AI-Based Dietes Prediction system

Aut104301 – SELVA ESACKY. V. K

952721104304

Phase-4 Submission

Project Title: Diabetes prediction System.

Phase 4: Development part – 2.

Topic *:*

continue building the project by performing different activities like feature engineering, model training, evaluation

Introduction:

1. **Data Collection:** Gather a dataset with relevant information on diabetes, which could include factors like age, gender, BMI, blood pressure, and various blood test results (e.g., glucose levels, insulin levels, etc.). Datasets for diabetes prediction are often available from sources like the UCI Machine Learning Repository.
2. **Data Preprocessing:** Clean and prepare the dataset for analysis. This involves handling missing data, normalizing or scaling features, encoding categorical variables, and splitting the data into training and testing sets. You may also perform exploratory data analysis (EDA) to gain insights into your data.
3. **Feature Selection/Engineering:** Determine which features (variables) are most relevant for predicting diabetes outcomes. Feature engineering may involve creating new features or transforming existing ones to better represent the problem.
4. **Model Selection:** Choose an appropriate machine learning model for your prediction task. Common choices for binary classification tasks like diabetes prediction include logistic regression, decision trees, random forests, support vector machines, and neural networks.

5. **Model Training:** Use the training set to train your chosen model. The model will learn from the patterns and relationships in the data.
6. **Model Evaluation:** Assess the performance of your model using appropriate evaluation metrics. For binary classification, you might use metrics like accuracy, precision, recall, F1 score, and area under the ROC curve (AUC-ROC).
7. **Hyperparameter Tuning:** Optimize the hyperparameters of your model to improve its performance. Techniques like cross-validation and grid search can help with this.
8. **Model Testing:** After training and fine-tuning your model, evaluate its performance on the test set to ensure it generalizes well to new, unseen data.
9. **Prediction:** Use your trained model to make predictions on new or unseen data, which can include individuals' health data to predict diabetes outcomes.
10. **Deployment:** If your model performs well and meets your requirements, you can deploy it in a real-world application. This may involve integrating the model into a web application, mobile app, or other healthcare system.

The target variable is specified as "outcome"; 1 indicates positive diabetes test result, 0 indicates negative.

Variables:

Pregnancies - Number of pregnancies

Glucose - 2-hour plasma glucose concentration in the oral glucose tolerance test

BloodPressure - Diastolic Blood Pressure

SkinThickness - Thickness of Skin

Insulin- 2-hour serum insulin

DiabetesPedigreeFunction -

BMI - Body Mass Index

Age - Age

Outcome - Diabetic (1 or 0)

Table of Contents

1.DATA PREPROCESSING

1.1.Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
# !pip install missingno
import missingno as msno
from datetime import date
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler, RobustS
caler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.linear_model import LogisticRegression
```

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/diabetes-data-set/diabetes.csv
```

