PHASE 5 AIML

AI -BASED DIABETES PREDICTION SYSTEM

TITLE : "AI-BASED DIABETES PREDICTION SYSTEM"

In this phase the entire document of our project is presented.

Problem Statement:

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.

Design Thinking:

1. Data Collection and Preparation:

- The Kaggle PIMA dataset is used for our project which includes blood pressure, skin thickness, insulin levels and other relevant data from the patient.
- Analyzation of data and standardization of the data will be performed since our model requires the input in similar range.

2. Feature Engineering:

- Followed by standardization , the labelling of our data will be done here.
- For training purpose , the test train split is selected.
- The training data will be fed into our model whereas the test data will be used for finding the accuracy score of our trained model.
- With this we can declare the performance of the prediction system.

3. Model Selection and Training:

- Support vector machine(SVM), a machine learning algorithms for classification is selected for this prediction system.
- SVM undergoes training and fine-tuning using the training dataset.

4. Evaluation:

- Evaluation of model performance is done by using accuracy score metrics.

- Validation of the model on the testing dataset is ensured as it generalizes well to new data.

5. Deployment:

- A user-friendly interface is developed for individuals to input their data and receive their predictions.

Key Features and Advancements:

- Comprehensive Data Integration:

The Diabetes predicter brings together diverse data sources, creating a holistic view of an individual's health. It seamlessly incorporates electronic health records, genetic data, lifestyle factors, and more, ensuring a complete and accurate assessment of diabetes risk.

- Personalized Predictive Models:

Using advanced machine learning techniques, the system generates personalized risk profiles. It considers an individual's unique genetic makeup, medical history, lifestyle choices, and more to deliver highly accurate predictions.

- Real-Time Risk Assessment:

The system offers real-time risk assessment, providing individuals with immediate insights into their diabetes risk. This enables proactive decision-making and timely interventions.

- User-Friendly Interface:

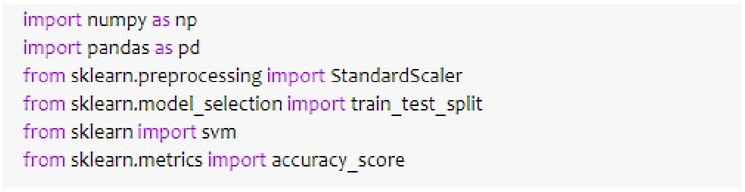
A user-centric design ensures that individuals can easily interact with the system. It features an intuitive interface accessible through web and mobile platforms, making it convenient for users to input data and receive personalized recommendations.

Impact and Benefits:

- Early Detection: By identifying individuals at risk well in advance, it facilitates early intervention and lifestyle modifications, potentially preventing the onset of diabetes.
- Personalized Guidance: With tailored recommendations for diet, exercise, and healthcare interventions, individuals receive actionable insights to proactively manage their health.

- Empowerment: This innovation empowers individuals to become active participants in their healthcare, fostering a sense of ownership and responsibility for their well-being.
- Healthcare Efficiency: By streamlining risk assessment and intervention, the system can lead to more efficient healthcare resource allocation and reduced healthcare costs.

Importing the Dependencies :



```
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
```

- Numpy is primarily used for various mathematical operations and it is easy to working with arrays
- Pandas is used for data analysis and machine learning tasks. It supports working with tabular data like CSV.
- StandardScalar is used standardize our data in similar range.
- To train our model we are using train test split method.
- SVM is the algorithm which we use to build this project.
- For evaluation purpose, accuracy score is imported.

Data Collection and Analysis :

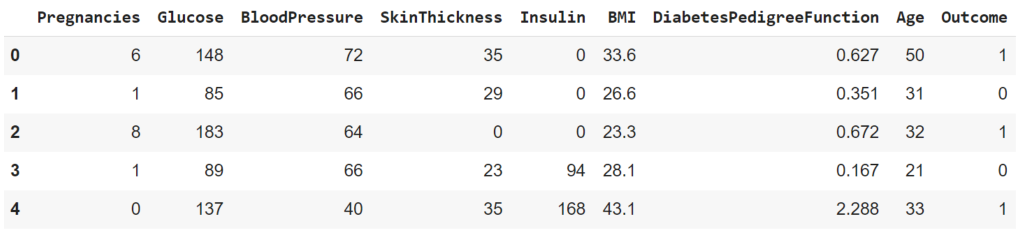
#loading the diabetes dataset to a pandas DataFrame

diabetes_dataset = pd.read_csv('/content/diabetes.csv')

