AI-Based Diabetes Prediction system

au952721104004-E.Arun Kumar.

Phase-5 Submission

Project Title: **Diabetes prediction System.**

Introduction to Diabetes Prediction System:

Diabetes is a chronic metabolic disorder characterized by high levels of blood sugar (glucose) due to the body's inability to produce or effectively utilize insulin. It is a global

health concern with a significant impact on the quality of life and healthcare costs. Early detection and effective management of diabetes are crucial in preventing complications and improving patient outcomes. To address this, Diabetes Prediction Systems have emerged as a vital tool in the field of healthcare.

A Diabetes Prediction System is a technological solution that leverages various data sources, advanced analytics, and machine learning algorithms to predict the risk of an individual developing diabetes. These systems play a pivotal role in identifying individuals at risk, enabling timely interventions, and promoting healthier lifestyles. They are widely used in clinical settings, as well as in public health initiatives.

Key components of a Diabetes Prediction System include:

1. **Data Collection**: These systems collect a wide range of data, including personal information, medical history, lifestyle factors (such as diet and physical activity), and clinical measurements (e.g., blood glucose levels, body mass index, and blood pressure). They can also incorporate data from wearable devices and electronic health records.
2. **Data Preprocessing**: Raw data is preprocessed to clean and transform it into a suitable format for analysis. This may involve data cleaning, normalization, and feature engineering to extract relevant information.
3. **Feature Selection**: Identifying the most relevant features (variables) that have the most impact on diabetes prediction is essential. Machine learning algorithms are often used to automatically select or rank features.
4. **Machine Learning Algorithms**: Various machine learning models, such as logistic regression, decision trees, support vector machines, and neural networks, are applied to analyze the data and make predictions. These models learn from historical data and use it to predict an individual's diabetes risk.
5. **Model Training**: The selected machine learning algorithms are trained on historical datasets, often involving thousands of patient records, to learn patterns and relationships within the data.
6. **Model Evaluation**: The performance of the prediction model is assessed using metrics like accuracy, sensitivity, specificity, and area under the receiver operating characteristic curve (AUC-ROC). Cross-validation techniques are commonly used to ensure the model's generalizability.
7. **Risk Assessment**: Once the model is trained and evaluated, it can be used to assess an individual's risk of developing diabetes based on their personal data and medical history.

8. **Interventions and Recommendations**: Diabetes Prediction Systems often provide recommendations and interventions for individuals at risk. These recommendations may include lifestyle modifications, regular health check-ups, or referrals to healthcare professionals for further evaluation and treatment.
9. **Continuous Monitoring**: Some systems incorporate continuous monitoring to track changes in an individual's health status and update their risk assessment over time.

Here a list of tools and software commonly used in the process:

1. Machine Learning Algorithms: AI-based systems use a variety of machine learning algorithms to analyze data and make predictions. Common algorithms include logistic regression, decision trees, random forests, support vector machines, and neural networks.
2. Electronic Health Records (EHRs): AI can analyze patient electronic health records to identify risk factors and patterns associated with diabetes. This data may include blood glucose levels, family medical history, BMI, and other health-related information.
3. Wearable Devices and Sensors: Wearable devices such as fitness trackers and continuous glucose monitors can provide real-time data that can be used in AI-based prediction systems. These devices track factors like physical activity, sleep, and glucose levels.
4. Genetic Data: Genetic information can be used to assess an individual's susceptibility to diabetes. AI can analyze genetic markers and gene expression data to predict the risk of developing the disease.
5. Mobile Apps: Mobile applications can collect user-generated data, including dietary habits, exercise routines, and glucose monitoring, which can be analyzed by AI to predict diabetes risk.
6. Image Analysis: AI can analyze medical images, such as retinal images and skin images, to detect signs of diabetes-related complications, like diabetic retinopathy and skin disorders. Early detection of these complications can indicate a higher risk of diabetes.
7. Natural Language Processing (NLP): NLP techniques can be used to extract valuable information from unstructured clinical notes, medical reports, and patient communication. This textual data can help in refining diabetes risk predictions.

