American International University- Bangladesh

PROGRAMMING IN PYTHON

Final term Report

Spring 22-23

Project Title: Classification based Machine learning Model Development
Section: B

Student Name	Student Id
Argho Proshad Singha	20-42906-1
Partha Malakar	20-42908-1
Ashma ul husna	20-42614-1

Project Overview:

The project involves developing a classification-based machine learning model in Python. The project requires the use of a real dataset with a minimum of 2000 instances. The project also involves data preprocessing, exploratory data analysis, and the development of several classification-based models, including naive Bayes, K-Nearest Neighbours (KNN), decision tree, logistic regression, and Support Vector Machines (SVM).
The comparison among the classifiers will be based on their predictive accuracy. The final project report will follow a standard template, including a cover page, an introduction, dataset overview, data pre-processing and exploratory data analysis, model development, and a conclusion.

Dataset Overview:

i. Data source with valid URL Diabetes Prediction Predict
ii. https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset

Description about dataset the goal of this project is to use Python to create a Classification based Machine learning Model Development. In this dataset we have 1,00,000 instances and 9 attributes. This project will use a large dataset of 1,00,000 individuals to explore the relationship between heart disease, BMI, HbA1c levels, blood glucose level and diabetes. The result of the analysis will provide to predict diabetes in patients based on their medical history and demographic information. Also, this can be useful for healthcare professionals in identifying patients.
Hypertension: Including hypertension status is crucial for identifying high-risk individuals and developing personalized prevention and management strategies. Hypertension is marked as 0 and 1 as present or not in this database.

Heart disease: heart disease is a common complication in individuals with diabetes due to various factors. The heart disease is leveled as 0 and 1 as present or not.

Smoking history: Smoking is a risk factor for the development of type 2 diabetes and diabetes-related complications, and including smoking, history can improve the accuracy of diabetes risk prediction and enable personalized prevention and management strategies. In this database, the diabetes is categorized as per their smoking status which includes 'Never' 'No info' 'Current' 'Former' 'Ever', and 'Not Current'.

BMI: Higher BMI levels are associated with a higher risk of insulin resistance and other diabetes-related risk factors. In this dataset, the BMI is enlisted from the range of 0.52 to 23.45.

HbA1c level: Including the HbA1c level can improve the accuracy of diabetes risk prediction and facilitate personalized prevention and management strategies. Here In this database, the HbA1c level varies from 3.5 to 9.0

Blood glucose level: Including blood glucose levels can help identify individuals at higher risk of developing diabetes and enable the development of targeted prevention and management strategies based on their risk profile. Here in this dataset, the Blood Glucose level varies from 80 to 300.

```
In [2]: df = pd.read_csv("diabetes_prediction_dataset.csv")
```

```
In [3]: df.head()
```

Out[3]:

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0

Figure: Diabetes Prediction Dataset

1. Importing all the necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
# Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets # for using built-in datasets
from sklearn import metrics # for checking the model accuracy
#To plot the graph embedded in the notebook
%matplotlib inline
```

Data Preprocessing and Exploratory Data Analysis:

2. Load the dataset

In [2]:
```
df = pd.read_csv("diabetes_prediction_dataset.csv")
```

3. Checking information about Dataset.

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   gender               100000 non-null  object 
 1   age                  100000 non-null  float64
 2   hypertension         100000 non-null  int64  
 3   heart_disease        100000 non-null  int64  
 4   smoking_history      100000 non-null  object 
 5   bmi                  100000 non-null  float64
 6   HbA1c_level          100000 non-null  float64
 7   blood_glucose_level  100000 non-null  int64  
 8   diabetes             100000 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 6.9+ MB
```

