**PHASE 05**

**PROBLEM STATEMENT:**

AI-BASED DIABETES PREDICTION SYSTEM

**PROBLEM DEFINITION:**

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health. AI-based diabetes prediction system is to accurately predict diabetic illness from patient data using machine learning algorithms. The system should be able to identify diabetes risk factors and evaluate different machine learning algorithms for diabetes.

**INTRODUCTION:**

We use logistic regression to predict whether a person has diabetic or not based on various features such as pregnancies, glucose level, blood pressure, skin thickness, insulin level, BMI, diabetes pedigree function, and age. We will load a dataset containing these features and the corresponding outcome (diabetic or non-diabetic) and train a logistic regression model to make predictions.

**DESIGN THINKING:**

Data Collection: We need a dataset containing medical features such as glucose levels, blood pressure, BMI, etc., along with information about whether the individual has diabetes or not.

Data Preprocessing: The medical data needs to be cleaned, normalized, and prepared for training machine learning models.

Feature Selection: We will select relevant features that can impact diabetes risk prediction.

Model Selection: We can experiment with various machine learning algorithms like Logistic Regression, Random Forest, and Gradient Boosting.

Evaluation: We will evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.

Iterative Improvement: We will fine-tune the model parameters and explore techniques like feature engineering to enhance prediction accuracy.

**DATASET LINK:** <https://www.kaggle.com/datasets/mathchi/diabetes-data-set>

**CODE EXPLANATION:**

**Data Loading and Exploration:** The code starts by importing the necessary libraries, including pandas, numpy, matplotlib, and seaborn. It then loads the diabetes dataset from a CSV file using the pd.read\_csv() function and assigns it to the variable dataset. The head() function is used to display the first few rows of the dataset, giving us a glimpse of the data. The shape attribute is used to display the dimensions of the dataset (number of rows and columns). The info() function provides information about the dataset, including the data types of each column. The describe() function calculates the descriptive statistics of the dataset, such as count, mean, standard deviation, minimum, and maximum values, for each numerical column. The isnull().sum() function is used to check for missing values in the dataset, returning the sum of missing values for each column.

**Data Visualization:** The code uses various visualization techniques to gain insights into the dataset. The countplot() function from the seaborn library is used to visualize the distribution of the target variable (Outcome column), which represents whether a person has diabetes or not. The pairplot() function is used to visualize the pairwise relationships between variables, with different colors representing the different outcomes. The heatmap() function from seaborn is used to visualize the correlation matrix of the dataset, showing the strength and direction of the linear relationship between variables.

**Data Preprocessing:** The code creates a copy of the dataset named dataset\_new to perform data preprocessing steps. It replaces zero values in selected columns (Glucose, BloodPressure, SkinThickness, Insulin, BMI) with NaN (Not a Number) using the replace() function. The isnull().sum() function is then used to check for missing values in the updated dataset. Missing values are filled with the mean of each column using the fillna() function.

**Data Splitting:** The code separates the target variable (Outcome) from the features by assigning it to the variable y and dropping the Outcome column from the dataset to create the feature matrix X.

**Data Splitting for Training and Testing:** The code uses the train\_test\_split() function from the sklearn library to split the dataset into training and testing sets. The test\_size parameter is set to 0.20, indicating that 20% of the data will be used for testing, while the remaining 80% will be used for training. The random\_state parameter is set to 42 for reproducibility, and the stratify parameter is set to the Outcome column to ensure that the distribution of the target variable is preserved in both the training and testing sets. The resulting splits are assigned to the variables X\_train, X\_test, Y\_train, and Y\_test.

**Model Training and Evaluation:** The code trains a logistic regression model using the LogisticRegression() class from the sklearn library. The fit() function is used to fit the model to the training data, with X\_train as the feature matrix and Y\_train as the target variable. The trained model is then used to make predictions on the testing set using the predict() function, with X\_test as the input. The confusion matrix is calculated using the confusion\_matrix() function, which compares the actual values (Y\_test) with the predicted values (y\_predict). The accuracy of the model is calculated using the accuracy\_score() function, which compares the predicted values with the actual values. The confusion matrix and accuracy score are printed to the console.

