Report on

Diabetes Prediction System using Machine Learning algorithms

Submitted by-

Samiksha S. Lohe

Abstract

There are several machine learning techniques that are used to train the diabetes predictive system. These system is predicting the whether the individual has diabetes or not. Diabetes is a chronic disease that occurs when the pancreas is no longer able to make insulin, or when the body cannot make good use of the insulin it produces. That's why with this result, he will confirm his health issues and consult to the doctor if needed. For experiment purpose, a dataset of patient's medical record is obtained and five different machine learning algorithms are applied on the dataset. Performance and accuracy of the applied algorithms is discussed and compared. Comparison of the different machine learning techniques used here for prediction of diabetes. Decision tree is one of popular machine learning methods in medical field, which has grateful classification power. Random forest algorithm generates many decision trees. This project aims to help doctors and practitioners significantly in early prediction of diabetes using machine learning techniques. Predictive techniques are using both classification and regression to classify vast amounts of data and measure its accuracy by using training and testing datasets.

Table of Contents

List of figures:

1. Introduction:

Diabetes dramatically increases the risk of various cardiovascular problems, including coronary artery disease with chest pain (angina), heart attack, and stroke and narrowing of arteries (atherosclerosis). If you have diabetes, you're more likely to have heart disease or stroke. It affects the hormone insulin, resulting in abnormal metabolism of crabs and improves level of sugar in the blood. Diabetes occurs when body does not make enough insulin. For this purpose we use the Pima Indian Diabetes Dataset, we apply various Machine Learning classification and ensemble Techniques to predict diabetes. For this project, we use various features in the dataset which are responsible to cause diabetes such as Age, obesity, lack of exercise, hereditary diabetes, living style, bad diet, high blood pressure, etc. Machine Learning is a method that is used to train computers or machines explicitly. Various Machine Learning Techniques provide efficient result to collect Knowledge by building various classification and ensemble models from collected dataset. Such collected data can be useful to predict diabetes. Various techniques of Machine Learning are capable to do prediction, however it is tough to choose best technique. Thus we train the model by using numerous methods. The performances of all these algorithms are evaluated on various measures like Precision, Accuracy, F-Measure, and Recall. Accuracy is measured over correctly and incorrectly classified instances.

The given diagram shows the workflow of the system to predict the diabetes of a person -

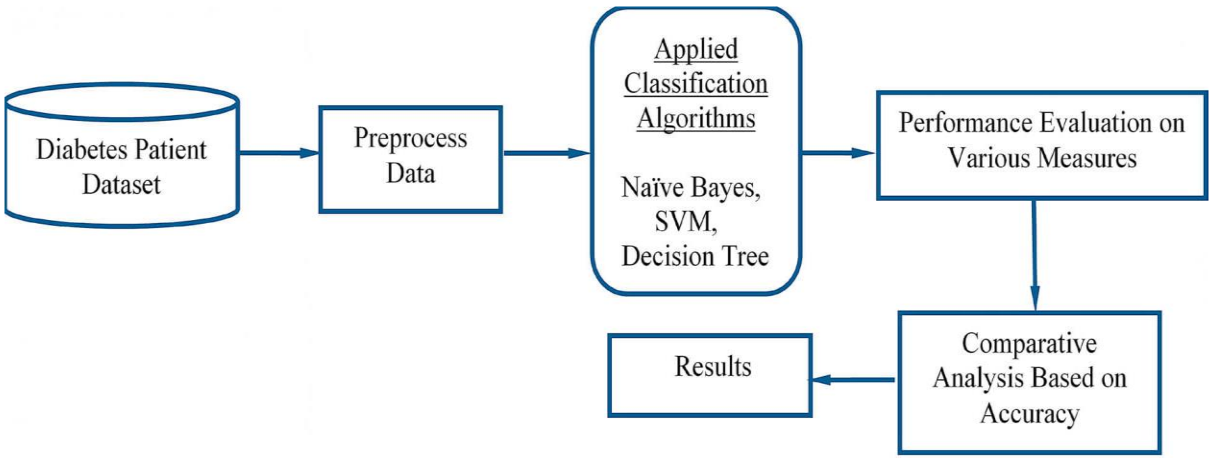

Diabetes Patient Dataset
Preprocess Data
Applied Classification Algorithms
Naïve Bayes, SVM, Decision Tree
Performance Evaluation on Various Measures
Comparative Analysis Based on Accuracy
Results

Fig. 1: Workflow to predict the Diabetes status

2. Methods with their architecture:

2.1 SVM:

SVM stands for Support Vector Machine. SVM is a supervised machine learning algorithm that is commonly used for classification and regression challenges. Common applications of the SVM algorithm are Intrusion Detection System, Handwriting Recognition, Protein Structure Prediction, and Detecting Steganography in digital images, etc. In the SVM algorithm, each point is represented as a data item within the n-dimensional space where the value of each feature is the value of a specific coordinate.

After plotting, classification has been performed by finding hype-plane, which differentiates two classes.

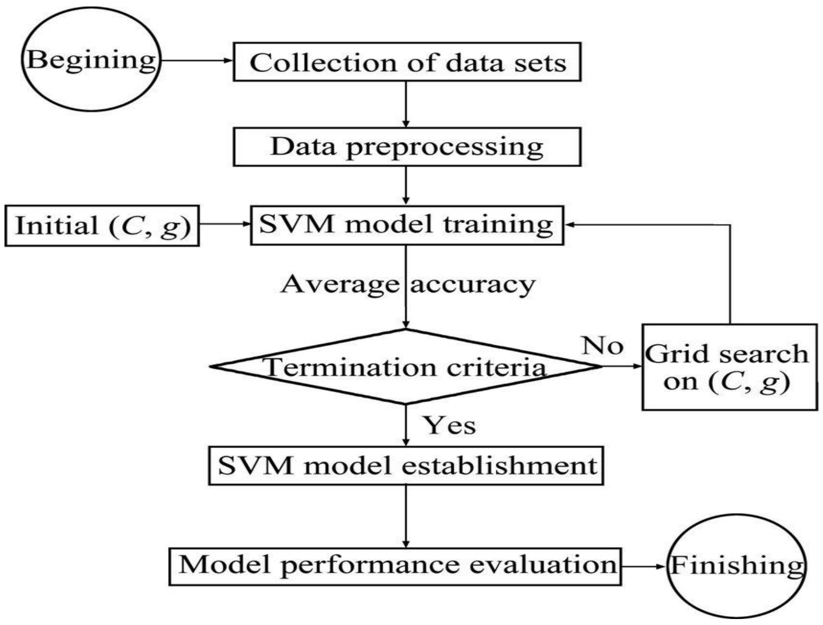


Fig. 2.1: Architecture of SVM

2.2 KNN:

K-nearest neighbours (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. K-nearest neighbours (KNN) algorithm uses ‘feature similarity’ to predict the values of new data points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps -

Algorithm

Step 1: For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2: Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3: For each point in the test data do the following −

> **3.1**: Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
>
> **3.2**: Now, based on the distance value, sort them in ascending order.
>
> **3.3**: Next, it will choose the top K rows from the sorted array.
>
> **3.4**: Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4: End

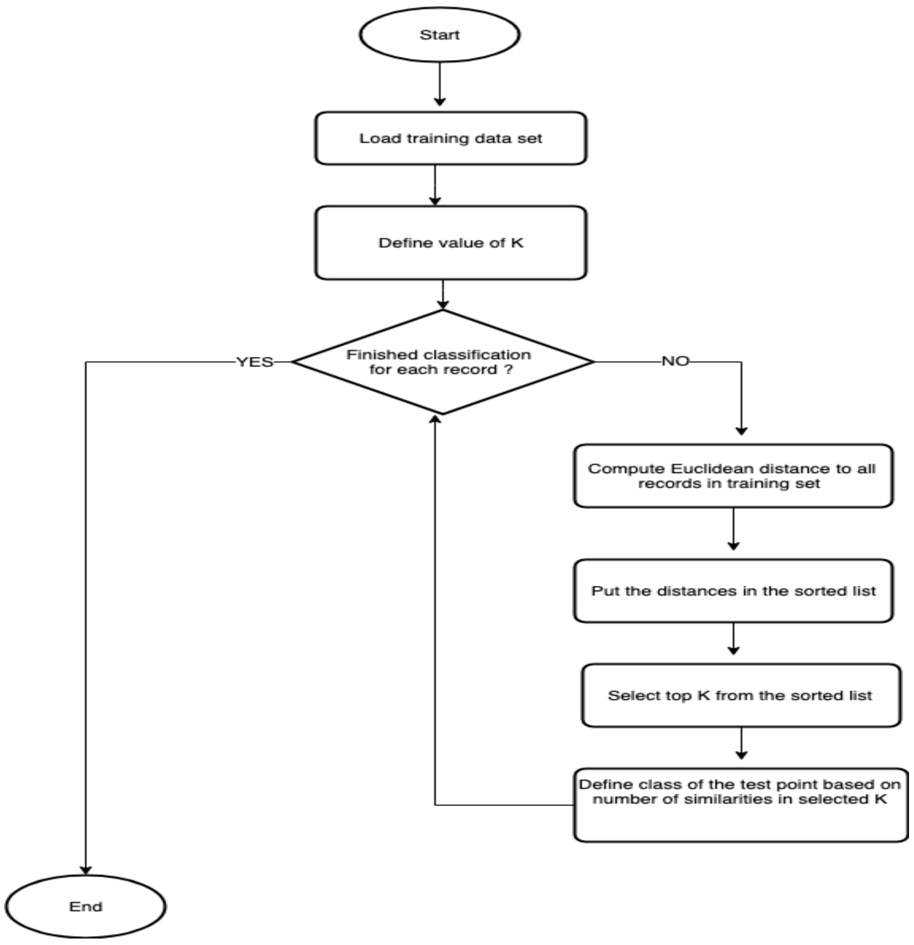

Start
Load training data set
Define value of K
Finished classification for each record ?
YES
NO
Compute Euclidean distance to all records in training set
Put the distances in the sorted list
Select top K from the sorted list
Define class of the test point based on number of similarities in selected K
End