4. Checking unique value in every variable in Dataset.

```
#checking unique values for every column
print("Gender",df['gender'].unique())
print("Age",df['age'].unique())
print("Hypertension",df['hypertension'].unique())
print("Heart disease",df['heart_disease'].unique())
print("Smoking history",df['smoking_history'].unique())
print("Bmi",df['bmi'].unique())
print("HbA1c level",df['HbA1c_level'].unique())
print("Blood glucose level",df['blood_glucose_level'].unique())
```

```
Gender ['Female' 'Male' 'Other']
Age [80.   54.   28.   36.   76.   20.   44.   79.   42.   32.   53.   78.
 67.   15.   37.   40.    5.   69.   72.    4.   30.   45.   43.   50.
 41.   26.   34.   73.   77.   66.   29.   60.   38.    3.   57.   74.
 19.   46.   21.   59.   27.   13.   56.    2.    7.   11.    6.   55.
  9.   62.   47.   12.   68.   75.   22.   58.   18.   24.   17.   25.
  0.08 33.   16.   61.   31.    8.   49.   39.   65.   14.   70.    0.56
 48.   51.   71.    0.88 64.   63.   52.    0.16 10.   35.   23.    0.64
  1.16  1.64  0.72  1.88  1.32  0.8   1.24  1.    1.8   0.48  1.56  1.08
  0.24  1.4   0.4   0.32  1.72  1.48]
Hypertension [0 1]
Heart disease [1 0]
Smoking history ['never' 'No Info' 'current' 'former' 'ever' 'not current']
Bmi [25.19 27.32 23.45 ... 59.42 44.39 60.52]
HbA1c level [6.6 5.7 5.  4.8 6.5 6.1 6.  5.8 3.5 6.2 4.  4.5 9.  7.  8.8 8.2 7.5 6.8]
Blood glucose level [140  80 158 155  85 200 145 100 130 160 126 159  90 260 220 300 280 240]
```

5. To know the and shape of the dataset

```
In [5]: df.keys()
```

```
Out[5]: Index(['gender', 'age', 'hypertension', 'heart_disease', 'smoking_history',
               'bmi', 'HbA1c_level', 'blood_glucose_level', 'diabetes'],
              dtype='object')
```

```
In [6]: print(df.shape)
```

```
(100000, 9)
```

6. To detect the null value

```
In [8]: df.isnull().sum()
```

```
Out[8]: gender                0
        age                   0
        hypertension          0
        heart_disease         0
        smoking_history       0
        bmi                   0
        HbA1c_level           0
        blood_glucose_level   0
        diabetes              0
        dtype: int64
```

7. dropna() method can be used to remove empty rows in the data set. As our dataset is big that's why we have dropped the empty rows here instead of replacing them with mean value.

```
In [10]: newdf= df.dropna(inplace=True)
```

8. Removing all the rows that have no information on smoking history in the dataset

```
for i in df.index:
    if df.loc[i,'smoking_history']=='No Info':
        df.drop(i,inplace=True)
```

9. describe() method shows each variable's instance count, mean, std, min, max, and 25%, 50%, and 75% of values in the dataset.

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

Out[24]:

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	diabetes
count	93777.000000	93777.000000	93777.000000	93777.000000	93777.000000	93777.000000	93777.000000
mean	42.466107	0.077802	0.040522	27.455258	5.533356	138.265385	0.087996
std	22.236435	0.267861	0.197180	6.632793	1.072698	40.838100	0.283291
min	0.080000	0.000000	0.000000	10.080000	3.500000	80.000000	0.000000
25%	25.000000	0.000000	0.000000	23.760000	4.800000	100.000000	0.000000
50%	43.000000	0.000000	0.000000	27.320000	5.800000	140.000000	0.000000
75%	60.000000	0.000000	0.000000	29.800000	6.200000	159.000000	0.000000
max	80.000000	1.000000	1.000000	95.690000	9.000000	300.000000	1.000000

