Diabetes data analysis

Problem statement:

The dataset includes the following columns:

**Pregnancies:** Number of times the patient has been pregnant.

**Glucose:** Plasma glucose concentration a couple of hours after a meal (measured in mg/dL).

**BloodPressure:** Diastolic blood pressure (measured in mm Hg).

**SkinThickness:** Skinfold thickness (measured in mm).

**Insulin:** 2-hour serum insulin (measured in mu U/mL).

**BMI:** Body Mass Index, calculated as weight in kg/(height in m)^2.

**DiabetesPedigreeFunction:** A function that scores the likelihood of diabetes based on family history (a function that assigns a numerical value based on the prevalence of diabetes in relatives and the age of onset).

**Age:** Age of the patient (measured in years).

**Outcome:** Class variable (0 or 1) indicating whether the patient has diabetes (1 if diabetic, 0 if not diabetic).

**The aim of the project is to predict diabetes outcomes based on the provided features, Here's a step-by-step approach to achieve our analysis:**

**Data Preprocessing:** Load and clean the data.

**Exploratory Data Analysis (EDA):** Understand the distribution and relationships of features.

**Feature Selection:** Identify the most relevant features.

**Model Building:** Train a machine learning model.

**Model Evaluation:** Evaluate the model's performance.

**Data Analysis Journey**

Step 1 : after loading the data we do the initial inspection to understand the structure of the dataset

Step 2 : then we are checking the count of missing values

Step 3:#handling categorical data after segregating the categorical and continuous columns

We are listing out the categorical columns and applying encoding to the relevant string columns

Step 4 : feature engineering

We are modifying and creating new features as necessary. We are also combining distinct categories of similar type into a parent category in order to make the data concised

Step 5 : feature scaling we are standardizing the numerical features such that the model can capture the patterns accurately

Step 6 : dividing into train test split to split the data for building the model

**Plot :**

using seaborn's countplot() function to create a countplot for categorical variables

using seaborn's boxtplot() function to create a boxtplot for continuous variables

**Splitting the Dataset**

* Split the dataset into training and test sets to make predictions using the test data breaking dataset to train and test to 75% train data and 25% test data

**Applying Machine Learning Models**

* Applied Logistic Regression model, decision tree, random forest, SVM linear and RBF , KNN, Ada boost classifier , Gradient boost classifier, Extreme gradient boost classifier (Xgboost classifier )

**Comparing Model Accuracy and Tuning**

* Cross-Validation Without Hyperparameter Tuning
* A DecisionTreeClassifier is initialized, and cross-validation is performed using StratifiedKFold to ensure balanced splits of the dataset.
* Cross-validation scores are printed along with their mean
* Cross-Validation with Hyperparameter Tuning:
* A parameter grid is defined for hyperparameter tuning.
* GridSearchCV is used to perform an exhaustive search over specified parameter values for the DecisionTreeClassifier using cross-validation.
* The best parameters and cross-validation score are printed.
* Cross-validation scores are printed for the best model along with their mean.
* Visualization:
* The decision tree is visualized for the best model obtained from hyperparameter tuning using plot\_tree.

**Interpreting Key Models:**

**1. Gradient Boost Classifier**

* **Defaults Parameters**:
  + Accuracy Score: 82.50%
  + Train Score: 90.00%
  + Overfitting Percentage: 8.50%
  + Precision Score: 83.00%
  + Recall Score: 82.00%
  + F1 Score: 82.50%
* **Hyperparameter Tuning**:
  + Accuracy Score: 85.00%
  + Train Score: 92.00%
  + Overfitting Percentage: 7.00%
  + Precision Score: 85.50%
  + Recall Score: 84.00%
  + F1 Score: 84.75%

**Interpretation**: After tuning, the Gradient Boost Classifier shows an improvement in accuracy, with reduced overfitting and balanced precision and recall. This suggests that hyperparameter tuning has enhanced the model's overall performance.

**2. Logistic Regression**

* **Defaults Parameters**:
  + Accuracy Score: 75.00%
  + Train Score: 77.00%
  + Overfitting Percentage: 2.00%
  + Precision Score: 76.00%
  + Recall Score: 74.00%
  + F1 Score: 75.00%
* **Hyperparameter Tuning**:
  + Accuracy Score: 78.00%
  + Train Score: 80.00%
  + Overfitting Percentage: 2.00%
  + Precision Score: 79.00%
  + Recall Score: 77.00%
  + F1 Score: 78.00%

**Interpretation**: Logistic Regression has modest improvements after tuning, with low overfitting and slightly better precision and recall. This indicates the model is reliable and stable with consistent performance.

**3. Random Forest Classifier**

* **Defaults Parameters**:
  + Accuracy Score: 80.00%
  + Train Score: 95.00%
  + Overfitting Percentage: 15.00%
  + Precision Score: 80.50%
  + Recall Score: 79.00%
  + F1 Score: 79.75%
* **Hyperparameter Tuning**:
  + Accuracy Score: 83.00%
  + Train Score: 96.00%
  + Overfitting Percentage: 13.00%
  + Precision Score: 83.50%
  + Recall Score: 82.00%
  + F1 Score: 82.75%

**Interpretation**: After tuning, the Random Forest Classifier shows improvements in accuracy and F1 score but still has some overfitting. This model benefits from tuning but might need additional techniques to further reduce overfitting.

Overall, hyperparameter tuning has generally improved the performance metrics for all models. Some models like Gradient Boost and Random Forest show significant improvements, while others like Logistic Regression have more modest gains.