AMERICAN INTERNATIONAL UNIVERSITY–BANGLADESH (AIUB)

FACULTY OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

PROGRAMMING IN PYTHON

Spring 2022-2023

Section: B

Supervised By

Akinul Islam Jony

Submitted By

Name	**ID**
Shamsunnahar Riya	19-41672-3
Abdul Aziz Sajib	20-42035-1
Farhan Sadik Ferdous	20-42072-1
Tapu Biswas	20-42073-1

Date of Submission: May 3, 2023

Project Overview

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed. Machine learning algorithms forecast new output values using historical data as input. Machine learning is frequently applied in recommendation engines. Other common applications include business process automation (BPA), predictive maintenance, spam filtering, malware threat detection, and fraud detection.

Machine learning is crucial because it enables the creation of new goods and provides businesses with a picture of trends in consumer behavior and operational business patterns. Many of today's leading companies, such as Facebook, Google, and Uber, make machine learning a central part of their operations. For many businesses, machine learning now significantly differentiates them from their competitors. An overview of a typical process for developing a classification-based machine learning model:

Data Collection and Preparation: Collect and clean the data you will use to train and test your machine-learning model. This includes tasks such as removing duplicates, dealing with missing values, and converting data to the appropriate format for analysis.

Exploratory data analysis: Exploratory Data Analysis (EDA) is the process of analyzing and understanding a dataset to discover patterns, relationships, and trends in the data. EDA is an important step in developing a machine learning model as it helps to identify potential issues with the data and provides insights into which features may be most relevant for predicting the outcome variable. Some common techniques used in EDA include:

- Summary statistics: This involves computing basic summary statistics such as mean, median, standard deviation, and range for each feature in the dataset. Summary statistics can provide a quick overview of the data and help to identify potential outliers or missing values.
- Visualization: Data visualization techniques such as histograms, scatter plots, and box plots can provide insights into the distribution of the data, and potential relationships between features, and identify outliers.
- Correlation analysis: Correlation analysis involves computing the correlation coefficient between pairs of features in the dataset. This can help to identify potential relationships between features and provide insights into which features may be most relevant for predicting the outcome variable.

Feature Selection and Engineering: Determine which features (or variables) are most important for predicting the outcome variable. This may involve selecting a subset of the available features or creating new features based on the available data.

Model Selection: Choose an appropriate algorithm for your classification problem. Common algorithms include Naïve Bayes, Logistic Regression, Decision trees, and support vector machines (SVM).

Model Training: Split your data into training and testing sets and train your model on the training data. Use cross-validation techniques to avoid overfitting and select the best accuracy for the dataset.

Model Deployment: it is the process of taking a trained machine learning model and making it available for use in a real-world environment. The goal of model deployment is to integrate the machine learning model into a larger system or application so that it can be used to make predictions on new data. After training and testing the dataset, the dataset needs to be evaluated through algorithms to get the accuracy for the dataset.

Comparing Model Accuracy: When comparing the accuracy of different machine learning models, it's important to consider several factors beyond just the reported accuracy metric. These factors include the size and quality of the dataset used to train and test the models, the complexity of the model and its ability to generalize to new data, and the computational resources required to train and run the model. In addition to accuracy, it's also important to consider other performance metrics such as precision, recall, and F1 score, as these metrics can provide additional insights into how well the model is performing on different aspects of the classification problem. When comparing multiple models, it's often useful to use cross-validation techniques to ensure that the models are evaluated on multiple subsets of the data, reducing the risk of overfitting to a particular subset. It's also important to consider the specific business problem being solved and the performance metrics that are most important for that problem. Ultimately, the choice of which model to use will depend on a combination of factors, including the available data, the performance metrics of interest, the computational resources available, and the specific business problem being solved. It's important to carefully evaluate multiple models and consider all of these factors before making a final decision.

Dataset Overview

i. Data source with valid URL

Diabetes Prediction Predict

https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset

ii. 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 who may be at risk of developing diabetes and in developing personalized treatment plans. The Diabetes prediction dataset is a collection of medical and demographic data from patients, along with their diabetes status (positive or negative). The data includes features such as age, gender, body mass index (BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level.

Gender: Males and females have different prevalence and risk factors for diabetes. Including gender can improve the accuracy of diabetes risk prediction and prevention strategies. In Gender, the existing categories are male, female, and other gender options.

Age: Age is a well-established risk factor for diabetes, and the prevalence, complications, and management of diabetes may differ depending on the patient's age. Here in this dataset, the age limit varies from 0.08 to 80.

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.

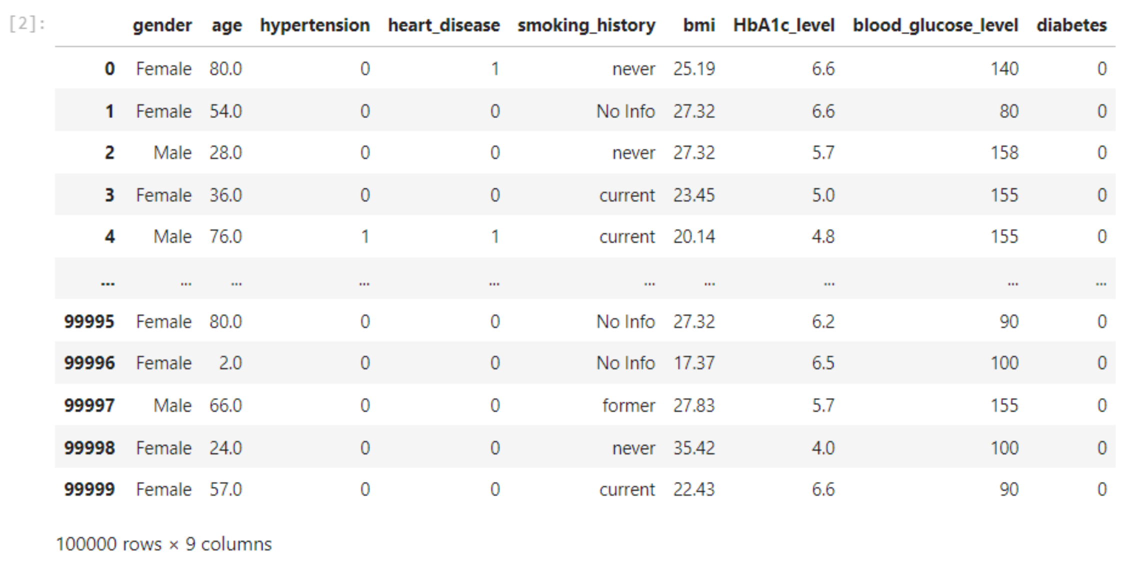
Heart disease: Heart disease is a common complication in individuals with diabetes due to various factors. The heart disease is leveled as 0 and 1 as present or not.

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.



```
[2]:
```

	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
...	...	...	...	...	...	...	...	...	...
99995	Female	80.0	0	0	No Info	27.32	6.2	90	0
99996	Female	2.0	0	0	No Info	17.37	6.5	100	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

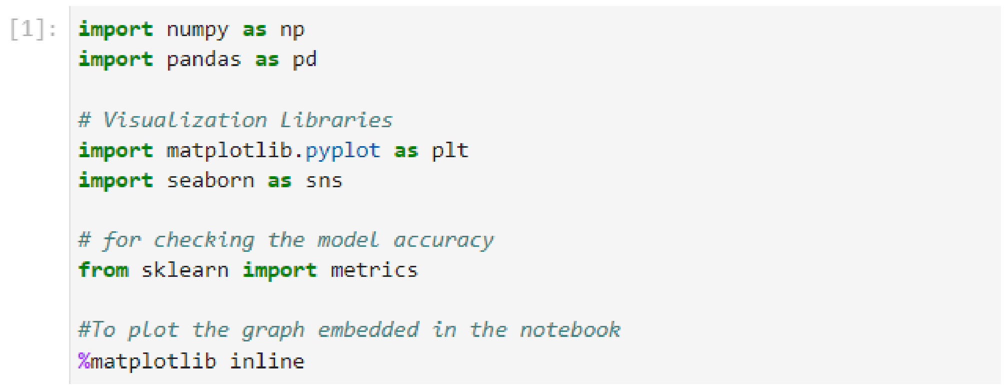
100000 rows × 9 columns

Figure: Diabetes Prediction Dataset

<u>Data Preprocessing and Exploratory Data Analysis</u>

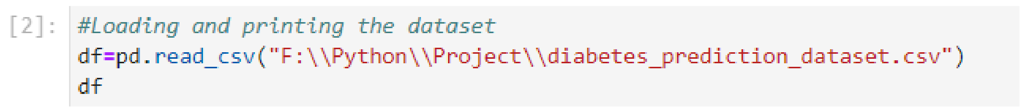
<u>Data pre-processing steps:</u>

1. Importing all the necessary libraries

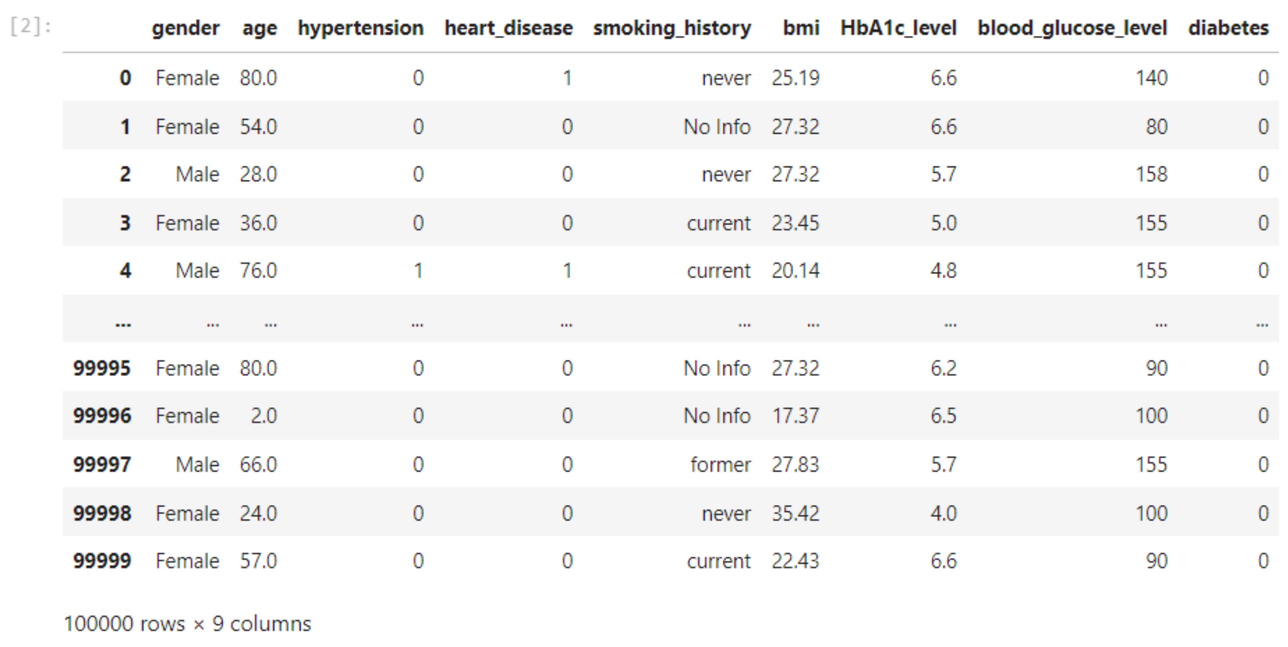


```
[1]: import numpy as np
     import pandas as pd

     # Visualization Libraries
     import matplotlib.pyplot as plt
     import seaborn as sns

     # for checking the model accuracy
     from sklearn import metrics

     #To plot the graph embedded in the notebook
     %matplotlib inline
```