10. To improve the clarity of the visualization, decimal values in the age variable (example:0.6) of the dataset have been rounded to 1 and any person whose age is under 9 and has smoking history has been removed from the dataset. This step has been taken to remove the outliner in the dataset.

```
]: #If the value of age is less than 1, set it to 1
for x in df.index:
    if df.loc[x, "age"] < 1:
        df.loc[x, "age"] = 1

#If the value of age is less than 6 and they have smkoing history (except never) drop the row
for x in df.index:
    if df.loc[x, "age"] < 9:
        if df.loc[x,'smoking_history']=='current' or df.loc[x,'smoking_history']=="former"
        or df.loc[x,'smoking_history']=="ever" or df.loc[x,'smoking_history']=="not current":
            df.drop(x, inplace=True)

df
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0
5	Female	20.0	0	0	never	27.32	6.6	85	0
...	...	...	...	...	...	...	...	...	...
99992	Female	26.0	0	0	never	34.34	6.5	160	0
99993	Female	40.0	0	0	never	40.69	3.5	155	0
99997	Male	66.0	0	0	former	27.83	5.7	155	0
99998	Female	24.0	0	0	never	35.42	4.0	100	0
99999	Female	57.0	0	0	current	22.43	6.6	90	0

63993 rows × 9 columns

11. duplicate() remove duplicate value from the dataset.

```
#Removing duplicates values
df.drop_duplicates(inplace = True)
df.info()
```

12. . Renaming the target variable

```
print("diabetes",df['diabetes'].unique())
```

```
diabetes [0 1]
```

```
# replace the target values with new names
df['diabetes'] = df['diabetes'].replace([0, 1], ['Negative', 'Positive'])
df
```

]:

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	Negative
2	Male	28.0	0	0	never	27.32	5.7	158	Negative
3	Female	36.0	0	0	current	23.45	5.0	155	Negative
4	Male	76.0	1	1	current	20.14	4.8	155	Negative
5	Female	20.0	0	0	never	27.32	6.6	85	Negative
...	...	...	...	...	...	...	...	...	...
99992	Female	26.0	0	0	never	34.34	6.5	160	Negative
99993	Female	40.0	0	0	never	40.69	3.5	155	Negative
99997	Male	66.0	0	0	former	27.83	5.7	155	Negative
99998	Female	24.0	0	0	never	35.42	4.0	100	Negative
99999	Female	57.0	0	0	current	22.43	6.6	90	Negative

63068 rows × 9 columns

Exploratory Data Analysis and Plot:

1. Stripplot is a type of plot used to display the distribution of a numerical variable across different categories of a categorical variable. It does so by plotting individual data points as dots along a horizontal or vertical axis corresponding to the categories. Here x axis and y axis represent bmi and gender. Blue and Yellow color indicates negative and positive diabetes result.

```
In [32]: sns.stripplot (x='bmi',y= 'gender', data=df, jitter=True,
         hue='diabetes',
         dodge=True)
         pit.title("Stripplot based on 'Diabete Prediction' dataset");
```

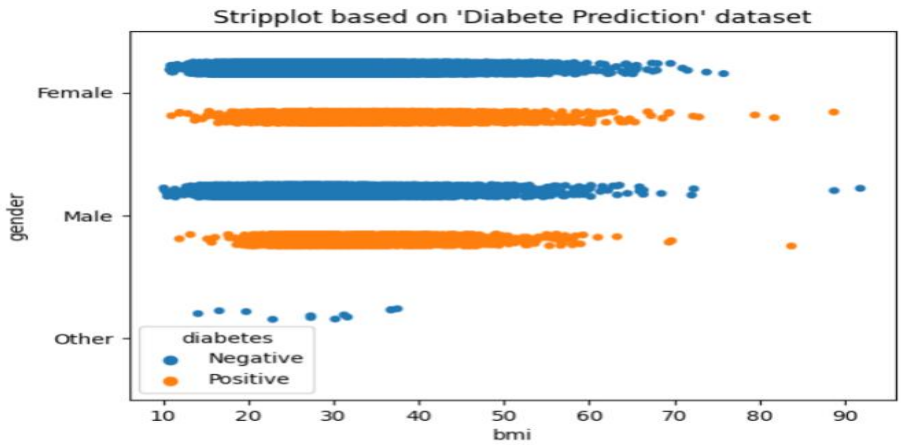


Figure: Visualization of BMI vs Gender on Stripplot

2. Bar plot is a type of chart used to visualize categorical data by representing each category as a bar of a certain height or length. It is a way to aggregate and compare the data across different categories. By default, the height of each bar represents the mean value of the data within that category. Here x-axis and y-axis represent diabetes and hypertension. Rocket fuel red and Peach color indicates negative and positive diabetes results. As hypertension increases the tendency to get diabetes also increases.