**LOGISTIC REGRESSION:**

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class or not. It is a kind of statistical algorithm, which analyze the relationship between a set of independent variables and the dependent binary variables. It is a powerful tool for decision-making.

Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value.

It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.

**User Input and Prediction:** The code prompts the user to enter input values for prediction. The user is asked to enter the number of pregnancies, glucose level, blood pressure, skin thickness, insulin level, BMI, diabetes pedigree function, and age. The input values are stored in variables with appropriate data types. An array of input values is created using the np.array() function, reshaped to match the input shape expected by the model. The trained model is then used to make a prediction on the input values using the predict() function. The input values and the prediction result are displayed to the console. Finally, based on the prediction result, the code prints whether the person is predicted to be diabetic or non-diabetic.

**CODE:**

Import pandas as pd

Import numpy as np

Import matplotlib.pyplot as plt

Import seaborn as sns

# Load the dataset

Dataset = pd.read\_csv(‘diabetes.csv’)

# Display the first few rows of the dataset

Dataset.head()

# Display the shape of the dataset

Dataset.shape

# Display information about the dataset

Dataset.info()

# Display the descriptive statistics of the dataset

Dataset.describe().T

# Check for missing values in the dataset

Dataset.isnull().sum()

# Visualize the distribution of the target variable

Sns.countplot(x=’Outcome’, data=dataset)

# Visualize the pairwise relationships between variables

Sns.pairplot(data=dataset, hue=’Outcome’)

Plt.show()

# Visualize the correlation matrix

Sns.heatmap(dataset.corr(), annot=True)

Plt.show()

# Create a copy of the dataset

Dataset\_new = dataset

# Replace zero values with NaN in selected columns

Dataset\_new[[“Glucose”, “BloodPressure”, “SkinThickness”, “Insulin”, “BMI”]] = dataset\_new[[“Glucose”, “BloodPressure”, “SkinThickness”, “Insulin”, “BMI”]].replace(0, np.NaN)

# Check for missing values in the updated dataset

Dataset\_new.isnull().sum()

# Fill missing values with the mean of each column

Dataset\_new[“Glucose”].fillna(dataset\_new[“Glucose”].mean(), inplace=True)

Dataset\_new[“BloodPressure”].fillna(dataset\_new[“BloodPressure”].mean(), inplace=True)

Dataset\_new[“SkinThickness”].fillna(dataset\_new[“SkinThickness”].mean(), inplace=True)

Dataset\_new[“Insulin”].fillna(dataset\_new[“Insulin”].mean(), inplace=True)

Dataset\_new[“BMI”].fillna(dataset\_new[“BMI”].mean(), inplace=True)

# Check for missing values in the updated dataset

Dataset\_new.isnull().sum()

# Separate the target variable from the features

Y = dataset\_new[‘Outcome’]

X = dataset\_new.drop(‘Outcome’, axis=1)

# Split the dataset into training and testing sets

From sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42, stratify=dataset\_new[‘Outcome’])

# Train a logistic regression model

From sklearn.linear\_model import LogisticRegression

Model = LogisticRegression()

Model.fit(X\_train, Y\_train)

# Make predictions on the testing set

Y\_predict = model.predict(X\_test)

Print(y\_predict)

# Evaluate the model using a confusion matrix

From sklearn.metrics import confusion\_matrix

Cm = confusion\_matrix(Y\_test, y\_predict)

Print(cm)

# Visualize the confusion matrix

Sns.heatmap(pd.DataFrame(cm), annot=True)

# Calculate the accuracy of the model

From sklearn.metrics import accuracy\_score

Accuracy = accuracy\_score(Y\_test, y\_predict)

Accuracy

# Prompt the user to enter input values for prediction

Pregnancies = float(input(“Enter the number of pregnancies: “))

Glucose = float(input(“Enter the glucose level: “))

BloodPressure = float(input(“Enter the blood pressure: “))

SkinThickness = float(input(“Enter the skin thickness: “))

Insulin = float(input(“Enter the insulin level: “))

BMI = float(input(“Enter the BMI: “))

DiabetesPedigreeFunction = float(input(“Enter the diabetes pedigree function: “))

