AI DIABETES PREDICTION SYSTEM

Team Members:

1. M. Keerthivasan
2. P. Thivahar
3. A.K. Praveenkumar
4. S. Muralidharan
5. A. Pachaiyappan

AI Diabetes Prediction System

Problem Definition:

The problem is to create an AI system that can accurately predict the risk of diabetes in individuals, allowing for early intervention and personalized health recommendations. This system should leverage relevant data sources, such as medical records, lifestyle information, genetics, and more, to provide actionable insights to individuals and healthcare providers.

Design Thinking Approach:

1. Empathize:

- ❖ Understand the challenges and concerns faced by individuals at risk of diabetes.
- ❖ Gather insights through interviews, surveys, and research about their health monitoring habits and needs.
- ❖ Engage with healthcare professionals to understand their perspectives on diabetes risk prediction.

2. Define:

- ❖ Clearly define the problem statement: "How might we develop an AI system that predicts diabetes risk accurately and provides actionable recommendations for individuals and healthcare providers?"
- ❖ Identify key stakeholders, including individuals, healthcare providers, data scientists, and regulatory authorities.

3. Ideate:

- ❖ Brainstorm potential solutions and AI models that can predict diabetes risk effectively.

- ❖ Explore various data sources, including electronic health records, wearable devices, dietary data, and genetic information.
- ❖ Consider how the predictions can be communicated to individuals in a way that is easily understood and actionable.

4. Prototype:

- ❖ Develop a prototype AI model using a diverse dataset that incorporates different types of health-related data.
- ❖ Create a user-friendly interface or mobile app to deliver predictions and recommendations to users.
- ❖ Conduct usability testing to refine the interface and improve user experience.

5. Test:

- ❖ Evaluate the accuracy and reliability of the AI model using a large and diverse dataset.
- ❖ Solicit feedback from individuals and healthcare providers on the usefulness and effectiveness of the system.
- ❖ Assess the system's compliance with data privacy and security regulations.

6. Implement:

- ❖ Deploy the AI diabetes prediction system in healthcare settings, such as clinics and hospitals.
- ❖ Train healthcare professionals on how to interpret and use the system's predictions.
- ❖ Ensure that the system complies with relevant healthcare standards and regulations.

7. Iterate:

- ❖ Continuously update and improve the AI model based on new data and research findings.
- ❖ Collect feedback from users and healthcare providers to enhance the system's features and usability.
- ❖ Stay informed about emerging technologies and best practices in diabetes risk prediction.

8. Scale:

- ❖ Expand the deployment of the AI system to reach a broader audience.
- ❖ Consider partnerships with insurance companies or public health organizations to promote diabetes prevention and management.
- ❖ Monitor the long-term impact of the system on diabetes prevention and healthcare outcomes.

Throughout this design thinking process, collaboration among healthcare experts, data scientists, user experience designers, and individuals at risk of diabetes is essential to ensure that the AI diabetes prediction system is accurate, user-friendly, and capable of making a meaningful impact on public health.

Algorithm for AI Diabetes Prediction:

To create a diabetes prediction AI project, you can use various machine learning algorithms and packages. Here's a basic outline of how to approach it:

1. Data Collection:

- ❖ Gather a dataset with relevant features such as age, BMI, blood pressure, glucose levels, etc., along with the corresponding diabetes outcomes (binary classification: diabetic or not).

2. Data Preprocessing:

- Preprocess the data by handling missing values, normalizing/standardizing features, and encoding categorical variables if necessary.

3. Algorithm Selection:

- You can use a variety of machine learning algorithms for binary classification, including:
- Logistic Regression
- Decision Trees
- Random Forest
- Support Vector Machines (SVM)
- Neural Networks (e.g., using TensorFlow or PyTorch)
- Gradient Boosting (e.g., XGBoost, LightGBM)
- k-Nearest Neighbors (k-NN)

4. Model Training:

- Split your dataset into training and testing sets to evaluate your model's performance.
- Train the selected algorithm(s) on the training data.

5. Model Evaluation:

- Use appropriate metrics (e.g., accuracy, precision, recall, F1-score, ROC AUC) to evaluate the model's performance on the test data.

6. Hyperparameter Tuning:

- Fine-tune hyperparameters of your chosen algorithm(s) to optimize performance.

7. Package Selection:

- Depending on the algorithm(s) you choose, you'll need packages like scikit-learn for traditional machine learning algorithms, TensorFlow or PyTorch for neural networks, and specific libraries for boosting algorithms or SVM.

8. Deployment:

- Once your model performs well, you can deploy it as a web application, mobile app, or integrate it into a healthcare system.

Remember that the choice of algorithm may depend on the specifics of your dataset and project goals. It's a good practice to experiment with multiple algorithms to find the one that works best for your diabetes prediction task.

AI Diabetes Prediction System

Introduction:

Our team at IBM, inspired by the vision of "Naan Mudhalvan," has transformed the design of our AI Diabetes prediction system into an innovative solution to tackle this pressing problem. Recognizing the urgent need for accurate and accessible predictive tools in healthcare, we embarked on this mission to make a difference.

Methodology:

In the core of our innovation lies the Logistic Regression Model, a robust analytical tool that serves as the foundation of our prediction system. This advanced statistical technique enables us to meticulously analyze complex datasets, identifying patterns and correlations that might otherwise go unnoticed. By utilizing this model, we not only predict the likelihood of diabetes but also empower individuals and healthcare professionals with valuable insights for early intervention.

By leveraging cutting-edge technology and the power of data analytics, our project aims to revolutionize diabetes prediction, paving the way for a healthier future.

