جامعة أم القرى
UMM AL-QURA UNIVERSITY

Project Data Analysis 2

Name	ID
Amany Saeed Fallatah	44411940
Nahla Mohammad AL osaimi	44412006

COURSE PRESENTER

“DR. Omaima Fallatah ”

Course Number: DS3114

Bachelor of science in Data Science

Kingdom of Saudi Arabia - Umm Al-Qura University

Here's a structured outline for documenting a Diabetes model using Different models using python.

Title: Diabetes Model Documentation.

- Introduction :

- The Diabetes Dataset contains information about individuals diagnosed with diabetes, including demographic attributes, medical history, and clinical measurements.

- This dataset serves as a valuable resource for studying diabetes management, risk factors, and predictive modeling for disease outcomes.

Dataset Features:

-Preg: To express the Number of pregnancies.

• Glucose: To express the Glucose level in blood.

• BPressure: To express the Blood pressure measurement.

• SThickness: To express the thickness of the skin.

• Insulin: To express the Insulin level in blood.

• BMI: To express the Body mass index.

• DiabetesPedigreeFunction: To express the Diabetes percentage.

• Age: To express the age.

• Outcome: To express the final result 1 is YES o is NO.

A brief overview of the project and its goal is to predict the outcomes if a person has diabetes or not, using different models.

2- Problem Statement :-

- Define the problem : predicting the outcomes if a person has diabetes or not.

- Highlight the importance and potential applications of such predictions.

Import library :

import pandas as pd: Imports the pandas library for data manipulation (loading, .cleaning, analyzing)

import numpy as np: Imports the NumPy library for numerical operations (arrays, matrices, linear algebra).

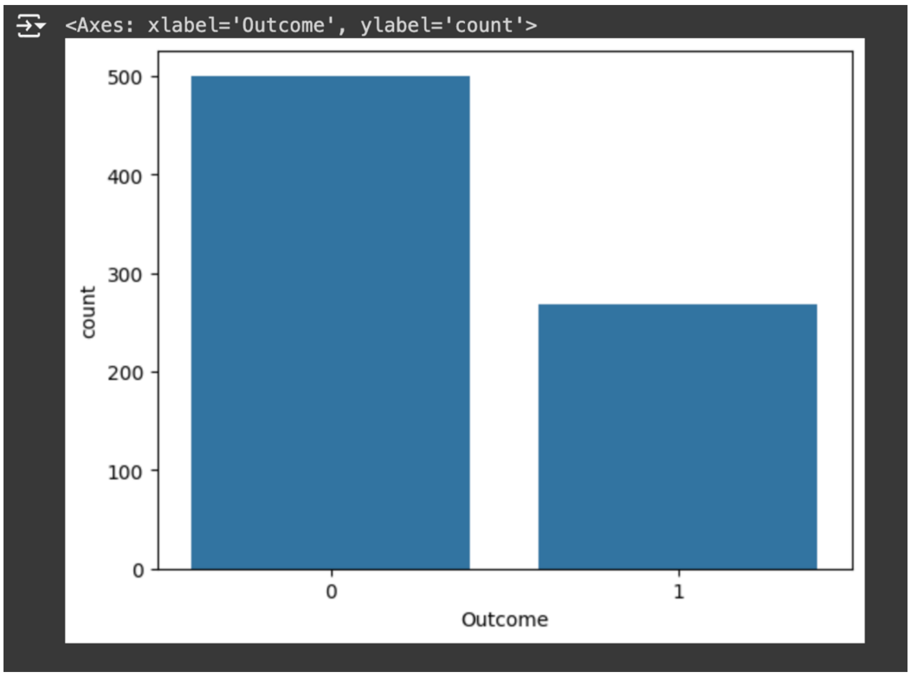
import seaborn as sns: Imports the seaborn library, built on top of matplotlib, for .advanced statistical data visualization (easier to create aesthetically pleasing plots).

Data Loading:

The train_test_split function from sklearn.model_selection is used to split a dataset into training and testing sets. This function divides the data by allocating a percentage of samples for training and the remaining samples for testing, based on the test_size parameter

from sklearn import preprocessing (commented out): Imports the preprocessing module from scikit-learn for data normalization or scaling if needed.

