AI Based Diabetes Prediction System

NAME: DANIEL DAVID.M
REG NO: 720421104011

AI_PHASE2 DOCUMENT SUBMISSION

PROJECT : 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. Early prediction of diabetes and prediabetes can reduce treatment cost and improve intervention. The development of (pre)diabetes is associated with various health conditions that can be monitored by routine health checkups. This study aimed to develop amachine learning-based model for predicting (pre)diabetes.

DESIGN THINKING :

1. 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.
2. Data Preprocessing: The medical data needs to be cleaned, normalized, and prepared for training machine learning models.
3. Feature Selection: We will select relevant features that can impact diabetes risk prediction.
4. Model Selection: We can experiment with various machine learning algorithms like Logistic Regression, Random Forest, and Gradient Boosting.

5. Evaluation: We will evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.
6. Iterative Improvement: We will fine-tune the model parameters and explore techniques like feature engineering to enhance prediction accuracy.

Data Source:

https://www.kaggle.com/datasets/mathchi/diabetes-data-set

PROGRAM :

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

#from sklearn.linear_model  import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.naive_bayes import BernoulliNB

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns
```

```
import pandas as pd

data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")

data.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.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
data.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

#here few misconception is there lke BMI can not be zero, BP can't be zero, glucose, insuline can't be zero so lets try to fix it

now replacing zero values with the mean of the column

```
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())

data['BloodPressure'] =
data['BloodPressure'].replace(0,data['BloodPressure'].mean())

data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())

data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())

data['SkinThickness'] =
data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

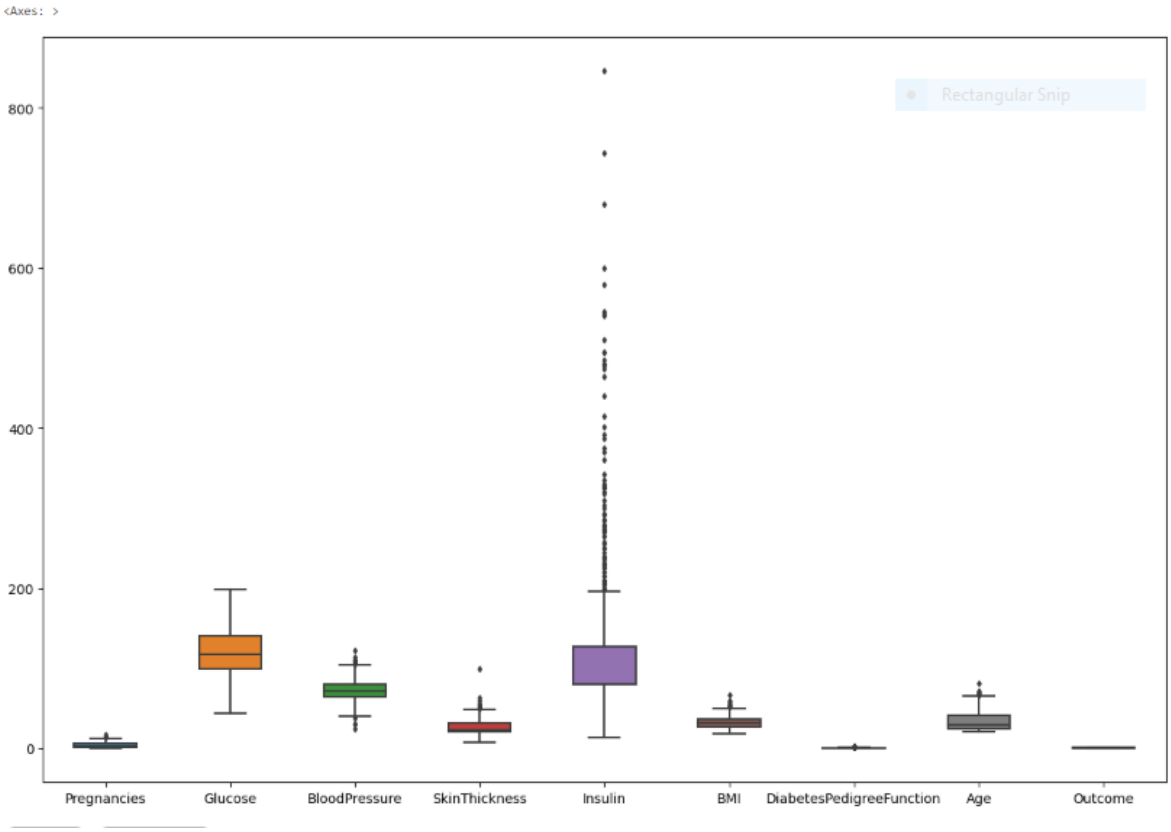
data.describe()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

#now we have dealt with the 0 values and data looks better. But, there still are outliers present in some columns.lets visualize it

```
fig, ax = plt.subplots(figsize=(15,10))

sns.boxplot(data=data, width= 0.5,ax=ax,  fliersize=3)
```



data.head()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50	1
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31	0
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32	1
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21	0
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33	1

#segregate the dependent and independent variable

X = data.drop(columns = ['Outcome'])

y = data['Outcome']

separate dataset into train and test

```
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.25,random_state=0)

X_train.shape, X_test.shape



import pickle

##standard Scaling- Standardization

def scaler_standard(X_train, X_test):

    #scaling the data

    scaler = StandardScaler()