```
In [33]: sns.barplot(x="diabetes".
         ', y="hypertension" data=df, palette="rocket")
         plt.title("Barplot based on 'Diabete Prediction' dataset");
```

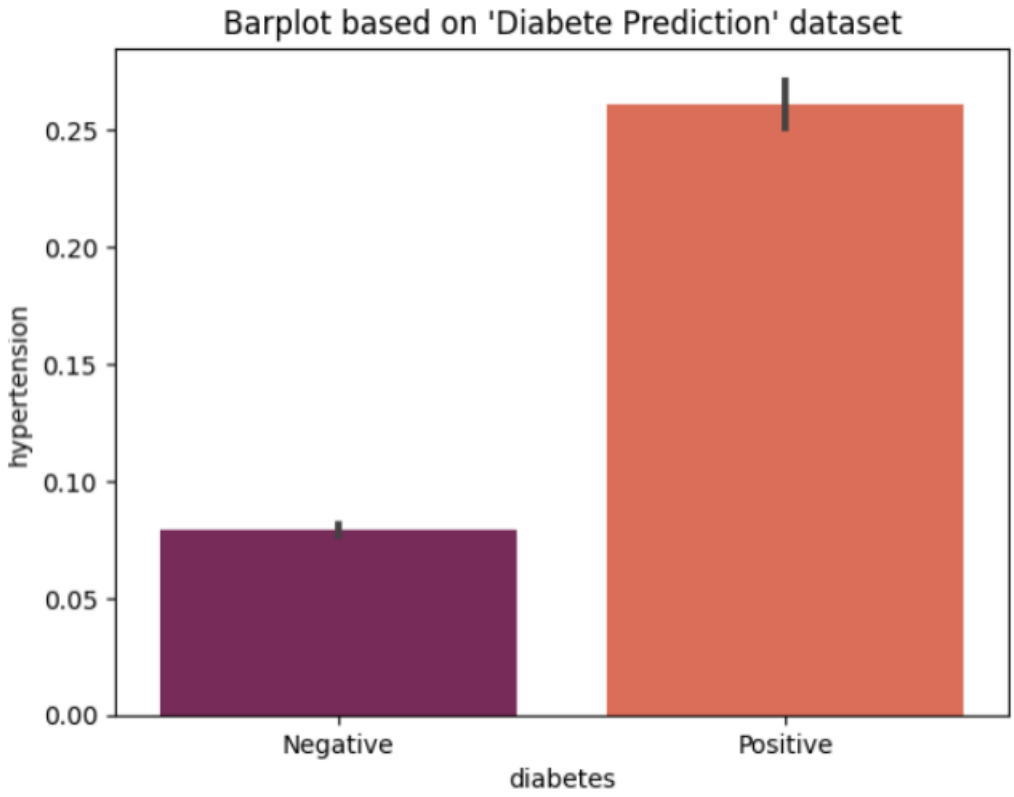


Figure: Visualization of Diabetes VS Hypertension on Barplot

3. Countplot is a chart that shows the number of occurrences of each category in a categorical variable. This plot is provided by the seaborn library and is a variation of a bar plot, where the height of each bar represents the frequency of each category.