8. Data Integration: AI systems often combine multiple data sources to create a more comprehensive picture of an individual's health. The integration of diverse data types can lead to more accurate predictions.
9. Risk Scoring Models: AI models can assign a risk score to individuals based on their unique health data. This score can help healthcare professionals prioritize interventions for those at higher risk of developing diabetes.
10. Telemedicine and Remote Monitoring: AI can support telemedicine and remote monitoring solutions, enabling healthcare providers to track patients' health in real-time and intervene when necessary.
11. Feedback and Alerts: AI-based systems can provide personalized feedback and alerts to individuals, encouraging them to make healthier choices, monitor their health, and seek medical attention when needed.

1. Definition and Design Thinking

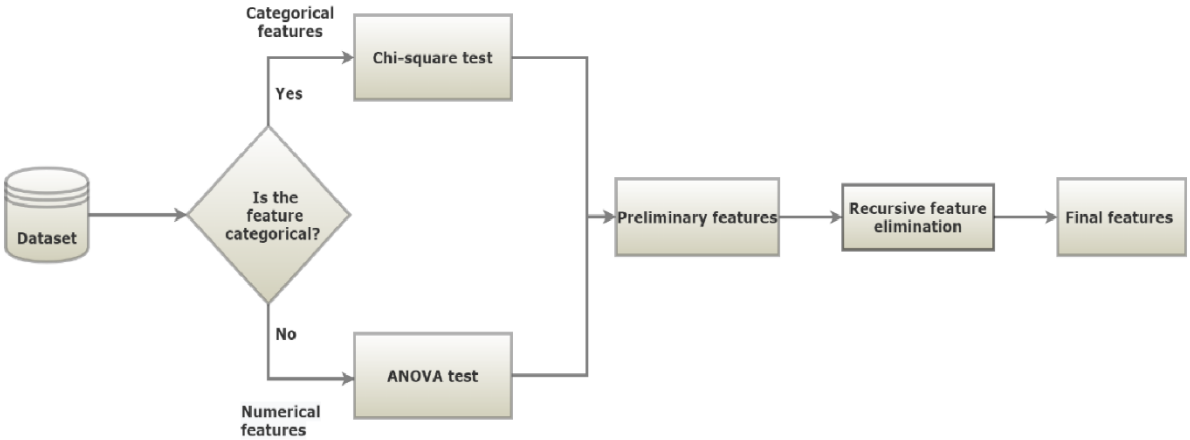
Key features of the AI-Based Diabetes Prediction System include:

1. **Data Integration:**

The system integrates electronic health records, wearable device data, and genetic information to create a comprehensive patient profile.

2. **Machine Learning Models:**

Utilizes advanced machine learning techniques, such as deep neural networks and ensemble methods, to analyze and extract patterns from diverse datasets.



3. **Risk Assessment:**

Generates personalized risk scores and identifies high-risk individuals who may benefit from early intervention and lifestyle modifications.

4. **Real-time Monitoring**:
   Allows for continuous monitoring of patients, enabling timely adjustments to treatment plans as needed.

5. **Interpretability:**

Provides healthcare professionals with explanations and visualizations of the factors contributing to risk, aiding in clinical decision-making.

6.Privacy and Security:

Adheres to strict privacy and security protocols to protect patient data and ensure compliance with healthcare regulations.

The AI-Based Diabetes Prediction System aims to empower healthcare providers with a proactive tool for diabetes prevention and management. By identifying at-risk individuals before the onset of symptoms, it can lead to early interventions, lifestyle modifications, and targeted healthcare resources, ultimately reducing the burden of diabetes on individuals and healthcare systems. This abstract provides a glimpse into the potential of AI in revolutionizing diabetes care and underscores the importance of leveraging technology for proactive healthcare solutions.