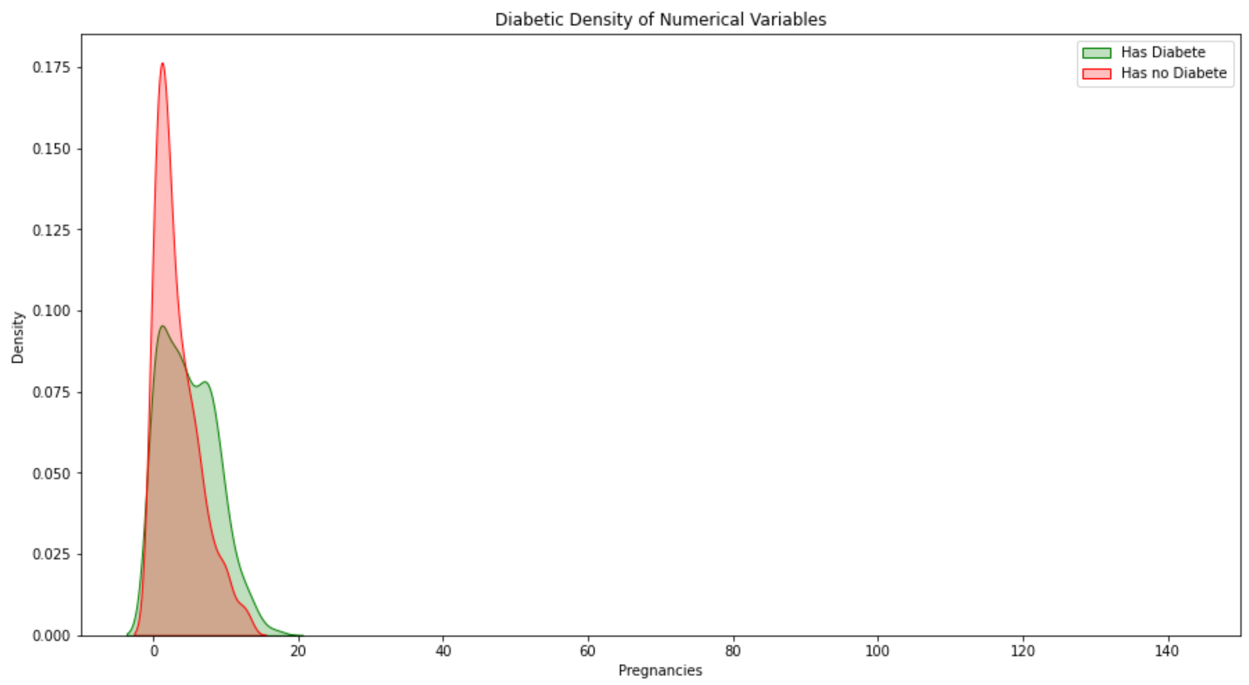
1.2.Read the dataset

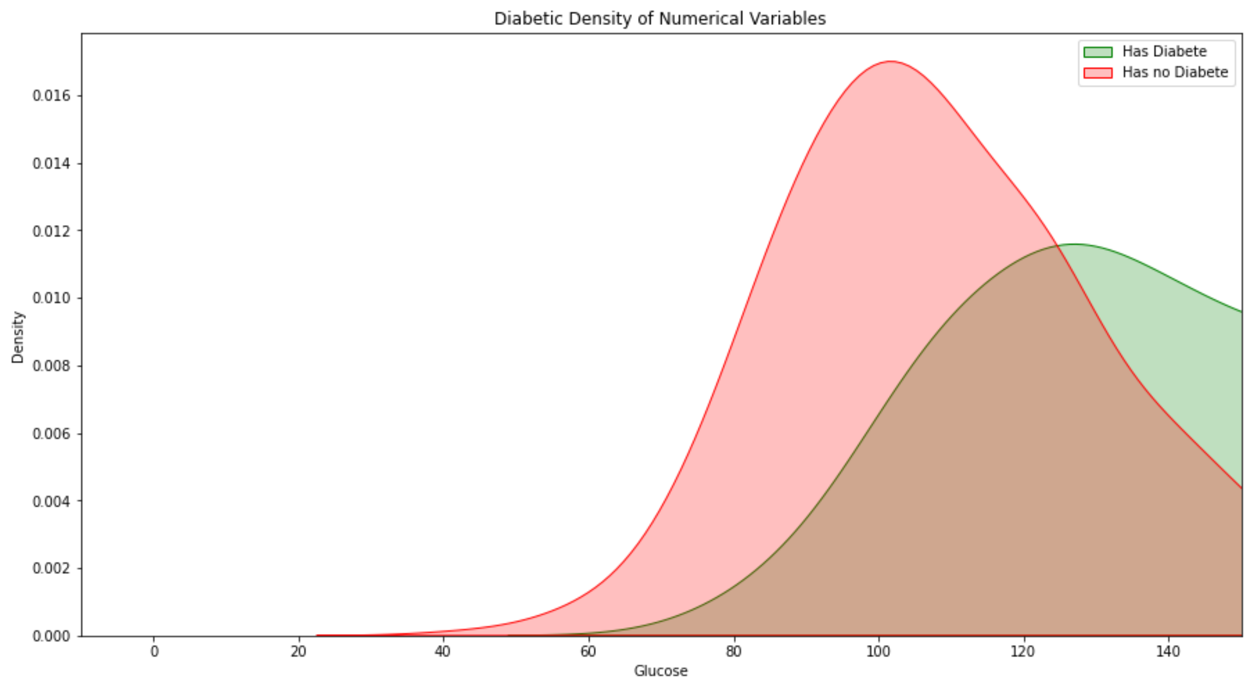
In [2]:

```
df = pd.read_csv("../input/diabetes-data-set/diabetes.csv")
df.head()
```