2. Import the dataset into RStudio. Here by default, the first 5 rows, and the last 5 rows will be shown.



```
[2]: #Loading and printing the dataset
     df=pd.read_csv("F:\\Python\\Project\\diabetes_prediction_dataset.csv")
     df
```

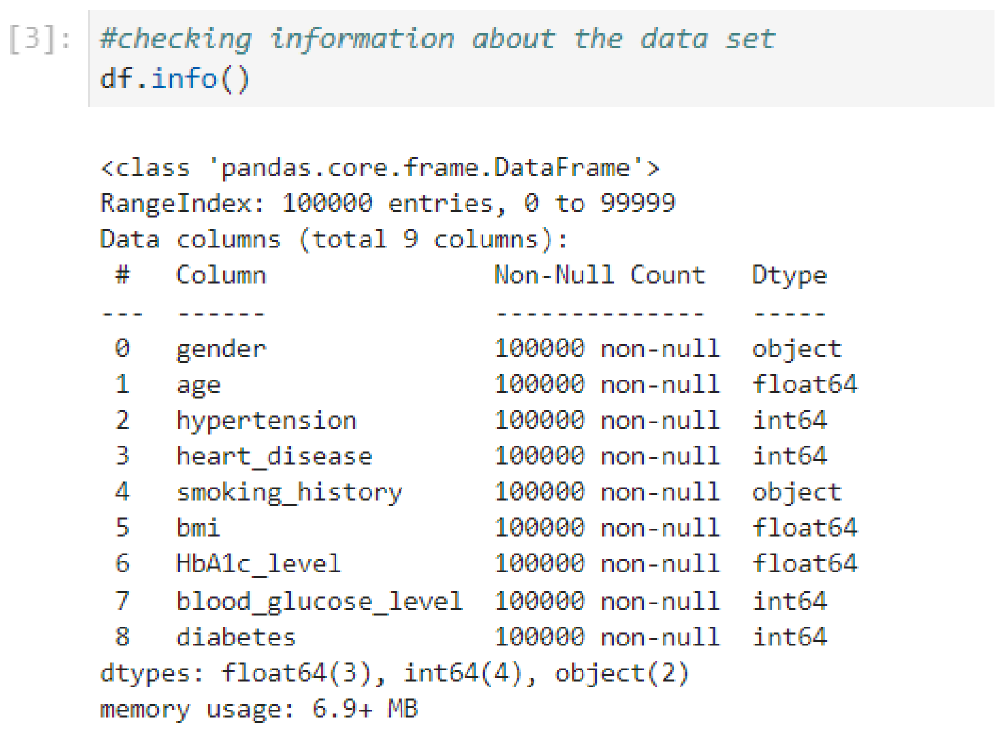


[2]:

	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
...	...	...	...	...	...	...	...	...	...
99995	Female	80.0	0	0	No Info	27.32	6.2	90	0
99996	Female	2.0	0	0	No Info	17.37	6.5	100	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

100000 rows × 9 columns

3. Checking information about Dataset.

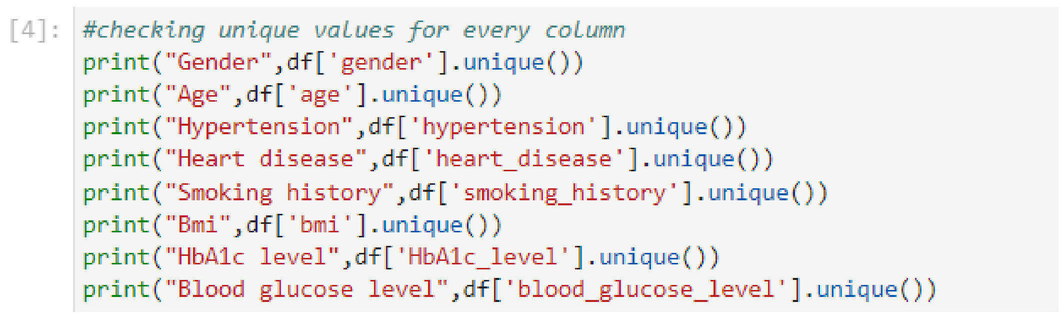


[3]:

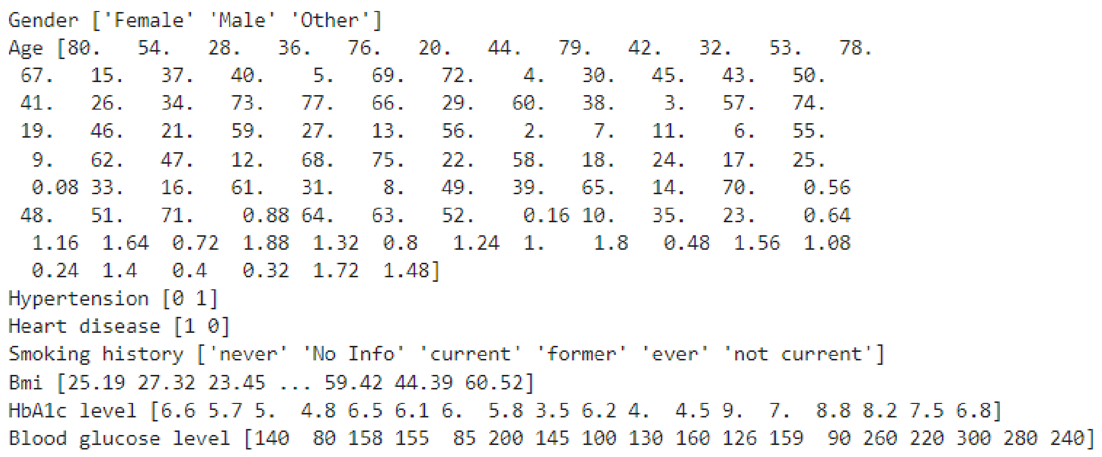
```
#checking information about the data set
df.info()
```

```
<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.

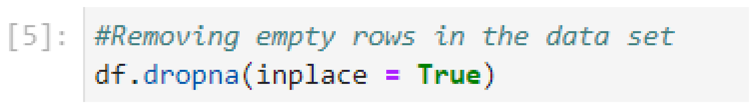


```
[4]: #checking unique values for every column
     print("Gender",df['gender'].unique())
     print("Age",df['age'].unique())
     print("Hypertension",df['hypertension'].unique())
     print("Heart disease",df['heart_disease'].unique())
     print("Smoking history",df['smoking_history'].unique())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



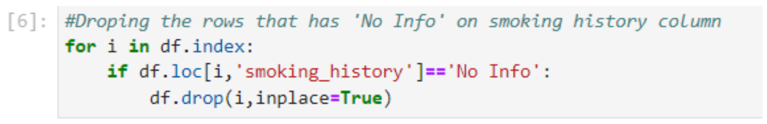
```
Gender ['Female' 'Male' 'Other']
Age [80.   54.   28.   36.   76.   20.   44.   79.   42.   32.   53.   78.
 67.   15.   37.   40.    5.   69.   72.    4.   30.   45.   43.   50.
 41.   26.   34.   73.   77.   66.   29.   60.   38.    3.   57.   74.
 19.   46.   21.   59.   27.   13.   56.    2.    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. 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.



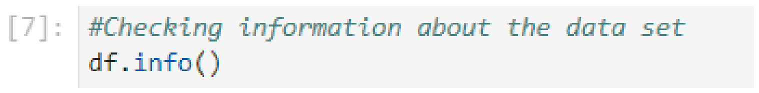
```
[5]: #Removing empty rows in the data set
     df.dropna(inplace = True)
```

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

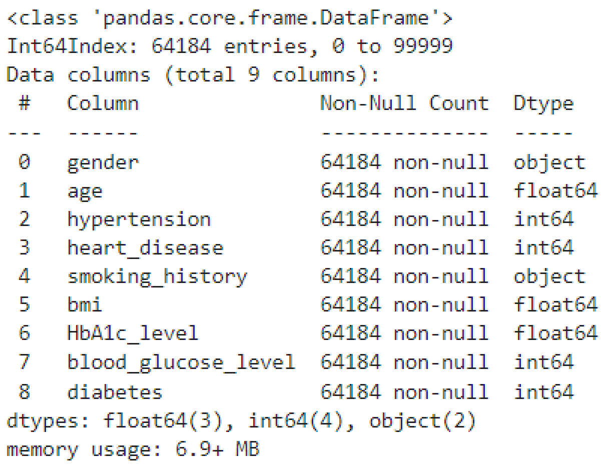


```
[6]: #Droping the rows that has 'No Info' on smoking history column
     for i in df.index:
         if df.loc[i,'smoking_history']=='No Info':
             df.drop(i,inplace=True)
```

7. info() returns information about the data set.

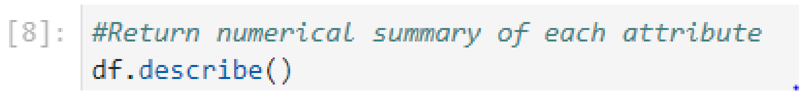


```
[7]: #Checking information about the data set
     df.info()
```

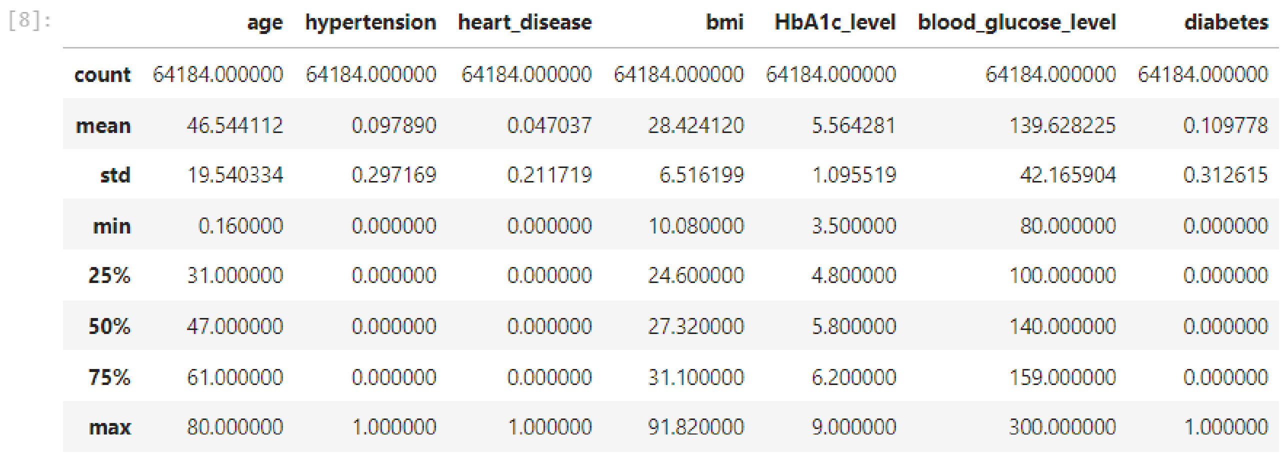