2. Innovation

1. Problem Definition:
Clearly define the problem you want to address, such as predicting blood glucose levels based on various features like age, BMI, diet, and physical activity.
2. Data Collection:
Gather a dataset that contains relevant features (independent variables) and the target variable (blood glucose levels). Data can be collected from sources like medical records, health sensors, or surveys.
3. Data Preprocessing:
Clean and preprocess the dataset to make it suitable for regression modeling. This may involve handling missing values, outlier detection and treatment, and feature scaling.
4. Feature Selection and Engineering:

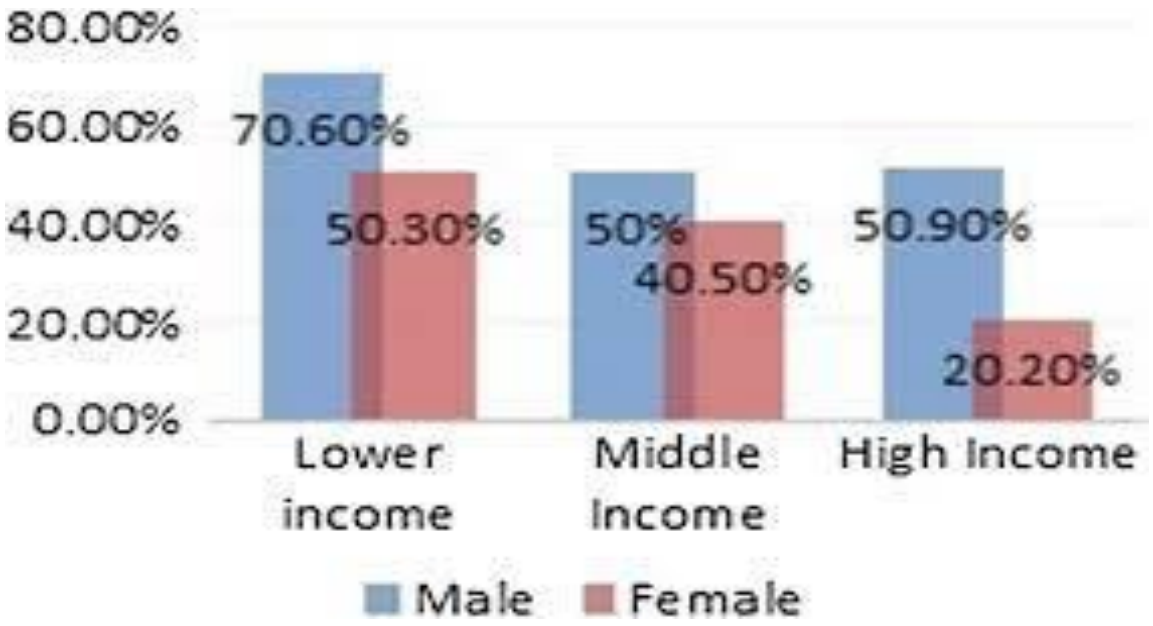
Select and engineer features that are relevant to the prediction task. You might also create new features, like interactions between variables or aggregations.

5. Data Split:

Divide the dataset into training, validation, and test sets. Common splits are 70-80% for training, 10-15% for validation, and 10-15% for testing.

6. Select Regression Models:

Choose one or more regression models suitable for the task. Common regression techniques include Linear Regression, Lasso Regression, Ridge Regression, Decision Trees, Random Forests, and Support Vector Regression.



Program:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
[1]:
model_nb = GaussianNB(var_smoothing=0.001)
model_lr = LogisticRegression()
model_dt = DecisionTreeClassifier(criterion='entropy', max_depth=3)
model_svc = SVC(kernel="linear")
[2]:
classifiers = [model_nb, model_lr, model_dt, model_svc]
[3]:
results = get_results(X_train_mm_scaled, y_train, X_test_mm_scaled, y_test, classifier
```

[4]:

Models		Accuracy	Precision	Recall	F1
0	Gaussian NB	0.716	0.597	0.561	0.578
1	Logistic Regression	0.753	0.667	0.576	0.618
2	Decision TreeClassifier	0.753	0.661	0.591	0.624
3	SVC	0.763	0.678	0.606	0.640

