4222-SURYA GROUP OF INSTITUTIONS

VIKRAVANDI-605652

AI BASED DIABETES PREDICTON SYSTEM

NAAN MUDHALVAN PROJECT

PREPARED BY

S.DINESH

REG.NO:422221106006

ECE DEPARTMENT

AI-based-diabetes-prediction-system:

INTRODUCTION:

Welcome to the future of healthcare with our AI-based Diabetes Prediction System. Harnessing the power of artificial intelligence, our innovative solution is designed to revolutionize the way we approach diabetes detection and prevention. By analyzing vast amounts of data and employing advanced machine learning algorithms, our system provides accurate predictions, enabling early intervention and personalized care. Stay ahead of diabetes with our cutting-edge technology, leading the way towards a healthier tomorrow.

AI MODELS USED:

- Logistic Regression
- Decision tress
- Random forest

STEPS TAKEN:

DATA COLLECTION:

	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

When encountered with a data set, first we should analyze and "**get to know**" the data set.This step is necessary to familiarize with the data, to gain some understanding of the potential features and to see if data cleaning is needed. First, we will import the necessary libraries and import our data set. We can observe the mentioned columns in the data set.

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as snsdiabetes = pd.read_csv('datasets/diabetes.csv')
diabetes.columns
```

Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
dtype='object')

We can examine the data set using the pandas' **head()** method**.**

```
diabetes.head()
```

```
print("Diabetes data set dimensions : {}".format(diabetes.shape))
```

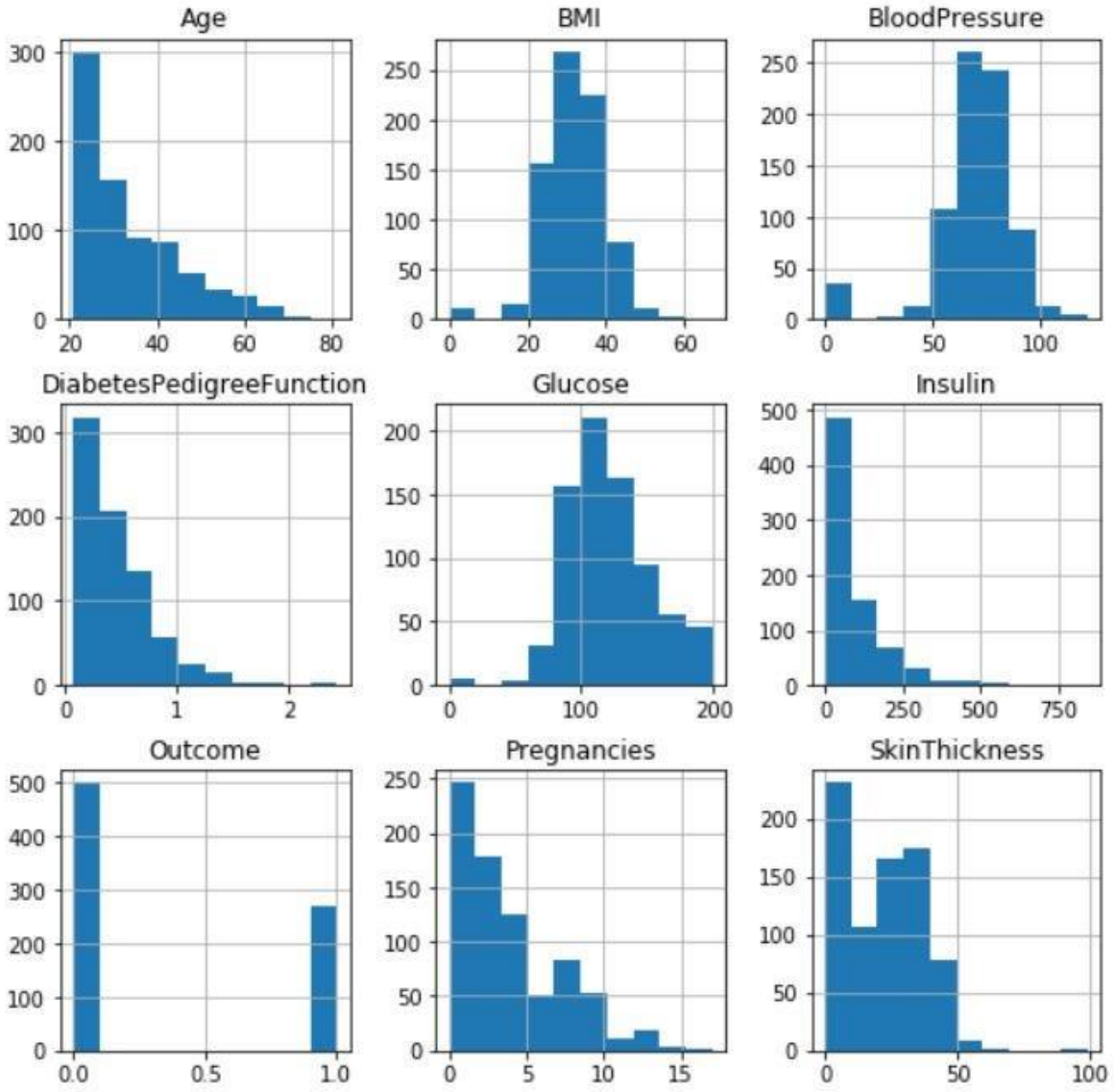
Diabetes data set dimensions : (768, 9) .

We can observe that the data set contain 768 rows and 9 columns. '*Outcome*' is the column which we are going to predict, which says if the patient is diabetic or not. 1 means the person is diabetic and 0 means a person is not. We can identify that out of the 768 persons, 500 are labeled as 0 (non-diabetic) and 268 as 1 (diabetic**).**