```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64184 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   gender               64184 non-null  object 
 1   age                  64184 non-null  float64
 2   hypertension         64184 non-null  int64  
 3   heart_disease        64184 non-null  int64  
 4   smoking_history      64184 non-null  object 
 5   bmi                  64184 non-null  float64
 6   HbA1c_level          64184 non-null  float64
 7   blood_glucose_level  64184 non-null  int64  
 8   diabetes             64184 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 6.9+ MB
```

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



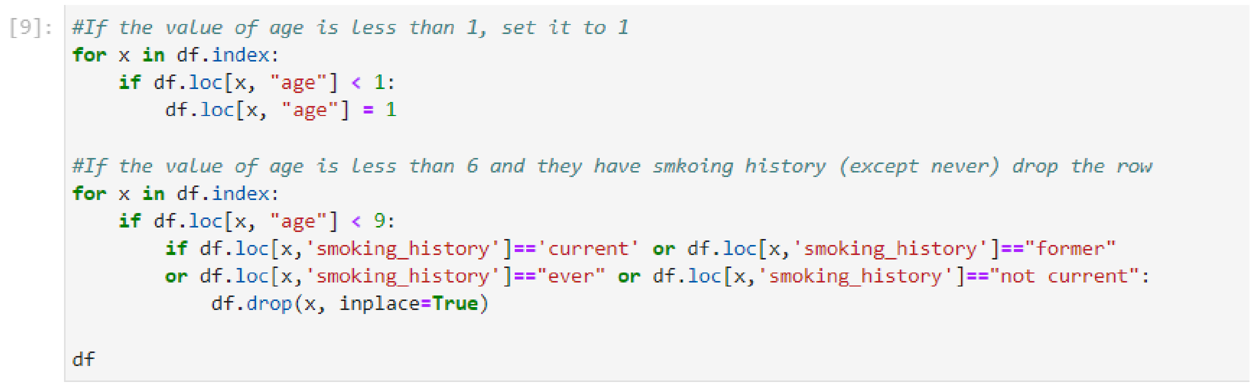
```
[8]: #Return numerical summary of each attribute
     df.describe()
```



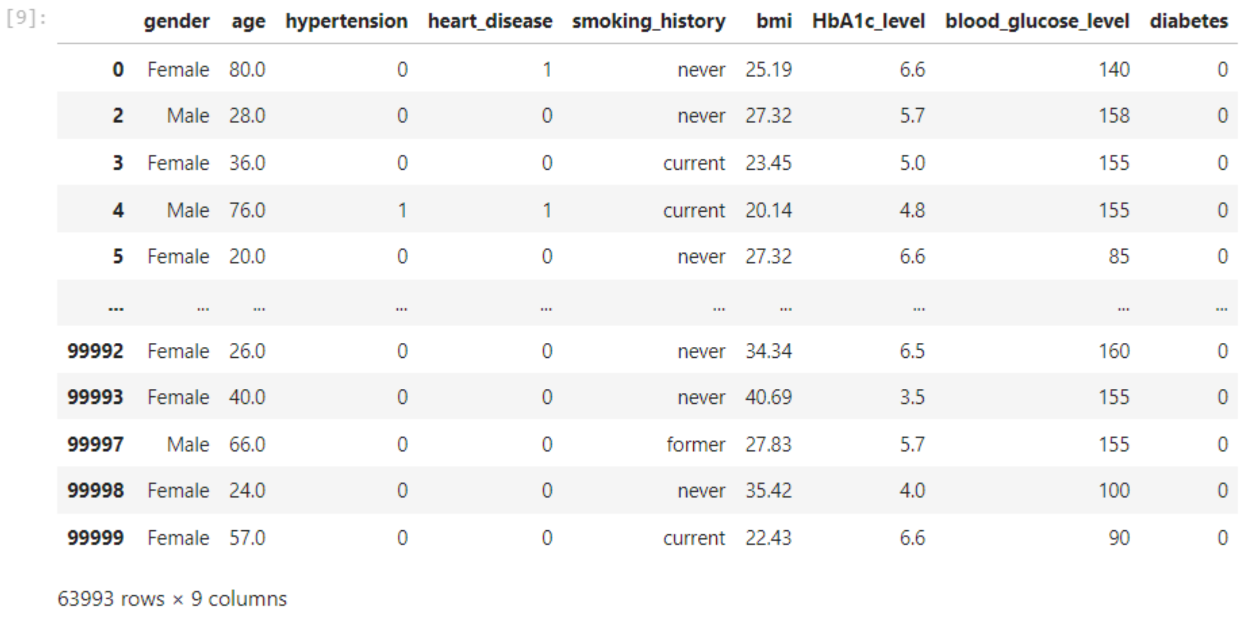
[8]:

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	diabetes
count	64184.000000	64184.000000	64184.000000	64184.000000	64184.000000	64184.000000	64184.000000
mean	46.544112	0.097890	0.047037	28.424120	5.564281	139.628225	0.109778
std	19.540334	0.297169	0.211719	6.516199	1.095519	42.165904	0.312615
min	0.160000	0.000000	0.000000	10.080000	3.500000	80.000000	0.000000
25%	31.000000	0.000000	0.000000	24.600000	4.800000	100.000000	0.000000
50%	47.000000	0.000000	0.000000	27.320000	5.800000	140.000000	0.000000
75%	61.000000	0.000000	0.000000	31.100000	6.200000	159.000000	0.000000
max	80.000000	1.000000	1.000000	91.820000	9.000000	300.000000	1.000000

9. 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.



```
[9]: #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
```

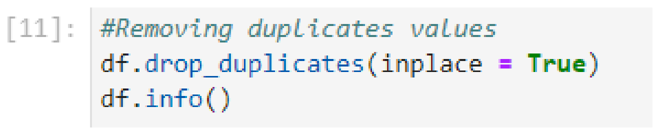


[9]:

	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

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



```
[11]: #Removing duplicates values
      df.drop_duplicates(inplace = True)
      df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 63068 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   gender               63068 non-null  object 
 1   age                  63068 non-null  float64
 2   hypertension         63068 non-null  int64  
 3   heart_disease        63068 non-null  int64  
 4   smoking_history      63068 non-null  object 
 5   bmi                  63068 non-null  float64
 6   HbA1c_level          63068 non-null  float64
 7   blood_glucose_level  63068 non-null  int64  
 8   diabetes             63068 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 4.8+ MB
```

11. Renaming the target variable.

```
[12]: #checking unique values for diabetes column
      print("diabetes",df['diabetes'].unique())
```

