0.627   50        1   Obesity 1        Abnormal
      1                     0.351   31        0  Overweight          Normal
      2                     0.672   32        1      Normal        Abnormal
      3                     0.167   21        0  Overweight          Normal
      4                     2.288   33        1   Obesity 3        Abnormal
```

```
[64]: # 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]
```

```
[65]: df.head()
```

```
[65]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
      0            6    148.0           72.0           35.0    169.5  33.6
      1            1     85.0           66.0           29.0    102.5  26.6
      2            8    183.0           64.0           32.0    169.5  23.3
      3            1     89.0           66.0           23.0     94.0  28.1
      4            0    137.0           40.0           35.0    168.0  43.1

         DiabetesPedigreeFunction  Age  Outcome      NewBMI NewInsulinScore  \
      0                     0.627   50        1   Obesity 1        Abnormal
      1                     0.351   31        0  Overweight          Normal
      2                     0.672   32        1      Normal        Abnormal
      3                     0.167   21        0  Overweight          Normal
      4                     2.288   33        1   Obesity 3        Abnormal

        NewGlucose
      0     Secret
      1     Normal
```

```
2     Secret
3     Normal
4     Secret
```

[66]:
```
# 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)
```

[67]:
```
df.head()
```

[67]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6    148.0           72.0           35.0    169.5  33.6
1            1     85.0           66.0           29.0    102.5  26.6
2            8    183.0           64.0           32.0    169.5  23.3
3            1     89.0           66.0           23.0     94.0  28.1
4            0    137.0           40.0           35.0    168.0  43.1

   DiabetesPedigreeFunction  Age  Outcome  NewBMI_Obesity 1  NewBMI_Obesity 2  \
0                     0.627   50        1              True             False
1                     0.351   31        0             False             False
2                     0.672   32        1             False             False
3                     0.167   21        0             False             False
4                     2.288   33        1             False             False

   NewBMI_Obesity 3  NewBMI_Overweight  NewBMI_Underweight  \
0             False              False               False
1             False               True               False
2             False              False               False
3             False               True               False
4              True              False               False

   NewInsulinScore_Normal  NewGlucose_Low  NewGlucose_Normal  \
0                   False           False              False
1                    True           False               True
2                   False           False              False
3                    True           False               True
4                   False           False              False

   NewGlucose_Overweight  NewGlucose_Secret
0                  False               True
1                  False              False
2                  False               True
3                  False              False
4                  False               True
```

```
[68]: 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']]
      categorical_df.head()
```

```
[68]:    NewBMI_Obesity 1  NewBMI_Obesity 2  NewBMI_Obesity 3  NewBMI_Overweight  \
      0              True             False             False              False
      1             False             False             False               True
      2             False             False             False              False
      3             False             False             False               True
      4             False             False              True              False

         NewBMI_Underweight  NewInsulinScore_Normal  NewGlucose_Low  \
      0               False                   False           False
      1               False                    True           False
      2               False                   False           False
      3               False                    True           False
      4               False                   False           False

         NewGlucose_Normal  NewGlucose_Overweight  NewGlucose_Secret
      0              False                  False               True
      1               True                  False              False
      2              False                  False               True
      3               True                  False              False
      4              False                  False               True
```

```
[69]: 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
```

```
[70]: X.head()
```

```
[70]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
      0            6    148.0           72.0           35.0    169.5  33.6
      1            1     85.0           66.0           29.0    102.5  26.6
      2            8    183.0           64.0           32.0    169.5  23.3
      3            1     89.0           66.0           23.0     94.0  28.1
      4            0    137.0           40.0           35.0    168.0  43.1

         DiabetesPedigreeFunction  Age
```

```
0                  0.627   50
1                  0.351   31
2                  0.672   32
3                  0.167   21
4                  2.288   33
```

[71]:
```
# 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)
```

[72]:
```
X.head()
```

[72]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0          0.6    0.775          0.000       1.000000  1.000000  0.177778
1         -0.4   -0.800         -0.375       0.142857  0.000000 -0.600000
2          1.0    1.650         -0.500       0.571429  1.000000 -0.966667
3         -0.4   -0.700         -0.375      -0.714286 -0.126866 -0.433333
4         -0.6    0.500         -2.000       1.000000  0.977612  1.233333