```
feature_names=X_test.columns)
```

7. Model Training:
Train the chosen regression model(s) on the training data. The model will learn the relationships between the features and the target variable (blood glucose levels).
8. Model Evaluation:
Assess the model's performance using appropriate regression metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) on the validation dataset.
9. Hyperparameter Tuning:
Fine-tune the model's hyperparameters to optimize its performance using techniques like grid search, random search, or Bayesian optimization.
10. Model Testing:
Evaluate the final model on the test dataset to ensure it generalizes well to new, unseen data.
11. Interpretability:
Ensure that the model's predictions are interpretable. This can be crucial in a healthcare context to understand the factors contributing to blood glucose predictions.
12.Deployment:
Deploy the regression model in a production environment. This could involve creating an API for real-time predictions or incorporating the model into a healthcare system.
13. Monitoring and Maintenance:
Continuously monitor the model's performance in the real world, and retrain it periodically with new data to maintain its accuracy

3. Development Part 1

Table of Contents

6. Modeling

1. First look to the Dataset

Business Problem: It is desired to develop a machine learning model that can predict whether people have diabetes when their characteristics are specified. You are expected to perform the necessary data analysis and feature engineering steps before developing the model.

Story of Dataset: The dataset is part of the large dataset held at the National Institutes of DiabetesDigestive-Kidney Diseases in the USA. Data used for diabetes research on Pima Indian women aged 21 and over living in Phoenix, the 5th largest city of the State of Arizona in the USA.

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

- **Pregnancies:** The number of pregnancies
- **Glucose:** 2-hour plasma glucose concentration in the oral glucose tolerance test
- **Blood Pressure:** Blood Pressure (Small blood pressure) (mmHg)
- **SkinThickness:** Skin Thickness
- **Insulin:** 2-hour serum insulin (mu U/ml)
- **DiabetesPedigreeFunction:** A function that calculates the probability of having diabetes according to the descendants
- **BMI:** Body mass index
  Age: Age (year)
- **Outcome:** Have the disease (1) or not (0)

2. EDA (Exploratory Data Analysis)

This is the part of the explore the dataset. The outliers and missing value analyses will be done in this part but the only point is to observe these in this part. The operation and editing for these will be done in Part 3: Pre-processing.

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

In [2]:

```
# Adjustment of visibility of Datafreames
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.set_option('display.width', 500)
```

In [3]:

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

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.600	0.627	50	1
1	1	85	66	29	0	26.600	0.351	31	0
2	8	183	64	0	0	23.300	0.672	32	1
3	1	89	66	23	94	28.100	0.167	21	0
4	0	137	40	35	168	43.100	2.288		

2.1 Analyse outliers and missing values

Now, check the missing values and outliers. The only process in EDA is to analyze these values. Editing of missing values and outliers will be done in the next part.

Before analyzing, firstly, an explanation of these terms will be done, the codes will be placed right after each explanation.

Outliers

Values that deviate considerably from the general trend in the data are called outliers. Especially in linear problems, the effects of outliers are more severe. They have less impact on tree methods, but still, need to be considered.

How are outliers determined?: The critical point is to determine the acceptable threshold value, which are up limit and low limit. After determining the threshold value, outliers are caught based on these values. Methods by which we can catch the threshold value:

- Industry knowledge
- Standard deviation approach
- Z-score approximation
- **Boxplot(interquartile range-IQR) method (as univariate)**

  LOF Method => Multivariate

We will emphasize and examine Boxplot(IQR Method) and LOF Method to analyse. After getting outliers, they may be deleted, or reassigned with thresholds(pressing outliers values to thresholds). These solution approaches will be in the next part.

In [12]:

```
df.describe().T
```

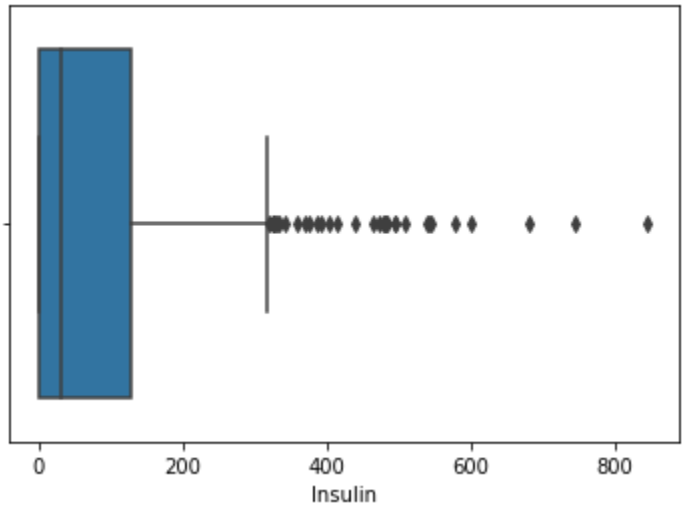
Out[12]:

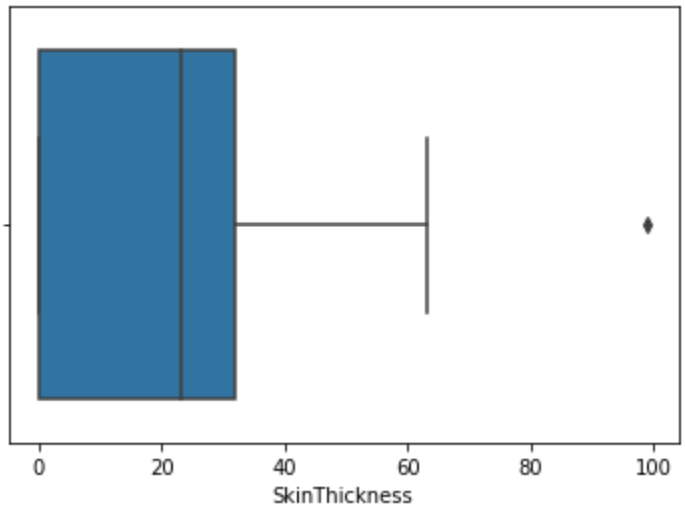
	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.000	3.845	3.370	0.000	1.000	3.000	6.000	17.000
Glucose	768.000	120.895	31.973	0.000	99.000	117.000	140.250	199.000
BloodPressure	768.000	69.105	19.356	0.000	62.000	72.000	80.000	122.000
SkinThickness	768.000	20.536	15.952	0.000	0.000	23.000	32.000	99.000
Insulin	768.000	79.799	115.244	0.000	0.000	30.500	127.250	846.000
BMI	768.000	31.993	7.884	0.000	27.300	32.000	36.600	67.100
DiabetesPedigreeFunction	768.000	0.472	0.331	0.078	0.244	0.372	0.626	2.420
Age	768.000	33.241	11.760	21.000	24.000	29.000	41.000	81.000

Outcome	768.000	0.349	0.477	0.000	0.000	0.000	1.000	1.000

```
# Boxplot
sns.boxplot(x=df["Insulin"])
plt.show()

sns.boxplot(x=df["SkinThickness"])
plt.show()
plt.show()
```





3. Pre-processing (Solve Outliers and Missing Values)

In part 2, the analysis has been completed. Examination of thresholds for outliers and how to reach missing values have been done. The next step is to interfere with these and prepare the dataset. This part will be separated into 2, like part 2. Firstly, the outliers problem will be examined, then, missing values will be processed.

Our threshold values can be affected after the operations on the missing value. Also, an outlier method, LOF, cannot be used when the dataset includes NaN values. Therefore, the missing value part will be taken firstly on preprocessing.

Missing Values

It mentioned on analyze part that there is 3 solution to missing values.

Deleting: It means dropping the rows that include missing values. Especially in a small dataset, it creates a loss of information. For example, the dataset has 768 rows, but Insuline has 374 missing values. If missing values are dropped, half of the dataset has been lost, and the information will be lost, as well. If the dataset would be large, and there are a couple of missing values that can be sacrificed, deleting can be an option.

In [31]:

```
df_new = df.copy() # copy dataset to see effect without damage the main
datas et df_new.shape
```

Out[31]:

```
(768, 9)
```

In [32]:

```
df_new.dropna(inplace=True)
df_new.shape
```

Out[32]:

```
(392, 9)
```

- **Value Assignment Methods**: We can fill the NaN values with the column's mean, median, mode, etc. If the distribution is homogeneous, filling with median or mean is logical. Also, in some scenarios, filling with a specific number like 0 would make sense.

In [33]:

```
df_fill = df.apply(lambda x: x.fillna(x.median()) if x.dtype != "O" else x, a
xis=0)
df_fill.head(10) # after filling
```

Out[33]:

	Pregnanci es	Glucos e	BloodPressu re	SkinThickne ss	Insulin	BMI	DiabetesPedigreeFunc tion	Age	Outcom e
0	6	148.000	72.000	35.000	125.000	33.600	0.627	50.000	1
1	1	85.000	66.000	29.000	125.000	26.600	0.351	31.000	0

2	8	183.000	64.000	29.000	125.000	23.300	0.672	32.000	1

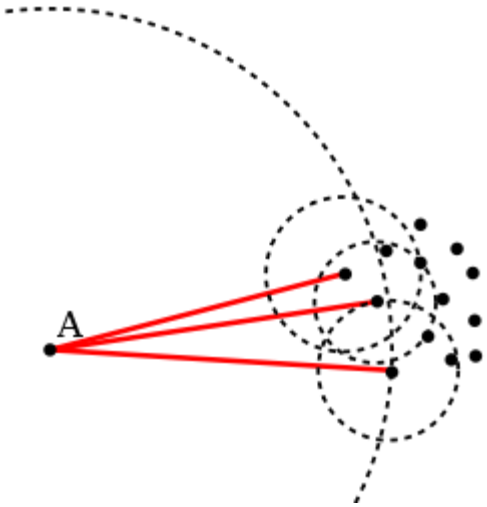
	Pregnanci es	Glucos e	BloodPressu re	SkinThickne ss	Insulin	BMI	DiabetesPedigree	Func tion	Age	Outcom e
3	1	89.000	66.000	23.000	94.000	28.100	0.167	21.000		0
4	0	137.000	40.000	35.000	168.000	43.100	2.288	33.000		1
5	5	116.000	74.000	29.000	125.000	25.600	0.201	30.000		0
6	3	78.000	50.000	32.000	88.000	31.000	0.248	26.000		1
7	10	115.000	72.000	29.000	125.000	35.300	0.134	29.000		0
8	2	197.000	70.000	45.000	543.000	30.500	0.158	53.000		1
9	8	125.00	96.000	29.000	125.00	32.30	0.232	0	0	0

Outliers

We mentioned on analyze part that there are 2 ways to solve this problem: IQR Method for the find outliers on univariate, and LOF Method for getting multivariate outliers.

Firstly, LOF Method will be explained because it controls the variables' meaningfulness to each other and gives a few outliers. Then, IQR Method will be implemented.

- **Local Outlier Factor (LOF):** Another approach is the Local Outlier Factor. It helps us to define outliers accordingly by ordering the observations based on the density at their location. The local density of a point means the neighborhoods around that point. If a point is significantly less dense than its neighbors, then that point is in a moresparse region, so there



4. Feature Extraction

When a data set is prepared, not only existing variables are tried to be edited, but also new, meaningful variables have to be created. New columns sometimes can be created with mathematical operations, sometimes named a numerical value to categorical, or categorical values' ranges, etc. This process is known as **Feature Engineering**, and this is one of the critical parts of data preparation.

In [56]:

```
# pregrancy cannot be float, it occurs due to calculation
of IQR df["Pregnancies"] = df["Pregnancies"].apply(lambda
x: int(x))
```

In [57]:

```
# create categorical columns from numerical columns


# if bins are 0, 3, 6 => 0 values become NaN due to bins
df["NumOfPreg"] = pd.cut(df["Pregnancies"], bins=[-1, 3,
6, df["Pregnancies"]
.max()], labels=["Normal", "Above Normal","Extreme"])
df["AgeGroup"] = pd.cut(df["Age"], bins=[18, 25, 40,
df["Age"].max()], labels
=["Young", "Mature", "Old"]) df["GlucoseGroup"] =
pd.qcut(df["Glucose"], 3, labels=["Low", "Medium", "High
"])
df["Patient"] = np.where(df["Outcome"] == 1, "Yes", "No")
```

In [58]:

```
# example of mathematical expression
```

```
"""Assume there is a variable named "BMIns", and it can
be found with the mul tiplication of BMI and Insuline.
Create and add it to data frame"""

df["BMIns"] = df["BMI"]*df["Insulin"] # numerical
df["BMInsGroup"] = pd.qcut(df["BMIns"], 3,
labels=["Low", "Medium", "High"])
# categorical
```

In [59]:

```
df.head()
```

Out[59]:

6	148.000	72.000	35.000	125.000	33.600	0.627	50.000	1	Old
1	85.000	66.000	29.000	125.000	26.600	0.351	31.000	0	Mature
8	183.000	64.000	29.000	125.000	23.300	0.672	32.000	1	Mature
1	89.000	66.000	23.000	114.500	28.100	0.167	21.000	0	Young
0	137.000	40.000	35.000	134.500	43.100	1.200			

5. Encoding & Scaling

Encoding

Changing the representation of variables. There are different types of encoding: If there are 2 classes, it is called **"Binary Encoding"**, if there are more than 2 classes, it is called **"Label Encoding"**.

It converts in alphabetical order, giving 0 to the first one it sees.

In [60]:

```
df_enc = df.copy()

le = LabelEncoder()
le.fit_transform(df_enc["Patient"])[0:5]
```

Out[60]:

```
array([1, 0, 1, 0, 1])
```

In [61]:

```
# let's say we forgot which 0 and which 1,
inverse_transform is used to detec t this
le.inverse_transform([0, 1])
```

Out[61]:

```
array(['No', 'Yes'], dtype=object)
```

In [62]:

```
# detect variables that have 2 unique numbers for Binary
Encoding binary_cols = [col for col in df.columns if
df[col].dtype not in [int, float]                 and
df[col].nunique() == 2] binary_cols
```

Out[62]:

```
['Patient']
```

6. Model

It is not within the scope of this topic, we are only doing it to see the model establishment while preparing the data.

Patient carries the same information with Outcome. It was created as an example of feature engineering. Also, BMIns is not a real feature, it was another example of feature engineering using mathematical expression. Therefore, these 2 variables will be dropped.

KNN algorithm will be used.

In [72]:

```
y = df["Outcome"]
X = df.drop(["Outcome", "Patient", "BMIns",
"BMInsGroup_Medium", "BMInsGroup_
High"], axis=1)


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, ran dom_state=17)


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)


0.8104575163398693
```

4. Development Part 2

.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,
RobustScaler
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
```