Fig. 2.2: Architecture of KNN

2.3 Decision tree:

Decision Tree is a supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset.

Algorithm

Step 1: Begin the tree with the root node, says S, which contains the complete dataset.

Step 2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

Step 3: Divide the S into subsets that contains possible values for the best attributes.

Step 4: Generate the decision tree node, which contains the best attribute.

Step 5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

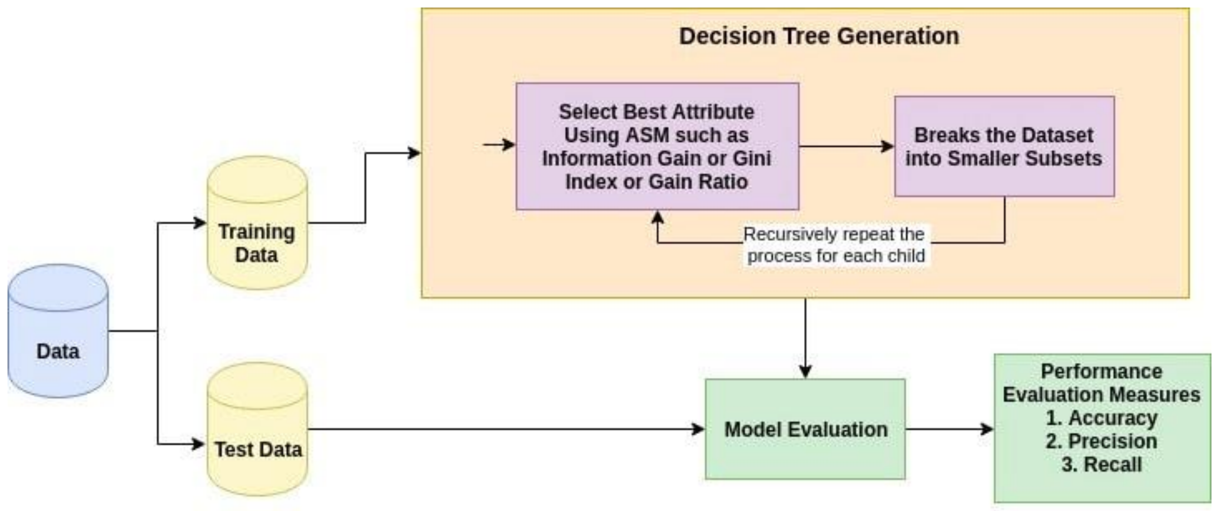

Decision Tree Generation
Select Best Attribute Using ASM such as Information Gain or Gini Index or Gain Ratio
Breaks the Dataset into Smaller Subsets
Recursively repeat the process for each child
Training Data
Data
Test Data
Model Evaluation
Performance Evaluation Measures
1. Accuracy
2. Precision
3. Recall

Fig. 2.3: Architecture of Random forest

2.4 Naïve Bayes:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset.

Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

The formula for Bayes' theorem is given as:

The performances of all the three algorithms are evaluated on various measures like Precision, Accuracy, F-Measure, and Recall. Accuracy is measured over correctly and incorrectly classified instances.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Algorithm

Step 1: Data Pre-processing step:

In this step, we will pre-process/prepare the data so that we can use it efficiently in our code. It is similar as we did in data-pre-processing.

Step 2: Fitting Naive Bayes to the Training Set:

After the pre-processing step, now we will fit the Naive Bayes model to the Training set.

Step 3: Prediction of the test set result:

Now we will predict the test set result. For this, we will create a new predictor variable **y_pred**, and will use the predict function to make the predictions.

Step 4: Creating Confusion Matrix:

Now we will check the accuracy of the Naive Bayes classifier using the Confusion matrix.

Step 5: Visualizing the training set result:

Next we will visualize the training set result using Naïve Bayes Classifier.

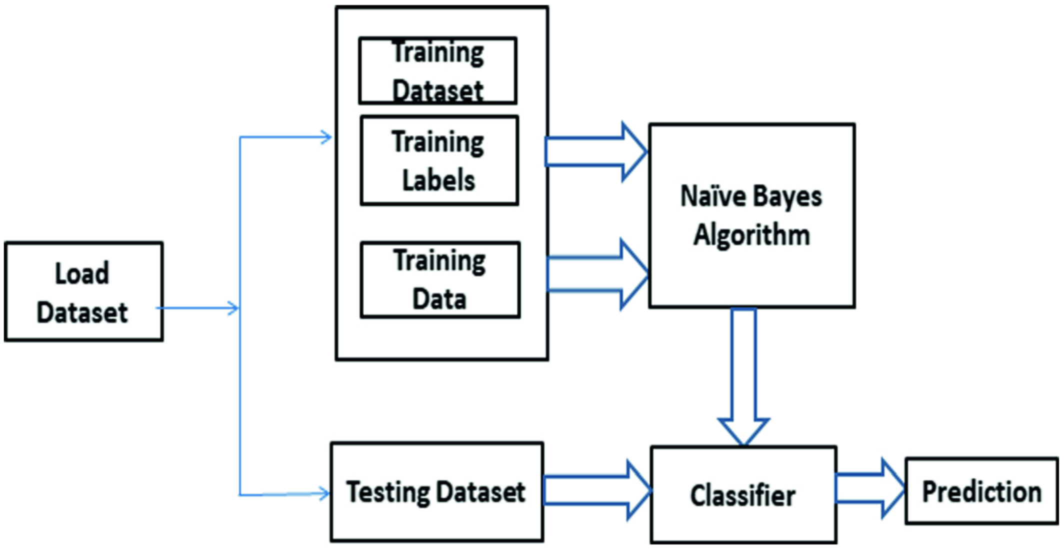


Fig. 2.4: Architecture of Naïve Bayes

2.5 Logistic Regression:

Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable (or output), y, can take only discrete values for given set of features (or inputs), X. Logistic regression is generally used where we have to classify the data into two or more classes. One is binary and the other is multi-class logistic regression. As the name suggests, the binary class has 2 classes that are Yes/No, True/False, 0/1, etc. In multi-class classification, there are more than 2 classes for classifying data.

Contrary to popular belief, logistic regression IS a regression model. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered as "1". Just like linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function.

Algorithm

Step 1: Data Preprocessing

Step 2: Unsupervised cluster (Use k-means clustering to remove incorrectly classified data).

Step 3: Calculate rate=remainder (data/sum).

Step 4: If the rate has lower accuracy, then execute again with another value of data.

Step 5: Use classification regression algorithm and model the validation.

