



## DATA ANALYSIS

COURSE PRESENTER  
(DR. Omainah)

### Pima Indians Diabetes Database

STUDENT	ID
SHAHAD AMER	444005703
AREEJ TALEB	444002403

DEPARTMENT OF (INFORMATION SCIENCE-DATA SCIENCE)  
COLLEGE OF COMPUTER AND INFORMATION SYSTEMS  
UMM AL-QURA UNIVERSITY

2024

## **INTRODUCTION**

The Pima Indians Diabetes Database is a significant dataset used for diabetes research and predictive modeling, this dataset consists of several medical predictor variables and one target variable, Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on. We used the Kagel website to obtain this data, and in our project, we will discuss the analysis of Pima Indians Diabetes Database.

## **OBJECTIVE**

The main objective of this project is to build a model that accurately predict whether the patients in the dataset have diabetes or not based on certain diagnostic measurements included in the dataset.

# Data preparation

## Import Libraries

Import the most important libraries that facilitate the analysis process.

## Read the File

Load the dataset from a specified file.

## Display Initial Data

Display the first few rows of the Data Frame to understand its structure.

## Identify Missing Values

Identify any zero values in the dataset, which may indicate missing data.

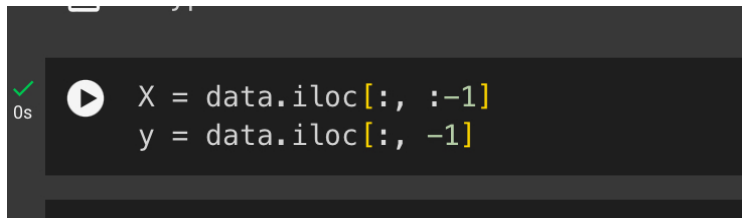
## Replace Zero Values

Replace the zero values in specific columns with missing values (pd.NA).

## Fill Missing Values

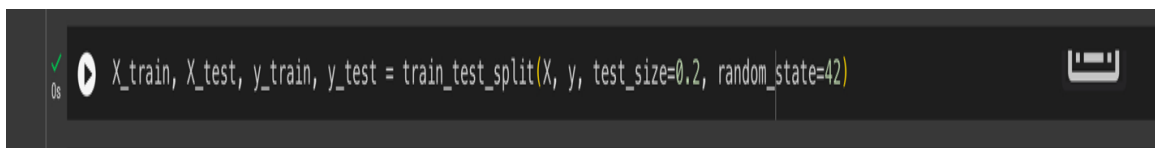
- Fill missing values with the mean for the columns **Glucose** and **Blood Pressure**.
- Fill missing values with the median for the columns **Skin Thickness**, **Insulin**, and **BMI**.

This approach helps ensure that analyses or models built on this data are more robust and informative.

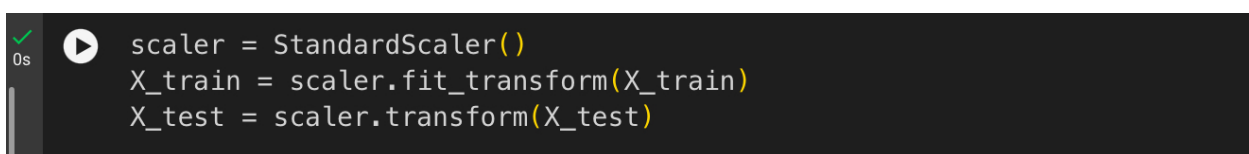
A screenshot of a Jupyter Notebook code cell. It contains two lines of Python code: `X = data.iloc[:, :-1]` and `y = data.iloc[:, -1]`. The code is highlighted in yellow. To the left of the code, there is a green checkmark and a play button icon, indicating the code has been executed successfully. The execution time is shown as '0s'.

Here we split the dataset into features (X) and labels (y).  
`data.iloc[:, :-1]`: This selects all rows (:) and all columns except the last one (:-1).  
`data.iloc[:, -1]`: This selects all rows and only the last column (-1).

This technique allows to feed the model input data (features) and teach it to predict the target variable (labels). This separation helps in understanding and evaluating the model's performance on unseen data.

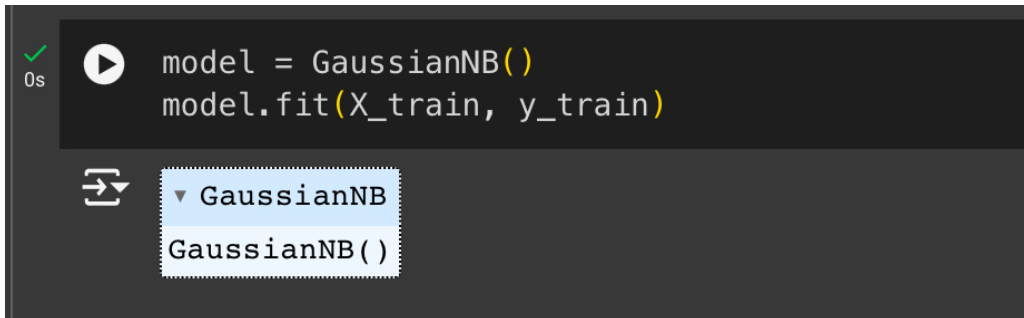
A screenshot of a Jupyter Notebook code cell. It contains one line of Python code: `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`. The code is highlighted in yellow. To the left of the code, there is a green checkmark and a play button icon, indicating the code has been executed successfully. The execution time is shown as '0s'. To the right of the code, there is a save icon.

Splitting the data into training and testing sets the test set should be smaller than the training set, here our test set is 2%.