```
sns.countplot(x="diabetes", data=df, palette="rainbow")
plt.title("Countplot based on 'Diabete Prediction' dataset");
```

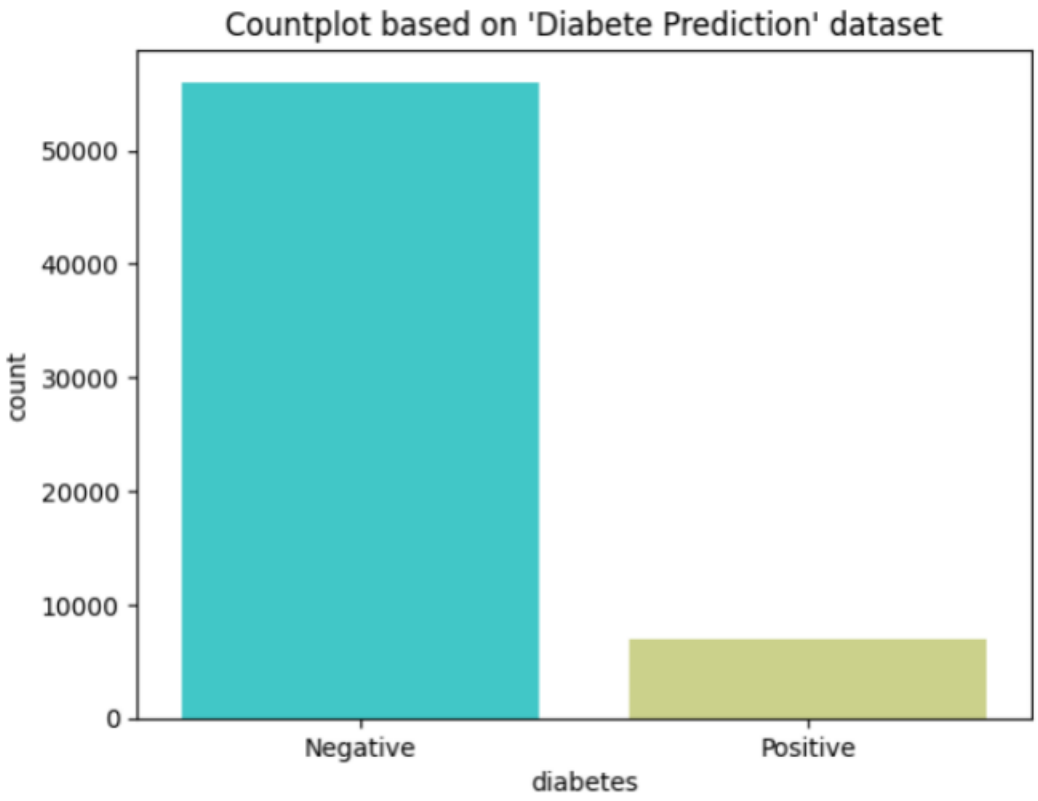


Figure: Visualization of Diabetes on Countplot

4. Box plot is a graphical representation of numerical data that displays the distribution of the data through their quartiles. The box in the plot represents the middle 50% of the data, with the median value marked by a line inside the box. The whiskers of the plot extend to the minimum and maximum values of the data, or a certain distance from the box if the data contains outliers. Here x-axis and y-axis represent blood glucose level and gender. Sky-blue and Peach color indicates negative and positive diabetes results.

