AI BASED ON DIABETES PREDICTION SYSTEM

INTRODUCTION:

Diabetes is a prevalent and chronic medical condition that affects millions of people worldwide. Early detection and effective management of diabetes are crucial for improving patient outcomes and reducing healthcare costs. Artificial Intelligence(AI) has emerged as a powerful tool in the field of healthcare, offering the potential to revolutionize the way we diagnose and manage diseases, including diabetes.

PRIMARY GOALS:

- Early Detection: The system aims to identify individuals at risk of diabetes at an early stage, allowing for timely intervention and lifestyle modification.
- Personalized Risk Assessment: It provides a personalized risk assessment for each

individual, taking into account their specific medical history, clinical parameters, and lifestyle factors.

- Improved Healthcare: By automating the prediction process, healthcare providers can allocate resources more efficiently, offer preventive care, and better mange diabetes patients.
- Data-Driven Insights: The system can offer valuable insights by analyzing patterns and relationships within the data, potentially leading to a better understanding of diabetes risk factors.

INNOVATIVE IDEAS FOR PROJECT:

Load the diabetes dataset: The code uses scikit-learn's load_diabets function to load the diabetes dataset, which contains data on diabetes patients, and assigns it to the diabetes variable.

Create a DataFrame: The code creates a dataframe, 'data', from the dataset. It uses the data anf feature names provided in diabetes data

and diabetes feature_name, respectively.It also creates a target variable ‘target’ from diabetes target and adds it to the dataframe as a new column.

Split the data: The dataset is split into features(X) and the target variable(Y).Features are extracted by dropping the ‘target’ column from the dataframe, and the target variable is set to be the ‘target’ column.

Split into training and testing sets: The code uses train_test_split from scikit-learn to split the data into training and testing sets. The training set (X_train, Y_train) is used to train the machine learning model, while the testing set (X_train, Y_train) is used to evaluate the model’s performance.

Creates a logistic regression model: A logistic regression model is created using logistic regression() from scikit-learn. However, it’s worth noting that using logistic regression for a regression problem(predicting a numerical target like diabetes) is not appropriate.

Fit the model: The model is trained on the training data by calling model.fit (X_train, Y_train). This step would be suitable for classification tasks but not for regression.

Make predictions: The model makes predictions on the test data using model. predict (X_test). However, since it's a regression dataset, this is not the correct approach.

Calculate accuracy: The code attempts to calculate the accuracy of the model's predictions using accuracy_score from scikit-learn. This metric is typically used for classification tasks, not regressions.

Print results: The code prints the accuracy and confusion matrix is relevant for classification tasks but not for regression.

CONCLUSION:

The main objective of this paper is to improve the correctness of predictive models. The accuracy can be achieved by either refining the performance of the data or with the help of an

algorithm. For achieving best results the data can be improved at the earlier phase. PIMA dataset has been taken for performing the accuracy checks on each classifier. Among all, the genetic algorithm leads over
others. In this research, the author has concluded one more important factor that the accuracy of a model is highly dependent on the database.

Annexure

PROGRAM:

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
confusion_matrix,classification_report

#load the dataset (replace'diabetes.cvs' with your dataset)
data=pd.read_cvs('diabetes.csv')
#split the dataset into features(X) and target (Y)
X= data.drop('outcome',axis=1)
```

```
Y= data['outcome']
#split data into training and testing sets
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size =
0.2,random_state=42)
#create a random forest classifier model
Model=  randomforestclassifier(n_estimators=
100,random_state=42

#Yrain the model
Model.fit(X_train,Y_train)

#Make prediction on the test set
Y_pred=model.predict(X_test)

#Evaluate the model
Accuracy=accuracy_score(Y_test,Y_pred)
Confusion=confusion_matrix(Y_test,Y_pred)
Classification _report_str=
Classification_report(Y_test,Y_pred)

#print the evaluation results
Print('Accuracy:{accuracy:.2f}')
Print('Confusion matrix:')
Print(confusion)
Print(Classification Report:')
Print(classification_report_str)
```

INPUT:

The patient is likely does not have diabetes

OUTPUT:
Pregnancies:5
Glucose:120
BloodPressure:70
SkinThickness:30
Insulin:80
BMI:25
DiabetePedigreefunction:0.4
Age:35

```
Model = RandomForestClassifier(n_estimators =
100,random_state =42)
#Train the model
Model.fit (X_train,Y_TRAIN)

#Make predictions on the test set
Y_pred = model.predict(X_test)

#Evaluate the model
accuracy = accuracy_score(y_test,y_pred)
confusion = confusion_matrix(y_test,y_pred)
classification_report_str =
classification_report(y_test,y_pred)

#print the evaluation results
Print(f'Accuracy:{accuracy:.2f}')
Print('Confusion Matrix:')
Print(confusion)
Print(Classification Report:')
Print(classification_report_str)
```