Age = float(input(“Enter the age: “))

# Create an array of input values

Input\_values = np.array([Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age], dtype=float).reshape(1, -1)

# Make a prediction using the trained model

Y\_predict = model.predict(input\_values)

# Display the input values

Print(“Pregnancies:”, Pregnancies)

Print(“Glucose:”, Glucose)

Print(“BloodPressure:”, BloodPressure)

Print(“SkinThickness:”, SkinThickness)

Print(“Insulin:”, Insulin)

Print(“BMI:”, BMI)

Print(“DiabetesPedigreeFunction:”, DiabetesPedigreeFunction)

Print(“Age:”, Age)

Print()

# Display the prediction

Print(y\_predict)

# Display the prediction result

If y\_predict == 1:

Print(“Diabetic”)

Else:

Print(“Non-Diabetic”)

**OUTPUT:**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 768 entries, 0 to 767

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pregnancies 768 non-null int64

1 Glucose 768 non-null int64

2 BloodPressure 768 non-null int64

3 SkinThickness 768 non-null int64

4 Insulin 768 non-null int64

5 BMI 768 non-null float64

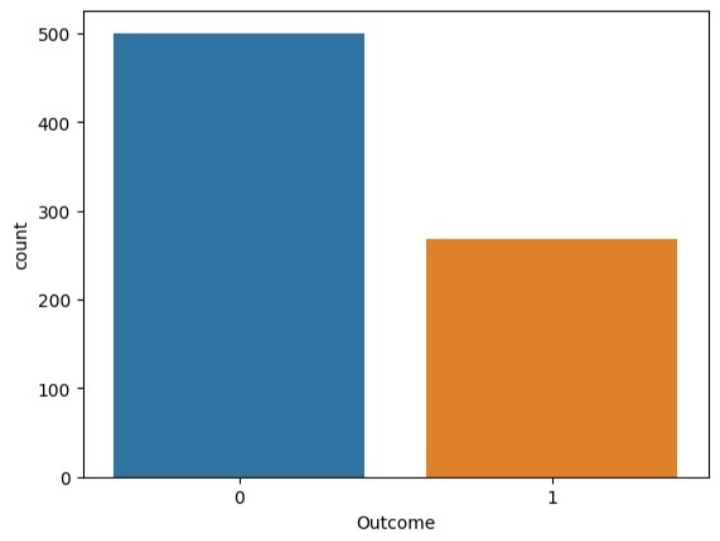
6 DiabetesPedigreeFunction 768 non-null float64

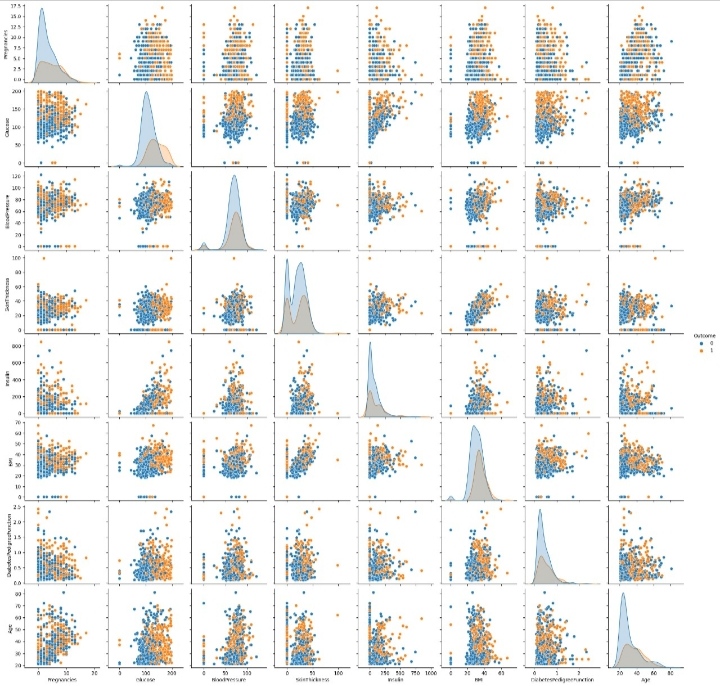
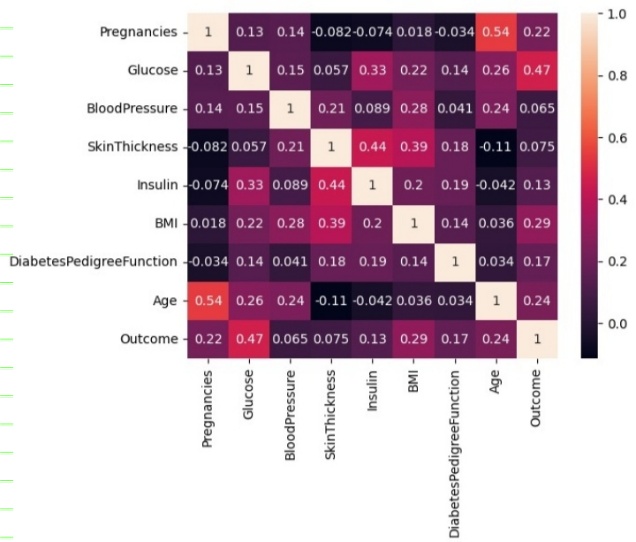
7 Age 768 non-null int64

