**AI-BASED DIABETES PREDICTION**

**SYSTEM**

# Problem Definition:

* The problem is to build an AI-powered diabetes prediction systemϖ 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 personalizedϖ preventive measures, allowing individuals to take proactive actions to manage their health

# 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,ϖ precision, recall, F1-score, and ROC-AUC.

# Iterative Improvement:

* We will fine-tune the model parameters and explore techniques likeϖ feature engineering to enhance prediction accuracy.

**DESIGNING AND INNOVACTION:**

# SYSTEM ARCHITECTURE:

* The AI Based Diabetes Prediction System Consists Of Several Key

# Components:

# DATA COLLECTION:

* Data Collection Data sources include electronic health records, patient demographics, lifestyle data, and genetic information.
* Data is collected securely and stored in a centralized database.

# DATA PREPROCESSSING :

* Raw data is preprocessed to remove noise, outliers, and missing values.Feature extraction techniques are applied to relevant attributes.

# MACHINE LEARNING MODELS :

* The system employs various machine learning models, such as logistic regression, support vector machines, and neural networks, to build predictive models.
* These models are trained on historical data and validated to ensure accuracy.

# FEATURE SELECTION :

* Feature selection techniques are used to identify the most relevant variables for diabetes prediction. This enhances model interpretability and reduces overfitting.

# 

# MODEL EVALUATION:

* Cross-validation and performance metrics like accuracy, precision, recall, and F1-score are used to evaluate model performance. Continuous monitoring and updates are implemented to maintain model accuracy.

# USER INTERFACE:

* The system provides an easy-to-use interface for healthcare professionals and individuals. Users can input their data and receive predictions and risk assessments.

# KEY FEATURES:

## PERSONALIZED RISK ASSESMENT :

* The system provides individualized risk assessments based on user data, allowing for tailored recommendations.

## DATA SECURITY :

* Robust security measures are implemented to protect sensitive health data in compliance with privacy regulations.

## 

## INTERPRETABILITY :

* The system offers explanations for predictions, making it transparent and interpretable for both users and healthcare providers.

## CONTINUOUS LEARNING :

* The model is updated with new data to adapt to changing trends and improve prediction accuracy.

DEPLOYMENT CLOUD BASED SOLUTION

* The system is deployed on a secure cloud infrastructure, allowing for scalability and accessibility from anywhere.

## MOBILE APPLICATION:

* A mobile app is developed for on-the-go access, ensuring convenience for users.

## ETHICAL CONSIDERATIONS INFORMED CONSENT

* Users are required to provide informed consent for data collection and usage.

## PRIVACY PROTECTION

* Stringent data protection measures are in place to safeguard user information.

## BIAS MITIGATION

* Efforts are made to address and mitigate biases in the data and the AI models to ensure fairness and equity.

# STEPS TO BE FOLLOWED :

* Importing required libraries
* Loading data set
* Checking for missing data
* Co-relation matrix
* Train test split
* Training the model
* Making predictions
* Evaluation

## IMPORTING REQUIRED LIBRARIES :

Here we importing some of the tools to process this diabetes prediction system are :

### Pandas :

Pandas is used for data handling,preprocessing and exploration. You can load and manipulate datasets with it.

### Matplotlib :

Matplotlib is for data visualization. You can create charts and graphs to explore and present your data.

### Seaborn :

Seaborn is an enhanced data visualization library that can make your plots more appealing and informative.