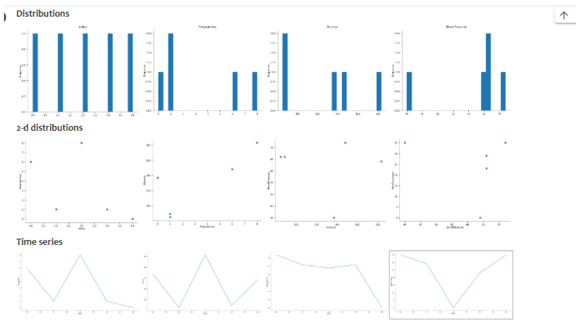
The dataset is downloaded from the Kaggle PIMA dataset and loaded into a csv file. The above function is used to read the stored csv file.

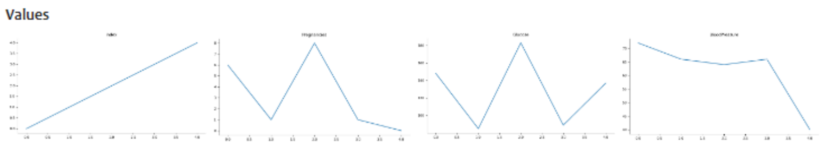
# printing the first 4 rows of the dataset

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





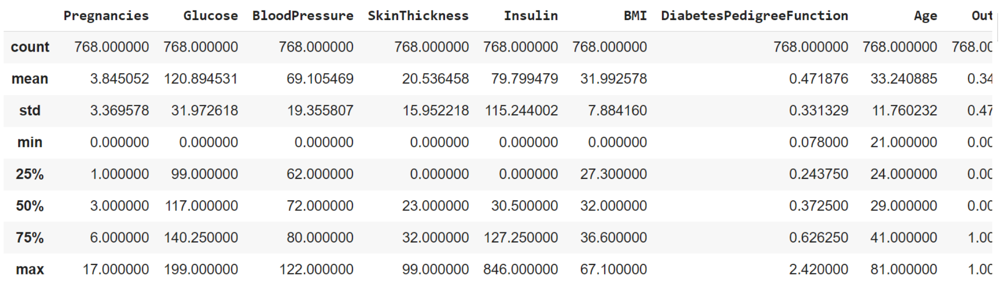
Data Preprocessing :

# number of rows and Columns in this dataset

- diabetes_dataset.shape
- [768, 9] : There are **768 rows and 9 columns.**

# getting the statistical measures of the data

diabetes_dataset.describe[ ]



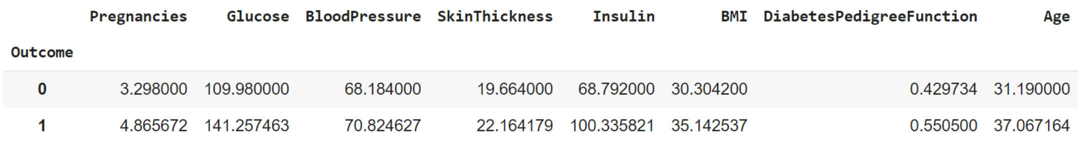
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Out
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.00
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.34
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.47
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.00
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.00
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.00
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.00
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.00

- diabetes_dataset['Outcome'].value_counts()
- 0   500
- 1   268

Name: Outcome, dtype: int64

1. 0 --> Non-Diabetic
2. 1 --> Diabetic

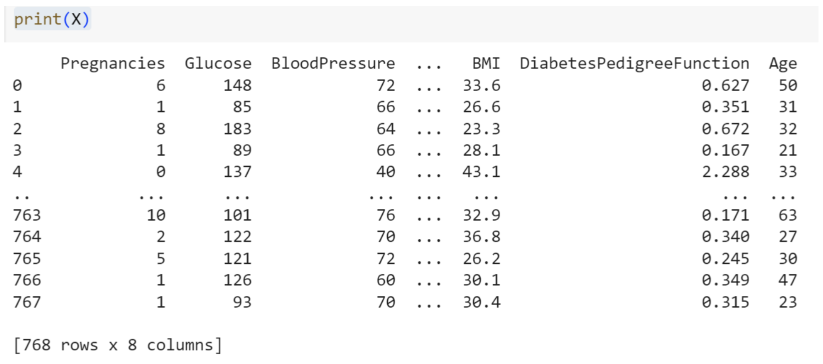
diabetes_dataset.groupby('Outcome').mean()