   DiabetesPedigreeFunction       Age
0                  0.669707  1.235294
1                 -0.049511  0.117647
2                  0.786971  0.176471
3                 -0.528990 -0.470588
4                  4.998046  0.235294
```

[73]:
```
X = pd.concat([X,categorical_df], axis = 1)
```

[74]:
```
X.head()
```

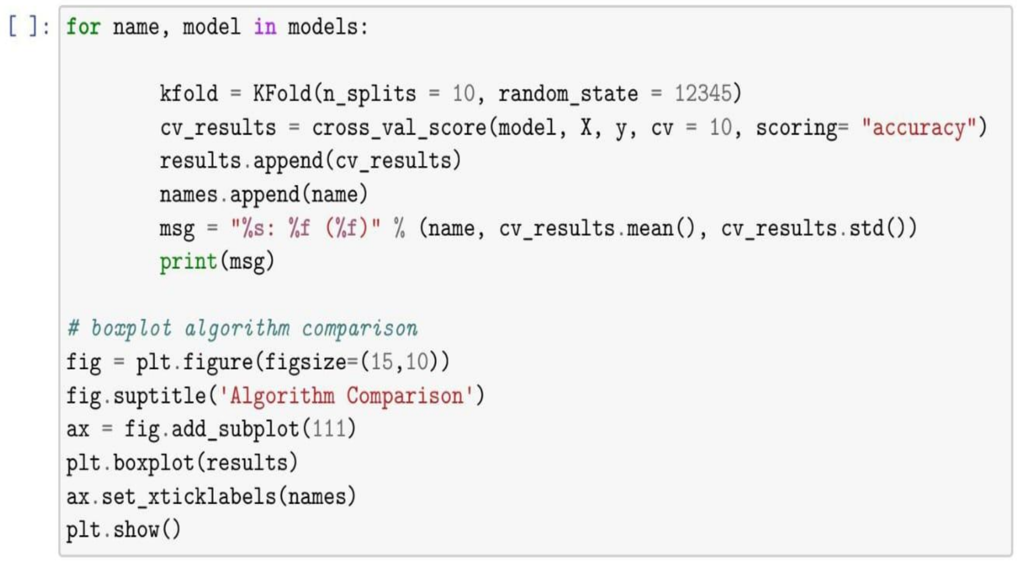
[74]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0          0.6    0.775          0.000       1.000000  1.000000  0.177778
1         -0.4   -0.800         -0.375       0.142857  0.000000 -0.600000
2          1.0    1.650         -0.500       0.571429  1.000000 -0.966667
3         -0.4   -0.700         -0.375      -0.714286 -0.126866 -0.433333
4         -0.6    0.500         -2.000       1.000000  0.977612  1.233333

   DiabetesPedigreeFunction       Age  NewBMI_Obesity 1  NewBMI_Obesity 2  \
0                  0.669707  1.235294              True             False
1                 -0.049511  0.117647             False             False
2                  0.786971  0.176471             False             False
3                 -0.528990 -0.470588             False             False
```

```
4                  4.998046  0.235294             False             False

   NewBMI_Obesity 3  NewBMI_Overweight  NewBMI_Underweight  \
0             False              False               False
1             False               True               False
2             False              False               False
3             False               True               False
4              True              False               False

   NewInsulinScore_Normal  NewGlucose_Low  NewGlucose_Normal  \
0                   False           False              False
1                    True           False               True
2                   False           False              False
3                    True           False               True
4                   False           False              False

   NewGlucose_Overweight  NewGlucose_Secret
0                  False               True
1                  False              False
2                  False               True
3                  False              False
4                  False               True
```

[75]:
```
y.head()
```

[75]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

[ ]:
```
# 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 = []
```



```
[ ]: 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()
```

BENEFITS

1. **Early Detection:** AI models can analyze large datasets of patient information, including medical records, genetics, lifestyle data, and more, to detect patterns and risk factors associated with diabetes. This early detection can help identify individuals at risk before symptoms become severe.

2. **Personalized Risk Assessment:** AI can provide personalized risk assessments by taking into account an individual's unique health history and factors, such as age, family history, and lifestyle. This allows for tailored interventions and preventive measures.

3. **Predictive Accuracy:** AI models have the ability to process and analyze vast amounts of data, leading to more accurate predictions. These predictions can help healthcare professionals make informed decisions and recommendations for patients.

4. **Cost-Effective:** Early detection and prevention of diabetes can lead to cost savings in healthcare by reducing the need for expensive treatments and hospitalizations. This can lower the overall economic burden of diabetes on healthcare systems.

5. **Continuous Monitoring:** AI models can be integrated with wearable devices and mobile apps to provide continuous monitoring of patients' health. This real-time data can alert individuals and healthcare providers to changes in their risk factors and overall health.

6. **Improved Healthcare Delivery:** Healthcare providers can use AI-based prediction models to prioritize patients at higher risk of developing diabetes, ensuring they receive necessary interventions and education. This can lead to more efficient and effective healthcare delivery.

7. **Research and Insights:** AI can help researchers analyze large-scale population data to identify new risk factors,

contributing to a better understanding of diabetes and potential new avenues for prevention and treatment.

8. Patient Empowerment: By providing individuals with information about their diabetes risk, AI-based models empower them to make informed decisions about their health and take proactive steps to prevent or manage the condition.

9. Reduction of Misdiagnosis: AI can assist in reducing misdiagnosis or delayed diagnosis by considering a wide range of patient data and reducing the impact of human bias in the diagnostic process.

10. Scalability: AI models can be deployed at scale, making them accessible to a larger population and ensuring that more people can benefit from early diabetes prediction and prevention.

CONCLUSION

However, as we embrace the potential of AI in diabetes prediction, it is crucial to consider ethical concerns, data privacy, and the need for a human-centered approach. AI should complement, rather than replace, the expertise of healthcare professionals. Furthermore, ensuring that these models are accessible and equitable is of paramount importance to maximize their positive impact.

In the years to come, the continued development and integration of AI-based diabetes prediction models will likely play a crucial role in reducing the burden of diabetes, improving patient outcomes, and advancing our understanding of this complex condition. As technology evolves and research progresses, we can anticipate even more refined and effective tools to tackle diabetes, ultimately leading to better health and well-being for individuals worldwide.

PRESENTED BY,

SABASTIN SELVIN JERALD J