Sigmoid Function:

The core of logistic regression is the sigmoid function (also called the logistic function). The sigmoid function maps any realvalued number to the range [0, 1]. It has an S-shaped curve and is defined as follows:

[\sigma(z) = \frac{1}{1 + e^{-z}} \]

Where (z \) is a linear combination of the input features and their respective weights, plus a bias term. In mathematical terms, for a binary classification task with \(n \) features:

[z = b + \sum_{i=1}^{n} (w_i \times x_i) \]

Here, (b \) represents the bias term, \(w_i \) represents the weights associated with each feature \(x_i \).

Probability Prediction:

The sigmoid function takes the linear combination of features and weights as input and squashes it between 0 and 1. This output represents the probability of the instance belonging to the positive class (class 1). For example, if the output is 0.7, it means there is a 70% chance that the instance belongs to the positive class.

Decision Boundary:

To make a binary decision (class 0 or class 1), a threshold is set (usually 0.5). If the predicted probability is greater than or equal to

0.5, the instance is classified as class 1; otherwise, it is classified as class 0. The decision boundary is the line (or hyperplane in higher dimensions) where the sigmoid function equals the threshold value.

Training the Model:

During the training process, the model adjusts the weights and the bias term to minimize the difference between the predicted probabilities and the actual class labels in the training data. This is typically done using optimization techniques such as gradient descent, which iteratively updates the weights to minimize the loss function.

Loss Function:

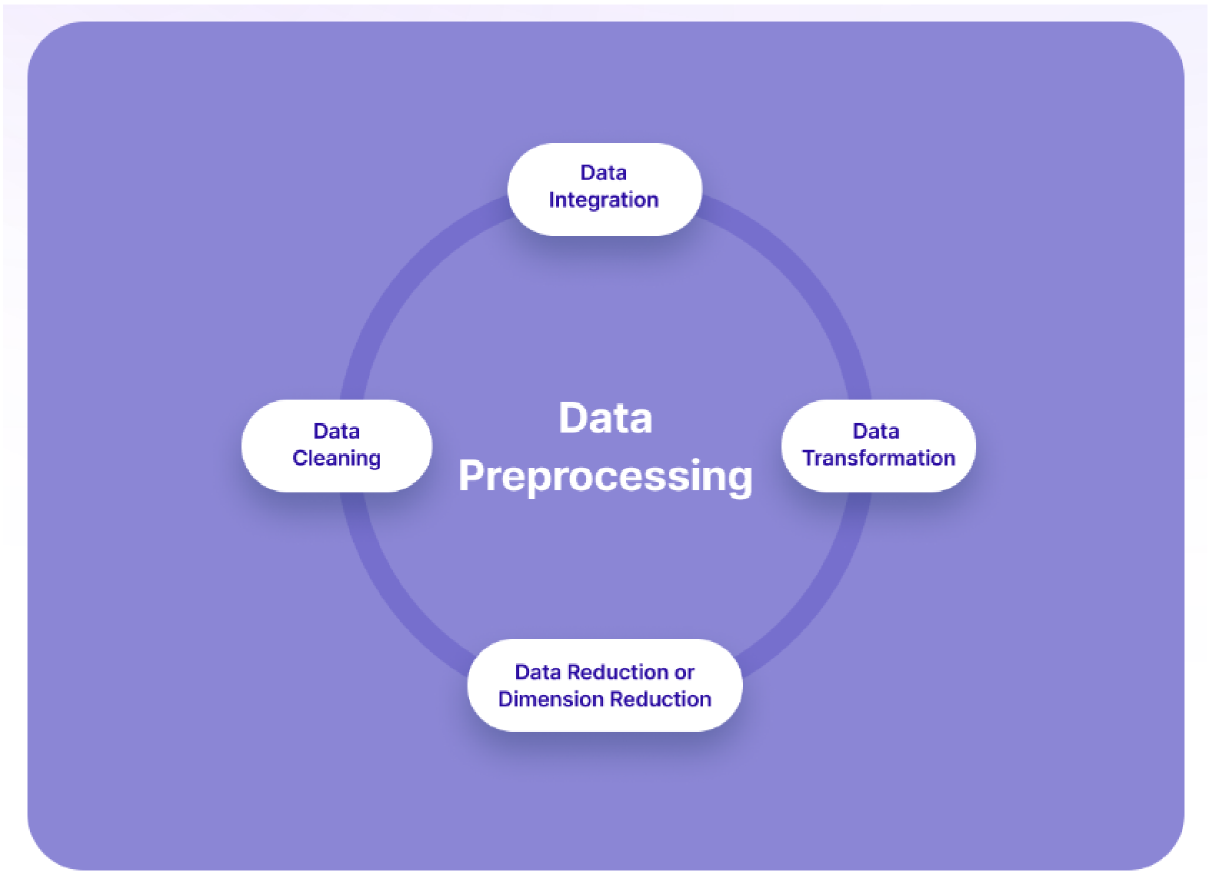
The loss function used in logistic regression is the log loss (or cross-entropy loss). It measures the dissimilarity between the predicted probabilities and the actual labels. Minimizing this loss function during training ensures that the model's predicted probabilities align well with the true labels.

Evaluation Metrics:

Common evaluation metrics for logistic regression models include accuracy, precision, recall, F1-score, and the area under the ROC curve (AUC-ROC). These metrics help assess the model's performance and its ability to correctly classify instances into their respective classes.

<u>Dataset preprocessing:</u>

In the merged dataset, we discovered a few exceptional zero values.



For example, skin thickness and Body Mass Index (BMI) cannot be zero. The zero value has been replaced by its corresponding mean value. The training and test dataset has been separated using the holdout validation technique, where 80% is the training data and 20% is the test data.