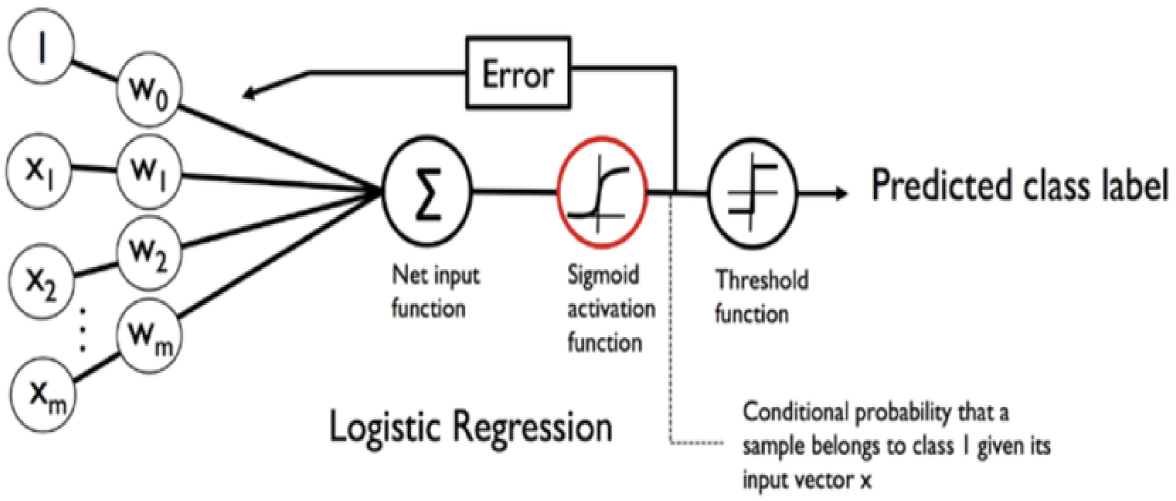
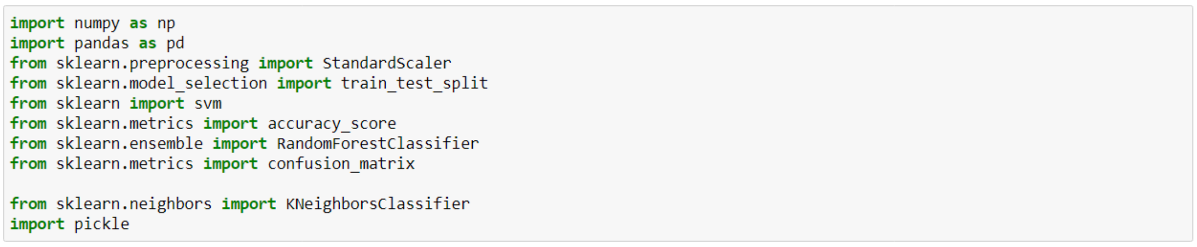

1
x1
x2
xm
w0
w1
w2
wm
Error
Σ
Net input function
Sigmoid activation function
Threshold function
Predicted class label
Logistic Regression
Conditional probability that a sample belongs to class 1 given its input vector x

Fig. 2.5: Architecture of Logistic Regression

3. Implementation:

The dataset used to implement diabetes prediction system was collected from PIMA Indian diabetes dataset from Kaggle. For this prediction, we use various features in the dataset which are responsible to cause diabetes such as Age, obesity, lack of exercise, hereditary diabetes, living style, bad diet, high blood pressure, etc.

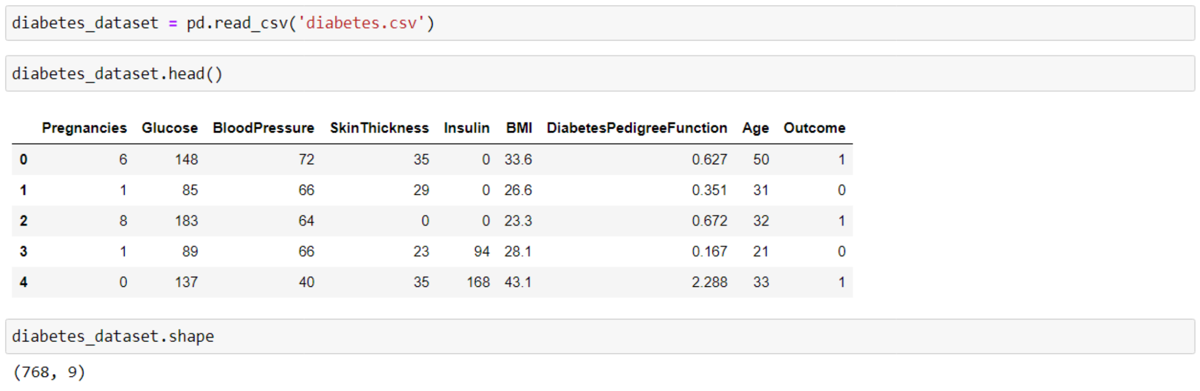
1. To create webapp for the prediction of diabetes status of person, first of all we have to import required libraries. Importing libraries in Jupyter notebook is as follows:



```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix

from sklearn.neighbors import KNeighborsClassifier
import pickle
```

Fig. 3.1: Importing the required libraries

2. Data pre-processing: Afterwards, we have to read the csv file with read_csv() function. Initialise it with the variable diabetes_dataset and display the top 5 rows of that dataset. And check the total number of rows and columns by using shape function



```python
diabetes_dataset = pd.read_csv('diabetes.csv')
```

```python
diabetes_dataset.head()
```

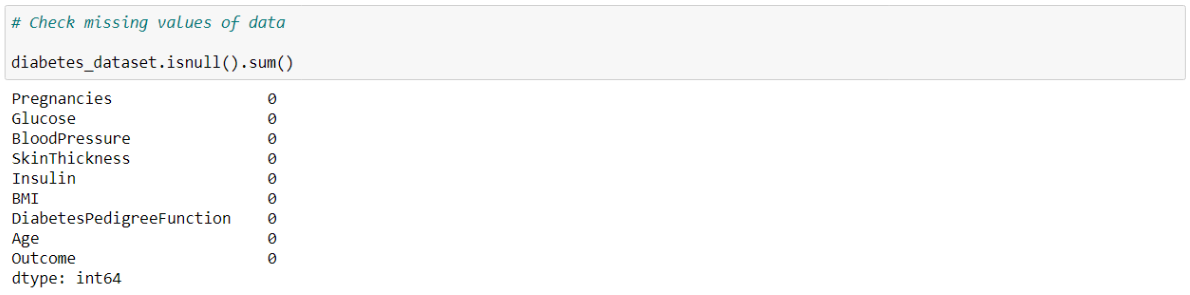
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```python
diabetes_dataset.shape
```

```
(768, 9)
```

Fig. 3.2: Reading the first 5 rows of imported data

To remove the further anomalies in the dataset, we need to check any null or zero values present there. Then check if there is any null values present in the dataset or not with *isnull().sum()* function. This function checks the null values column by column. Here, zero value indicates that our dataset is clean for further operations.

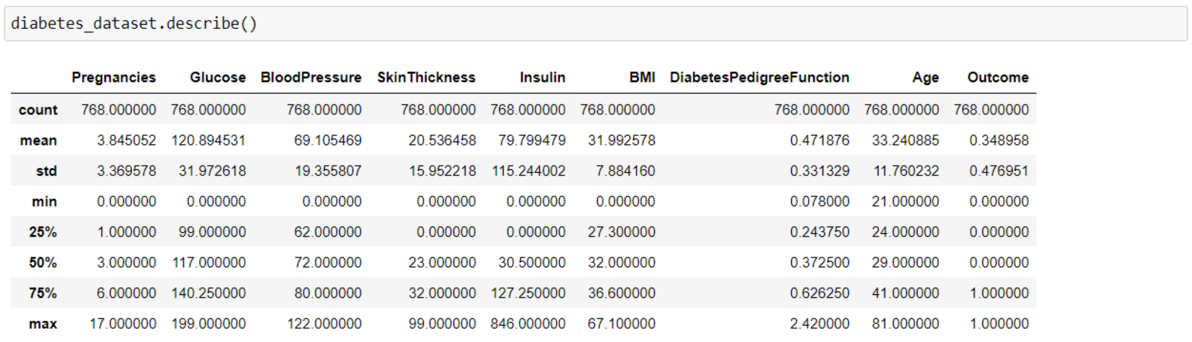


```
# Check missing values of data

diabetes_dataset.isnull().sum()
```

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

Fig. 3.3: Checking the missing values of data

3. Statistical measures of data: Then describe() function returns values of count, mean, standard deviation, minimum and maximum value, and so forth for every column of dataset. This function computes the statistical values of each column and further used to analyse dataset.

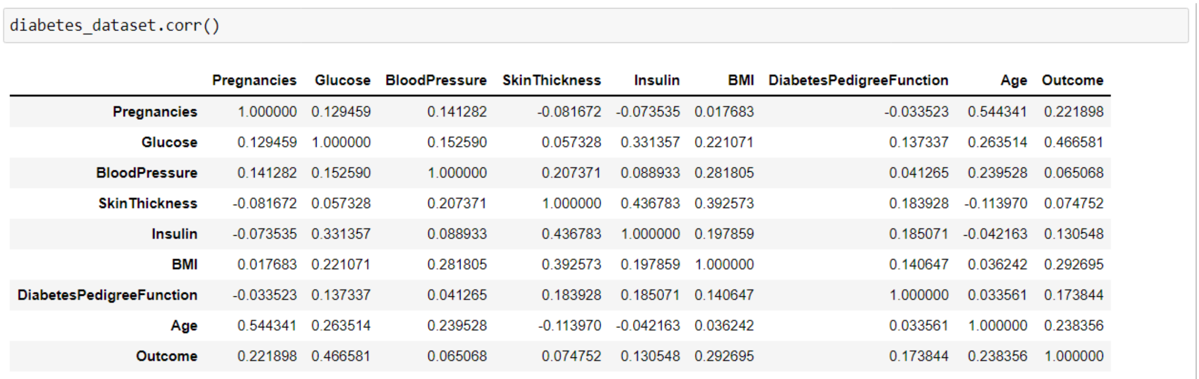