```
diabetes [0 1]
```

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

[13]:

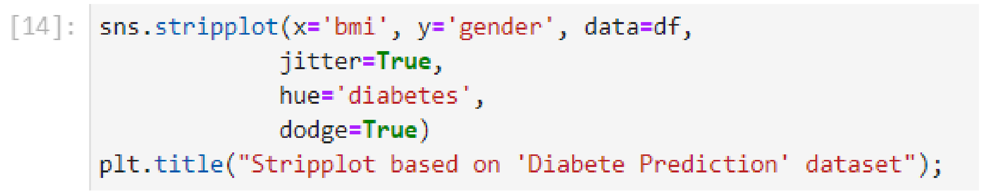
	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	Negative
2	Male	28.0	0	0	never	27.32	5.7	158	Negative
3	Female	36.0	0	0	current	23.45	5.0	155	Negative
4	Male	76.0	1	1	current	20.14	4.8	155	Negative
5	Female	20.0	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.



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

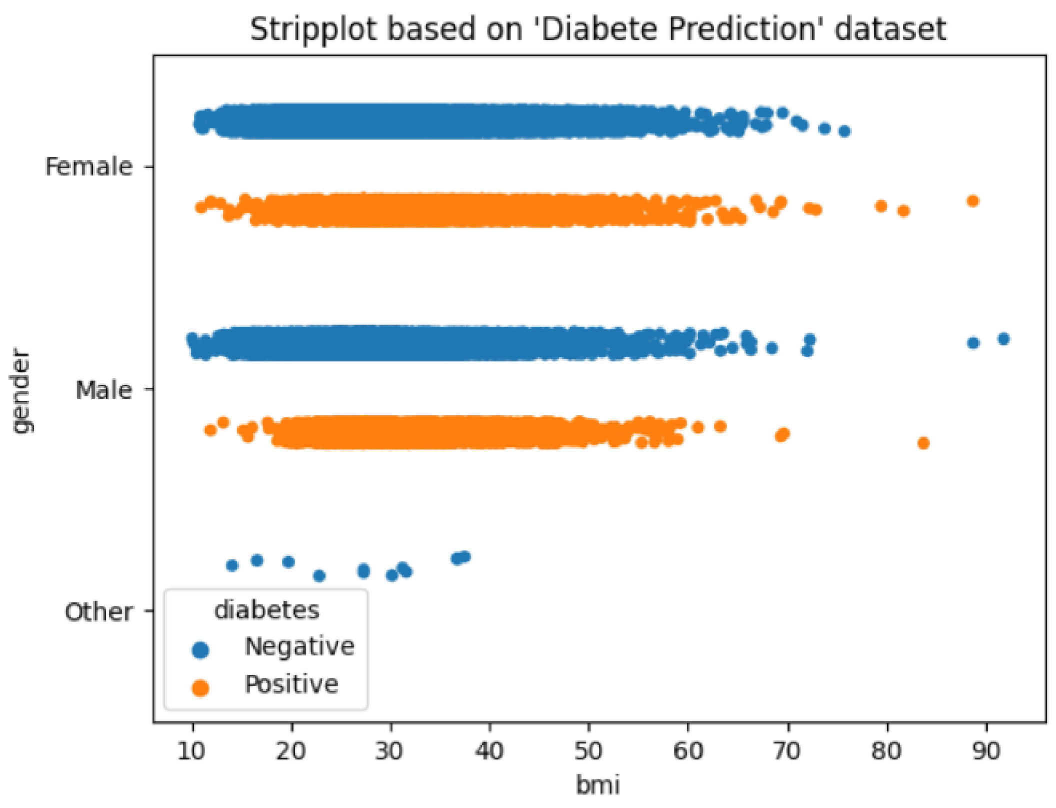
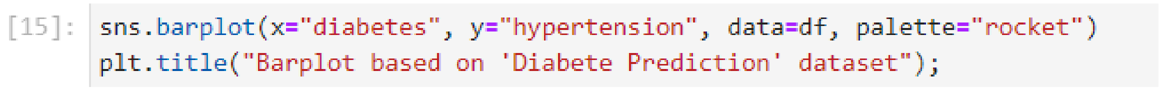


Figure: Visualization of BMI vs Gender on Stripplot

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.



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

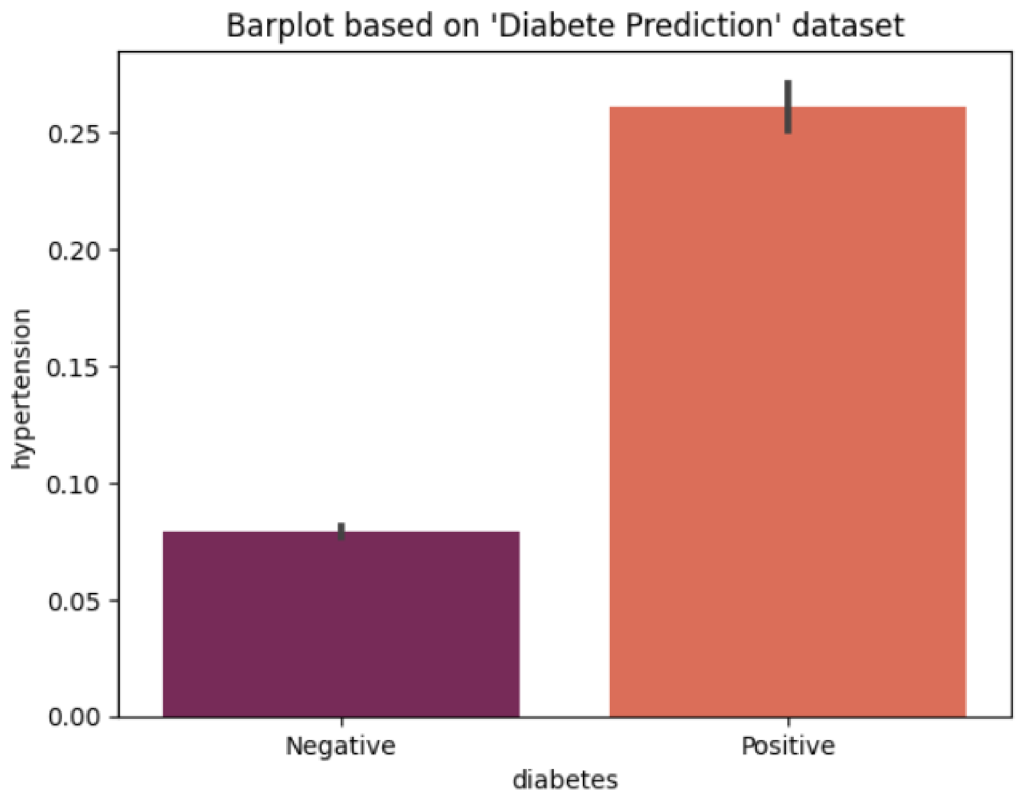
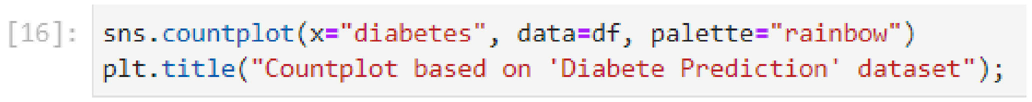


Figure: Visualization of Diabetes VS Hypertension
on Barplot

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.

Here x-axis represents diabetes. Sky-blue and Paste color indicates negative and positive diabetes results. As hypertension increases the tendency to get diabetes also increases. In our dataset people who don’t have exist more than affected one.



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

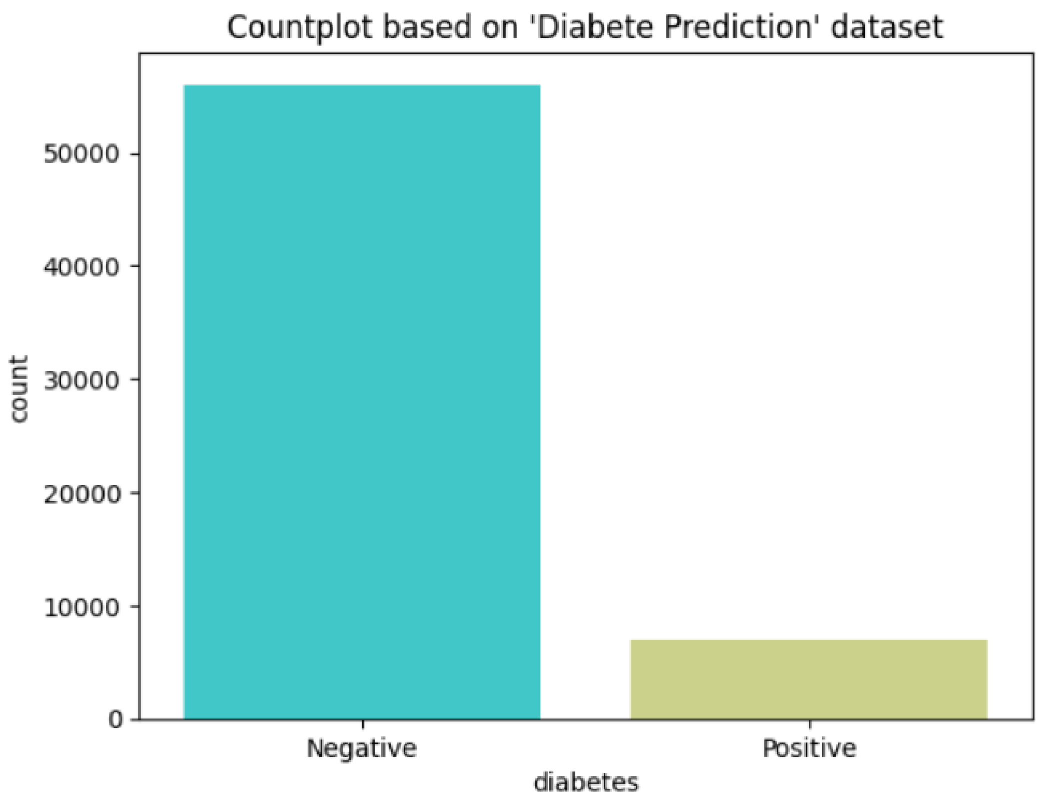
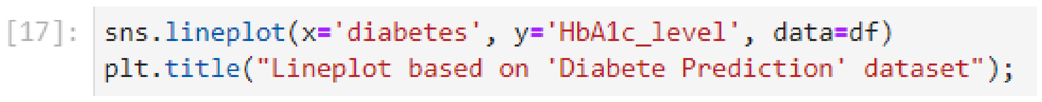


Figure: Visualization of Diabetes on Countplot

4. Line plot is a type of graph that displays the relationship between two variables, usually with time on the x-axis. In a line plot, data points are connected by straight lines, making it easy to see trends or patterns in the data over time.

Here x-axis and y-axis represent diabetes and HbA1c level. As HbA1c level increase the tendency to get diabetes also increases.



```
[17]: sns.lineplot(x='diabetes', y='HbA1c_level', data=df)
      plt.title("Lineplot based on 'Diabete Prediction' dataset");
```

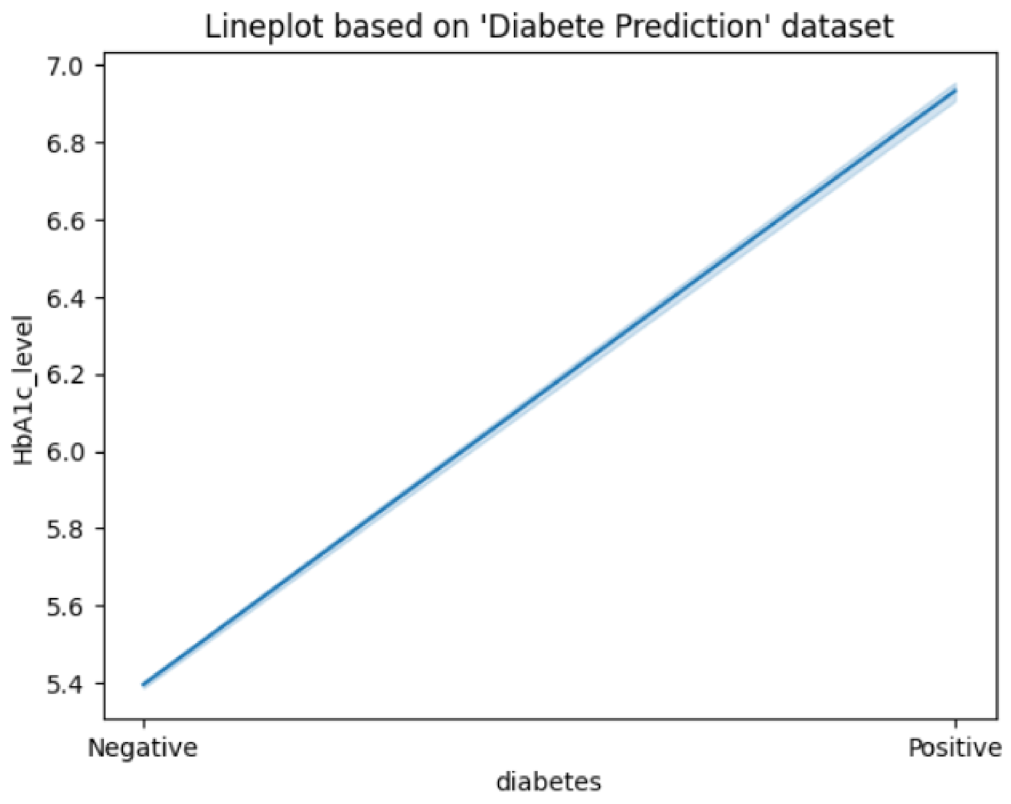


Figure: Visualization of Diabetes Vs HbA1c Level
on Lineplot

5. 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.

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

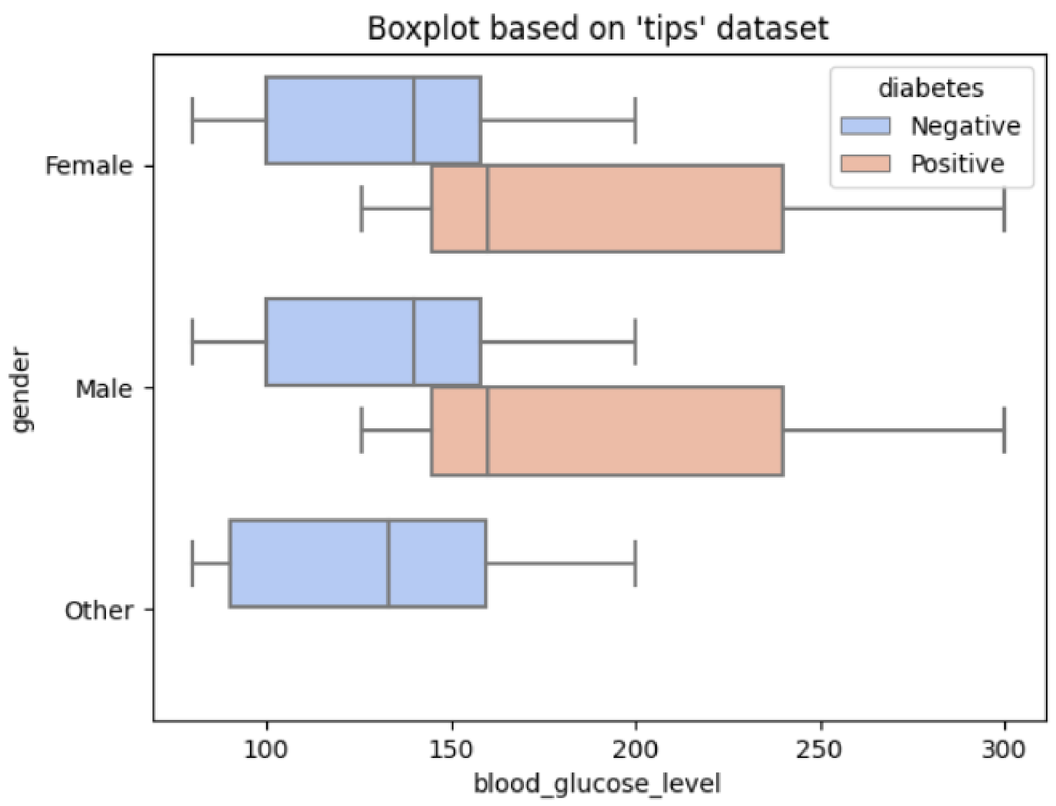


Figure: Visualization of Blood Glucose Level Vs
Gender on Boxplot

6. Violin plot is a type of data visualization that is similar to a box plot but provides a more detailed and informative display of the data distribution. It uses kernel density estimation to show the shape of the distribution, which can provide a better understanding of how the data is spread out across different values.

Here x-axis and y-axis represent heart disease and age perspective to diabetes. Rocket fuel red and Peach color indicates negative and positive diabetes results.