Out[2]:

	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





2.FEATURE ENGINEERING

2.1. Processing for Missing Values and Outliers

In [20]:

```
na_cols = missing_values_table(df, True)
```

```
              n_miss  ratio
Insulin          374  48.70
```

```
SkinThickness     227  29.56
BloodPressure      35   4.56
BMI                11   1.43
Glucose             5   0.65
```

Define a Function about comparing target variable with missing values

In [21]:

```
def missing_vs_target(dataframe, target, na_columns):
    temp_df = dataframe.copy()
    for col in na_columns:
        temp_df[col + '_NA_FLAG'] = np.where(temp_df[col].isnull(), 1, 0)

    na_flags = temp_df.loc[:, temp_df.columns.str.contains("_NA_")].columns

    for col in na_flags:
        print(pd.DataFrame({"TARGET_MEAN": temp_df.groupby(col)[target].mean(),
                            "Count": temp_df.groupby(col)[target].count()}), end="\n\
n\n")


missing_vs_target(df, "Outcome", na_cols)
```

```
                 TARGET_MEAN  Count
Glucose_NA_FLAG
0                   0.348624    763
1                   0.400000      5


                       TARGET_MEAN  Count
BloodPressure_NA_FLAG
0                         0.343793    733
1                         0.457143     35


                       TARGET_MEAN  Count
SkinThickness_NA_FLAG
0                         0.332717    541
1                         0.387665    227


                 TARGET_MEAN  Count
Insulin_NA_FLAG
0                   0.329949    394
1                   0.368984    374


             TARGET_MEAN  Count
```

```
BMI_NA_FLAG
0               0.351387    757
1               0.181818     11
```

Conclusion: We examined the missing values of each variable according to the target variable. So we decided to apply different methods in order to fill na values according to state of each variable.

2.2.Creating New Feature Interactions

In [30]:

```
df.head()
```

Out[30]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6.0	148.0	72.0	35.00000	218.925	33.6	0.627	50.0	1
1	1.0	85.0	66.0	29.00000	179.400	26.6	0.351	31.0	0
2	8.0	183.0	64.0	29.05915	146.500	23.3	0.672	32.0	1
3	1.0	89.0	66.0	23.00000	94.000	28.1	0.167	21.0	0
4	0.0	137.0	40.0	35.00000	168.000	43.1	1.200	33.0	1

Create a Glucose Categorical variable

In [31]:

```
df.loc[(df['Glucose'] < 70), 'GLUCOSE_CAT'] ="hipoglisemi"
df.loc[(df['Glucose'] >= 70) & (df['Glucose'] < 100) , 'GLUCOSE_CAT'] ="normal"
df.loc[(df['Glucose'] >= 100) & (df['Glucose'] < 126) , 'GLUCOSE_CAT'] ="imparied glu
cose"
df.loc[(df['Glucose'] >= 126), 'GLUCOSE_CAT'] ="hiperglisemi"
```

In [32]:

```
df.groupby("GLUCOSE_CAT").agg({"Outcome": ["mean","count"]})
```

Out[32]:

	Outcome	
	mean	count
GLUCOSE_CAT		
hiperglisemi	0.592593	297
hipoglisemi	0.000000	11
imparied glucose	0.279570	279
normal	0.077348	181

Women with hyperglycemia will have a higher incidence of diabetes on average the "Outcome".