    X_train_scaled = scaler.fit_transform(X_train)

    X_test_scaled = scaler.transform(X_test)



    #saving the model

    file = open('standardScalar.pkl','wb')

    pickle.dump(scaler,file)

    file.close()



    return X_train_scaled, X_test_scaled



X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
```

X_train_scaled

```
array([[ 1.50755225, -1.09947934, -0.89942504, ..., -1.45561965,
        -0.98325882, -0.04863985],
       [-0.82986389, -0.1331471 , -1.23618124, ...,  0.09272955,
        -0.62493647, -0.88246592],
       [-1.12204091, -1.03283573,  0.61597784, ..., -0.03629955,
         0.39884168, -0.5489355 ],
       ...,
       [ 0.04666716, -0.93287033, -0.64685789, ..., -1.14021518,
        -0.96519215, -1.04923114],
       [ 2.09190629, -1.23276654,  0.11084355, ..., -0.36604058,
        -0.5075031 ,  0.11812536],
       [ 0.33884418,  0.46664532,  0.78435594, ..., -0.09470985,
         0.51627505,  2.953134  ]])
```

Decision Tree Model Training With Hyperparameter Tuning

```
import warnings

warnings.filterwarnings('ignore')
```

```
parameter={

 'criterion':['gini','entropy','log_loss'],

  'splitter':['best','random'],

  'max_depth':[1,2,3,4,5],

  'max_features':['auto', 'sqrt', 'log2']


}
```

```
from sklearn.model_selection import GridSearchCV

classifier=DecisionTreeClassifier()
```

```
clf=GridSearchCV(classifier,param_grid=parameter,cv=3,scoring='accuracy',verbose=3)
clf.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 90 candidates, totalling 270 fits
[CV 1/3] END criterion=gini, max_depth=1, max_features=auto, splitter=best;, score=0.578 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=1, max_features=auto, splitter=best;, score=0.641 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=1, max_features=auto, splitter=best;, score=0.641 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=1, max_features=auto, splitter=random;, score=0.646 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=1, max_features=auto, splitter=random;, score=0.641 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=1, max_features=auto, splitter=random;, score=0.641 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=best;, score=0.635 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=best;, score=0.641 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=best;, score=0.688 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=random;, score=0.646 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=random;, score=0.641 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=random;, score=0.641 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=1, max_features=log2, splitter=best;, score=0.646 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=1, max_features=log2, splitter=best;, score=0.783 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=1, max_features=log2, splitter=best;, score=0.641 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=1, max_features=log2, splitter=random;, score=0.708 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=1, max_features=log2, splitter=random;, score=0.609 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=1, max_features=log2, splitter=random;, score=0.630 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=2, max_features=auto, splitter=best;, score=0.714 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=2, max_features=auto, splitter=best;, score=0.698 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=2, max_features=auto, splitter=best;, score=0.788 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=2, max_features=auto, splitter=random;, score=0.656 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=2, max_features=auto, splitter=random;, score=0.672 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=2, max_features=auto, splitter=random;, score=0.734 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=2, max_features=sqrt, splitter=best;, score=0.641 total t
```

```
clf.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 5,
 'max_features': 'auto',
 'splitter': 'best'}
```

```
classifier=DecisionTreeClassifier(criterion='entropy',max_depth=5,max_features='auto',splitter='random')
```

```
classifier.fit(X_train,y_train)

## Support Vector Classifier With Hyperparameter Tuning

# defining parameter range
param_grid = {'C': [0.1, 1, 10],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel':['linear','rbf','polynomial']
              }


grid=GridSearchCV(SVC(),param_grid=param_grid,refit=True,cv=3,verbose=3,scorin
g='accuracy')

grid.fit(X_train,y_train)


grid.best_params_
```

```
{'C': 0.1, 'gamma': 1, 'kernel': 'linear'}
```

```
svc_clf=SVC(C=0.1,gamma=1,kernel='linear')
svc_clf.fit(X_train,y_train)



## Decision Tree prediction
y_pred = classifier.predict(X_test_scaled)


## SVC prediction
y_pred_svc = svc_clf.predict(X_test_scaled)


conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

```
array([[118,  12],
       [ 48,  14]])
```

```
conf_mat = confusion_matrix(y_test,y_pred_svc)
conf_mat
```

```
array([[130,   0],
       [ 62,   0]])
```

```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
```

```
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]

Accuracy = (true_positive + true_negative) / (true_positive +false_positive +
false_negative + true_negative)
Accuracy
```

```
0.6770833333333334
```

```
Accuracy = (true_positive + true_negative) / (true_positive +false_positive +
false_negative + true_negative)
Accuracy
```

```
0.6770833333333334
```

```
Precision = true_positive/(true_positive+false_positive)
Precision
```

```
1.0
```

```
Recall = true_positive/(true_positive+false_negative)
Recall
```

```
0.6770833333333334
```

```
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

```
0.8074534161490683
```

```
import pickle
file = open('modelForPrediction.pkl','wb')
pickle.dump(classifier,file)
file.close()
```

Conclusion:

In summary, building an AI-powered diabetes prediction system is a complex endeavor involving data preprocessing, model training, and deployment. The

provided code outline is a starting point. However, full development includes user interface integration, personalized preventive measures, data security, and ongoing updates. Ethical considerations, data privacy, and collaboration with healthcare professionals are crucial in real-world healthcare applications.