```
[19]: sns.violinplot(x='heart_disease', y='age', data=df,hue='diabetes',split=True, palette="flare")
      plt.title("Violinplot based on 'Diabete Prediction' dataset");
```

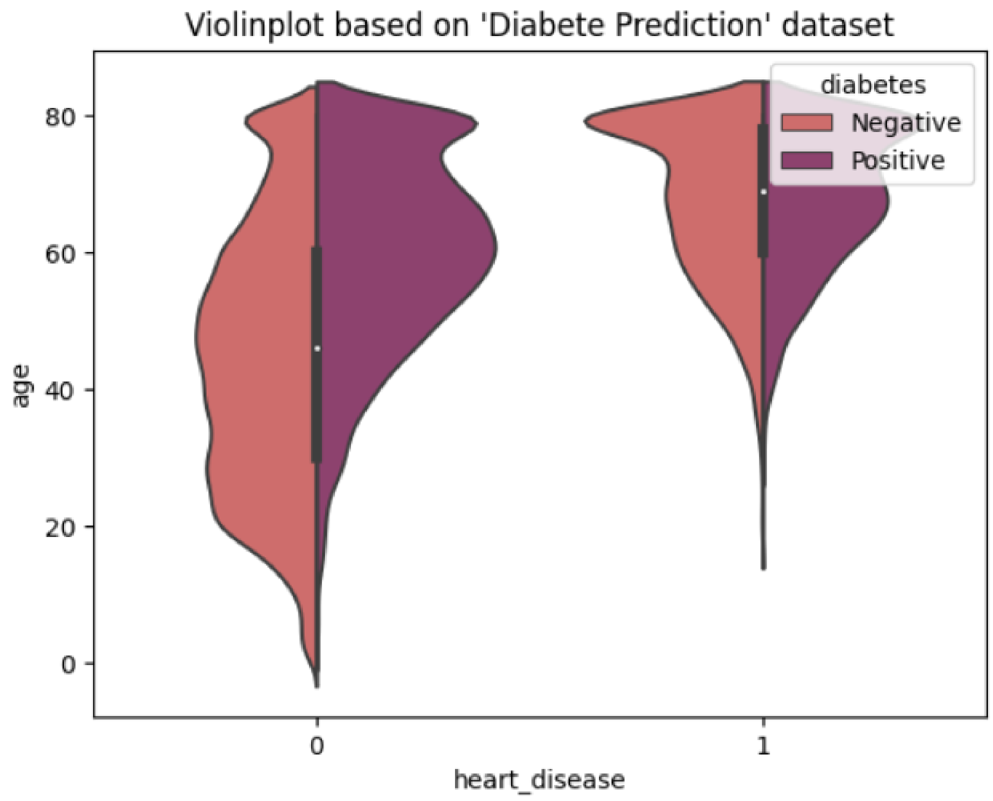


Figure: Visualization of Heart Disease Vs Age on Violinplot

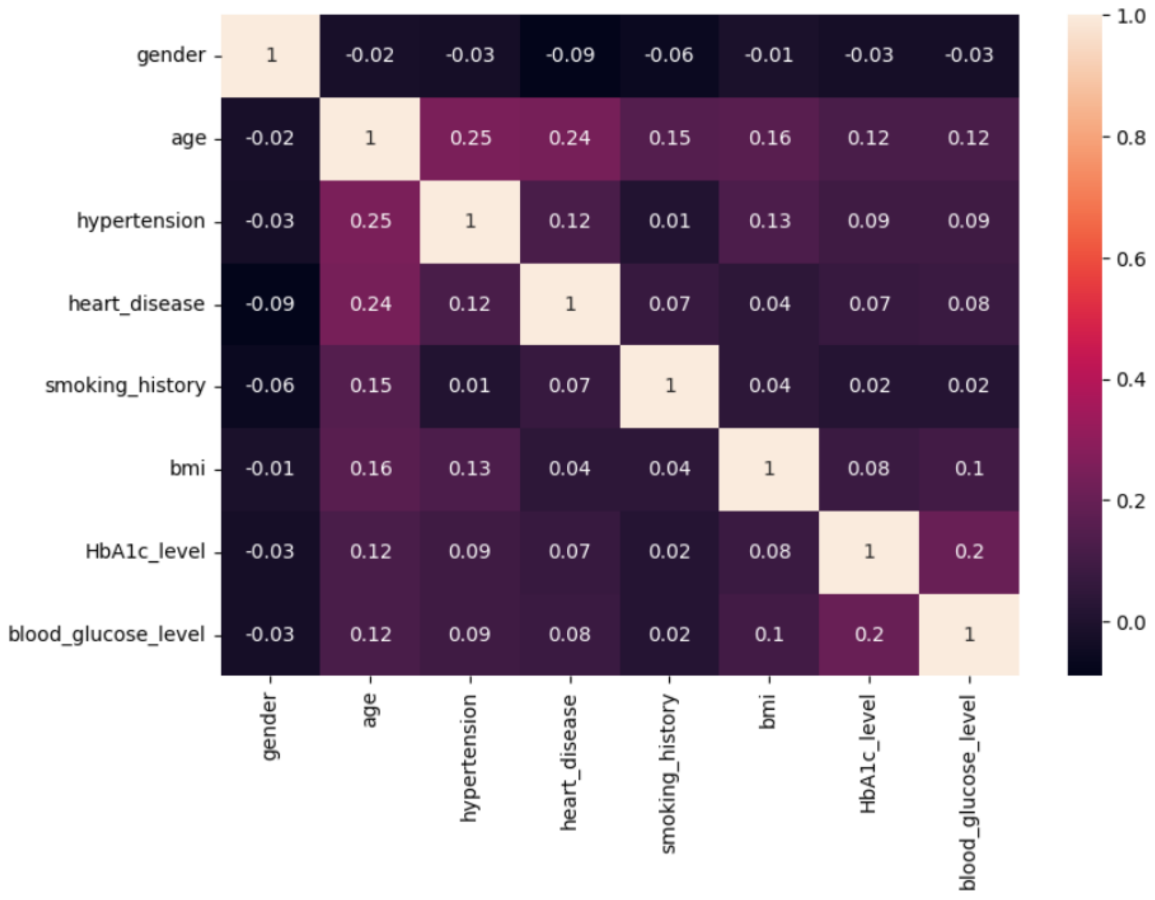
Model Development:

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.

Heatmap:

To identify highly correlated features, a correlation matrix can be created using the corr function from pandas. The correlation coefficient ranges from -1 to 1, with values close to 1 indicating a strong positive correlation and values close to -1 indicating a strong negative correlation between two variables. To visualize the correlation matrix, the heatmap function from the seaborn library can be used.

```
[21]: # corr() to calculate the correlation between variables
      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);
```



	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level
gender	1	-0.02	-0.03	-0.09	-0.06	-0.01	-0.03	-0.03
age	-0.02	1	0.25	0.24	0.15	0.16	0.12	0.12
hypertension	-0.03	0.25	1	0.12	0.01	0.13	0.09	0.09
heart_disease	-0.09	0.24	0.12	1	0.07	0.04	0.07	0.08
smoking_history	-0.06	0.15	0.01	0.07	1	0.04	0.02	0.02
bmi	-0.01	0.16	0.13	0.04	0.04	1	0.08	0.1
HbA1c_level	-0.03	0.12	0.09	0.07	0.02	0.08	1	0.2
blood_glucose_level	-0.03	0.12	0.09	0.08	0.02	0.1	0.2	1