### Sklearn :

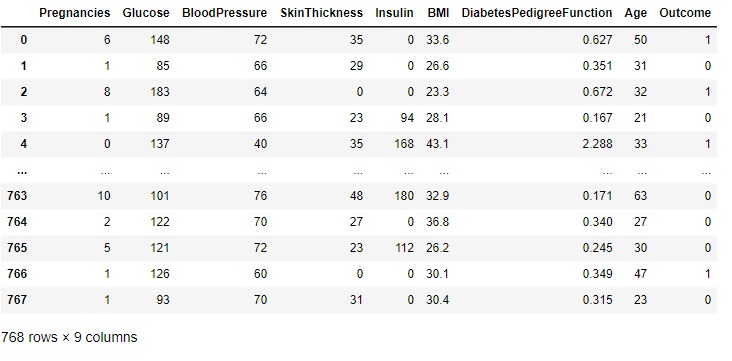
* train\_test\_split from sklearn.model\_selection: It is used to split the dataset into training and testing sets to train and evaluate the model.
* LogisticRegression from sklearn.linear\_model: An example classification model. You can replace it with other machine learning or deep learning models.
* accuracy\_scorefrom sklearn.metrics: These functions are for evaluating the model's performance.

|  |
| --- |
| **import pandas as pd**  **import matplotlib.pyplot as plt**  **import seaborn as sns**  **from sklearn.model\_selection import train\_test\_split**  **from sklearn.linear\_model import LogisticRegression**  **from sklearn.metrics import accuracy\_score** |

## LOADING DATA SET :

* Loading a dataset in the context of machine learning refers to the process of acquiring and bringing a set of data into a format that can be easily processed by a machine learning algorithm. This typically involves reading data from a file (e.g., CSV, Excel, JSON), a database, or an external source, and converting it into a structured data format that can be used for training, testing, and validating machine learning models.

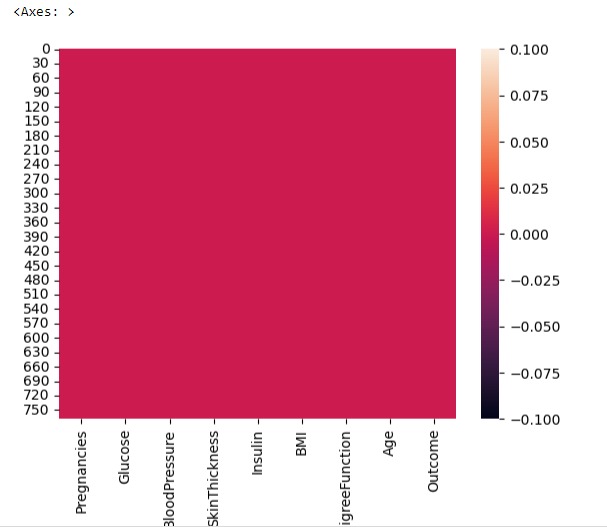
|  |
| --- |
| **data = pd.read\_csv(r"C:\Users\admin\Downloads\archive\diabetes\_dataset.csv")**  **data** |

**d**

# CHECKING FOR MISSING DATA:

checking for missing data" refers to the process of examining a dataset to identify and handle instances where certain data points or values are absent or incomplete. Missing data can significantly impact the performance and accuracy of machine learning models, so it's crucial to address it appropriately.

|  |
| --- |
| sns.heatmap(data.isnull()) |



The steps involved in checking for missing data typically include:

# Identification:

The first step is to identify missing values within the dataset. This can be done using various techniques such as statistical summaries or visualization.

# Handling Missing Data:

* After identifying missing data, you have several options for handling it, including:
* Removing rows or columns with missing values.Imputing missing values with statistical measures (e.g., mean, median, mode) or using more advanced imputation techniques.
* Encoding missing values as a separate category if applicable.

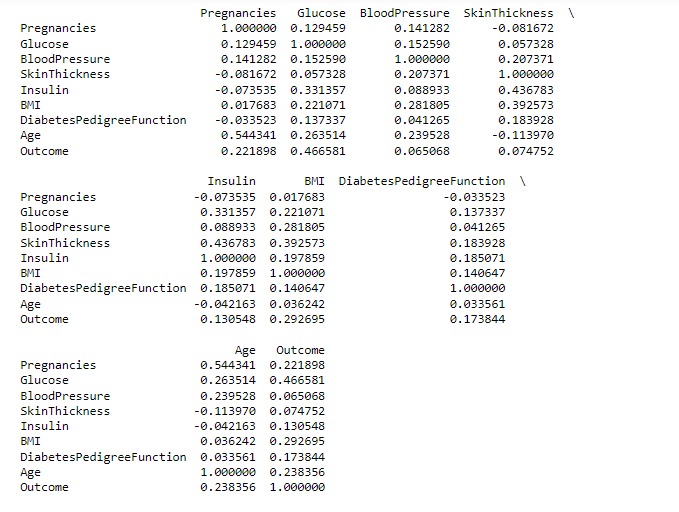
# Data Preprocessing:

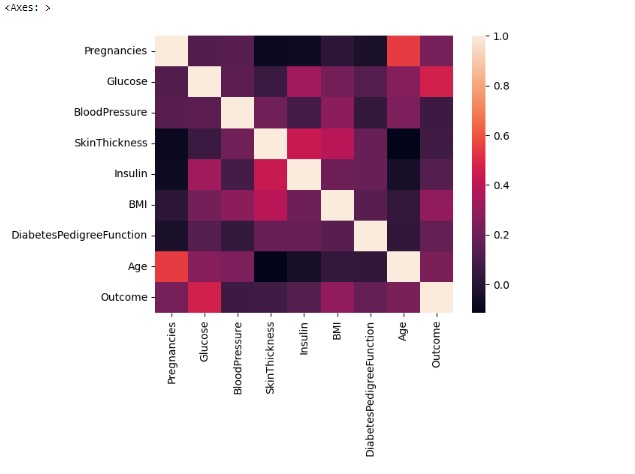
Once missing data is handled, you can proceed with data preprocessing, which may involve feature scaling, encoding categorical variables, and splitting the dataset into training and testing sets.

# CO-RELATION MATRIX

Correlation matrices are often used in feature selection, data exploration, and dimensionality reduction in machine learning. They help identify relationships between variables, which can be important for choosing the right features for a model, understanding multicollinearity (high correlations between independent variables), and gaining insights into the data's structure.

|  |
| --- |
| correlation = data.corr()  print(correlation)  sns.heatmap(correlation) |





# TRAIN TEST SPLIT:

"train-test split" refers to the process of dividing a dataset into two separate subsets: the training set and the testing set. This division is a fundamental step in building and evaluating machine learning models.

X = data.drop("Outcome", axis=1)

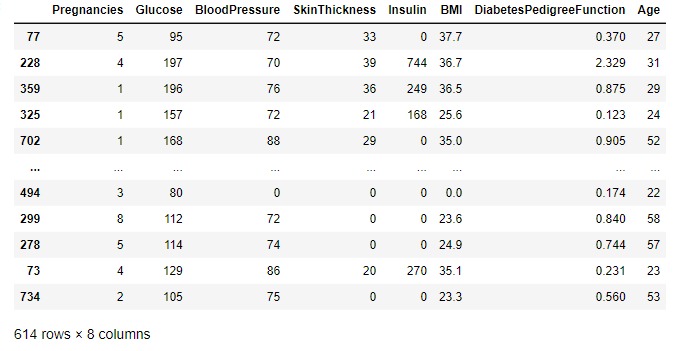
Y = data['Outcome']

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2)

X\_train

model = LogisticRegression()

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



Here's a definition of train-test split:

# Training Set:

The training set is a portion of the dataset that is used to train the machine learning model. It is the data on which the model learns the relationships and patterns between the input features and the target variable. The model "learns" from this data during the training process.

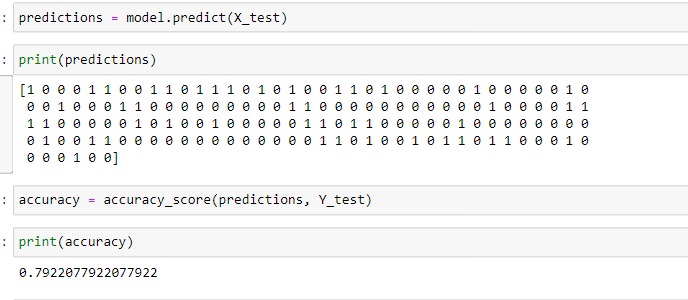
# Testing Set:

The testing set is a separate portion of the dataset that is not used during the model training phase. It is kept aside and used to evaluate the model's performance. The model's predictions are compared to the actual target values in the testing set to assess how well the model generalizes to unseen data.