Need of Data Preprocessing:

For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.

Another aspect is that the data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithm are executed in one data set, and best out of them is chosen.

Data Preprocessing Steps:

There are four major steps in Data Preprocessing they are:

1. Data quality assessment
2. Data Cleaning
3. Data Transformation
4. Data Reduction

1. Data quality assessment:

There are a number of data anomalies and inherent problems to look out for in almost any data set, for example:

- Mismatched data types
- Mixed data values
- Data outliers
- Missing data

2. Data Cleaning:

Data cleaning is the process of adding missing data and correcting, repairing, or removing incorrect or irrelevant data from a data set. Dating cleaning is the most important step of preprocessing because it will ensure that your data is ready to go for your downstream needs.

3. Data Transformation:

With data cleaning, we’ve already begun to modify our data, but data transformation will begin the process of turning the data into the proper formats.

4. Data Reduction:

The more data you’re working with, the harder it will be to analyze, even after cleaning and transforming it. Depending on your task at hand, you may actually have more data than you need. Data reduction not only makes the analysis easier and more accurate, but cuts down on data storage.

Importing Libraries:

Code:

```
import numpy as np
```

AI Diabetes Prediction System

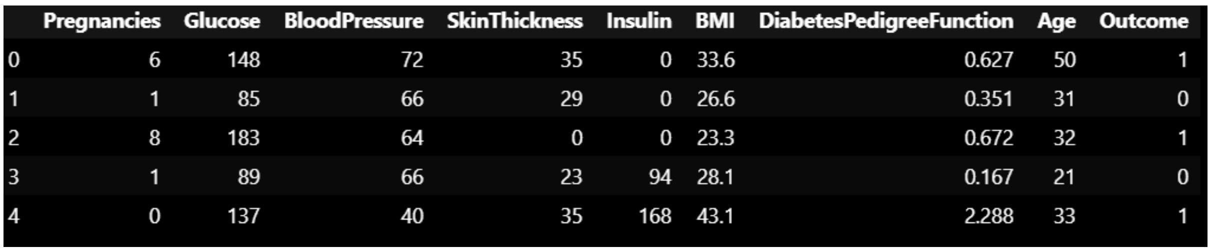
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

from mlxtend.plotting import plot_decision_regions
import missingno as msno
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Reading the dataset:

Code:

```
diabetes_df = pd.read_csv('diabetes.csv')
diabetes_df.head()
```

Output:



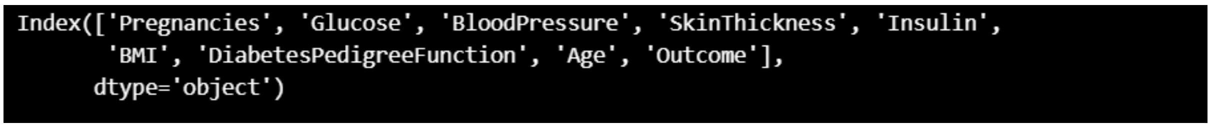
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Columns:

Code:

```
diabetes_df.columns
```

Output:



```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

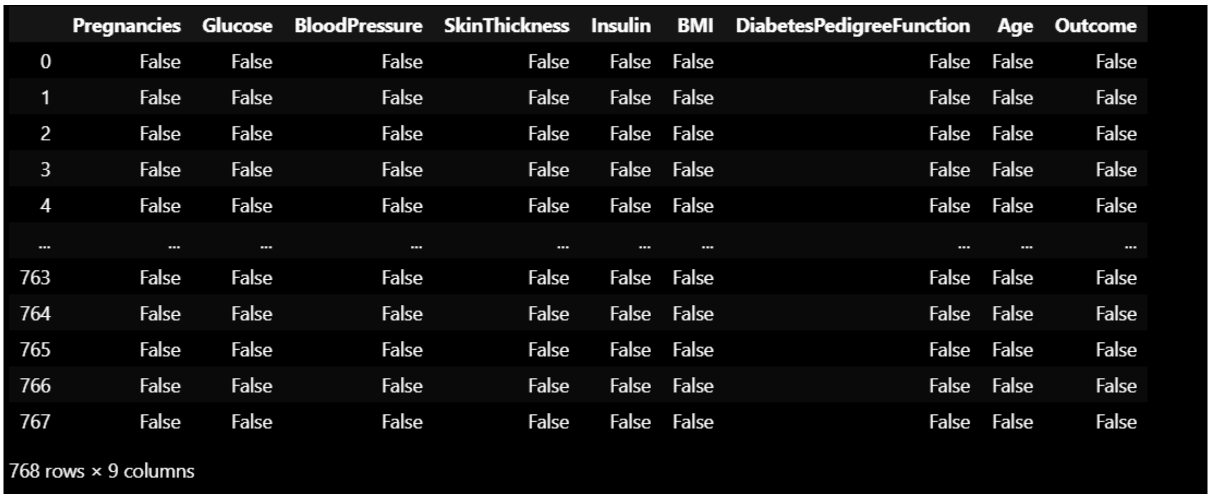
Data Cleaning:

Check the dataset have null values or not:

Code:

```
diabetes_df.isnull()
```

Output:



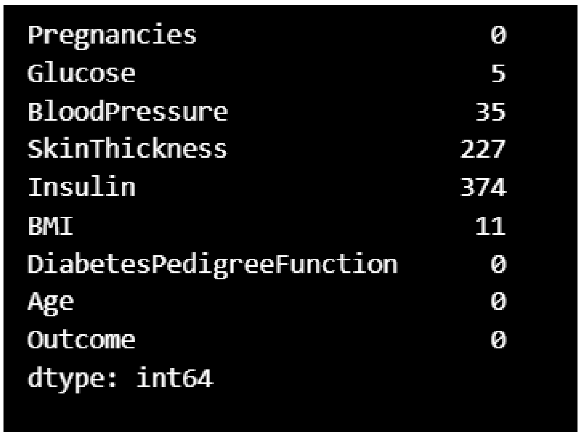
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI  DiabetesPedigreeFunction    Age  Outcome
  0        False    False          False          False    False  False                     False  False    False
  1        False    False          False          False    False  False                     False  False    False
  2        False    False          False          False    False  False                     False  False    False
  3        False    False          False          False    False  False                     False  False    False
  4        False    False          False          False    False  False                     False  False    False
...          ...      ...            ...            ...      ...    ...                       ...    ...      ...
763        False    False          False          False    False  False                     False  False    False
764        False    False          False          False    False  False                     False  False    False
765        False    False          False          False    False  False                     False  False    False
766        False    False          False          False    False  False                     False  False    False
767        False    False          False          False    False  False                     False  False    False

768 rows × 9 columns
```

Replace the 0 value with the NAN:

Code:

```
diabetes_df_copy = diabetes_df.copy(deep = True)
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin
','BMI']] =
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin
','BMI']].replace(0,np.NaN)

print(diabetes_df_copy.isnull().sum())
```

Output:



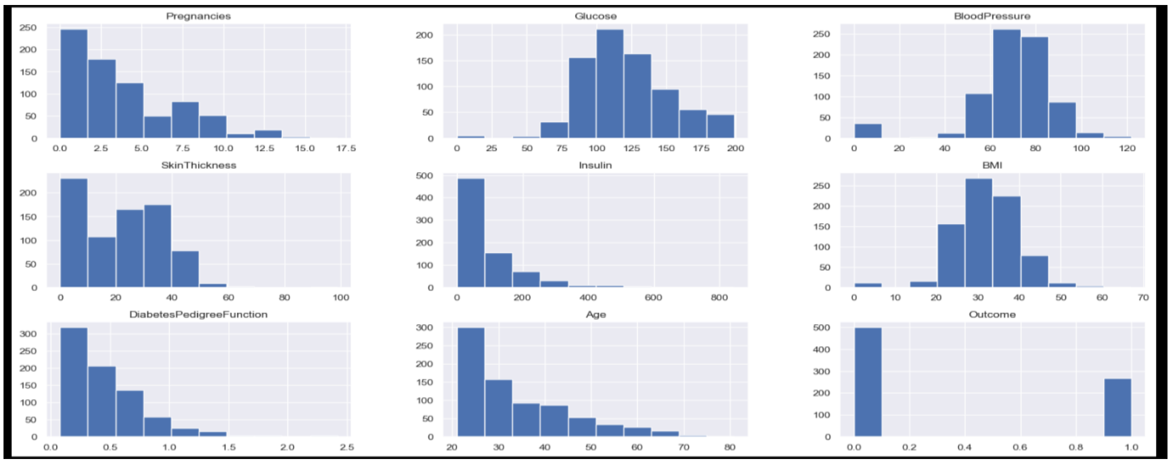
```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64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

Data Visualization:

Code:

```
p = diabetes_df.hist(figsize = (10,20))
```

Output:



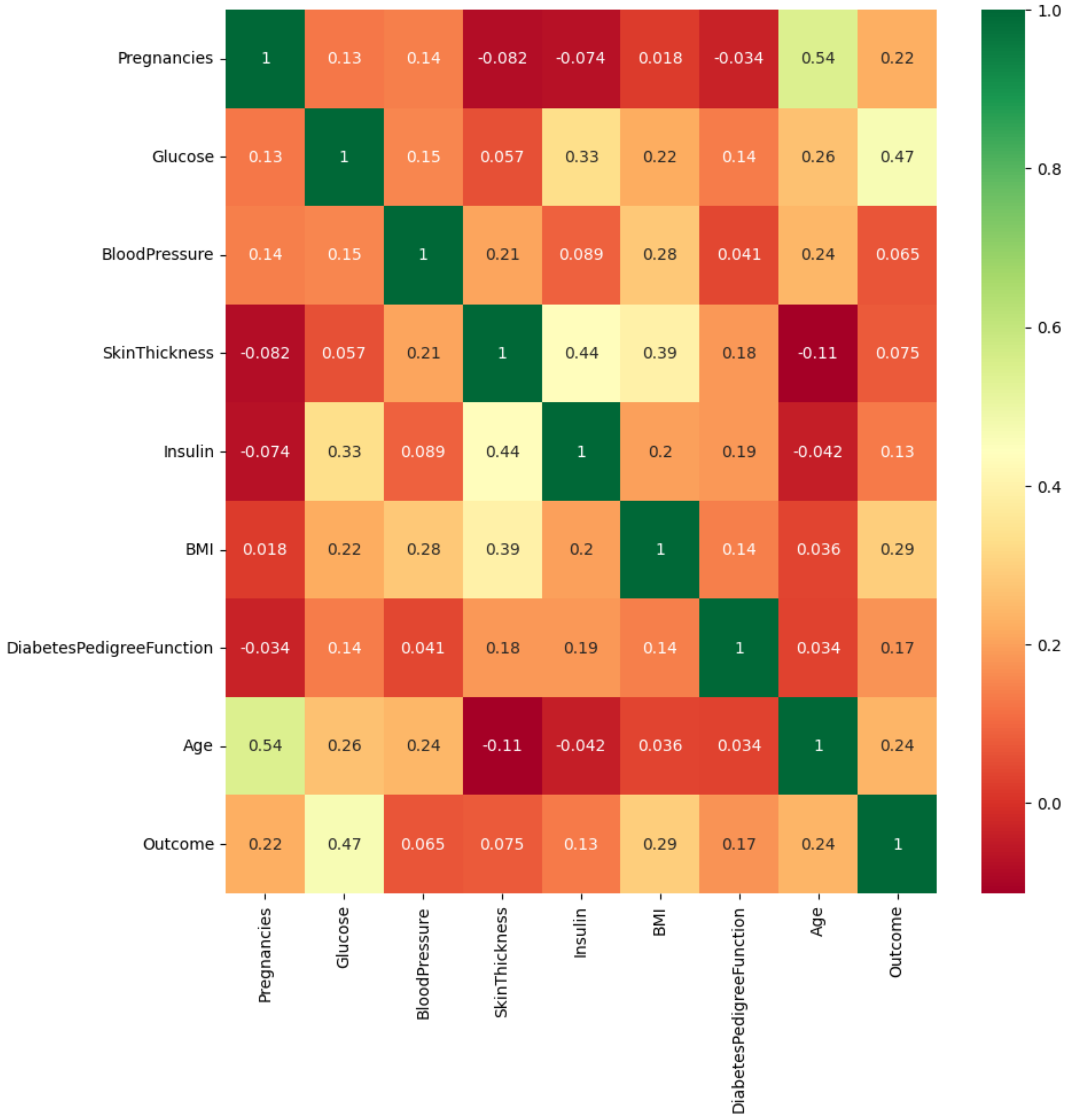
Correlation between all the features:

Correlation between all the features before cleaning:

Code:

```
plt.figure(figsize=(10,10))
p = sns.heatmap(diabetes_df.corr(), annot=True,cmap ='RdYlGn')
```

Output:



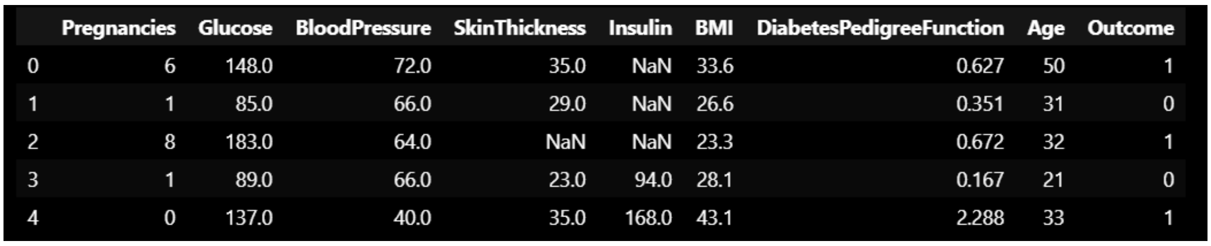
Scaling the Data:

Before scaling down the data:

Code:

```
diabetes_df_copy.head()
```

Output:



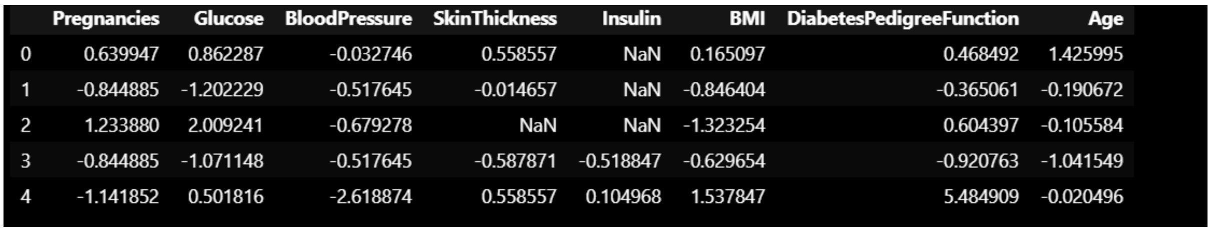
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50	1
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31	0
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

After Standard scaling down the data:

Code:

```
sc_X = StandardScaler()
X =
pd.DataFrame(sc_X.fit_transform(diabetes_df_copy.drop(["Outcome"],ax
is = 1),), columns=['Pregnancies',
'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age'])
X.head()
```

Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.862287	-0.032746	0.558557	NaN	0.165097	0.468492	1.425995
1	-0.844885	-1.202229	-0.517645	-0.014657	NaN	-0.846404	-0.365061	-0.190672
2	1.233880	2.009241	-0.679278	NaN	NaN	-1.323254	0.604397	-0.105584
3	-0.844885	-1.071148	-0.517645	-0.587871	-0.518847	-0.629654	-0.920763	-1.041549
4	-1.141852	0.501816	-2.618874	0.558557	0.104968	1.537847	5.484909	-0.020496

AI Diabetes Prediction System

In our "IBM Naan Mudhalvan" project, titled "AI Diabetes Prediction System," we utilized the Logistic Regression Algorithm. This document includes the Logistic Regression code we employed, as well as the coding for training the model.

Algorithm (Logistic Regression):

```
# === libraries ===
import numpy as np
# === sigmoid ===
# description: the S shape function which gives us one or zero.
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
# === Logistic Regression ===
class LogisticRegression():
    def __init__(self, lr=0.001, n_iters=1000):
        self.lr = lr
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    # X ===> Training inputs samples
    # y ===> Target values
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        # this is gradient descent
        for _ in range(self.n_iters):
            linear_pred = np.dot(X, self.weights) + self.bias
            predictions = sigmoid(linear_pred)