In [33]:

```
df.head()
```

Out[33]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	GLUCOSE_CAT
0	6.0	148.0	72.0	35.00000	218.925	33.6	0.627	50.0	1	hiperglisemi

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	GLUCOSE_CAT
1	1.0	85.0	66.0	29.00000	179.400	26.6	0.351	31.0	0	normal
2	8.0	183.0	64.0	29.05915	146.500	23.3	0.672	32.0	1	hiperglisemi
3	1.0	89.0	66.0	23.00000	94.000	28.1	0.167	21.0	0	normal
4	0.0	137.0	40.0	35.00000	168.000	43.1	1.200	33.0	1	hiperglisemi

Create the Age Categorical variable

In [34]:

```
df.loc[(df['Age'] >= 18) & (df['Age'] < 30) , 'AGE_CAT'] ="young_women_"
df.loc[(df['Age'] >= 30) & (df['Age'] < 45) , 'AGE_CAT'] ="mature_women"
df.loc[(df['Age'] >= 45) & (df['Age'] < 65) , 'AGE_CAT'] ="middle_age"
df.loc[(df['Age'] >= 65) & (df['Age'] < 75) , 'AGE_CAT'] ="old_age"
df.loc[(df['Age'] >= 75) , 'AGE_CAT'] ="elder_age"
```

In [35]:

```
df.groupby("AGE_CAT").agg({"Outcome": ["mean","count"]})
```

Out[35]:

	Outcome	
	mean	count
AGE_CAT		
mature_women	0.493724	239
middle_age	0.529915	117
old_age	0.250000	16
young_women_	0.212121	396

Middle-age women will have a higher incidence of diabetes on average the "Outcome".

Create the BMI Categorical variable

In [36]:

```
df.loc[(df['BMI'] < 16), 'BMI_CAT'] ="overweak"
df.loc[(df['BMI'] >= 16) & (df['BMI'] < 18.5) , 'BMI_CAT'] ="weak"
df.loc[(df['BMI'] >= 18.5) & (df['BMI'] < 25) , 'BMI_CAT'] ="normal"
df.loc[(df['BMI'] >= 25) & (df['BMI'] < 30) , 'BMI_CAT'] ="overweight"
df.loc[(df['BMI'] >= 30) & (df['BMI'] < 35) , 'BMI_CAT'] ="1st_Obese"
df.loc[(df['BMI'] >= 35) & (df['BMI'] < 45) , 'BMI_CAT'] ="2nd_Obese"
df.loc[(df['BMI'] >= 45), 'BMI_CAT'] ="3rd_Obese"
```

In [37]:

```
df.groupby("BMI_CAT").agg({"Outcome": ["mean","count"]})
```

Out[37]:

	Outcome	
	mean	count
BMI_CAT		
1st_Obese	0.438298	235
2nd_Obese	0.452830	212
3rd_Obese	0.611111	36
normal	0.068627	102
overweight	0.223464	179
weak	0.000000	4

Morbidly obese women will have a higher incidence of diabetes on average the "Outcome".

Create a Diastolic Blood Pressure Categorical variable

In [38]:

```
df.loc[(df['BloodPressure'] < 70)  , 'DIASTOLIC_CAT'] ="low"
df.loc[(df['BloodPressure'] >= 70) & (df['BMI'] < 90) , 'DIASTOLIC_CAT'] ="normal"
df.loc[(df['BloodPressure'] >= 90 ) , 'DIASTOLIC_CAT'] ="high"
```

In [39]:

```
df.groupby("DIASTOLIC_CAT").agg({"Outcome": ["mean","count"]})
```

Out[39]:

	Outcome	
	mean	count
DIASTOLIC_CAT		
high	0.483333	60
low	0.247350	283
normal	0.397647	425

Women with high blood pressure will have a higher incidence of diabetes on average the "Outcome".