# MAKING PREDICTIONS:

* MakingPredictions refers to the process of using a trained machine learning model to generate output values or classifications for new, previously unseen data. The goal is to leverage the learned patterns and relationships from the training data to make informed predictions or decisions on new data points.
* Making accurate and reliable predictions is a key objective in machine learning, as it allows models to provide valuable insights, automate decision-making processes, and solve a wide range of real-world problems.

|  |
| --- |
| predictions = model.predict(X\_test)  print(predictions) |



# EVALUATION:

Evaluation refers to the process of assessing the performance and quality of a machine learning model. It involves using various metrics and techniques to measure how well the model can make predictions or classifications on new, unseen data. The goal of evaluation is to determine how effective and reliable the model is in solving a specific problem.

|  |
| --- |
| accuracy = accuracy\_score(predictions, Y\_test)  print(accuracy) |

# Performance Metrics:

Evaluation involves the use of specific performance metrics that are appropriate for the type of problem you are working on. Common metrics include accuracy, precision, recall, F1-score, mean squared error, and others, depending on whether it's a classification or regression problem.

# Testing Data:

Evaluation typically uses a testing dataset that was not used during model training. This unseen data helps assess how well the model generalizes to real-world situations.

# Cross-Validation:

To mitigate the impact of random variations in data splitting, cross-validation techniques like k-fold cross-validation may be employed. These involve splitting the data into multiple subsets, training and testing the model on different combinations, and aggregating the results.

# Confusion Matrix:

In classification tasks, a confusion matrix is often used to provide a detailed breakdown of true positives, true negatives, false positives, and false negatives.

**DEPLOYMENT PROCESS:**

# MODEL DEPLOYMENT:

* Model deployment refers to the process of making a machine learning model or artificial intelligence (AI) model available for use in a real-world, operational environment.
* It involves taking a trained model, which has learned patterns from historical data, and making it accessible for making predictions or decisions in a production setting.
* Model deployment is a crucial step in the machine learning lifecycle, as it enables the practical application of the model to solve specific problems and deliver value.
* The deployed model can be accessed through APIs, integrated into software applications, or used in various systems to automate tasks, provide insights, or make informed decisions based on new data inputs.
* Successful model deployment requires considerations related to scalability, security, monitoring, and ongoing maintenance to ensure the model operates reliably and efficiently in its target environment.

# 

# USING TECHNOLOGYS:

# PYTHON:

* Python is a high-level, versatile, and interpreted programming language known for its simplicity and readability.

MANAGE.PY

* #!/usr/bin/env python  
  *"""Django's command-line utility for administrative tasks."""*import os  
  import sys  
    
    
  def main():  
   *"""Run administrative tasks."""* os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'diabetesprediction.settings')  
   try:  
   from django.core.management import execute\_from\_command\_line  
   except ImportError as exc:  
   raise ImportError(  
   "Couldn't import Django. Are you sure it's installed and "  
   "available on your PYTHONPATH environment variable? Did you "  
   "forget to activate a virtual environment?"  
   ) from exc  
   execute\_from\_command\_line(sys.argv)  
    
    
  if \_\_name\_\_ == '\_\_main\_\_':  
   main()

VIEW.PY

import pandas as pd  
from django.shortcuts import render  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score  
  
def home(request):  
 return render(request,'home.html')  
def predict(request):  
 return render(request,'predict.html')  
def result(request):  
 data = pd.read\_csv(r"C:\Users\admin\Downloads\archive\diabetes\_dataset.csv")  
  
 X = data.drop("Outcome", axis=1)  
 Y = data['Outcome']  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2)  
 X\_train  
  
 model = LogisticRegression()  
 model.fit(X\_train, Y\_train)  
 val1 = float(request.GET['n1'])  
 val2 = float(request.GET['n2'])  
 val3 = float(request.GET['n3'])  
 val4 = float(request.GET['n4'])  
 val5 = float(request.GET['n5'])  
 val6 = float(request.GET['n6'])  
 val7 = float(request.GET['n7'])  
 val8 = float(request.GET['n8'])  
  
 pred = model.predict([[val1, val2, val3, val4, val5, val6, val7, val8]])  
 result1 = ""  
 if pred==[0]:  
 result1 = "positive"  
 else:  
 result1 = "negative"  
 return render(request,'predict.html', {"result2":result1})

**url.py**

*"""  
URL configuration for diabetes prediction project.  
  