            # formula: dw = (1 / n_samples) * X.T * (predictions - y)



            # formula: db = (1 / n_samples) * sum(predictions - y)
            dw = (1 / n_samples) * np.dot(X.T, (predictions - y))
            db = (1 / n_samples) * np.sum(predictions - y)

            self.weights = self.weights - self.lr * dw
            self.bias = self.bias - self.lr * db

    # labeling the data
    def predict(self, X):
        linear_pred = np.dot(X, self.weights) + self.bias
        y_pred = sigmoid(linear_pred)
        class_pred = [0 if y <= 0.5 else 1 for y in y_pred]
        return class_pred
```

Training the Model:

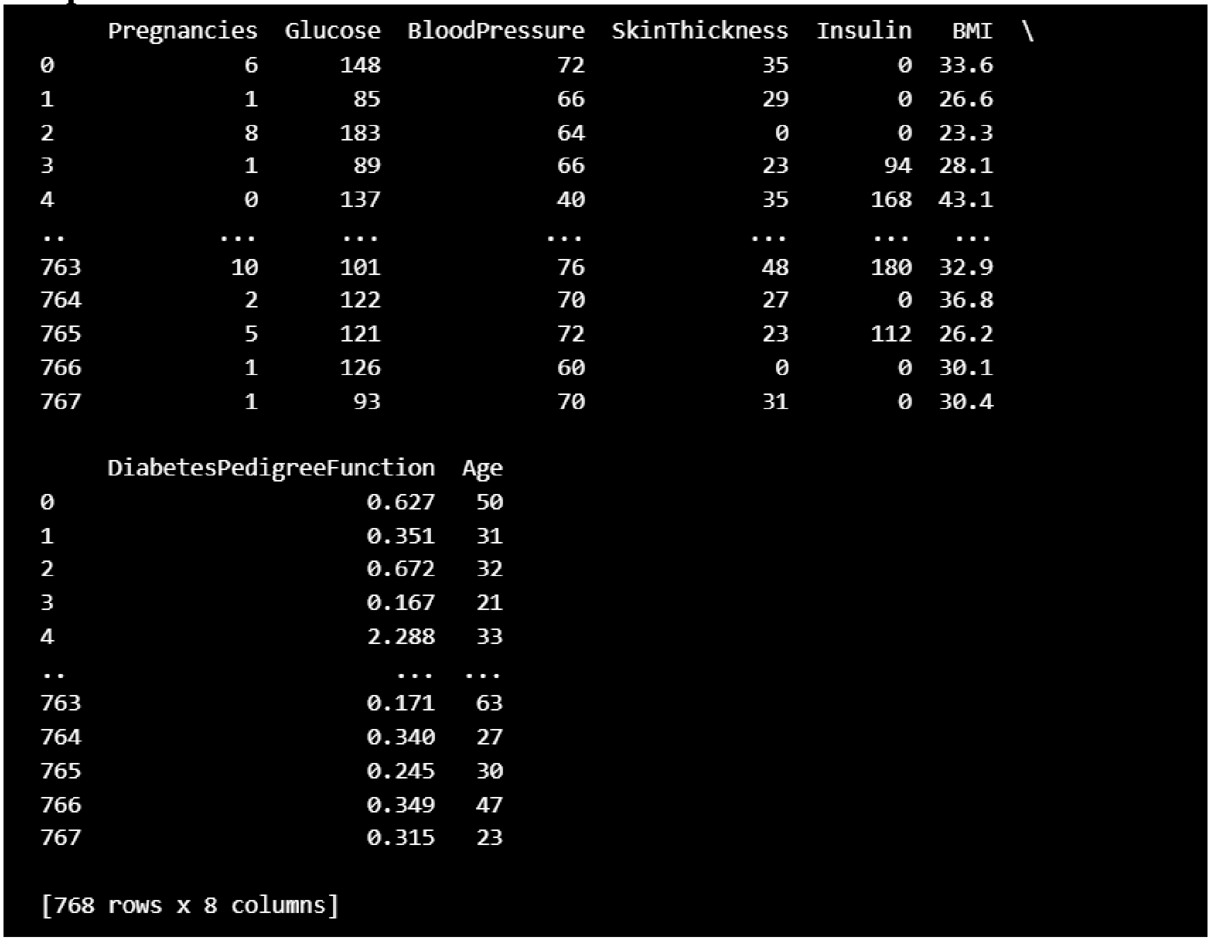
Code:

```
import numpy as np
from sklearn.model_selection import train_test_split
from logisticRegression import LogisticRegression
import pandas as pd
```

Code:

```
learn_d = pd.read_csv("diabetes.csv")
X = learn_d.drop('Outcome', axis=1)  # Assuming 'Outcome' is the diabetes label
y = learn_d['Outcome']
print(X)
```

Output:

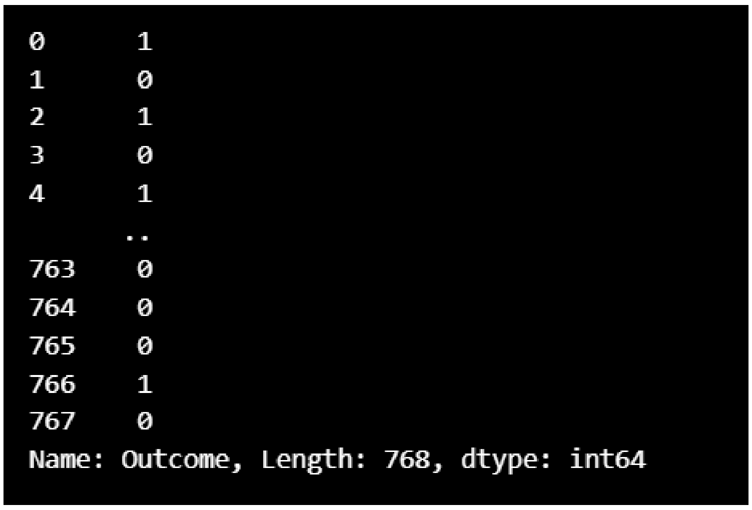