```
[22]: # Steps to remove redundant values
      # Return a array filled with zeros
      mask = np.zeros_like(correlation_matrix)
      # Return the indices for the upper-triangle of array
      mask[np.triu_indices_from(mask)] = True
      # 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, mask=mask);
```

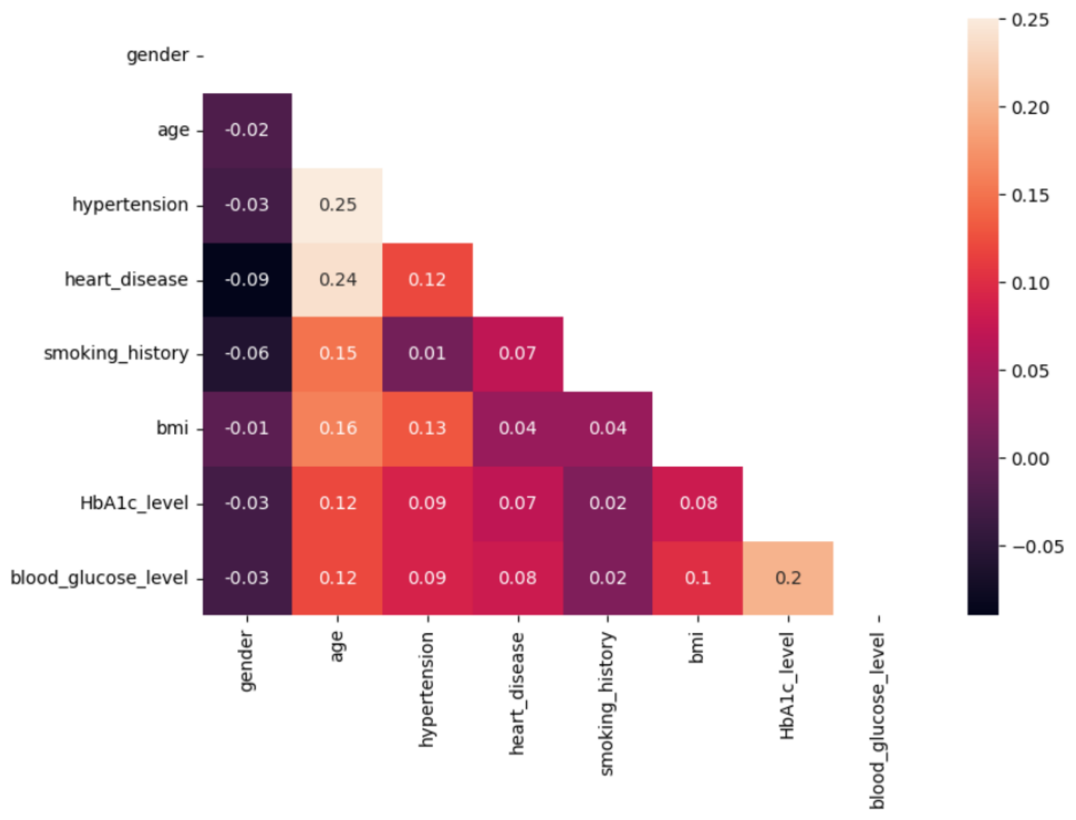
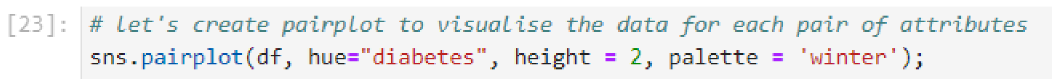


Figure: Heat Map

Pair plot:

Pair plot is used to visualize the target variable with the other variable as a pair.



```
[23]: # let's create pairplot to visualise the data for each pair of attributes
      sns.pairplot(df, hue="diabetes", height = 2, palette = 'winter');
```

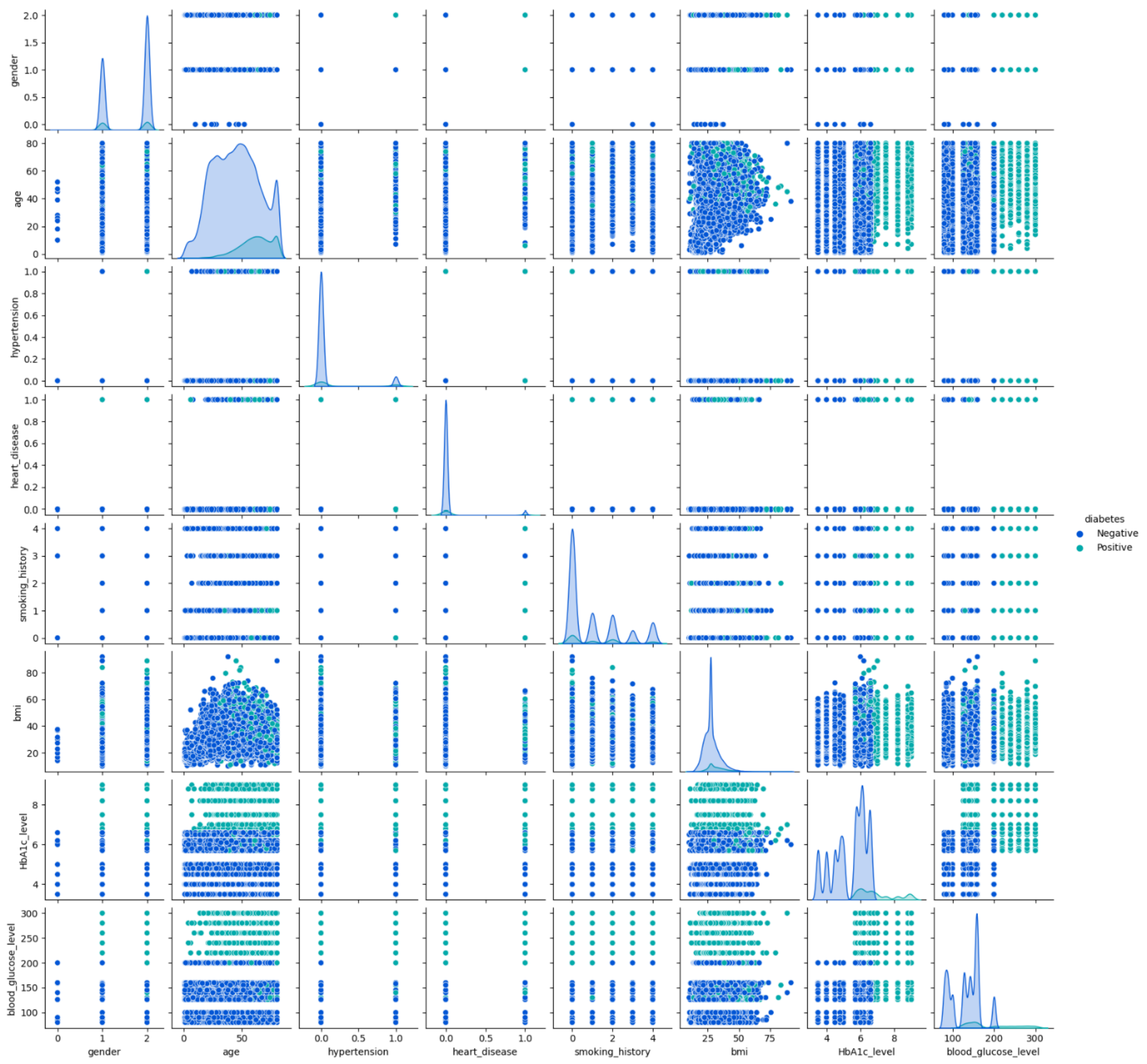
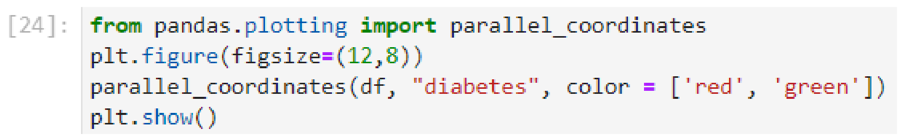


Figure: Pair plot

<u>Parallel Coordinate:</u>

For this dataset, another useful visualization plot is parallel coordinate, which represents each row as a line. A parallel plot allows comparing the feature of several individual observations (series) on a set of numeric variables.