```
diabetes_dataset.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Fig. 3.4: Representing the statistical values of data

This *corr()* method is used to find the pairwise correlation of all columns in the dataset. Also, it can be used to find out dependencies between column variables. Dependency of same variable is always 1; hence we find out dependency as 1 diagonally.



```
diabetes_dataset.corr()
```

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

Fig. 3.5: Evaluation of correlation between columns

A heatmap is a two-dimensional graphical representation of data where the individual values that are contained in a matrix are represented as colors. The seaborn python package allows the creation of annotated heatmaps which can be tweaked using Matplotlib tools. Usually the darker shades of the chart represent higher values than the lighter shade.

```
#correlation map without any preprocessing

plt.figure(dpi = 125,figsize= (5,4))
mask = np.triu(diabetes_dataset.corr())
sns.heatmap(diabetes_dataset.corr(),mask = mask, fmt = ".1f",annot=True,lw=0.1,cmap = 'YlGnBu')
plt.title('Correlation Map')
plt.show()
```

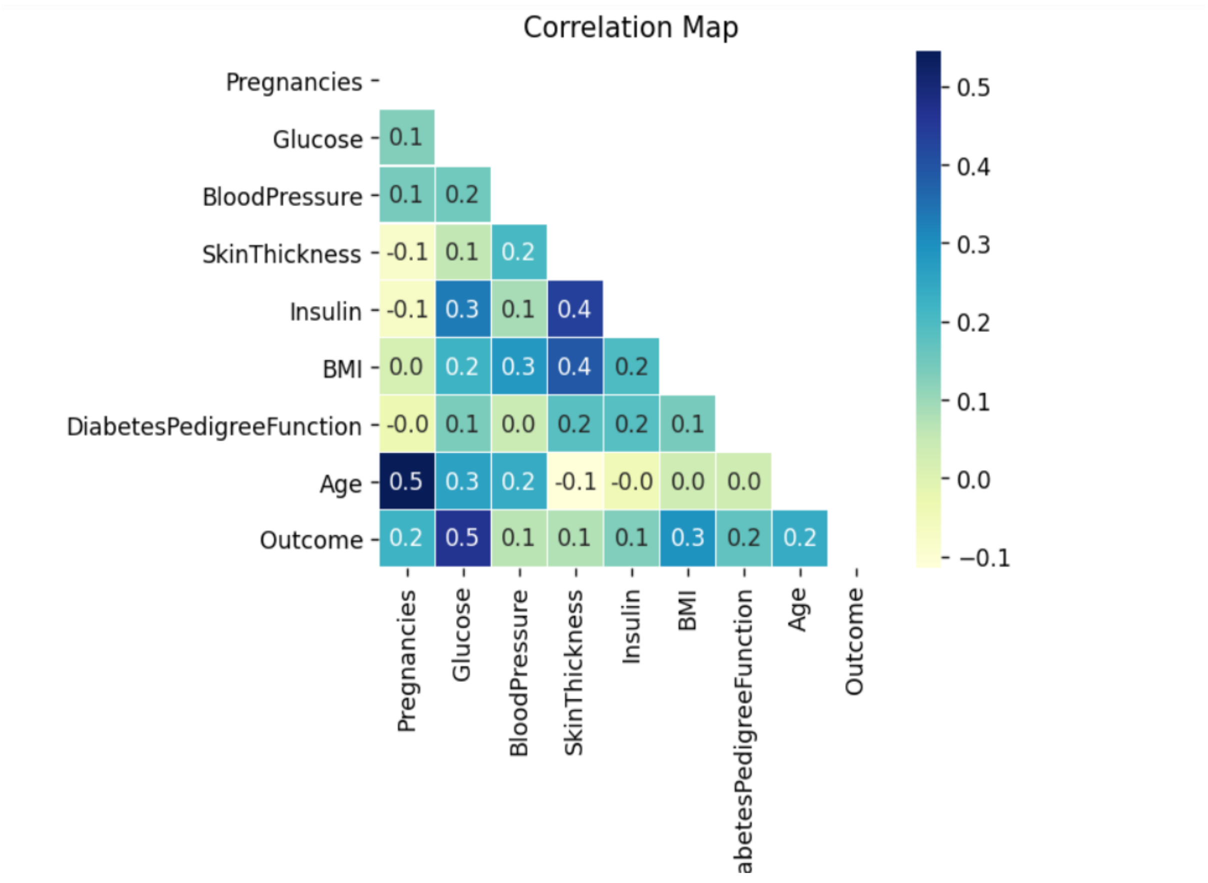
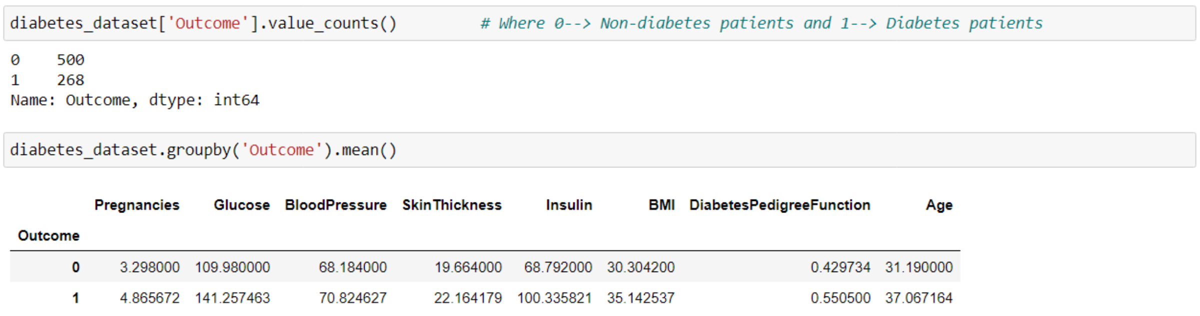


Fig. 3.6: Displaying correlation using heatmap

4. The target column is ‘Outcome’ column; check the distribution of diabetes patient with numeric data. Here, *groupby*() clause is used to find its distribution per column.



```
diabetes_dataset['Outcome'].value_counts()          # Where 0--> Non-diabetes patients and 1--> Diabetes patients
```