The `urlpatterns` list routes URLs to views. For more information please see:  
 https://docs.djangoproject.com/en/dev/topics/http/urls/  
Examples:  
Function views  
 1. Add an import: from my\_app import views  
 2. Add a URL to urlpatterns: path('', views. home, name='home')  
Class-based views  
 1. Add an import: from other\_app.views import Home  
 2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')  
Including another URLconf  
 1. Import the include() function: from django.urls import include, path  
 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))  
"""*from django.contrib import admin  
from django.urls import path  
from . import views  
  
urlpatterns = {  
 path('admin/', admin.site.urls),  
 path("", views.home),  
 path("predict/", views.predict),  
 path("predict/result", views.result),  
  
}

*""" Django settings for diabetesprediction project.  
  
Generated by 'django-admin startproject' using Django 5.0a1.  
  
For more information on this file, see  
https://docs.djangoproject.com/en/dev/topics/settings/  
  
For the full list of settings and their values, see  
https://docs.djangoproject.com/en/dev/ref/settings/  
"""*import os  
from pathlib import Path  
  
# Build paths inside the project like this: BASE\_DIR / 'subdir'.  
BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent  
  
  
# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/dev/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!  
SECRET\_KEY = 'django-insecure-ufap#\_3y\*r5-xq#uqzu4f2=vril\*s=b@\*ewa\*bvrxntjgv+ne2'  
  
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True  
  
ALLOWED\_HOSTS = []  
  
  
# Application definition  
  
INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
]  
  
MIDDLEWARE = [  
 'django.middleware.security.SecurityMiddleware',  
 'django.contrib.sessions.middleware.SessionMiddleware',  
 'django.middleware.common.CommonMiddleware',  
 'django.middleware.csrf.CsrfViewMiddleware',  
 'django.contrib.auth.middleware.AuthenticationMiddleware',  
 'django.contrib.messages.middleware.MessageMiddleware',  
 'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]  
  
ROOT\_URLCONF = 'diabetesprediction.urls'  
  
TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [os.path.join(BASE\_DIR,'templates')],  
 'APP\_DIRS': True,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = 'diabetesprediction.wsgi.application'  
  
  
# Database  
# https://docs.djangoproject.com/en/dev/ref/settings/#databases  
  
DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.sqlite3',  
 'NAME': BASE\_DIR / 'db.sqlite3',  
 }  
}  
  
  
# Password validation  
# https://docs.djangoproject.com/en/dev/ref/settings/#auth-password-validators  
  
AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',  
 },  
]  
  
  
# Internationalization  
# https://docs.djangoproject.com/en/dev/topics/i18n/  
  
LANGUAGE\_CODE = 'en-us'  
  
TIME\_ZONE = 'UTC'  
  
USE\_I18N = True  
USE\_L10N = True  
  
USE\_TZ = True  
  
  
# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/dev/howto/static-files/  
  
STATIC\_URL = '/static/'  
STATICFILES\_DIRS = (  
 os.path.join(BASE\_DIR,'static'),  
)  
STATIC\_ROOT = os.path.join(os.path.dirname(BASE\_DIR),'static')

# HTML&CSS:

* HTML is a markup language used to structure the content of a web page. It defines the elements and their organization on a web page, such as text, headings, paragraphs, lists, images, links, forms, and more.
* CSS is a stylesheet language used to control the presentation and layout of HTML elements. It allows web developers to define how elements should appear on a web page, such as their colors, fonts, spacing, positioning, and responsiveness to different screen sizes.