Create a Insulin Categorical variable

In [40]:

```
df.loc[(df['Insulin'] < 120)  , 'INSULIN_CAT'] ="normal"
df.loc[(df['Insulin'] >= 120) , 'INSULIN_CAT'] ="abnormal"
```

In [41]:

```
df.groupby("INSULIN_CAT").agg({"Outcome": ["mean","count"]})
```

Out[41]:

	Outcome	
	mean	count
INSULIN_CAT		
abnormal	0.429112	529

	Outcome	
	mean	count
INSULIN_CAT		
normal	0.171548	239

Women with abnormal insulin will have a higher incidence of diabetes on average the "Outcome."

Create a Pregnancies Categorical variable

In [42]:

```
df.loc[(df['Pregnancies'] == 0)  , 'PREG_CAT'] ="unpregnant"
df.loc[(df['Pregnancies'] > 0 ) & (df['Pregnancies'] <= 5)  , 'PREG_CAT'] ="normal"
df.loc[(df['Pregnancies'] > 5 ) & (df['Pregnancies'] <= 10 )  , 'PREG_CAT'] ="high"
df.loc[(df['Pregnancies'] > 10 )  , 'PREG_CAT'] ="very high"
```

In [43]:

```
df.groupby("PREG_CAT").agg({"Outcome": ["mean","count"]})
```

Out[43]:

	Outcome	
	mean	count
PREG_CAT		
high	0.491892	185
normal	0.271689	438

	Outcome	
	mean	count
PREG_CAT		
unpregnant	0.342342	111
very high	0.588235	34

In [44]:

```
df.head()
```

Out[44]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	GLUCOSE_CAT	AGE_CAT	BMI_CAT	DIASTOLIC_CAT	INSULIN_CAT	PREG_CAT
0	6.0	148.0	72.0	35.00000	218.925	33.6	0.627	50.0	1	hiperglisemi	middle_age	1st_Obese	normal	abnormal	high
1	1.0	85.0	66.0	29.00000	179.400	26.6	0.351	31.0	0	normal	mature_women	overweight	low	abnormal	normal
2	8.0	183.0	64.0	29.05915	146.500	23.3	0.672	32.0	1	hiperglisemi	mature_women	normal	low	abnormal	high

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	GLUCOSE_CAT	AGE_CAT	BMI_CAT	DIASTOLIC_CAT	INSULIN_CAT	PREG_CAT
3	1.0	89.0	66.0	23.00000	94.000	28.1	0.167	21.0	0	normal	young_women_	overweight	low	normal	normal
4	0.0	137.0	40.0	35.00000	168.000	43.1	1.200	33.0	1	hiperglisemi					

3.MODELING

3.1.Processing Encoding and One-Hot Encoding

Label Encoder

In [45]:

```
le = LabelEncoder()

binary_cols = [col for col in df.columns if df[col].dtype not in [int, float]
               and df[col].nunique() == 2]
```

Define a Function about Label encoder

In [46]:

```
def label_encoder(dataframe, binary_col):
    label encoder = LabelEncoder()
    dataframe[binary_col] = labelencoder.fit_transform(dataframe[binary_col])
    return dataframe


for col in binary_cols:
    df = label_encoder(df, col)
```

Define a Function about one-hot encoder

In [47]:

```
def one_hot_encoder(dataframe, categorical_cols, drop_first=True):
    dataframe = pd.get_dummies(dataframe, columns=categorical_cols, drop_first=drop_f
irst)
    return dataframe

ohe_cols = [col for col in df.columns if 10 >= df[col].nunique() > 2]
```