```
[24]: from pandas.plotting import parallel_coordinates
      plt.figure(figsize=(12,8))
      parallel_coordinates(df, "diabetes", color = ['red', 'green'])
      plt.show()
```

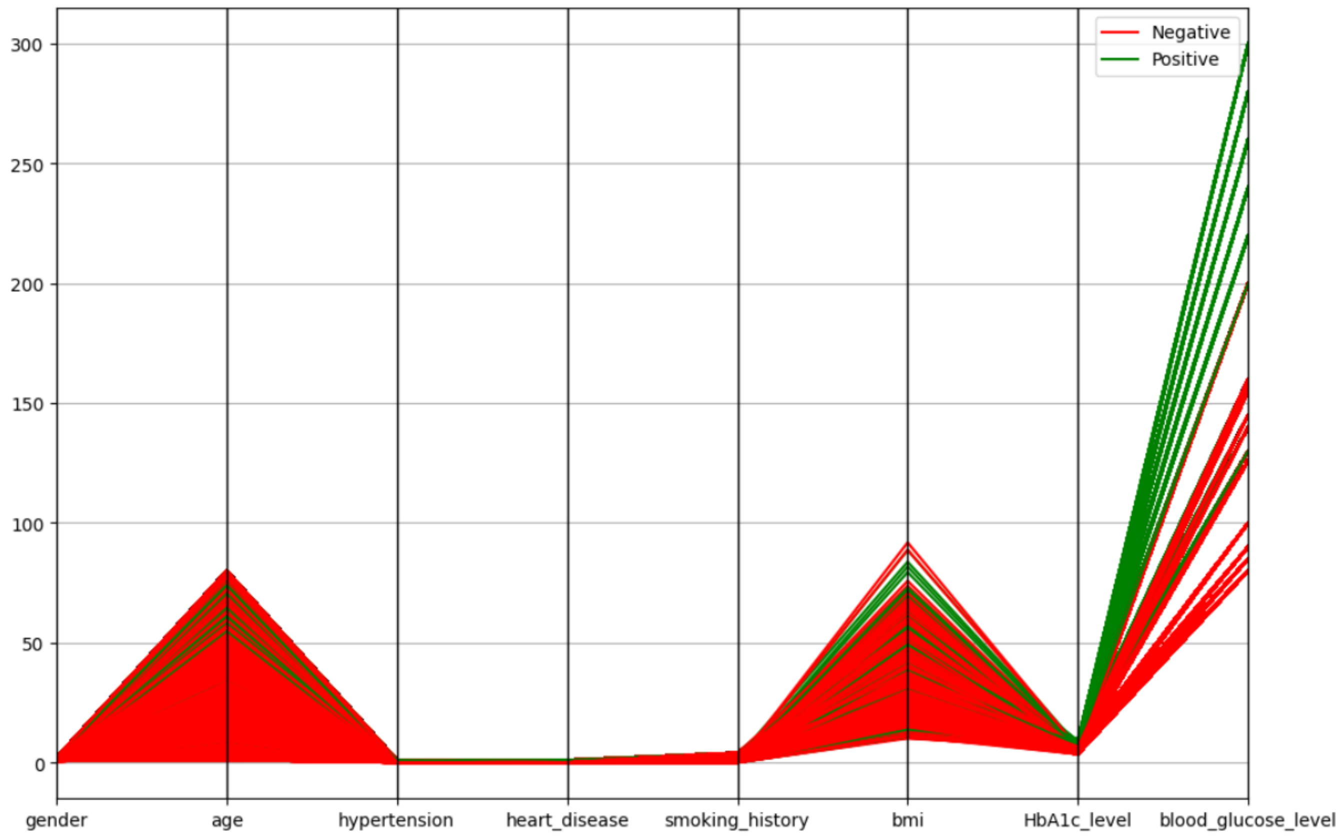
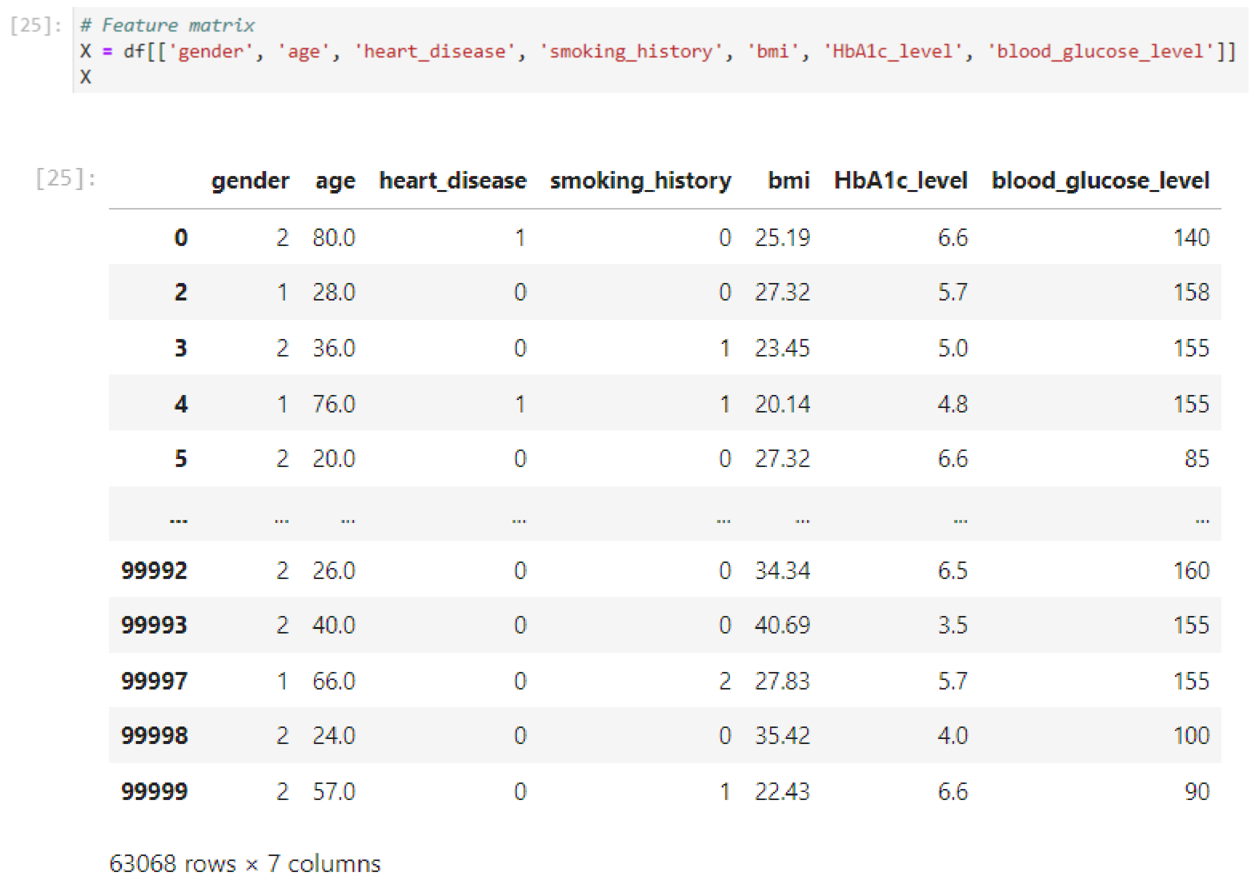


Figure: Parallel Co-ordinate

Features Matrix and Target Variable:

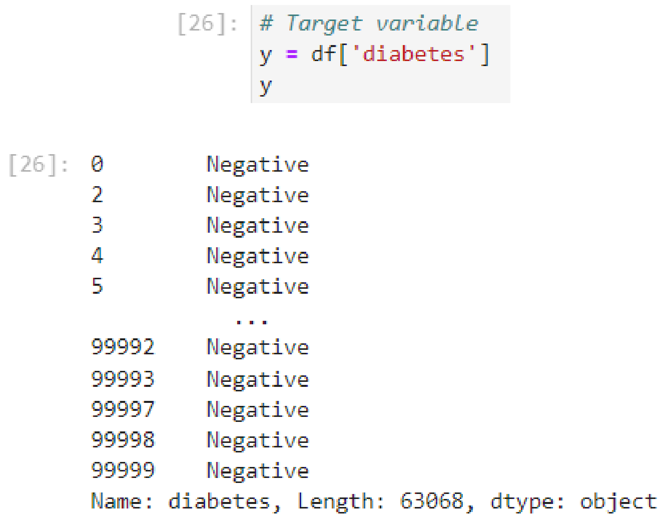


```
[25]: # Feature matrix
      X = df[['gender', 'age', 'heart_disease', 'smoking_history', 'bmi', 'HbA1c_level', 'blood_glucose_level']]
      X
```

[25]:

	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

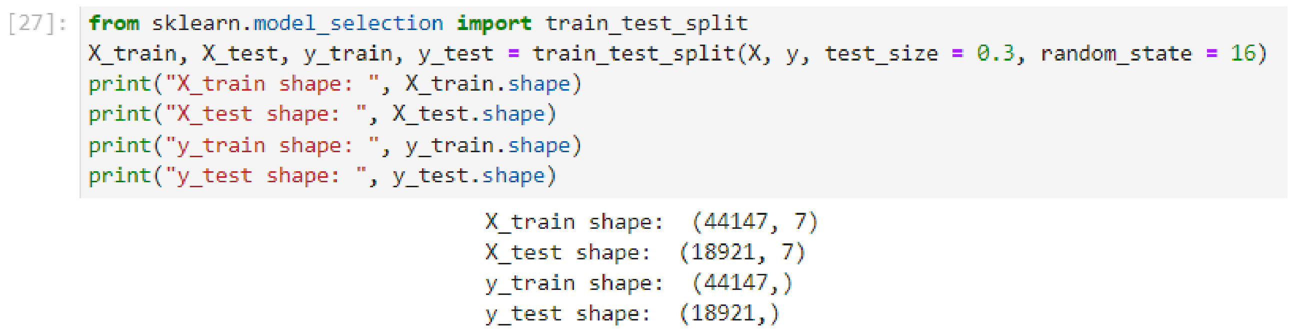


```
[26]: # Target variable
      y = df['diabetes']
      y
```

```
[26]: 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:

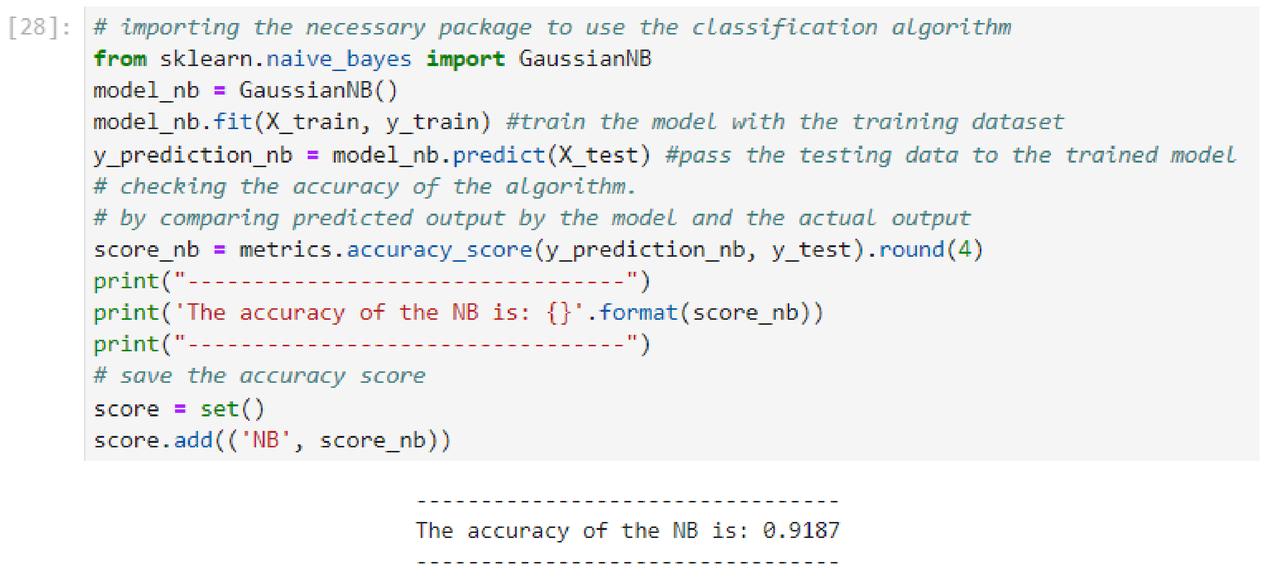


```
[27]: from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 16)
      print("X_train shape: ", X_train.shape)
      print("X_test shape: ", X_test.shape)
      print("y_train shape: ", y_train.shape)
 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

Figure: Train and Test Matrix

Naïve Bayes Model:

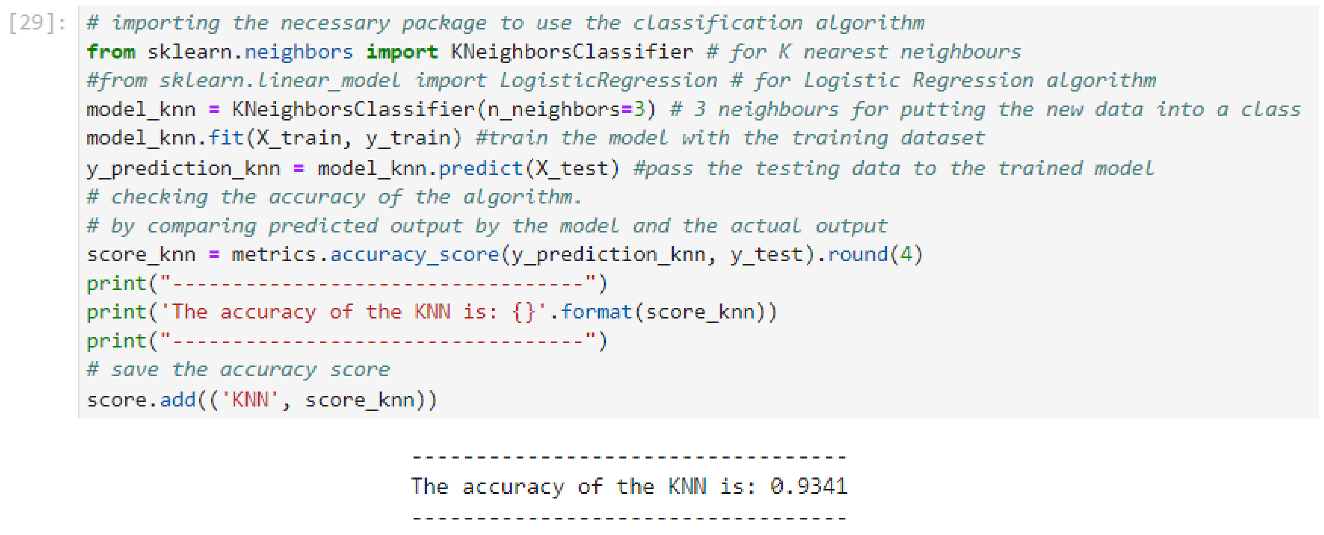