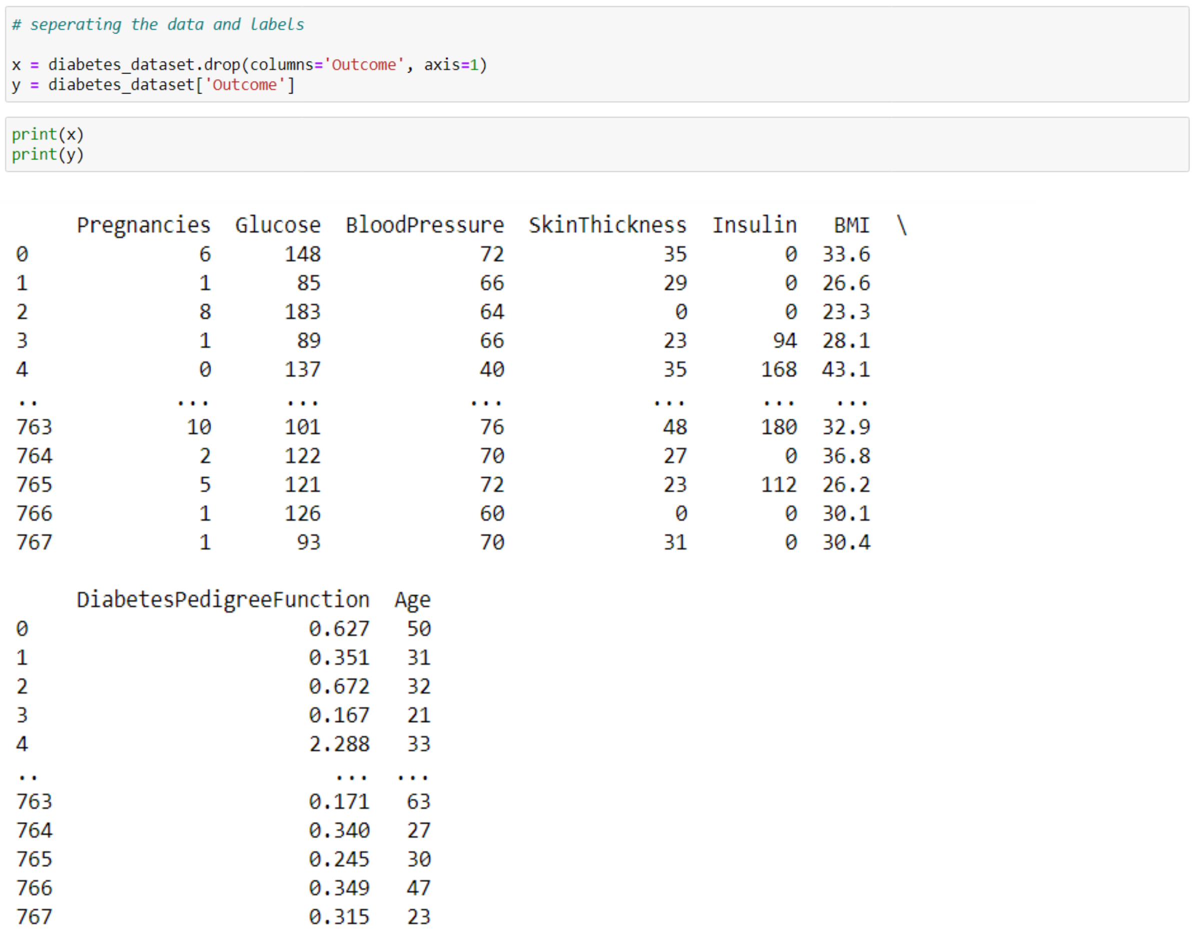
```
0    500
1    268
Name: Outcome, dtype: int64
```

```
diabetes_dataset.groupby('Outcome').mean()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Outcome								
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

Fig. 3.7: Distribution of number of diabetes and non-diabetes patients

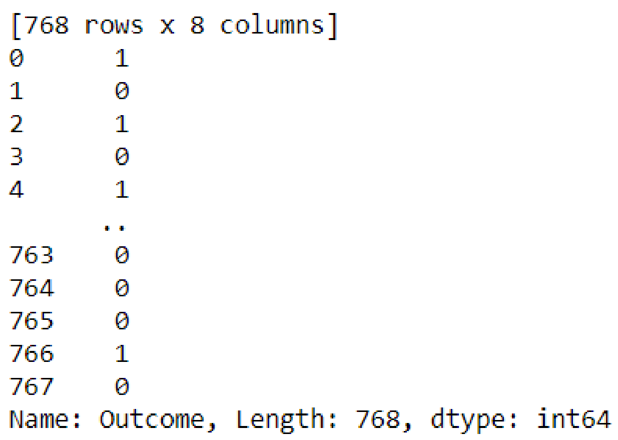
5. Separating the data and labels: We created two variables separating the data columns and the target column. X and Y variables divide the dataset by dropping the outcome column and saved it into Y variable.



```
# seperating the data and labels

x = diabetes_dataset.drop(columns='Outcome', axis=1)
y = diabetes_dataset['Outcome']
```

```
print(x)
print(y)
```

```
     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
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23
```

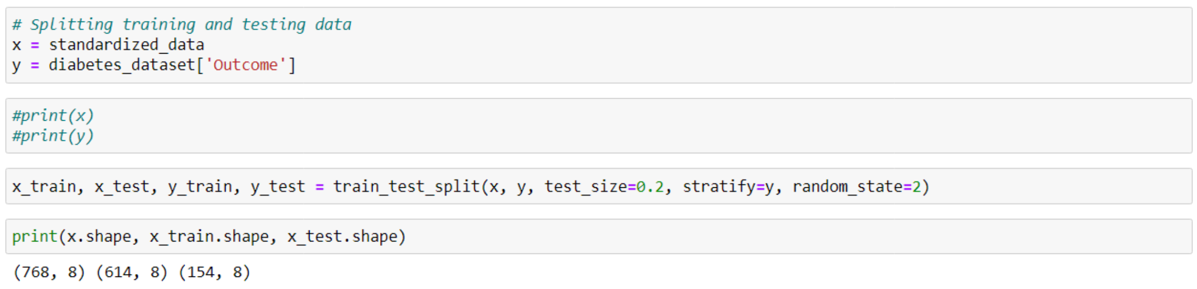


```
[768 rows x 8 columns]
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Fig. 3.8: Separating the data and target columns

6. Splitting the data into training and testing dataset: We splits the dataset into two categories i.e. training data and test data. Training data is used to train the model and we perform prediction on test data. It is such data where input and output is known to model and it finds pattern accordingly. Test data is such a data where model predicts the output with respect to the inputs provided to it. And then we measure the accuracy of test data.

   Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.



```
# Splitting training and testing data
x = standardized_data
y = diabetes_dataset['Outcome']
```

```
#print(x)
#print(y)
```

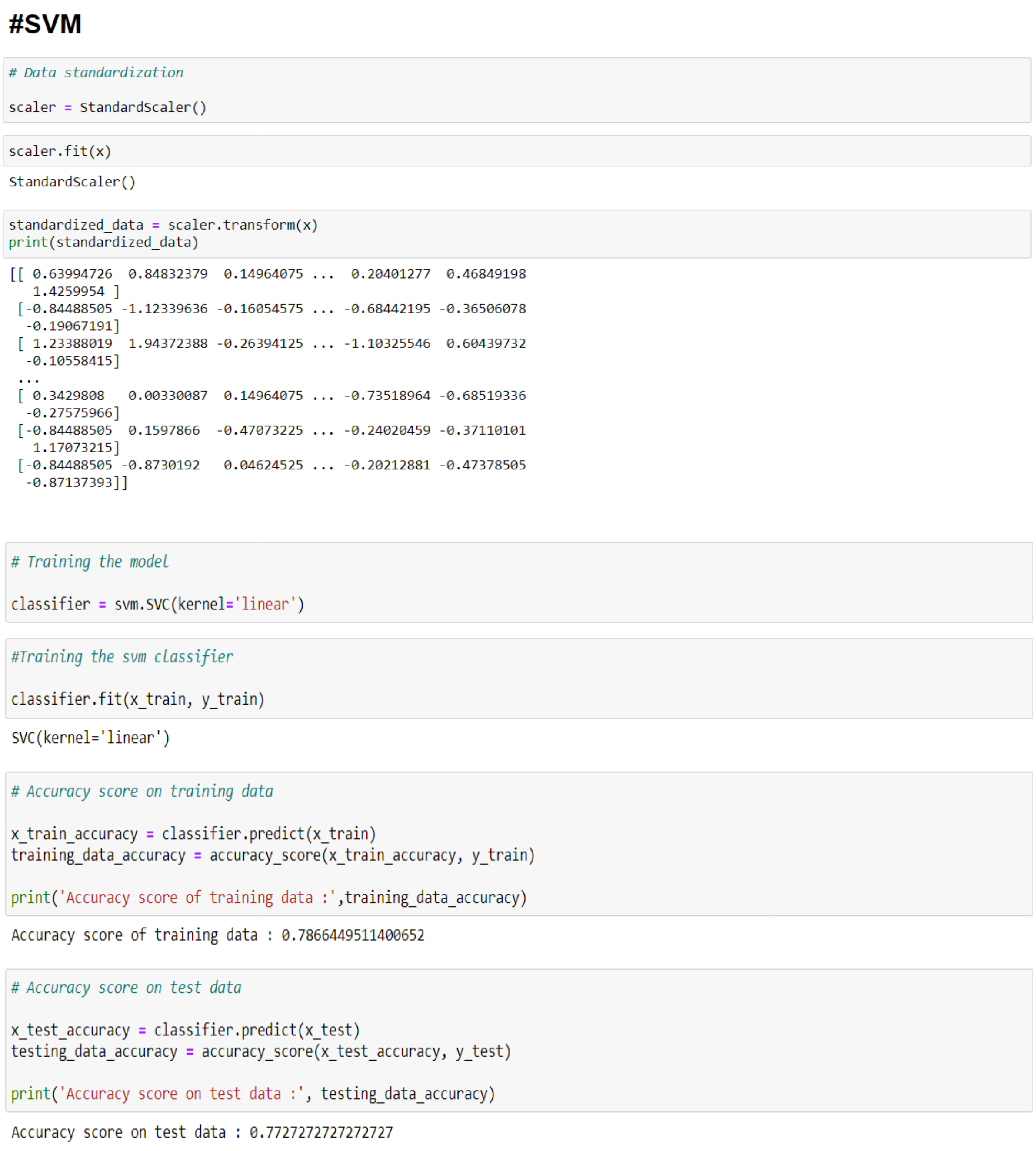
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y, random_state=2)
```

```
print(x.shape, x_train.shape, x_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

Fig. 3.9: Splitting the data into training and testing dataset

7. Methodologies used to train the model:
    a. SVM



#SVM

```
# Data standardization

scaler = StandardScaler()
```

```
scaler.fit(x)
```

```
StandardScaler()
```

```
standardized_data = scaler.transform(x)
print(standardized_data)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