```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              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]
```

Code:

```
print(y)
```

Output:



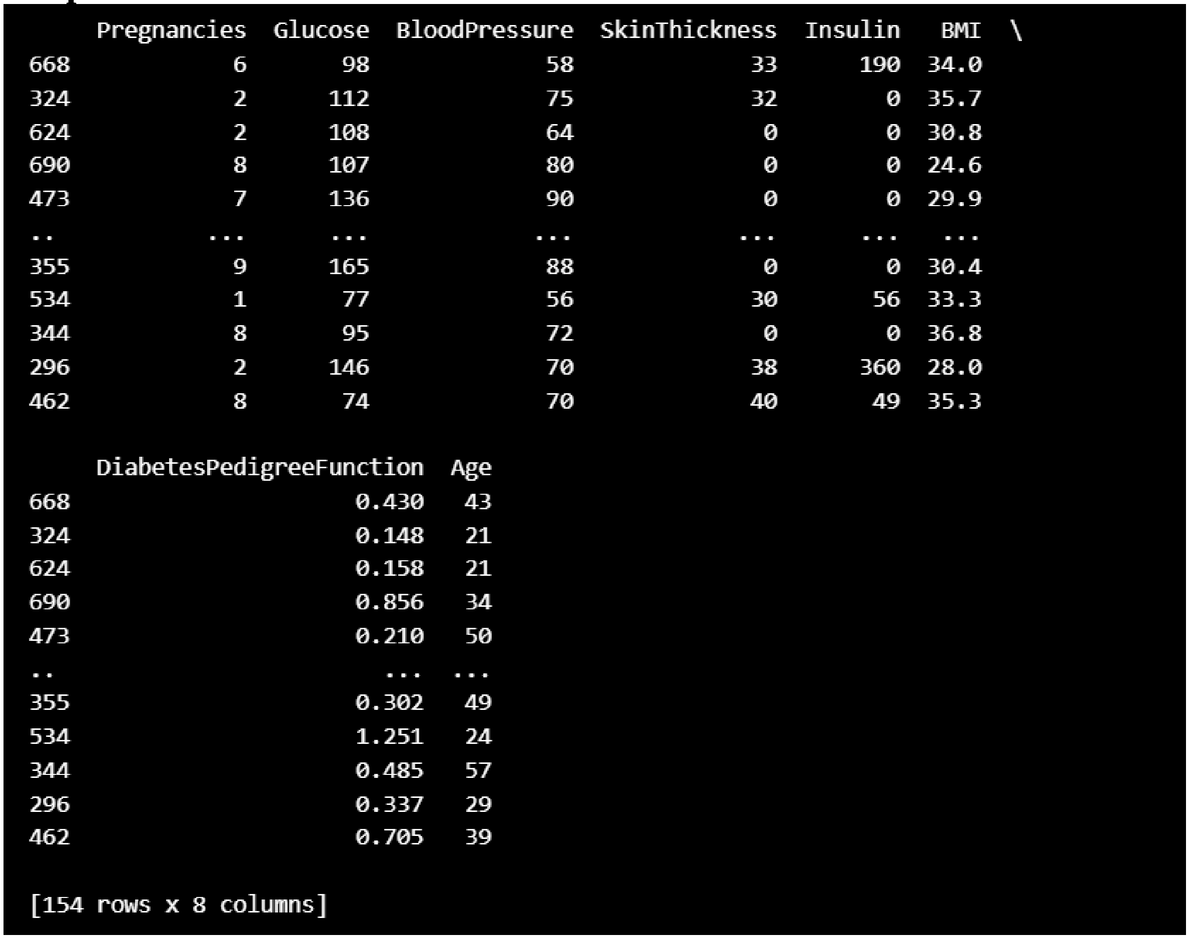
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Code:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(X_test)

Output:

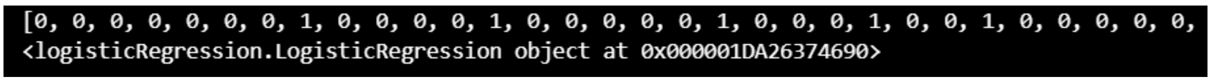


```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
668            6       98             58             33      190  34.0
324            2      112             75             32        0  35.7
624            2      108             64              0        0  30.8
690            8      107             80              0        0  24.6
473            7      136             90              0        0  29.9
..           ...      ...            ...            ...      ...   ...
355            9      165             88              0        0  30.4
534            1       77             56             30       56  33.3
344            8       95             72              0        0  36.8
296            2      146             70             38      360  28.0
462            8       74             70             40       49  35.3

     DiabetesPedigreeFunction  Age
668                     0.430   43
324                     0.148   21
624                     0.158   21
690                     0.856   34
473                     0.210   50
..                        ...  ...
355                     0.302   49
534                     1.251   24
344                     0.485   57
296                     0.337   29
462                     0.705   39

[154 rows x 8 columns]
```

Code:

```
LR = LogisticRegression(lr = 0.0001, n_iters= 100000)
LR.fit(X_train, y_train)