```
[28]: # importing the necessary package to use the classification algorithm
      from sklearn.naive_bayes import GaussianNB
      model_nb = GaussianNB()
      model_nb.fit(X_train, y_train) #train the model with the training dataset
      y_prediction_nb = model_nb.predict(X_test) #pass the testing 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 = set()
      score.add(('NB', score_nb))
```

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

Figure: Naïve Bayes Model

K-Nearest Neighbor Model:

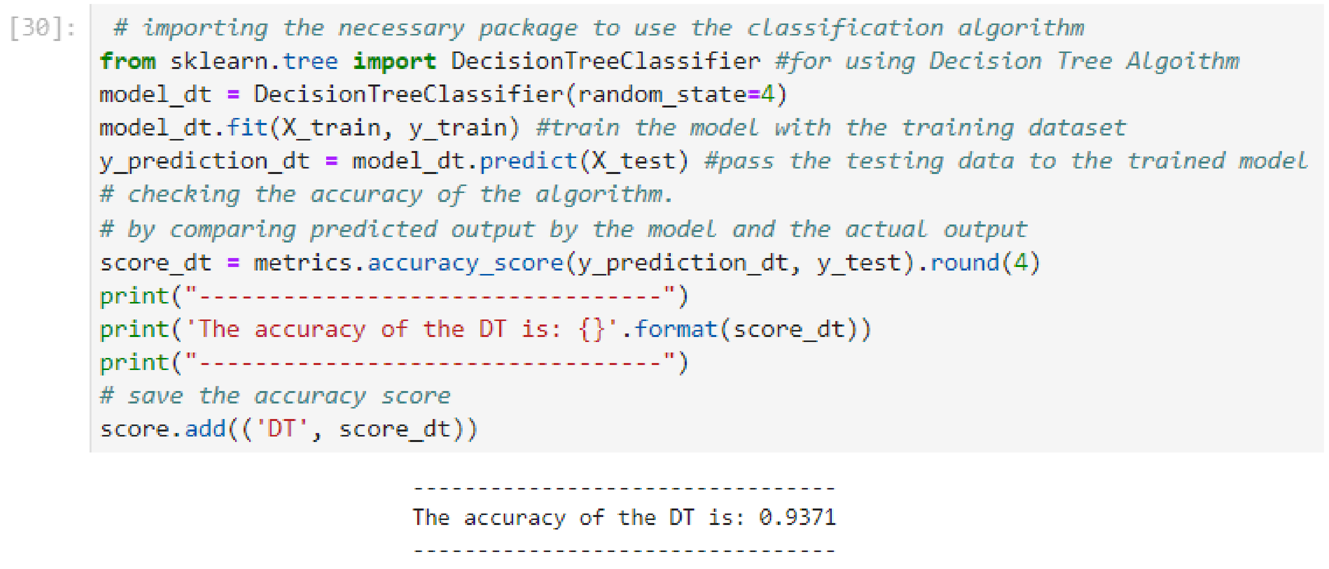


```
[29]: # importing the necessary package to use the classification algorithm
      from sklearn.neighbors import KNeighborsClassifier # for K nearest neighbours
      #from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
      model_knn = KNeighborsClassifier(n_neighbors=3) # 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
----------------------------------
```

Figure: K-Nearest Neighbor (KNN) Model

Decision Tree Model:

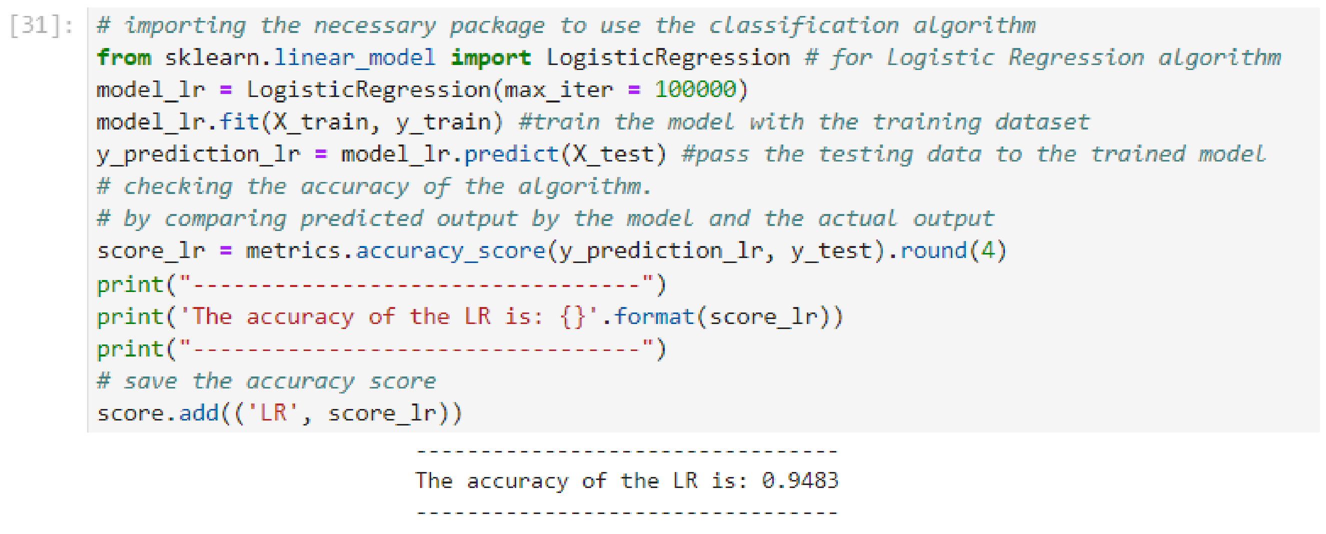


```
[30]:  # importing the necessary package to use the classification algorithm
      from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm
      model_dt = DecisionTreeClassifier(random_state=4)
      model_dt.fit(X_train, y_train) #train the model with the training dataset
      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
--------------------------------
```

Figure: Decision Tree Model

Logistic Regression Model:

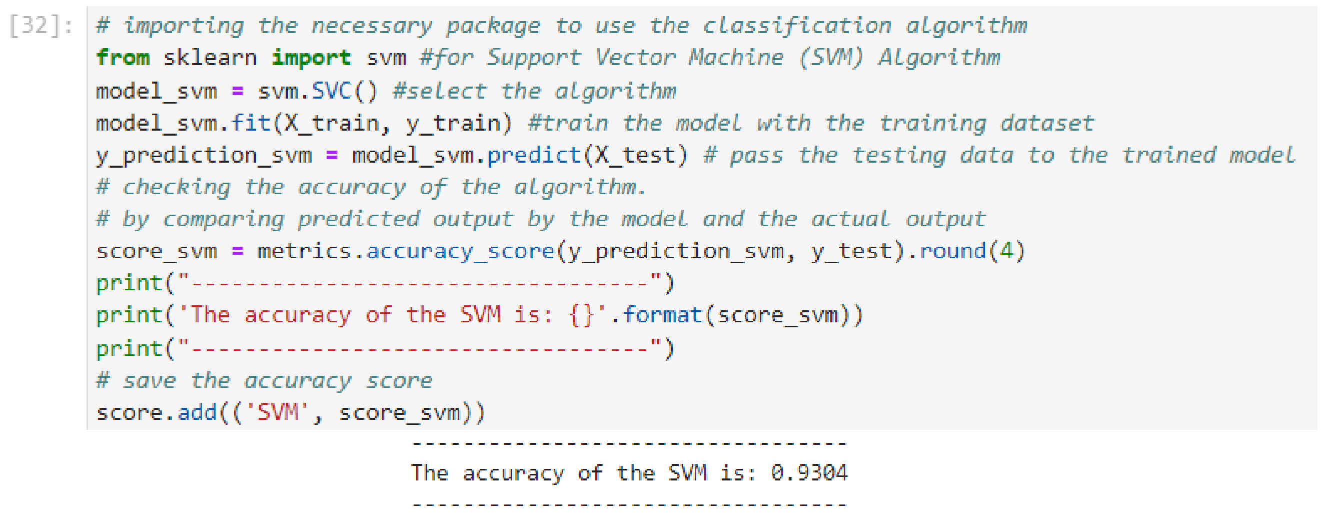


```
[31]: # importing the necessary package to use the classification algorithm
      from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
      model_lr = LogisticRegression(max_iter = 100000)
      model_lr.fit(X_train, y_train) #train the model with the training 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(4)
      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
---------------------------------
```

Figure: Logistic Regression Model

Support Vector Machine Model:



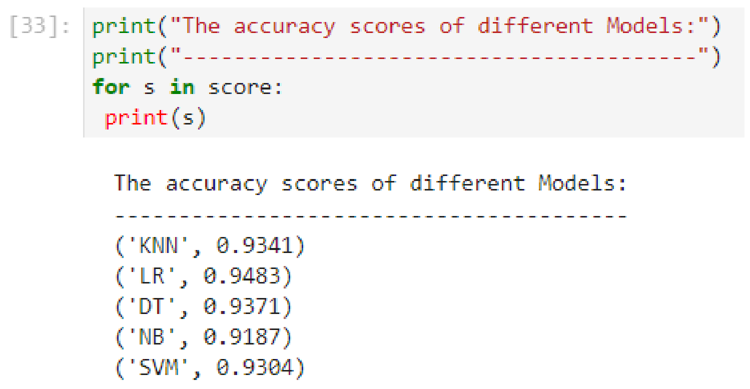
```
[32]: # importing the necessary package to use the classification algorithm
      from sklearn import svm #for Support Vector Machine (SVM) Algorithm
      model_svm = svm.SVC() #select the algorithm
      model_svm.fit(X_train, y_train) #train the model with the training dataset
      y_prediction_svm = model_svm.predict(X_test) # pass the testing data to the trained model
      # 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.add(('SVM', score_svm))
```

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

Figure: Support Vector Machine Model

Discussion and Conclusion:

Comparison Models:



```
[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)
```

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:

Here the purpose of this report was to identify a better accuracy for the diabetes prediction dataset that will be able to categorize the reason to get diabetes as correctly as possible and be able to forecast future cases from prediction. 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. This also helps to give an unbiased evaluation of the accuracy of the model itself.