Data Visualization:



```
<Axes: xlabel='Outcome', ylabel='count'>
```

It appears that the chart shows the distribution of two outcomes: 0 and 1. As you mentioned, the number of individuals with an outcome of 0 (indicating they do not have diabetes) is significantly higher compared to those with an outcome of 1 (indicating they have diabetes).

Machine Learning Models:

from sklearn.linear_model import LogisticRegression: Imports the LogisticRegression class from scikit-learn for building logistic regression models, which are suitable for binary classification problems.

from sklearn.naive_bayes import GaussianNB: Imports the GaussianNB class from scikit-learn for building Gaussian Naive Bayes models, which are suitable for classification tasks assuming a Gaussian (normal) distribution for features in each class.

Model Evaluation :

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score: Imports the confusion_matrix, classification_report, and accuracy_score functions from scikit-learn for evaluating model performance. These metrics provide insights into how well the models classify different classes in the data.

from sklearn import metrics (commented out): Imports the metrics module from scikit-learn, which might contain additional evaluation metrics.

Data Splitting: The code currently lacks the train_test_split function from sklearn.model_selection to split the loaded dataset into training and testing sets. This is crucial for training the models on a portion of the data (training set) and evaluating their performance on unseen data (testing set).

Model Training: The code doesn't explicitly create and train any machine learning models using the imported classes. You'll need to instantiate these models, fit them to the training data, and make predictions on the testing data.

Evaluation and Refinement: After training, evaluate the models' performance using the imported metrics like confusion_matrix, classification_report, and accuracy_score. Based on the results, you might need to adjust hyperparameters or try different models to improve performance.

The main part of the code utilizes the pd.read_csv() function from the pandas library to read data from a CSV file named 'Diabetes Dataset_Training Part.csv'. This function takes the path to the CSV file as an argument and returns a Pandas DataFrame object, which is a tabular data structure that represents the data from the file.

Data Loading into DataFrame: The pd.read_csv() function automatically infers the data types of the columns based on the values in the file. It also handles missing values (represented by NaN or empty cells) and can handle various encoding formats.

The code data.head() is used to display the first few rows of the DataFrame data created earlier.

Certainly, **the code data.info()** provides a summary of the DataFrame data created earlier using pd.read_csv('Diabetes Dataset_Training Part.csv').

The data.describe(): method in Python is used to generate a summary of descriptive statistics for numerical columns in a Pandas DataFrame. It provides insights into the central tendency, dispersion, and distribution of the data.

The data.isnull().sum() : code in Python is used to count the number of missing values (represented by NaN or empty cells) in each column of a Pandas DataFrame. It provides a quick overview of the extent of missing data in your dataset.

The data.corr() method in Python is used to calculate the correlation matrix for the numerical columns in a Pandas DataFrame. It provides insights into the linear relationships between pairs of columns in your data.

Graph Analysis:

sns.countplot displays the outcome distribution, showing the count of diabetic vs. non-diabetic cases.

Feature Selection and Target Definition:

Model Features (x): Columns 1-7 (e.g., glucose, blood pressure).

Target (y): Indicates diabetes diagnosis.

Data Preparation:

Splitting Data: train_test_split divides data into 75% training and 25% testing sets.

Scaling: StandardScaler reduces data variability, enhancing model accuracy.

Model Building and Evaluation:

Gaussian Naive Bayes:

Training: classifier.fit applied on training data.

Accuracy: 76.04% (training), 72.92% (testing).

Logistic Regression:

Training: Training data fed to LogisticRegression.

Accuracy: 78.47% (training), 73.44% (testing), showing superior performance.

Advanced Evaluation:

Confusion Matrix: Displays correct and incorrect predictions by class.

Classification Report: Provides details on precision, recall, and F1-score for each class.

x=data.drop('Outcome',axis=1) .