```
diabetes.groupby('Outcome').size()
```

```
Outcome
0    500
1    268
dtype: int64
```

Visualization of data is an imperative aspect of data science. It helps to understand data and also to explain the data to another person. Python has several interesting visualization libraries such as Matplotlib, Seaborn, etc.visualization which is built on top of matplotlib, to find the data distribution of the features.



diabetes.groupby('Outcome').hist(figsize=(9, 9))

DATA PREPROCESSING:

The next phase of the machine learning work flow is data cleaning. Considered to be one of the crucial steps of the workflow, because it can make or break the model. There is a saying in machine learning **"Better data beats fancier algorithms"**, which suggests better data gives you better resulting models.

There are several factors to consider in the data cleaning process

1.Duplicate or irrelevant observations

2. Bad labeling of data, same category occurring multiple times

3. Missing or null data points.

4.Unexpected outliers.

diabetes.isnull().sum()

diabetes.isna().sum()

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

Blood pressure: By observing the data we can see that there are 0 values for blood pressure. And it is evident that the readings of the data set seem wrong because a living person cannot have a diastolic blood pressure of zero. By observing the data we can see 35 counts where the value is 0.

```
print("Total : ", diabetes[diabetes.BloodPressure == 0].shape[0])Total :
35print(diabetes[diabetes.BloodPressure == 0].groupby('Outcome')['Age'].count())Outcome
0   19
1   16
Name: Age, dtype: int64
```

Plasma glucose levels: Even after fasting glucose levels would not be as low as zero. Therefore zero is an invalid reading. By observing the data we can see 5 counts where the value is 0.

```
print("Total : ", diabetes[diabetes.Glucose == 0].shape[0])Total :  5print(diabetes[diabetes.Glucose
== 0].groupby('Outcome')['Age'].count())Total :  5
Outcome
0   3
1   2
Name: Age, dtype: int64
```

Skin Fold Thickness: For normal people, skin fold thickness can't be less than 10 mm better yet zero. Total count where value is 0: 227.

```
print("Total : ", diabetes[diabetes.SkinThickness == 0].shape[0])Total :
227print(diabetes[diabetes.SkinThickness == 0].groupby('Outcome')['Age'].count())Outcome
0   139
```

1 88
Name: Age, dtype: int64
BMI: Should not be 0 or close to zero unless the person is really underweight which could be life-threatening.
print("Total : ", diabetes[diabetes.BMI == 0].shape[0])Total : 11print(diabetes[diabetes.BMI == 0].groupby('Outcome')['Age'].count())Outcome
0 9
1 2
Name: Age, dtype: int64
Insulin: In a rare situation a person can have zero insulin but by observing the data, we can find that there is a total of 374 counts.
print("Total : ", diabetes[diabetes.Insulin == 0].shape[0])Total : 374print(diabetes[diabetes.Insulin == 0].groupby('Outcome')['Age'].count())Outcome
0 236
1 138
Name: Age, dtype: int64

Here are several ways to handle invalid data values :

1. Ignore/remove these cases: This is not actually possible in most cases because that would mean losing valuable information. And in this case “skin thickness” and “insulin” columns mean to have a lot of invalid points. But it might work for “BMI”, “glucose ”and “blood pressure” data points.

2. Put average/mean values: This might work for some data sets, but in our case putting a mean value to the blood pressure column would send a wrong signal to the model.

3. Avoid using features: It is possible to not use the features with a lot of invalid values for the model. This may work for “skin thickness” but it's hard to predict that.

4. We will remove the rows which the “BloodPressure”, “BMI” and “Glucose” are zero.
5. diabetes_mod = diabetes[(diabetes.BloodPressure != 0) & (diabetes.BMI != 0) & (diabetes.Glucose != 0)]print(diabetes_mod.shape)(724, 9)

MODEL SELECTION:

Feature Selection is the process of transforming the gathered data into features that better represent the problem that we are trying to solve to the model, to improve its performance and accuracy.Feature selection creates more input features from the existing features and also combines several features to produce more intuitive features to feed to the model.
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']X = diabetes_mod[feature_names]
y = diabetes_ mod.

MODEL SELECTION:

Model selection or algorithm selection phase is the most exciting and the heart of machine learning. It is the phase where we select the model which performs best for the data set at hand .First, we will be calculating the **"Classification Accuracy (Testing Accuracy)"** of a given set of classification models with their default parameters to determine which model performs better with the diabetes data set. We will import the necessary libraries for the notebook. We import 7 classifiers namely **K-Nearest Neighbours , Support Vector Classifier, Logistic Regression, Gaussian Naive Bayes, Random Forest, and Gradient Boost** to be contenders for the best classifier.