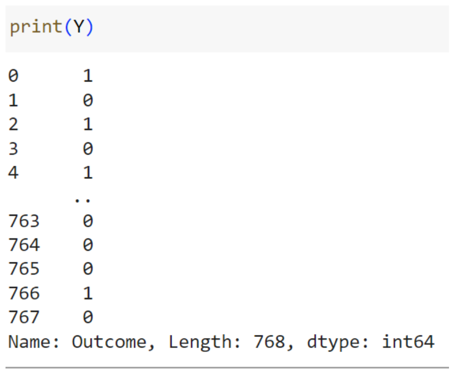
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Outcome								
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

# separating the data and labels

- X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
- Y = diabetes_dataset['Outcome']



```
print(X)

     Pregnancies  Glucose  BloodPressure  ...   BMI  DiabetesPedigreeFunction  Age
0              6      148             72  ...  33.6                     0.627   50
1              1       85             66  ...  26.6                     0.351   31
2              8      183             64  ...  23.3                     0.672   32
3              1       89             66  ...  28.1                     0.167   21
4              0      137             40  ...  43.1                     2.288   33
..           ...      ...            ...  ...   ...                       ...  ...
763           10      101             76  ...  32.9                     0.171   63
764            2      122             70  ...  36.8                     0.340   27
765            5      121             72  ...  26.2                     0.245   30
766            1      126             60  ...  30.1                     0.349   47
767            1       93             70  ...  30.4                     0.315   23

[768 rows x 8 columns]
```

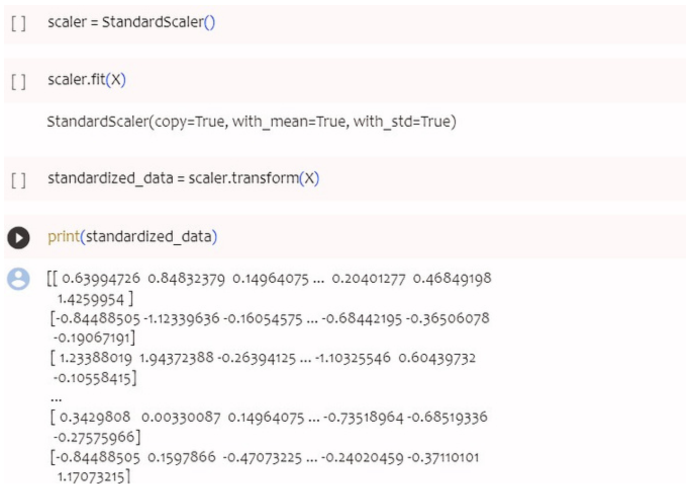


```
print(Y)

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
```

- For further model selection and training the data set has been separated into two with x & y labels.

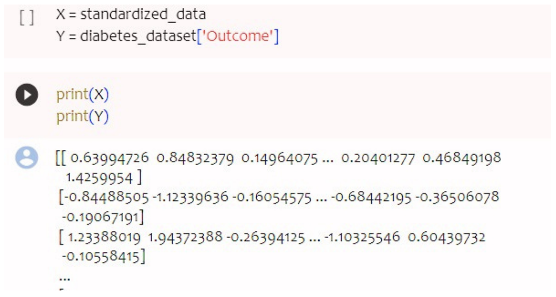
Data Standardization:

- It is difficult for a machine learning model to make some predictions when the range of the values in the dataset are in vast difference. So it is required to standardize our data in a particular range.
- As we already imported the standard scaler from sklearn, we can use it for fitting and transforming the data in same range.



```
[ ]  scaler = StandardScaler()

[ ]  scaler.fit(X)

     StandardScaler(copy=True, with_mean=True, with_std=True)

[ ]  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]
```

- we can see that values are in the range similar to each other. lets assign the variable X to the standardized data .

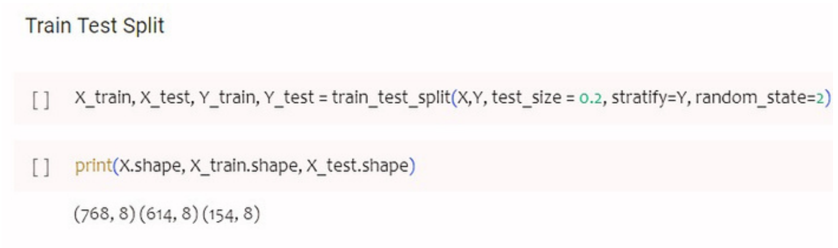


```
[ ]  X = standardized_data
     Y = diabetes_dataset['Outcome']

     print(X)
     print(Y)

[[ 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]
 ...
```

Algorithm :

- We are using Support Vector Machine Algorithm for our project since they can generate more complex decision boundaries.
- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.
- However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.
- This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and
- Hence algorithm is termed as Support Vector Machine.