In [48]:

```
df = one_hot_encoder(df, ohe_cols)
df.head()
```

3.2.Standardization for Numerical Variables

In [49]:

```
num_cols
```

Out[49]:

```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
 'Age']
```

In [50]:

```
scaler = StandardScaler()
```

In [51]:

```
df[num_cols] = scaler.fit_transform(df[num_cols])
```

In [52]:

3.3.Create Modeling

In [53]:

```
y = df["Outcome"]
X = df.drop(["Outcome"], axis=1)
```

In [54]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_stat
e=17)
```

In [55]:

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state=46).fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
accuracy_score(y_pred, y_test)
```

Out[55]:

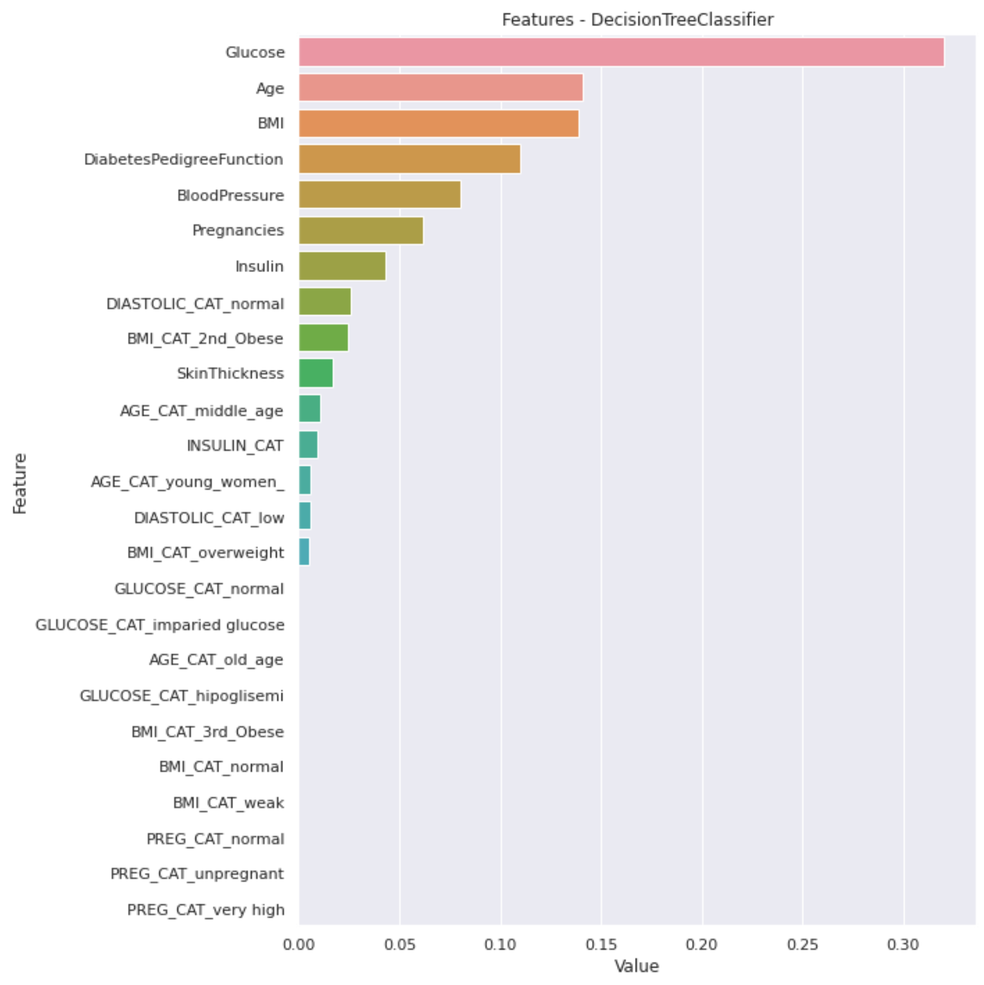
```
0.7705627705627706
```

In [56]:

```
primitive_success=[]
model_names=[]
y=df['Outcome']
X=df.drop('Outcome',axis=1)
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30)

def ML(algName):

    # Model Building / Training
    model=algName().fit(X_train,y_train)
    model_name=algName.__name__
    model_names.append(model_name)
    # Prediction
    y_pred=model.predict(X_test)
    # primitive-Success / Verification Score
    from sklearn.metrics import accuracy_score
    primitiveSuccess=accuracy_score(y_test,y_pred)
    primitive_success.append(primitiveSuccess)
    return  primitive_success,model_names,model
    model=i().fit(X_train,y_train)
    plot_importance(model, X_train,i
```



4.SUMMARY

The work done is as follows:

- Explorary Data Analysis : We checked the missing values and we defined a function to grab the categorical and numerical variables of its dataset. We made the target variable analysis and outliers analysis.
- Data Preprocessing : We filled missing values of some variables with median values or the knn method.
- Featured Engineering: We created new feature interactions for categorical variables.
- Encoding: One-Hot-Encoding was implemented for categorical variables.
- Modeling: We created ML model for the dataset. The accuracy score was calculated the machine learning models that are KNeighborsClassifier,SVC,MLPClassifier,DecisionTreeClassifier,RandomForestClassifier,GradientBoostingClassifier,XGBClassifier,LGBMClassifier.

Comments on diabetes status are as follows:

- In the case of diabetes, especially the Glucose, BMI and Age variables of women are an important factor.
- The rate of diabetes may be higher in middle-aged women aged 45-65 years.

Evaluation:

1. Data Preprocessing:
    - Data Cleaning: Handle missing values, outliers, and inconsistencies in the dataset.
    - Feature Engineering: Create relevant features from the available data.