```
. neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn models = []models.append(('KNN', KNeighborsClassifier()))
models.append(('SVC', SVC()))
models.append(('LR', LogisticRegression()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('GNB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('GB', GradientBoostingClassifier()))
```

EVALUATION:

It is a general practice to avoid training and testing on the same data. The reasons are that the goal of the model is to predict **out-of-sample data**, and the model could be overly complex leading to **overfitting**. To avoid the aforementioned problems, there are two precautions.

1. Train/Test Split
2. K-Fold Cross-Validation

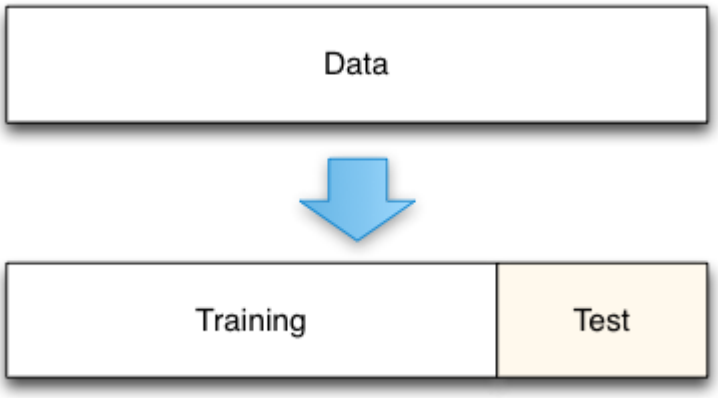
We will import *"train_test_split"* for train/test split **and *"cross_val_score" for*** *k-fold cross-validation.* ***"accuracy_score"*** **is to evaluate the accuracy of the model in the train/test split method.**

TRAIN/TEST SPLIT:

This method split the data set into two portions: a **training set** and a **testing set**. The **training set** is used to train the model. And the **testing set** is used to test the model, and evaluate the accuracy.

Pros : *But, train/test split is still useful because of its* ***flexibility and speed***

Cons : *Provides a* ***high-variance estimate*** *of out-of-sa*mple `accuracy`




```
X_train, X_test, y_train, y_test = train_test_split(X, y,
stratify = diabetes mod.Outcome, random state=0)
```

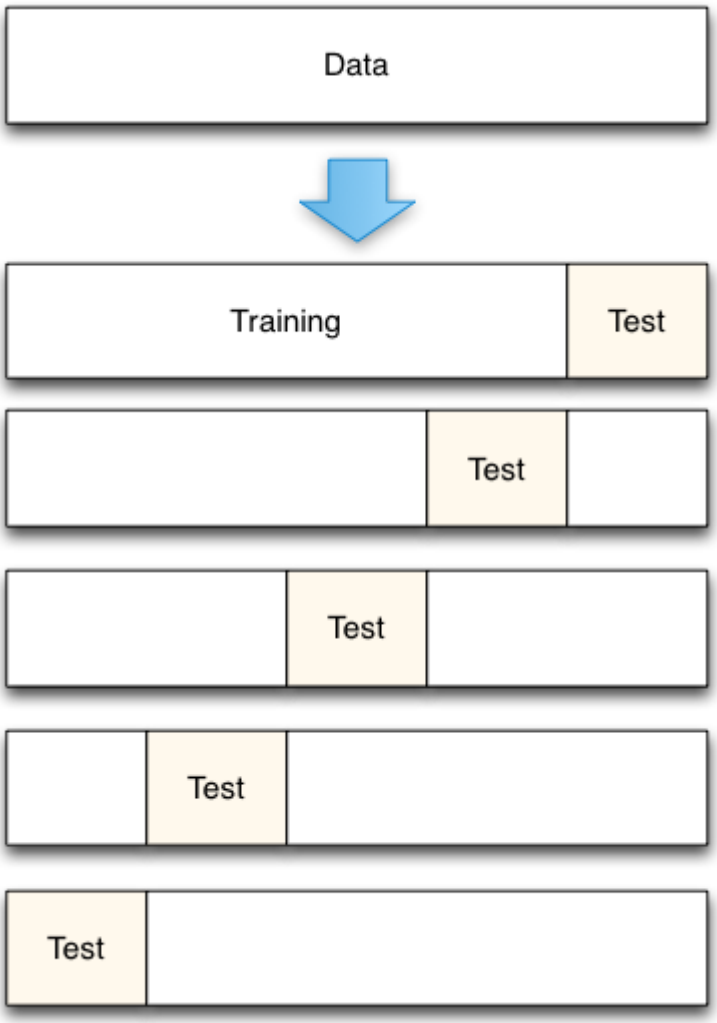
Then we fit each model in a loop and calculate the accuracy of the respective model using the *"accuracy_score"*.