```
]: sns.boxplot(x="blood_glucose_level", y="gender", data=df, hue="diabetes", palette="coolwarm")
   plt.title("Boxplot based on 'tips' dataset");
```

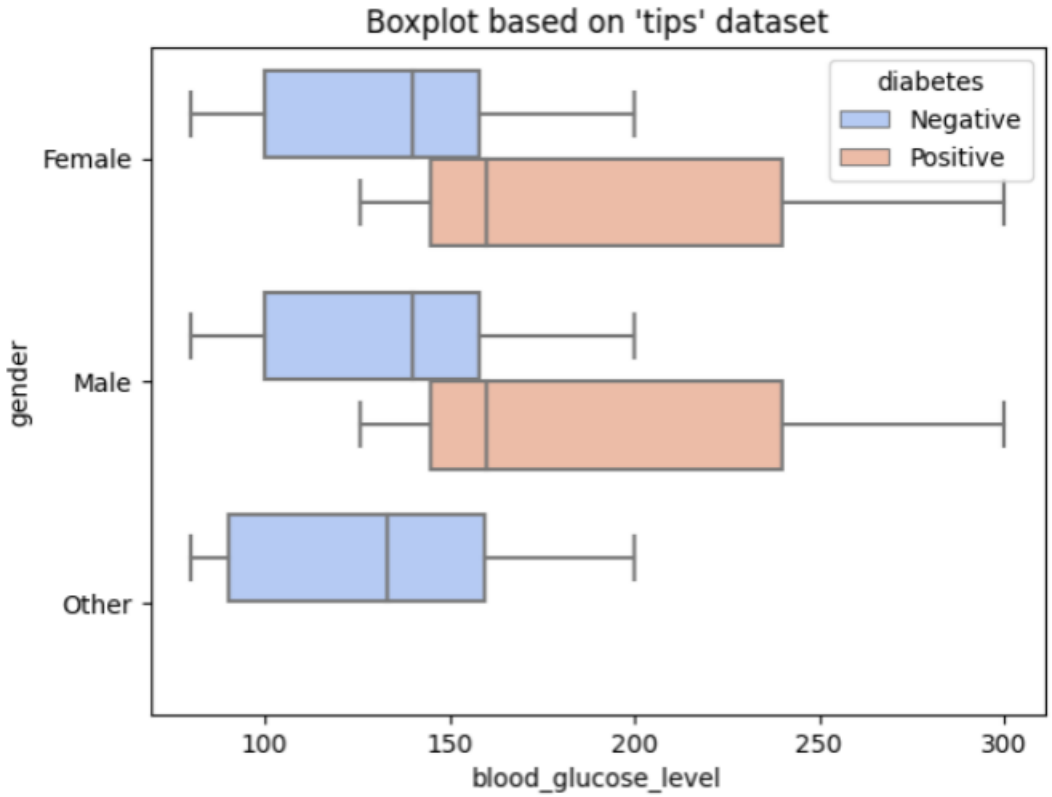
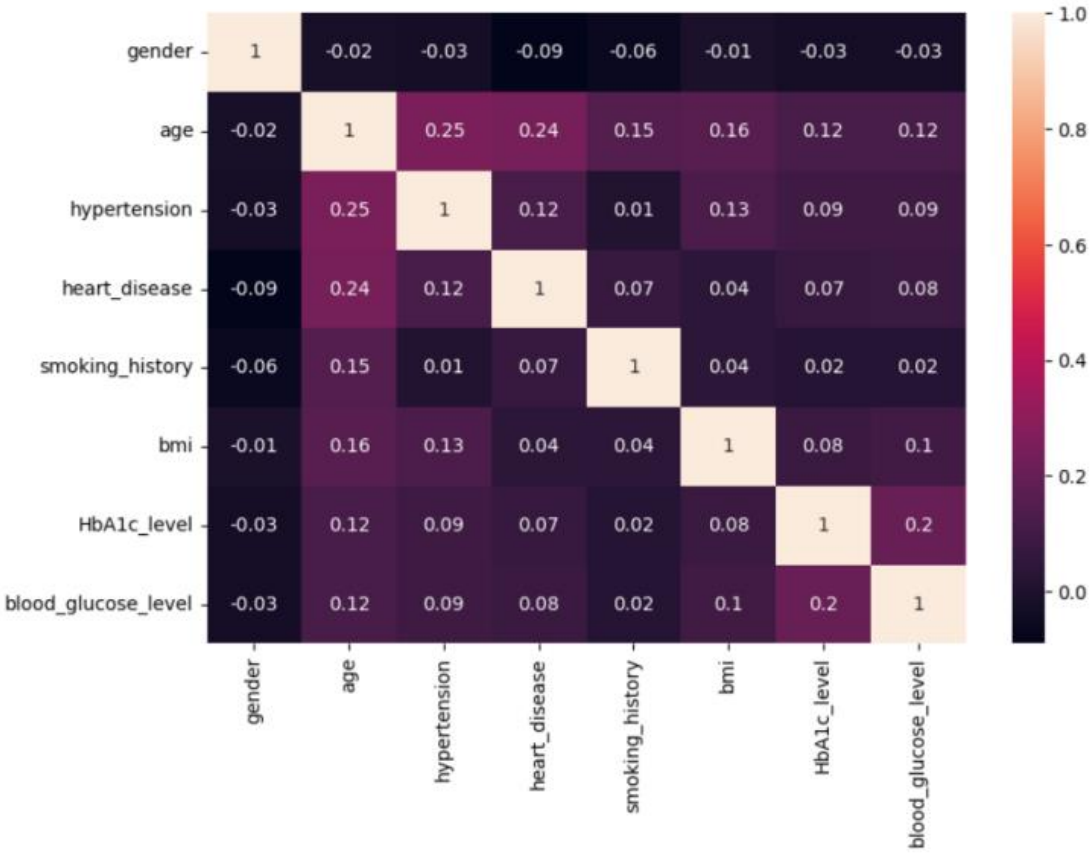


Figure: Visualization of Blood Glucose Level Vs Gender on Boxplot

Model Development: Classification in data mining is a common technique that separates data points into different classes. It allows you to organize data sets of all sorts, including complex and large datasets as well as small and simple ones. It primarily involves using algorithms that you can easily modify to improve the data quality. Here, we have applied three algorithms Naive Bayes, KNN, Decision Tree, Logistic Regression, SVM on RStudio software in this dataset. To predict the accuracy and its breakdown we have used a prediction accuracy.

Correlation:

```
correlation_matrix = df.corr().round(2)
# changing the figure size
plt.figure(figsize = (9, 6))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True);
```



Features Matrix and Target Variable:

```
# Feature matrix
X = df[['gender', 'age', 'heart_disease', 'smoking_history', 'bmi', 'HbA1c_level', 'blood_glucose_level']]
X
```

]:

	gender	age	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level
0	2	80.0	1	0	25.19	6.6	140
2	1	28.0	0	0	27.32	5.7	158
3	2	36.0	0	1	23.45	5.0	155
4	1	76.0	1	1	20.14	4.8	155
5	2	20.0	0	0	27.32	6.6	85
...	...	...	...	...	...	...	...
99992	2	26.0	0	0	34.34	6.5	160
99993	2	40.0	0	0	40.69	3.5	155
99997	1	66.0	0	2	27.83	5.7	155
99998	2	24.0	0	0	35.42	4.0	100
99999	2	57.0	0	1	22.43	6.6	90

63068 rows × 7 columns

```
# Target variable
y = df['diabetes']
y
```

```
0        Negative
2        Negative
3        Negative
4        Negative
5        Negative
           ...
99992    Negative
99993    Negative
99997    Negative
99998    Negative
99999    Negative
Name: diabetes, Length: 63068, dtype: object
```

Figure: Features Matrix and Target Variable

Test and Train Matrix:

```
']: from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 16)
    print("X_train shape: ", X_train.shape)
    print("X_test shape: ", X_test.shape)
    print("y_train shape: ", y_train.shape)
    print("y_test shape: ", y_test.shape)
```

```
X_train shape:  (44147, 7)
X_test shape:  (18921, 7)
y_train shape:  (44147,)
y_test shape:  (18921,)
```

Naïve Bayes Model:

```
In [333]: from sklearn.naive_bayes import GaussianNB
          model_nb = GaussianNB()
          model_nb.fit(X_train, y_train) #train the model with the training dataset
          y_prediction_nb = model_nb.predict(X_test) #pass the testing data to the trained model
          # checking the accuracy of the algorithm.
          # by comparing predicted output by the model and the actual output
          score_nb = metrics.accuracy_score(y_prediction_nb, y_test).round(4)
          print("---------------------------------")
          print('The accuracy of the NB is: {}'.format(score_nb))
          print("---------------------------------")
          # save the accuracy score
          score.add(('NB', score_nb))
```