Y_pred = LR.predict(X_test)
print(Y_pred)
```

Output:

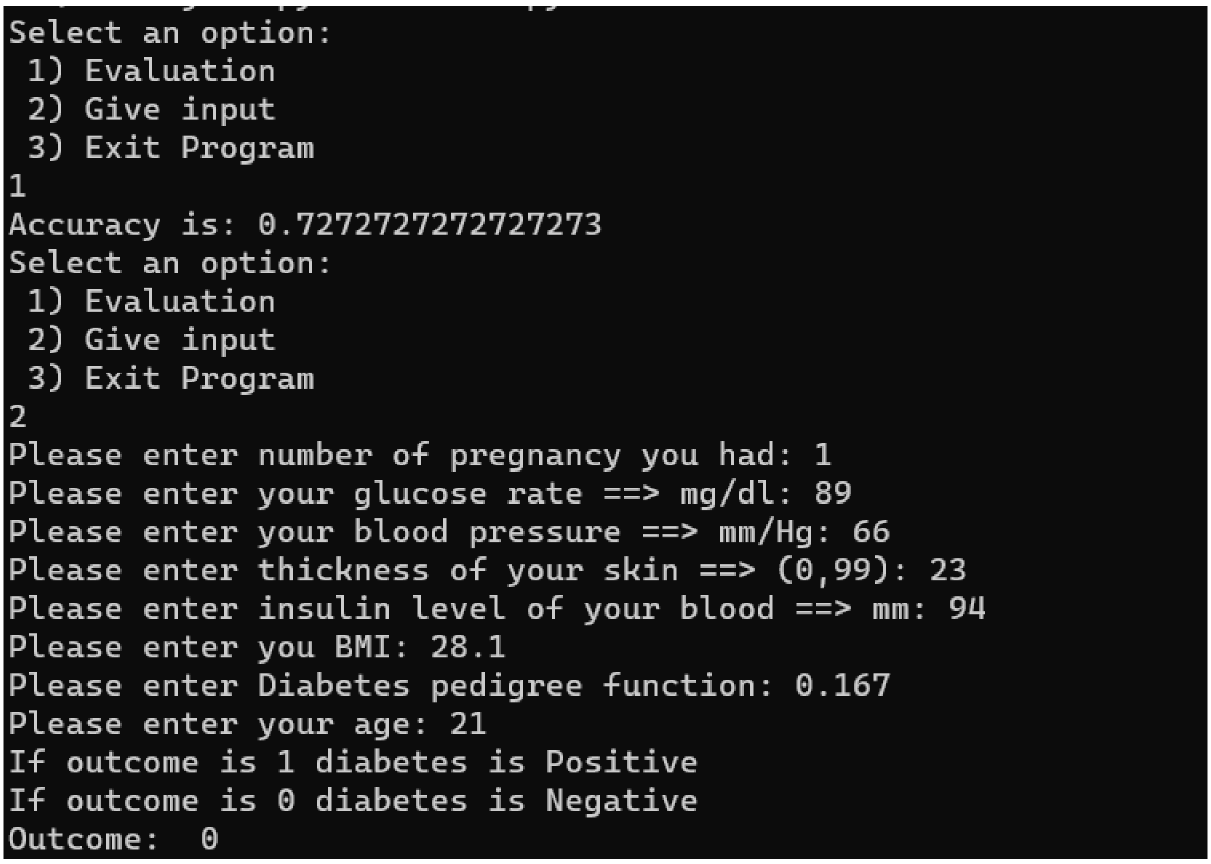


```
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
<logisticRegression.LogisticRegression object at 0x000001DA26374690>
```

Code:

```
def accuracy(y_pred, y_test):
    return np.sum(y_pred == y_test)/len(y_test)

# result = accuracy(Y_prediction, Y_test)
# print(result)

# === main ===
def new_func():
    option = int(input("select the option:"))
    return option
while(True):
    print("Select an option: \n 1) Evaluation\n 2) Give input\n 3) Exit Program")
    option = new_func()
    if(option == 1):
        acc = accuracy(y_test, Y_pred)
        print("Accuracy is:",acc)

    elif(option == 2):
        pregnancies = float(input("Please enter number of pregnancy you had: "))
        glucose = float(input("Please enter your glucose rate ==> mg/dl: "))
        bloodPressure = float(input("Please enter your blood pressure ==> mm/Hg: 
"))
        skinThickness = float(input("Please enter thickness of your skin ==> 
(0,99): "))
        insulin = float(input("Please enter insulin level of your blood ==> mm: "))
        bmi = float(input("Please enter you BMI: "))
        diabetesPedigreeFunction = float(input("Please enter Diabetes pedigree 
function: "))
        age = float(input("Please enter your age: "))

        x_input = [[pregnancies, glucose, bloodPressure, skinThickness, insulin, 
bmi, diabetesPedigreeFunction, age]]
        prob = LR.predict(x_input)
        print("Outcome: ", prob[0])
elif(option == 3):
        print("exit")
        break
```

Output:



```
Select an option:
 1) Evaluation
 2) Give input
 3) Exit Program
1
Accuracy is: 0.7272727272727273
Select an option:
 1) Evaluation
 2) Give input
 3) Exit Program
2
Please enter number of pregnancy you had: 1
Please enter your glucose rate ==> mg/dl: 89
Please enter your blood pressure ==> mm/Hg: 66
Please enter thickness of your skin ==> (0,99): 23
Please enter insulin level of your blood ==> mm: 94
Please enter you BMI: 28.1
Please enter Diabetes pedigree function: 0.167
Please enter your age: 21
If outcome is 1 diabetes is Positive
If outcome is 0 diabetes is Negative
Outcome:  0
```