AI-Based Diabetes Prediction System

Phase-1 Documentation Submission

Register Number:963521104017

Name: GURU RAHUL RU

Project Title: **Diabetes Prediction System**

OBJECTIVE:

The objective of this project is to design and implement an AI-based Diabetes Prediction System. This system aims to leverage advanced machine learning algorithms and predictive modelling to identify individuals at risk of developing diabetes.

ABSTRACT:

The "AI-Based Diabetes Prediction System" project endeavour's to create an intelligent and proactive solution for the early detection and management of diabetes. Drawing inspiration from the modular approach used in building chatbots with Python, this project adopts a structured methodology. It employs state-of-the-art machine learning and artificial intelligence techniques to predict the risk of diabetes in individuals, fostering early intervention and personalized healthcare.

Key phases of the project include selecting the appropriate machine learning libraries and frameworks for diabetes risk prediction, designing the predictive model, developing the AI algorithm to process medical data, deploying the system for accessible user interaction, and ensuring robustness through continuous monitoring and analytics.

Incorporating Natural Language Processing (NLP) concepts analogous to intent recognition in chatbots, the system analyses health-related data and user information to identify potential diabetes risks. A thoughtful dialogue management system helps in maintaining context during user interactions, while response generation employs various techniques, such as rule-based systems and generative models, to provide relevant feedback.

This project also addresses the critical aspects of security and privacy concerning sensitive healthcare data. Best practices for securing user information and safeguarding against malicious inputs are emphasized throughout the development process.

By adopting a modular approach akin to chatbot development, this "AI-Based Diabetes Prediction System" aims to empower healthcare professionals and individuals alike with a versatile and effective tool for diabetes risk assessment and proactive health management. This approach showcases the adaptability of Python's rich ecosystem to create intelligent solutions addressing pressing healthcare challenges.

PYTHON PROGRAM

In [1]:

```
#Let's start with importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import LogisticRegression
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
```

In [2]:

```
#read the data file
data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

In [3]:

```
data.describe()
```

Out[3]:

	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

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

In [4]:

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

Out[4]:

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

We can see there few data for columns Glucose , Insulin, skin thickenss, BMI and Blood Pressure which have value as 0. That's not possible,right? you can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

In [5]:

```
#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())
```

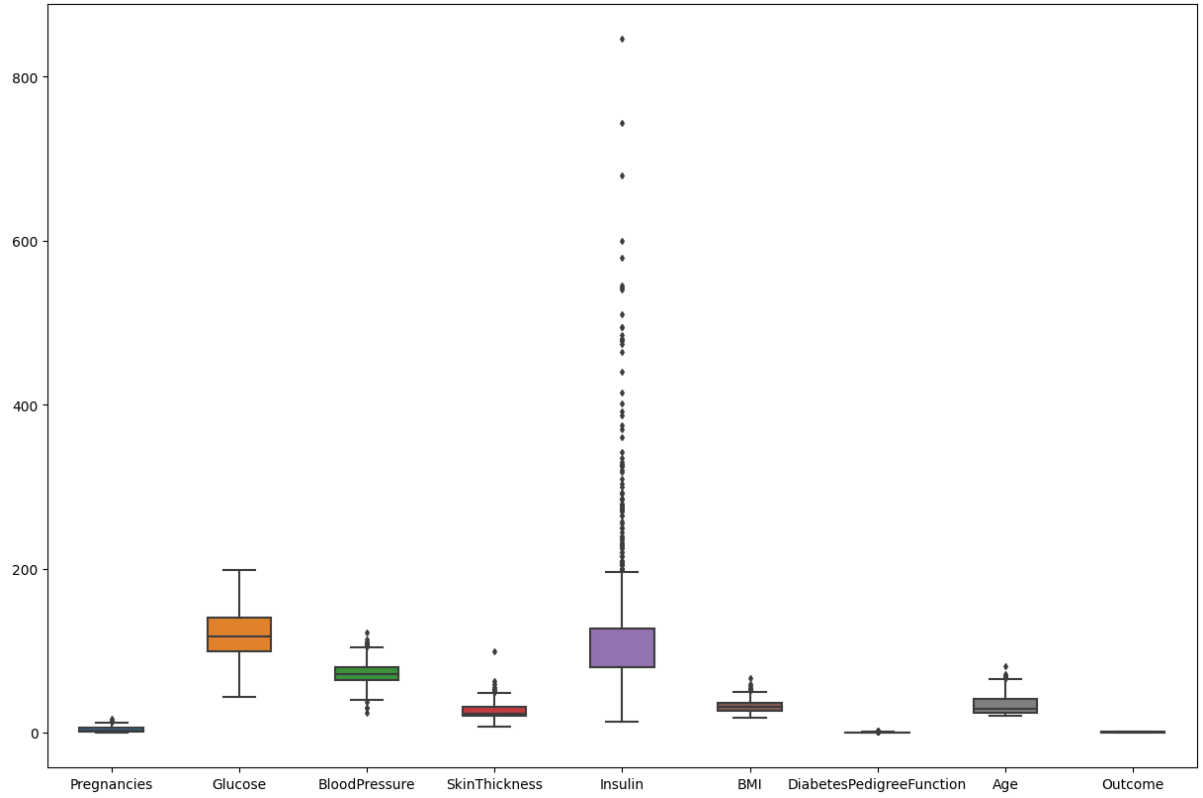
```
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness
'].mean())
```

In [6]:

```
#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)
```

Out[6]:

<Axes: >



In [7]:

```
data.head()
```

Out[7]:

	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

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

In [8]:
```
#segregate the dependent and independent variable
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
```

In [9]:
```
# separate dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,rand
om_state=0)
X_train.shape, X_test.shape
```

Out[9]:
```
((576, 8), (192, 8))
```

In [10]:
```
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
```

In [11]:
```
X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
```

In [12]:
```
X_train_scaled
```

Out[12]:
```
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  ]])
```

In [13]:
```
log_reg = LogisticRegression()

log_reg.fit(X_train_scaled,y_train)
```

Out[13]:
```
LogisticRegression
LogisticRegression()
```

In [14]:
```
## Hyperparameter Tuning
## GridSearch CV
from sklearn.model_selection import GridSearchCV
import numpy as np
import warnings
warnings.filterwarnings('ignore')
# parameter grid
parameters = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3,3,7),
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}
```

In [15]:
```
logreg = LogisticRegression()
clf = GridSearchCV(logreg,                    # model
                   param_grid = parameters,   # hyperparameters
                   scoring='accuracy',        # metric for scoring
                   cv=10)                     # number of folds
```

```
clf.fit(X_train_scaled,y_train)
```

Out[15]:

GridSearchCV
estimator: LogisticRegression

```
LogisticRegression
```

In [16]:

```
clf.best_params_
```

Out[16]:

```
{'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
```

In [17]:

```
clf.best_score_
```

Out[17]:

```
0.763793103448276
```

let's see how well our model performs on the test data set.

In [18]:

```
y_pred = clf.predict(X_test_scaled)
```

accuracy = accuracy_score(y_test,y_pred) accuracy

In [19]:

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

Out[19]:

```
array([[117,  13],
       [ 26,  36]])
```

In [20]:

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

In [21]:

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

Out[21]:

```
0.796875
```

In [22]:
```
Precision = true_positive/(true_positive+false_positive)
Precision
```

Out[22]:
```
0.9
```

In [23]:
```
Recall = true_positive/(true_positive+false_negative)
Recall
```

Out[23]:
```
0.8181818181818182
```

In [24]:
```
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

Out[24]:
```
0.8571428571428572
```

In [25]:
```
import pickle
file = open('modelForPrediction.pkl','wb')
pickle.dump(log_reg,file)
file.close()
```

OUTPUT

Accuracy: 0.796875

Confusion Matrix:

[[117 13]

 [26 36]]

Precision: 0.9

Recall: 0.8181818181818182

F1 Score: 0.8571428571428572