```
# Training the model

classifier = svm.SVC(kernel='linear')
```

```
#Training the svm classifier

classifier.fit(x_train, y_train)
```

```
SVC(kernel='linear')
```

```
# Accuracy score on training data

x_train_accuracy = classifier.predict(x_train)
training_data_accuracy = accuracy_score(x_train_accuracy, y_train)

print('Accuracy score of training data :',training_data_accuracy)
```

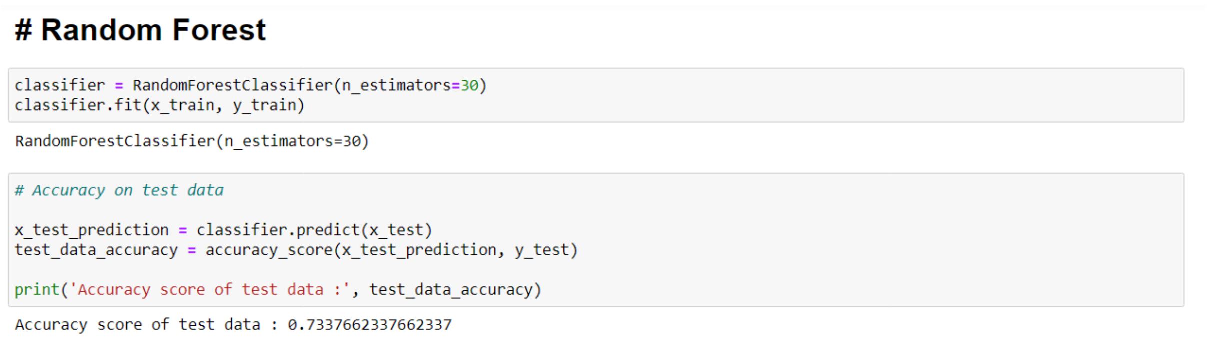
```
Accuracy score of training data : 0.7866449511400652
```

```
# Accuracy score on test data

x_test_accuracy = classifier.predict(x_test)
testing_data_accuracy = accuracy_score(x_test_accuracy, y_test)

print('Accuracy score on test data :', testing_data_accuracy)
```

```
Accuracy score on test data : 0.7727272727272727
```

Fig. 3.10: Code to train the model using SVM with their accuracies

b. Decision Tree



Random Forest

```
classifier = RandomForestClassifier(n_estimators=30)
classifier.fit(x_train, y_train)
```

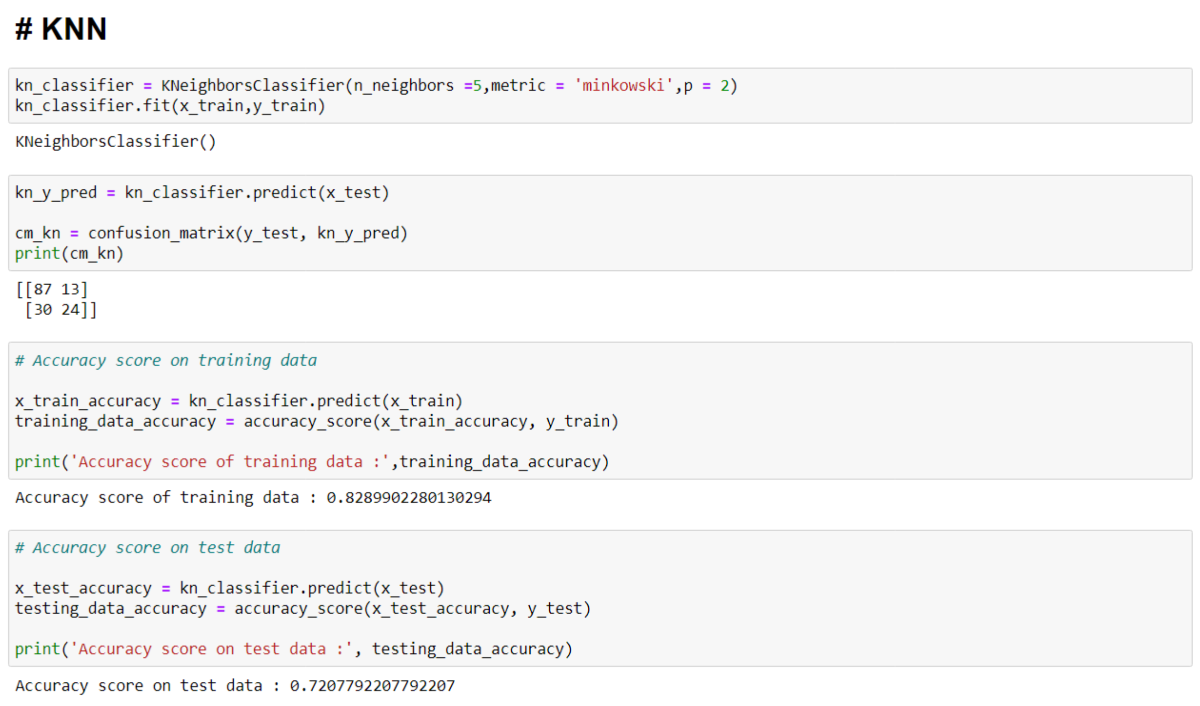
```
RandomForestClassifier(n_estimators=30)
```

```
# Accuracy on test data

x_test_prediction = classifier.predict(x_test)
test_data_accuracy = accuracy_score(x_test_prediction, y_test)

print('Accuracy score of test data :', test_data_accuracy)
```

```
Accuracy score of test data : 0.7337662337662337
```

Fig. 3.11: Code to train the model using Random Forest with their accuracies

c. KNN



KNN

```
kn_classifier = KNeighborsClassifier(n_neighbors =5,metric = 'minkowski',p = 2)
kn_classifier.fit(x_train,y_train)
```

```
KNeighborsClassifier()
```

```
kn_y_pred = kn_classifier.predict(x_test)

cm_kn = confusion_matrix(y_test, kn_y_pred)
print(cm_kn)
```

```
[[87 13]
 [30 24]]
```

```
# Accuracy score on training data

x_train_accuracy = kn_classifier.predict(x_train)
training_data_accuracy = accuracy_score(x_train_accuracy, y_train)

print('Accuracy score of training data :',training_data_accuracy)
```

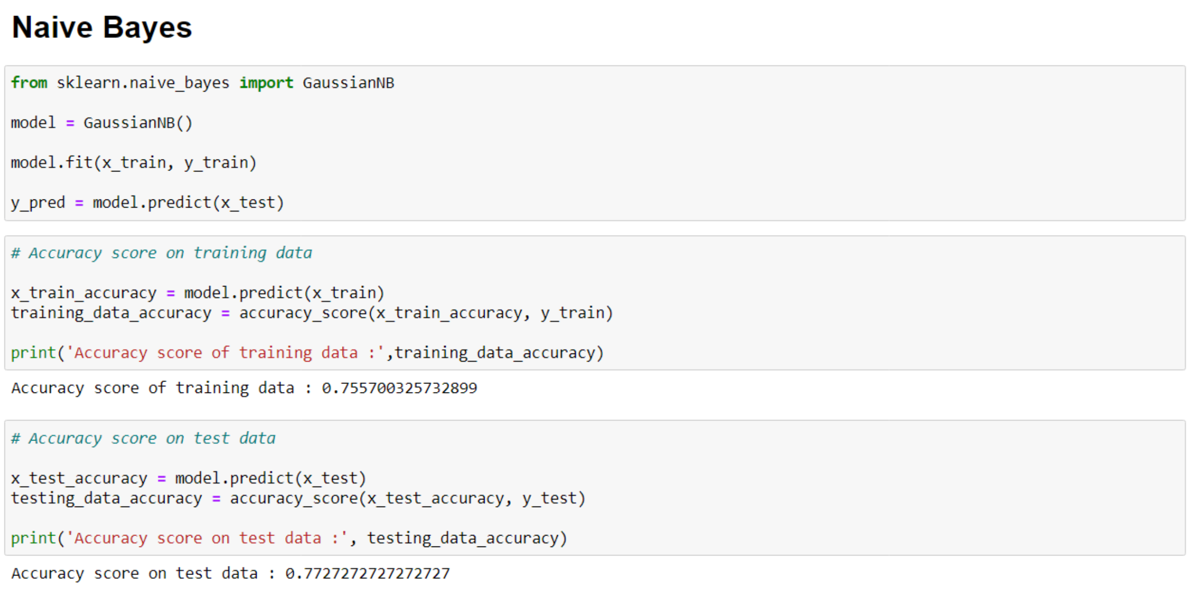
```
Accuracy score of training data : 0.8289902280130294
```

```
# Accuracy score on test data

x_test_accuracy = kn_classifier.predict(x_test)
testing_data_accuracy = accuracy_score(x_test_accuracy, y_test)

print('Accuracy score on test data :', testing_data_accuracy)
```

```
Accuracy score on test data : 0.7207792207792207
```

Fig. 3.12: Code to train the model using KNN with their accuracies

d. Naïve Bayes



Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()

model.fit(x_train, y_train)

y_pred = model.predict(x_test)
```

```
# Accuracy score on training data

x_train_accuracy = model.predict(x_train)
training_data_accuracy = accuracy_score(x_train_accuracy, y_train)

print('Accuracy score of training data :',training_data_accuracy)
```