```
---------------------------------
The accuracy of the NB is: 0.9187
---------------------------------
```

K-Nearest Neighbor Model:

In [330]:
```
from sklearn.neighbors import KNeighborsClassifier # for K nearest neighbours
#from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
model_knn = KNeighborsClassifier(n_neighbors=3) # 3 neighbours for putting the new data into a class
model_knn.fit(X_train, y_train) #train the model with the training dataset
y_prediction_knn = model_knn.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)
print("---------------------------------")
print('The accuracy of the KNN is: {}'.format(score_knn))
print("---------------------------------")
# save the accuracy score
score.add(('KNN', score_knn))
```

```
---------------------------------
The accuracy of the KNN is: 0.9341
---------------------------------
```

Decision Tree Model:

In [334]:
```
from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm
model_dt = DecisionTreeClassifier(random_state = 4)
model_dt.fit(X_train, y_train) #train the model with the training dataset
y_prediction_dt = model_dt.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
print("---------------------------------")
print('The accuracy of the DT is: {}'.format(score_dt))
print("---------------------------------")
# save the accuracy score
score.add(('DT', score_dt))
```

```
---------------------------------
The accuracy of the DT is: 0.9371
---------------------------------
```

Logistic Regression Model:

```
332]: # importing the necessary package to use the classification algorithm
from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train) #train the model with the training dataset
y_prediction_lr = model_lr.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_lr = metrics.accuracy_score(y_prediction_lr, y_test).round(5)
print("---------------------------------")
print('The accuracy of the LR is: {}'.format(score_lr))
print("---------------------------------")
# save the accuracy score
score.add(('LR', score_lr))
```

```
---------------------------------
The accuracy of the LR is: 0.9483
---------------------------------
```

Support Vector Machine Model:

```
In [329]: from sklearn import svm #for Support Vector Machine (SVM) Algorithm
model_svm = svm.SVC() #select the algorithm
model_svm.fit(X_train, y_train) #train the model with the training dataset
y_prediction_svm = model_svm.predict(X_test) # pass the testing data to the t
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)
print("---------------------------------")
print('The accuracy of the SVM is: {}'.format(score_svm))
print("---------------------------------")
# save the accuracy score
score = set()
score.add(('SVM', score_svm))
```

```
---------------------------------
The accuracy of the SVM is: 0.9304
---------------------------------
```

Discussion and Conclusion:

```
33]: print("The accuracy scores of different Models:")
     print("----------------------------------------")
     for s in score:
      print(s)

     The accuracy scores of different Models:
     ----------------------------------------
     ('KNN', 0.9341)
     ('LR', 0.9483)
     ('DT', 0.9371)
     ('NB', 0.9187)
     ('SVM', 0.9304)
```

FIGURE: Comparison Models

As we can see, the percentage of cases of KNN that are successfully categorized is 0.9341, the percentage of instances of Logistic regression that are correctly classified is 0.9483, and the percentage of instances of a Decision Tree that is correctly classified is 0.9371. Again, the percentage of instances of Naive Bayes and SVM that are correctly classified is 0.9187 and 0.9304. We may infer that the Logical Regression algorithm's accuracy performs better in this dataset since it correctly classifies more instances than other algorithms.

Conclusion:

Following the application of 5 different algorithms KNN, Logistic Regression, Naive Bayes, Decision Tree and SVM the dataset's best accuracy with a 0.9483 accuracy is in Logistic Regression. Then, a training and test set was taken from the original dataset to create a machine-learning model. The model was tested using a test dataset, and the results showed that the model's accuracy was obtained from the algorithms for the dataset. TO measure the accuracy to predict a result of a situation is an important concept in data science as it is used to improve generalization and minimize overfitting.