# Home.html:

{% load static %}  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport"  
 content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">  
 <meta http-equiv="X-UA-Compatible" content="ie=edge">  
 <title>Document</title>  
</head>  
<body>  
  
</body>  
</html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Home</title>  
 <style type="text/css">  
 div{  
 color:'white'  
 }  
 h1{  
 color:'white';  
 font-family:arial, sans-serif;  
 font-size:60px;  
 font-weight:bold;  
 margin-top:200px;  
 }  
 h2{  
 color:'white';  
 font-family:arial, sans-serif;  
 font-size:15px;  
 font-weight:bold;  
 margin-top:200px;  
 }  
 body{  
 background-color:#cceeff;  
 background-repeat:no-repeat;  
 background-attachment:fixed;  
 background-size:cover;  
 }  
 input[type=submit]{  
 background-color:#4dc3ff;  
 border:2px;  
 color:white;  
 padding:16px 32px;  
 cursor:pointer;  
 margin-top:15px;  
 }  
 </style>  
</head>  
<body>  
  
<div align = 'center'>  
 <h1>  
 DIABETES\_PREDICTION SYSTEM  
 </h1>  
 <form action="predict">  
 <input type="submit" value = "Lets get started">  
 </form>  
  
</div>  
</body>  
</html>

# Predict.html:

{%load static%}  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>PredictionSpace</title>  
 <style type="text/css">  
 body{  
 background-color:#d3d3d3;  
 background-repeat:no-repeat;  
 background-attachment:fixed;  
 background-size:cover;  
 }  
 .main{  
 position:fixed;  
 top:140px;  
 left:410;  
 width:550px;  
 background-colour:#ffffff;  
 border-radius:10px;  
 align-items:center;  
 padding:5%;  
 }  
 h1{  
 color:#00086b3;  
 font-size:30px;  
 font-weight:bold;  
 }  
 input[type=submit]{  
 background-color:#4dc3ff;  
 border:2px;  
 color:white;  
 padding;8px 16px;  
 cursor:pointer;  
 margin-top:15px;  
 }  
 </style>  
  
</head>  
<body>  
<p>  
 Welcome to Prediction Page  
</p>  
<div align = 'center' class="main">  
 <h1>please enter the following information: </h1>  
 <form action="result">  
 <table>  
 <tr>  
 <td align="right">pregnancies:</td>  
 <td align="left"><input type="text" name="n1"></td>  
 </tr>  
 <tr>  
 <td align="right">Glucose:</td>  
 <td align="left"><input type="text" name="n2"></td>  
 </tr>  
 <tr>  
 <td align="right">Blood Pressure:</td>  
 <td align="left"><input type="text" name="n3"></td>  
 </tr>  
 <tr>  
 <td align="right">Skin Thickness:</td>  
 <td align="left"><input type="text" name="n4"></td>  
 </tr>  
 <tr>  
 <td align="right">Insulin:</td>  
 <td align="left"><input type="text" name="n5"></td>  
 </tr>  
 <tr>  
 <td align="right">MBI:</td>  
 <td align="left"><input type="text" name="n6"></td>  
 </tr>  
 <tr>  
 <td align="right">Diabetes Pedigree function:</td>  
 <td align="left"><input type="text" name="n7"></td>  
 </tr>  
 <tr>  
 <td align="right">Age:</td>  
 <td align="left"><input type="text" name="n8"></td>  
 </tr>  
 </table>  
 <input type="submit">  
 </form>  
 Result:{{result2}}  
</div>  
  
</body>  
</html>

# OUTPUT:

# Screenshot (10).png

# CONCLUSION:

The development of an AI-based diabetes prediction system for this project represents a significant step towards harnessing advanced technology for healthcare. The system holds the potential to improve early diagnosis and risk management, ultimately enhancing patient outcomes. However, it is essential to acknowledge the need for rigorous testing, ongoing data collection, and collaboration with medical professionals to ensure its effectiveness, accuracy, and ethical use in real-world healthcare settings.