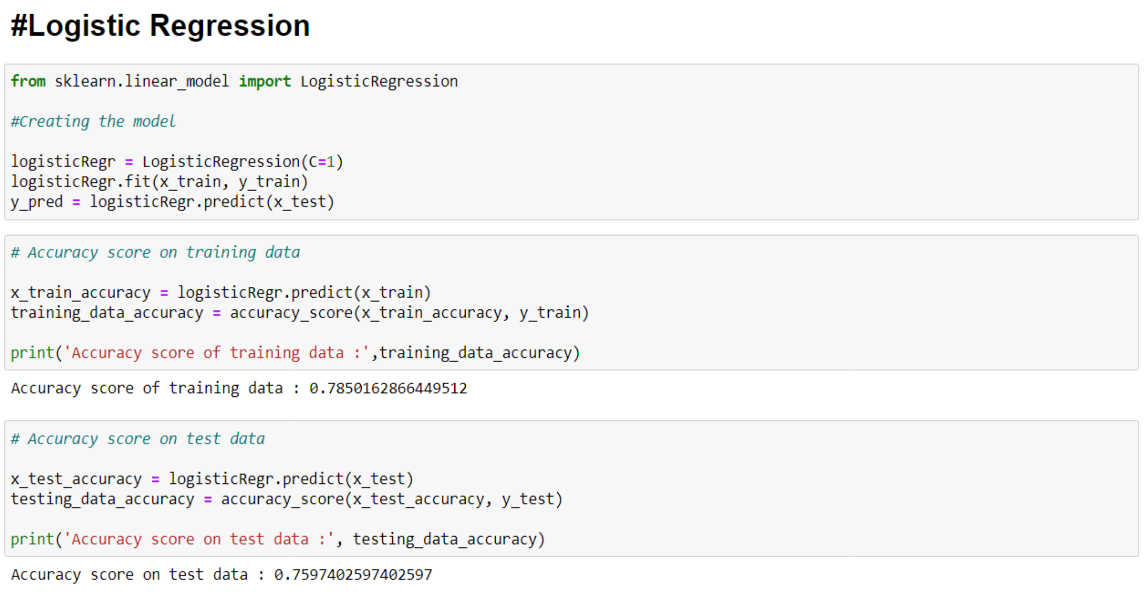
```
Accuracy score of training data : 0.755700325732899
```

```
# Accuracy score on test data

x_test_accuracy = model.predict(x_test)
testing_data_accuracy = accuracy_score(x_test_accuracy, y_test)

print('Accuracy score on test data :', testing_data_accuracy)
```

```
Accuracy score on test data : 0.7727272727272727
```

Fig. 3.13: Code to train the model using Naïve Bayes with their accuracies

e. Logistic Regression



#Logistic Regression

```
from sklearn.linear_model import LogisticRegression

#Creating the model

logisticRegr = LogisticRegression(C=1)
logisticRegr.fit(x_train, y_train)
y_pred = logisticRegr.predict(x_test)
```

```
# Accuracy score on training data

x_train_accuracy = logisticRegr.predict(x_train)
training_data_accuracy = accuracy_score(x_train_accuracy, y_train)

print('Accuracy score of training data :',training_data_accuracy)
```

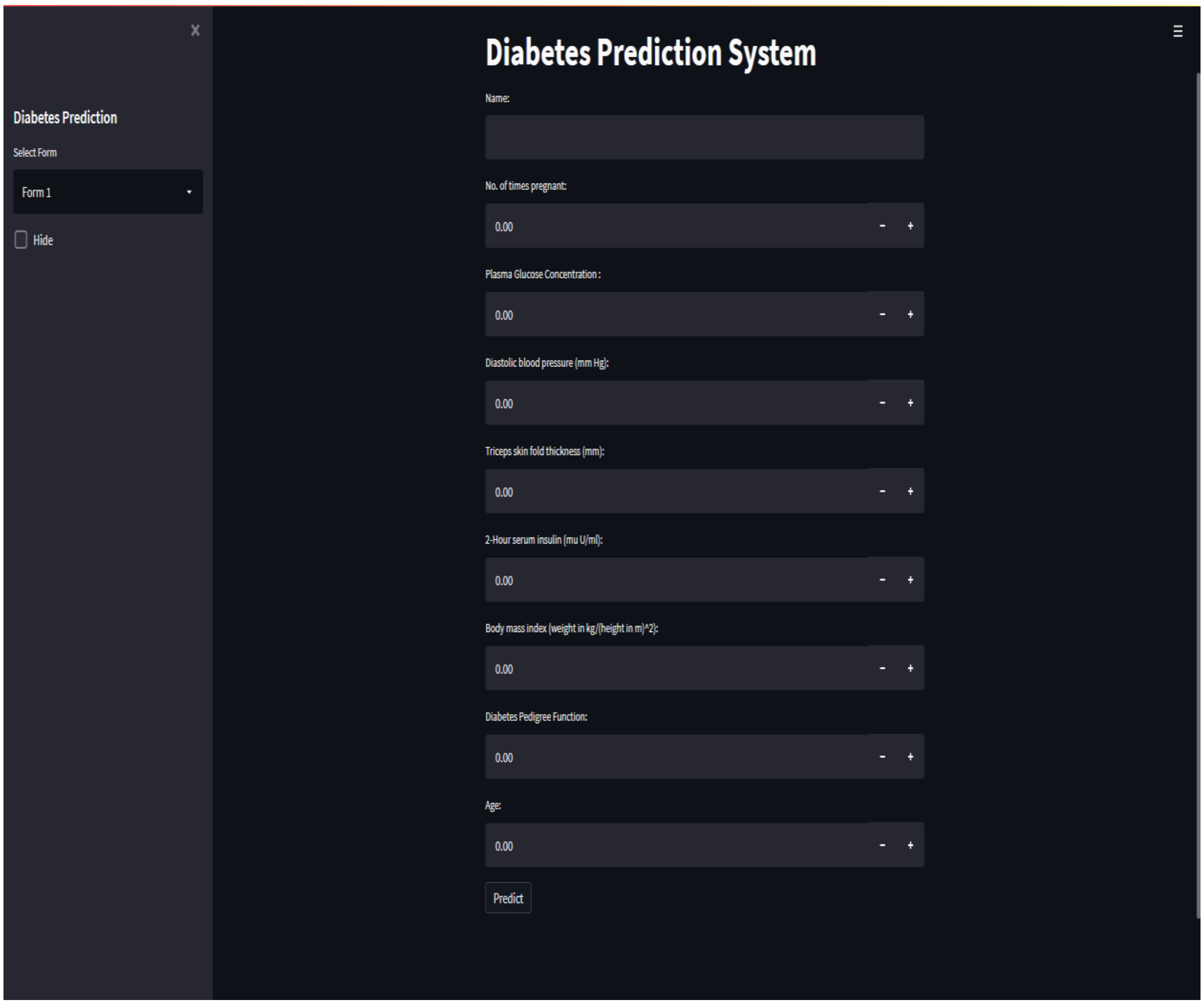
```
Accuracy score of training data : 0.7850162866449512
```

```
# Accuracy score on test data

x_test_accuracy = logisticRegr.predict(x_test)
testing_data_accuracy = accuracy_score(x_test_accuracy, y_test)

print('Accuracy score on test data :', testing_data_accuracy)
```

```
Accuracy score on test data : 0.7597402597402597
```

Fig. 3.14: Code to train the model using Logistic Regression with their accuracies

4. Prediction:



Diabetes Prediction

Select Form

Form 1

Hide

Diabetes Prediction System

Name:

No. of times pregnant: 0.00

Plasma Glucose Concentration : 0.00

Diastolic blood pressure (mm Hg): 0.00

Triceps skin fold thickness (mm): 0.00

2-Hour serum insulin (mu U/ml): 0.00

Body mass index (weight in kg/(height in m)^2): 0.00

Diabetes Pedigree Function: 0.00

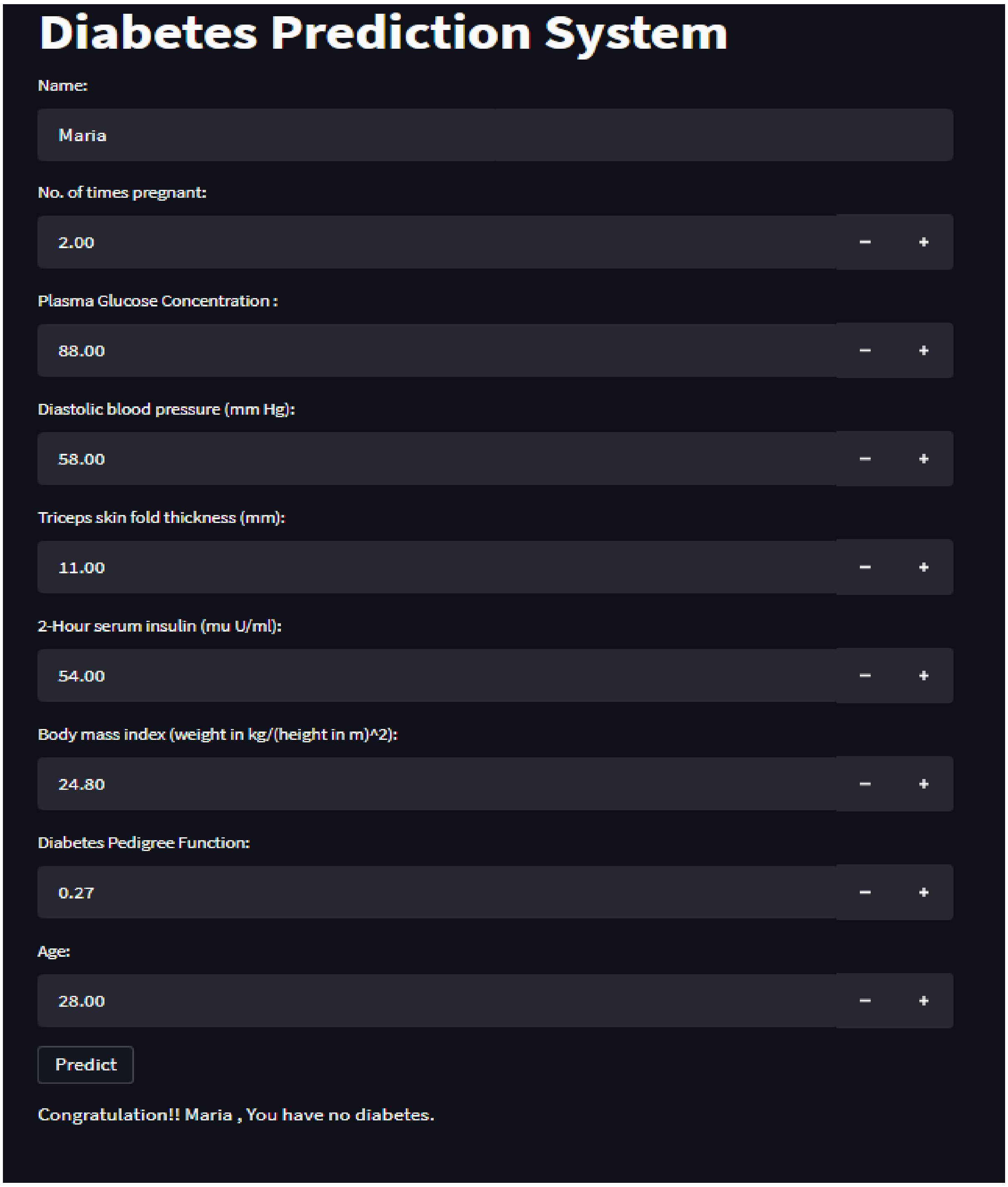
Age: 0.00

Predict

Fig. 4.1: Screenshot of diabetes prediction system

Results:

The below figure shows how the trained model is predicting the diabetes status of a person. The model is trained by using various machine learning algorithms and has the high accuracy.



Diabetes Prediction System

Name:

Maria

No. of times pregnant:

2.00

Plasma Glucose Concentration :

88.00

Diastolic blood pressure (mm Hg):

58.00

Triceps skin fold thickness (mm):

11.00

2-Hour serum insulin (mu U/ml):

54.00

Body mass index (weight in kg/(height in m)^2):

24.80

Diabetes Pedigree Function:

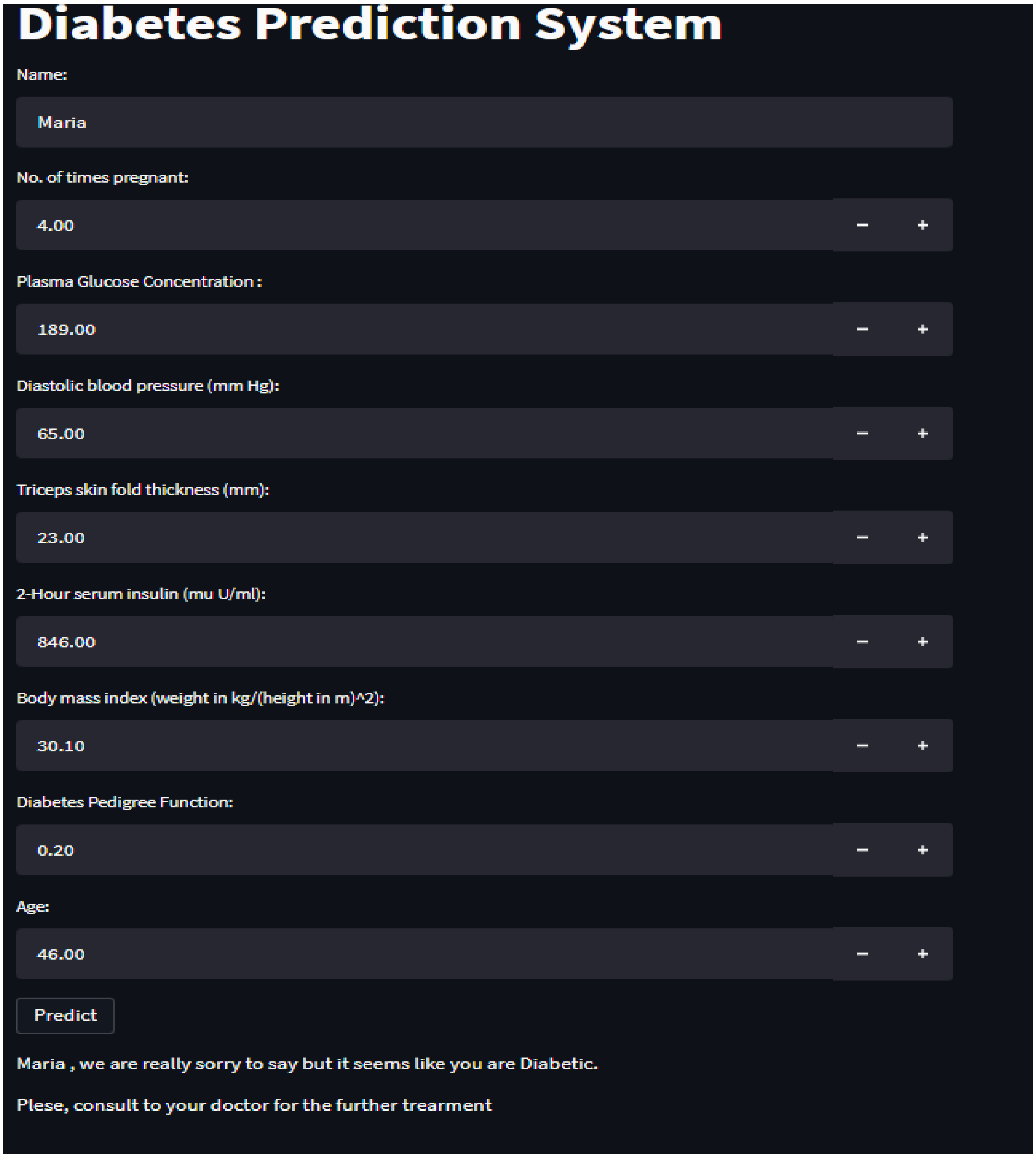
0.27

Age:

28.00

Congratulation!! Maria , You have no diabetes.

Fig. 4.2: The prediction of the person who has no diabetes



Diabetes Prediction System

Name:

Maria

No. of times pregnant:

4.00

Plasma Glucose Concentration :

189.00

Diastolic blood pressure (mm Hg):

65.00

Triceps skin fold thickness (mm):

23.00

2-Hour serum insulin (mu U/ml):

846.00

Body mass index (weight in kg/(height in m)^2):

30.10

Diabetes Pedigree Function:

0.20

Age:

46.00

Maria , we are really sorry to say but it seems like you are Diabetic.

Plese, consult to your doctor for the further trearment

Fig. 4.3: The prediction of the person who has diabetes and recommend to visit the doctor

5. Conclusion:

We trained the dataset with a wide array of machine learning models for classification and regression for the diabetes prediction. After applying various machine learning algorithms we achieved the satisfied accuracies for diabetes prediction. We saw that for many of the algorithms, setting the right parameters is important for good performance. We should be able to know how to apply, tune, and analyse the models to predict the diabetes status. Further this work can be extended to find how likely non-diabetes people have danger of having diabetes.