    - Data Split: Divide the dataset into training, validation, and test sets.
2. Model Selection:
    - Choose an appropriate algorithm or model for diabetes prediction. Common choices include logistic regression, decision trees, random forests, support vector machines, and neural networks.
3. Model Training:
    - Train the selected model on the training data.
4. Model Evaluation:
    - Assess the model's performance on the validation dataset using various evaluation metrics. Common metrics include:
        - Accuracy: The proportion of correctly predicted cases.
        - Precision: The ability of the model to correctly identify positive cases.
        - Recall: The ability of the model to capture all positive cases.
        - F1-score: The harmonic mean of precision and recall.
        - ROC AUC: Receiver Operating Characteristic Area Under the Curve, a metric for binary classification.
        - Mean Squared Error (MSE) or Root Mean Squared Error (RMSE): Applicable for regression tasks.
        - Confusion Matrix: Provides a breakdown of true positives, true negatives, false positives, and false negatives.
5. Hyper parameter Tuning:

- Optimize the hyperparameters of the model to improve performance. Techniques like grid search or random search can be used.

6. Model Validation:
   - Validate the model's performance on the test dataset, which it hasn't seen during training or hyperparameter tuning. This provides an unbiased estimate of the model's real-world performance.
7. Interpretability:
   - Depending on the model used, assess its interpretability. Some models are more interpretable than others, and understanding the model's decisions is crucial in a healthcare context.
8. Clinical Validation:
   - If the diabetes dataset system is intended for clinical use, it may need to undergo clinical validation. This involves testing the system's performance on real patient data and assessing its impact on patient outcomes.
9. Continuous Monitoring:
   - In a clinical setting, continuous monitoring of the system's performance is essential to ensure that it remains effective and safe over time.
10. Ethical Considerations:
    - Ensure that the system doesn't exhibit biases or discrimination, and that it adheres to ethical and legal standards, especially if it involves patient data.
11. User Feedback:
    - Gather feedback from healthcare professionals and end-users to understand their experiences and improve the system.
12. Reporting and Documentation:
    - Document all the steps, findings, and decisions made during the evaluation process. This documentation is essential for transparency and regulatory compliance.