This line creates a new DataFrame named x

The .drop('Outcome', axis=1) part removes a column from the DataFrame data

Outcome': This specifies the name of the column to be removed, which is assumed to ' be the target variable in your data

axis=1: This argument indicates that the removal should happen along axis 1, which corresponds to columns in a DataFrame

y=data['Outcome']

This line creates a new Series named y

It directly selects the column named "Outcome" from the DataFrame data.

x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2, random_state=0): This line splits the loaded digits dataset into training and testing sets for machine learning purposes. Here's a breakdown of the arguments

digits.data: This represents the data portion of the loaded digits dataset. It likely contains an array-like structure where each element represents an image of a handwritten digit.

digits.target: This represents the target labels (correct digits) associated with each data point in digits.data. It's likely an array-like structure containing integer values (0-9) corresponding to the correct digits in the images.

test_size=0.2: This argument specifies the proportion of data to be allocated for the testing set. Here, 20% (0.2) of the data will be used for testing, and the remaining 80% will be used for training.

random_state=0: This argument sets a seed for the random number generation used to split the data. Using a seed ensures reproducibility, meaning you'll get the same split of data into training and testing sets if you run this code multiple times with the same random_state value.

You'll have four separate variables

x_train: This will contain the training data (images of digits) used to train a machine learning model.

x_test: This will contain the testing data (images of digits) used to evaluate the performance of the trained model.

y_train: This will contain the training labels (correct digits) corresponding to the images in x_train.

y_test: This will contain the testing labels (correct digits) corresponding to the images in x_test.

The code len(data) in Python is used to calculate the length of a sequence or container object. In the context of a DataFrame, it returns the total number of rows in the DataFrame.

The code you provided imports and creates instances of seven different machine learning classification models from scikit-learn and XGBoost :

model1 = LogisticRegression(): This line creates an instance of the LogisticRegression classifier from scikit-learn. Logistic regression is a linear model commonly used for binary classification problems (predicting one of two classes).

Model2 = GaussianNB(): This line creates an instance of the GaussianNB classifier from scikit-learn. This classifier is based on the assumption that features in each class follow a Gaussian (normal) distribution. It's a simple and efficient classifier for certain types of data.

Columns List (columns): This line creates a list named columns. It contains the names of the different machine learning models you instantiated in the previous code block. These names will likely be used as labels when storing or displaying results.

This line creates an empty list named result1. This list is likely intended to store the results (performance metrics) obtained when you evaluate each of the machine learning models on your data.

Model Fitting

model.fit(x_train, y_train): This line fits the provided machine learning model (model) on the training data (x_train) and training labels (y_train). This training process allows the model to learn the patterns within the data.

Prediction:

pre = model.predict(x_test): This line uses the fitted model to predict the class labels for the unseen testing data (x_test). The predictions are stored in the variable pre.

Accuracy Calculation:

accuracy = accuracy_score(pre, y_test) * 100: This line calculates the accuracy of the model's predictions on the testing data. It uses the accuracy_score function from scikit-learn and multiplies the result by 100 to express it as a percentage. The accuracy is then stored in the variable accuracy.

Appending Accuracy:

result1.append(accuracy): This line assumes result1 is a list (defined earlier). It appends the calculated accuracy for this model to the result1 list, potentially for storing results of multiple models.

Classification Report

print(classification_report(pre, y_test)): This line prints a classification report using scikit-learn's classification_report function. The report provides detailed information about the model's performance, including precision, recall, F1-score, and support for each class.

Accuracy Printing:

print(f'Accuracy is : {round(accuracy, 2)} %'): This line prints a formatted string that displays the calculated accuracy with two decimal places.

Creating a Dictionary (finalResult):

This line creates a dictionary named finalResult

The dictionary has two key-value pairs:

Algorithm': This key likely corresponds to the list columns you defined earlier, ' containing the names of the models you evaluated (e.g., "LogisticRegression", " GaussianNB ", etc.).

- Supplementary information such as code snippets, additional analysis details, and visualizations.

Ensure that each step in the documentation provides clear explanations, examples where applicable, and relevant insights into the use of models for developing the diabetes prediction model.

.We must point out that logical registration is better than NB.