Training the Model:



Train Test Split

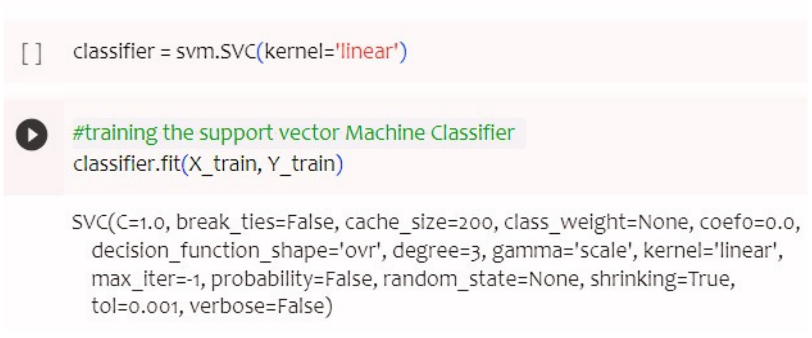
```
[ ]  X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, stratify=Y, random_state=2)

[ ]  print(X.shape, X_train.shape, X_test.shape)

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

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- **Train Dataset: Used to fit the machine learning model.**
- **Test Dataset: Used to evaluate the fit machine learning model**

The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

#training the support vector Machine Classifier



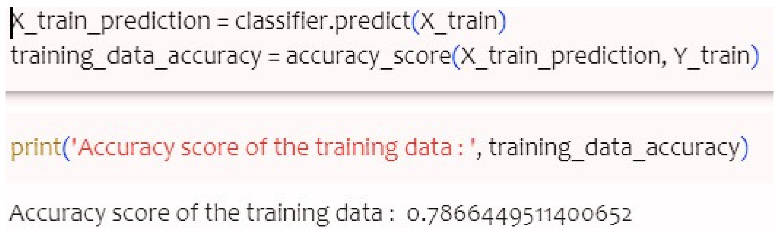
```
[ ]  classifier = svm.SVC(kernel='linear')
```

```
#training the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Model Evaluation:

# accuracy score on the training data



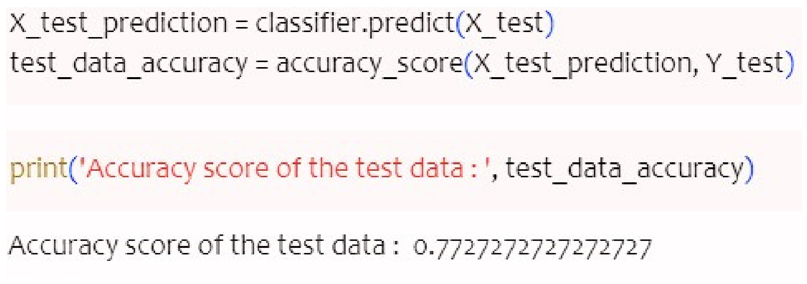
```
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

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

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

- We are comparing the X_train predictions with the original label Y_train in order to get the accuracy of our model.
- We got the accuracy score of our model as 0.78 which is pretty good.

# accuracy score on the test data



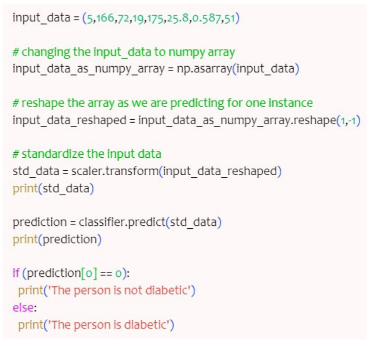
```
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

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

```
Accuracy score of the test data :  0.7727272727272727
```

- We got the accuracy score as 0.77 which represents that our model has not been over trained. Thus our model works good and we can build our predictive system.