```
names = []
scores = []for name, model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))
    names.append(name)tr_split = pd.DataFrame({'Name': names,
'Score': scores})
print(tr_split)
```

```
   Name     Score
0   KNN  0.711521
1   SVC  0.656075
2    LR  0.776440
3    DT  0.681327
4   GNB  0.755681
5    RF  0.739165
6    GB  0.765442
```

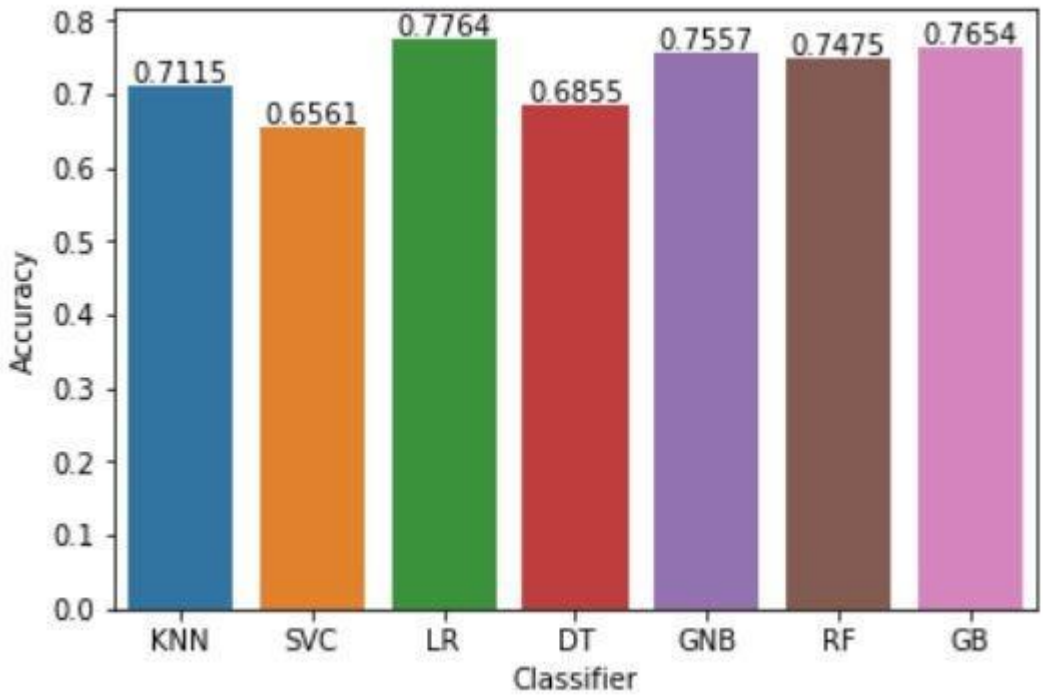
K-FOLD CROSS VALIDATION:

```
names = []
scores = []for name, model in models:

    kfold = KFold(n_splits=10, random_state=10)
    score = cross_val_score(model, X, y, cv=kfold,
scoring='accuracy').mean()

    names.append(name)
    scores.append(score)kf_cross_val = pd.DataFrame({'Name':
names, 'Score': scores})
print(kf_cross_val)
```



	Name	Score
0	KNN	0.711521
1	SVC	0.656075
2	LR	0.776440
3	DT	0.685494
4	GNB	0.755681
5	RF	0.747519
6	GB	0.765442

```
axis = sns.barplot(x = 'Name', y = 'Score', data = kf_cross_val)
axis.set(xlabel='Classifier', ylabel='Accuracy')for p in
axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.005,
'{:1.4f}'.format(height), ha="center")

plt.show()
```



We can see the Logistic Regression, Gaussian Naive Bayes, Random Forest and Gradient Boosting have performed better than the rest. From the base level we can observe that the Logistic Regression performs better than the other algorithms.