A screenshot of a Jupyter Notebook code cell. It contains three lines of Python code: `scaler = StandardScaler()`, `X_train = scaler.fit_transform(X_train)`, and `X_test = scaler.transform(X_test)`. The code is highlighted in yellow. To the left of the code, there is a green checkmark and a play button icon, indicating the code has been executed successfully. The execution time is shown as '0s'.

this code ensures that both the training and testing feature sets are on the same scale, which can improve the performance.


## Try Gaussian naïve bayes



```
0s [play] model = GaussianNB()  
model.fit(X_train, y_train)
```

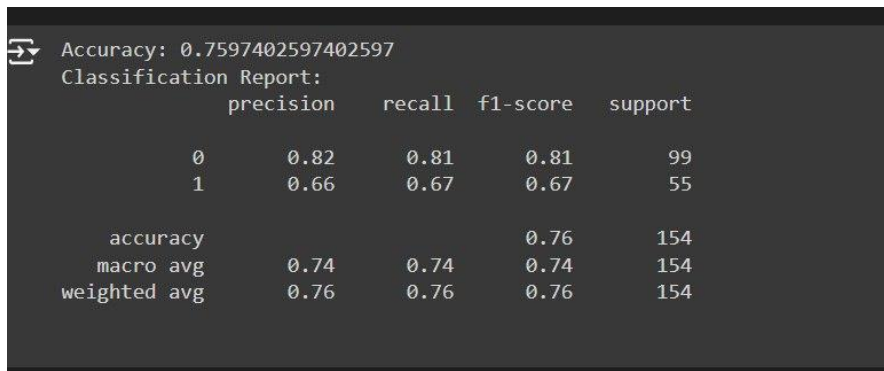
Below the code, a dropdown menu is open showing the class `GaussianNB` and its constructor `GaussianNB()`.

this code initializ Gaussian Naive Bayes classifier and trains it on the training dataset, allowing it to learn the relationship between the features and the target labels. Once trained, the model can be used to make predictions on new data.



```
1s [play] predictions = model.predict(X_test)
```

This code uses the trained model to generate predictions for the test data.



```
[play] Accuracy: 0.7597402597402597  
Classification Report:  
              precision    recall  f1-score   support  
  
      0               0.82        0.81         0.81         99  
      1               0.66        0.67         0.67         55  
  
   accuracy                0.76         154  
  macro avg               0.74         154  
 weighted avg               0.76         154
```

this code evaluates and reports the performance of the Gaussian Naive Bayes model on the test dataset. The accuracy is %75.

## Try Random Forest

```
[ model = RandomForestClassifier(random_state=42)
  model.fit(X_train, y_train)
```

```
[ predictions = model.predict(X_test)
```

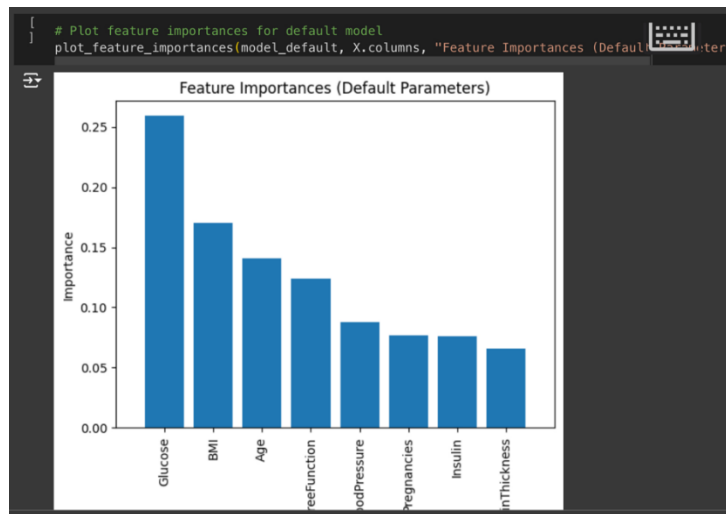
```
⇒ Accuracy: 0.7597402597402597
Classification Report:
              precision    recall  f1-score   support

     0           0.82       0.81      0.81         99
     1           0.66       0.67      0.67         55

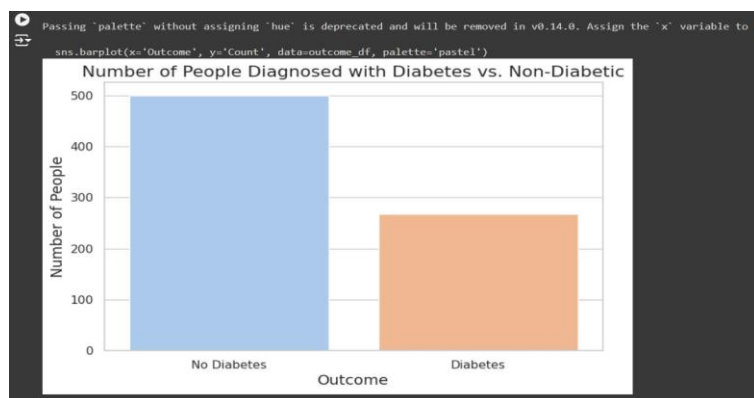
 accuracy          0.76         154
 macro avg         0.74         154
 weighted avg      0.76         154
```

the accuracy in random forest is %75.  
Which means GaussianNB and Random Forest has the same accuracy.

## Visualisation



From the plot we can see the most important features, which they are BMI and Glucose, they Are the most common reasons for diabetes.



From the graph we can see the number of people who have been diagnosed with diabetes and those who have not been diagnosed with diabetes.