8 Outcome 768 non-null int64

dtypes: float64(2), int64(7)

memory usage: 54.1 KB





/usr/local/lib/python3.10/dist-packages/sklearn/linear\_model/\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

<https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression>

n\_iter\_i = \_check\_optimize\_result(

[1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 0 0

0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1

0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 1 0 1 0 1 1 0

1 0 0 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1

0 0 0 0 1 0]

[[84 16]

[25 29]]

Enter the number of pregnancies: 6

Enter the glucose level: 148

Enter the blood pressure: 72

Enter the skin thickness: 35

Enter the insulin level: 0

Enter the BMI: 33.6

Enter the diabetes pedigree function: 0.627

Enter the age: 51

Pregnancies: 6.0

Glucose: 148.0

BloodPressure: 72.0

SkinThickness: 35.0

Insulin: 0.0

BMI: 33.6

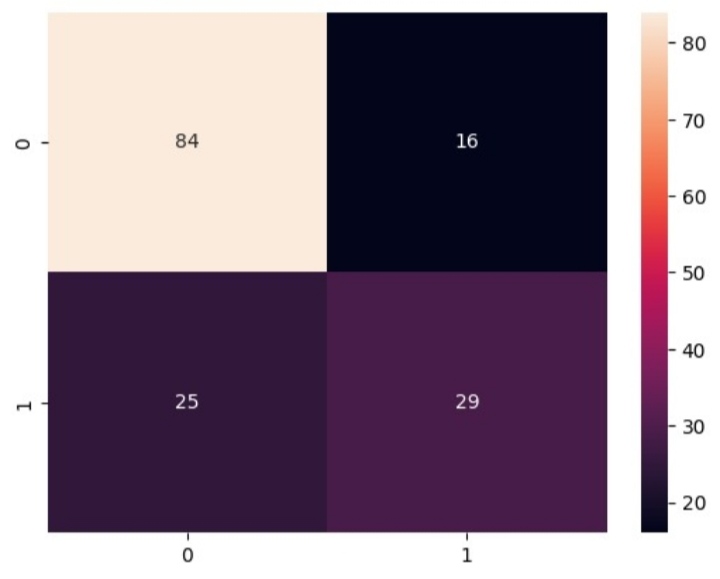
DiabetesPedigreeFunction: 0.627

Age: 51.0

[1]

Diabetic

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names

 Warnings.warn(

**CONCLUSION:**

We used logistic regression to predict whether a person is diabetic or not based on various features. We loaded a dataset, performed data preprocessing by handling missing values, split the dataset into training and testing sets, trained a logistic regression model, made predictions on the testing set, and evaluated the model’s performance using a confusion matrix and accuracy score. Finally, we allowed the user to enter input values for prediction and displayed the prediction result. Logistic regression is a powerful algorithm for binary classification tasks like diabetes prediction, and it can be further optimized and improved by tuning hyperparameters and using more advanced techniques.