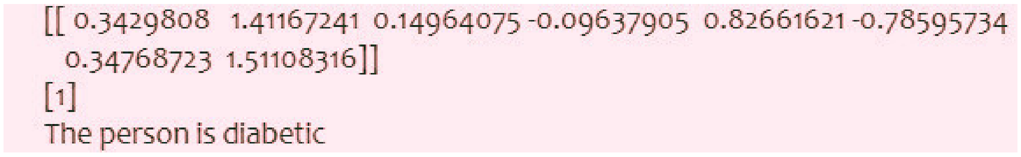
Making a Predictive System:



```
input_data = (5,166,72,19,175,25.8,0.587,51)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

- We have trained our model with 768 data points. But here we are using just one data point. If we don't reshape our input data, what our model expects is 768 data points. Thus it will make confusions. So we are re- shaping the arrays and standardizing.
- Now our trained model will predict the predictions for the given values and displays whether the person is diabetic or not by using the if condition.

Result:



```
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person is diabetic
```

- The given input is a random data from our PIMA dataset with the outcome 1 which denotes positive for diabetes.
- Our model has also predicted it correctly. Thus we have built a diabetes predictive system successfully.

Deployment:

Deployment can be done by using streamlit.

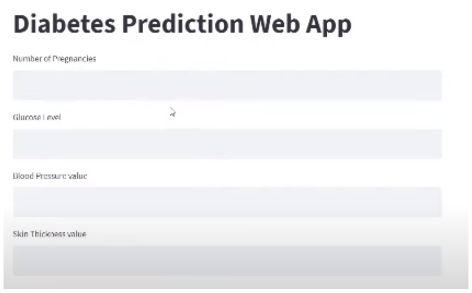
Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps.

web app code:

```
import numpy as np
import pickle
import streamlit as st


# loading the saved model
loaded_model = pickle.load(open('D:/Work/Machine Learning/Deploying Machine
Learning model/trained_model.sav', 'rb'))


# creating a function for Prediction

def diabetes_prediction(input_data):
    

    # changing the input_data to numpy array
    input_data_as_numpy_array = np.asarray(input_data)

    # reshape the array as we are predicting for one instance
    input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

    prediction = loaded_model.predict(input_data_reshaped)
    print(prediction)



    if (prediction[0] == 0):
      return 'The person is not diabetic'
    else:
      return 'The person is diabetic'
  


def main():
    
    
    # giving a title
    st.title('Diabetes Prediction Web App')
    
    
    # getting the input data from the user
    
    
    Pregnancies = st.text_input('Number of Pregnancies')
    Glucose = st.text_input('Glucose Level')
    BloodPressure = st.text_input('Blood Pressure value')
    SkinThickness = st.text_input('Skin Thickness value')
    Insulin = st.text_input('Insulin Level')
    BMI = st.text_input('BMI value')
```

```
    BMI = st.text_input('BMI value')
    DiabetesPedigreeFunction = st.text_input('Diabetes Pedigree Function
value')
    Age = st.text_input('Age of the Person')
    
    
    # code for Prediction
    diagnosis = ''
    
    # creating a button for Prediction
    
    if st.button('Diabetes Test Result'):
        diagnosis = diabetes_prediction([Pregnancies, Glucose, BloodPressure,
SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age])
        
        
    st.success(diagnosis)
    
    
    
    
    
if __name__ == '__main__':
    main()
```

__our simple user interface:__


Diabetes Prediction Web App

Number of Pregnancies

Glucose Level

Blood Pressure value

Skin Thickness value

Conclusion:

Our project is to design an Ai-based diabetes prediction system. We were provided with the PIMA diabetes dataset to build our prediction system. The project is done by using SVM algorithm. Among several algorithms, SVM worked better as they got 77% accuracy which is better than other classifier algorithms. The objective is to diagnostically predict whether or not a person has diabetes and our goal is successful.

__References__:

S. SIVARANJANI, S. ANANYA, J. ARAVINTH AND R. KARTHIKA, "DIABETES PREDICTION USING MACHINE LEARNING ALGORITHMS WITH FEATURE SELECTION AND DIMENSIONALITY REDUCTION," 2021 7TH INTERNATIONAL CONFERENCE ON ADVANCED COMPUTING AND COMMUNICATION SYSTEMS (ICACCS), COIMBATORE, INDIA, 2021, PP. 141-146, DOI: 10.1109/ICACCS51430.2021.9441935.

G. PARIMALA, R. KAYALVIZHI AND S. NITHIYA, "DIABETES PREDICTION USING MACHINE LEARNING," 2023 INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATION AND INFORMATICS (ICCCI), COIMBATORE, INDIA, 2023, PP. 1-10, DOI: 10.1109/ICCCI56745.2023.10128216

A. ALMAHDAWI, Z. S. NAAMA AND A. AL-TAIE, "DIABETES PREDICTION USING MACHINE LEARNING," 2022 3RD INFORMATION TECHNOLOGY TO ENHANCE E-LEARNING AND OTHER APPLICATION (IT-ELA), BAGHDAD, IRAQ, 2022, PP. 186-190, DOI: 10.1109/IT-ELA57378.2022.10107919

Thanking you.