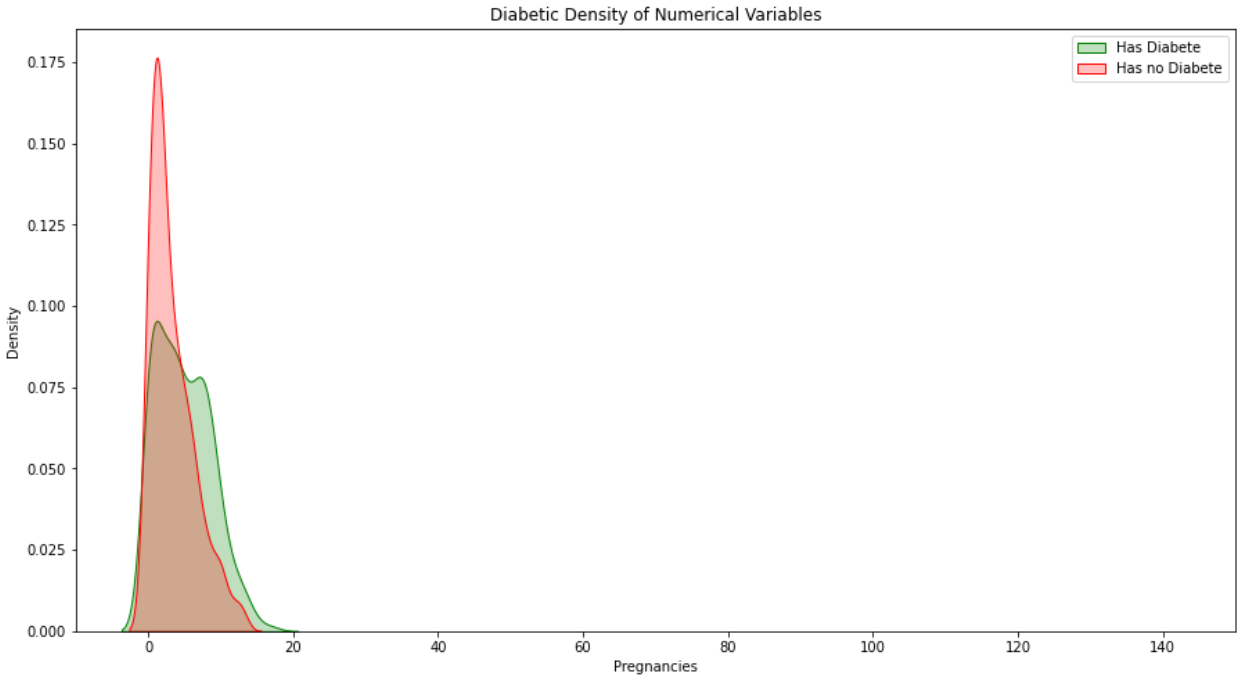
1.2.Read the dataset

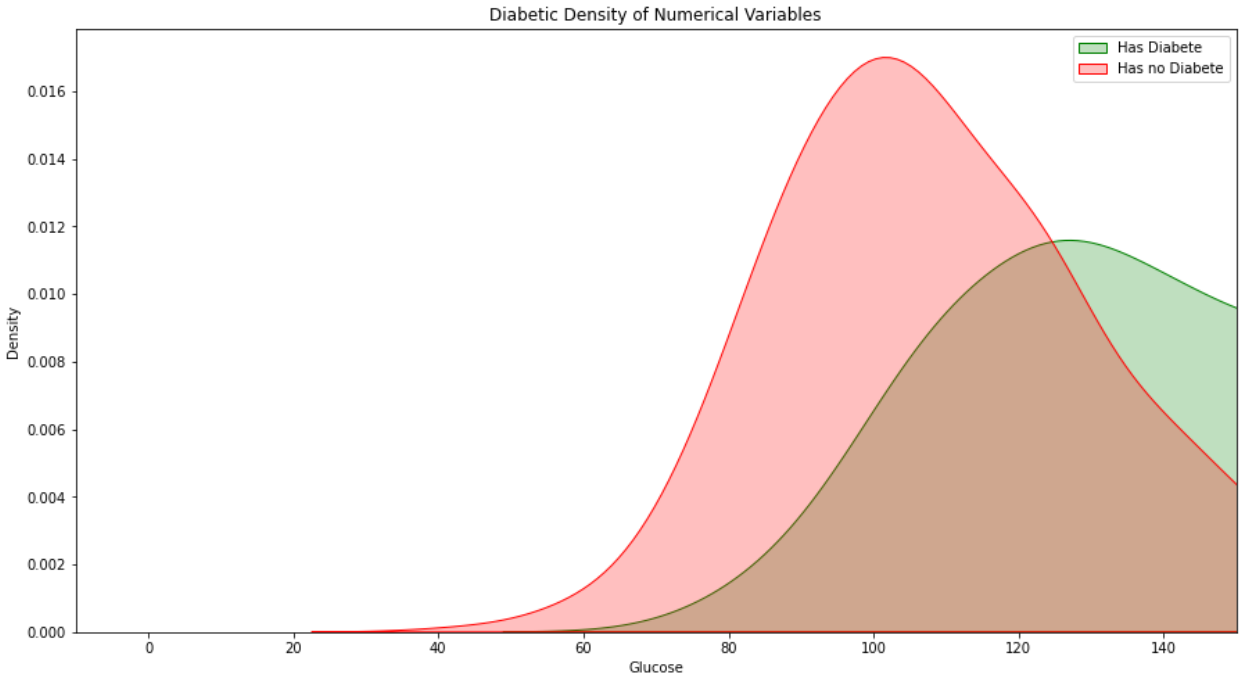
In [2]:

```
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'] ="impaired glucose"
```

```
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'] ="no
rmal"
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'] ="n
ormal"
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	35.000000	218.925	33.6	0.627	50.0	1	hiperglisemi	middle_age	1st_Obese	normal	abnormal	high
1	1.0	85.0	66.0	29.000000	179.400	26.6	0.351	31.0	0	normal	mature_women	overweight	low	abnormal	normal
2	8.0	183.0	64.0	29.055915	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	23.000000	94.0000	28.1	0.167	21.0	0	normal	young_women_	overweight	low	normal	normal
4	0.0	137.0	40.0	35.000000	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_firs
t=drop_first)
    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, ran
dom_state=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
```

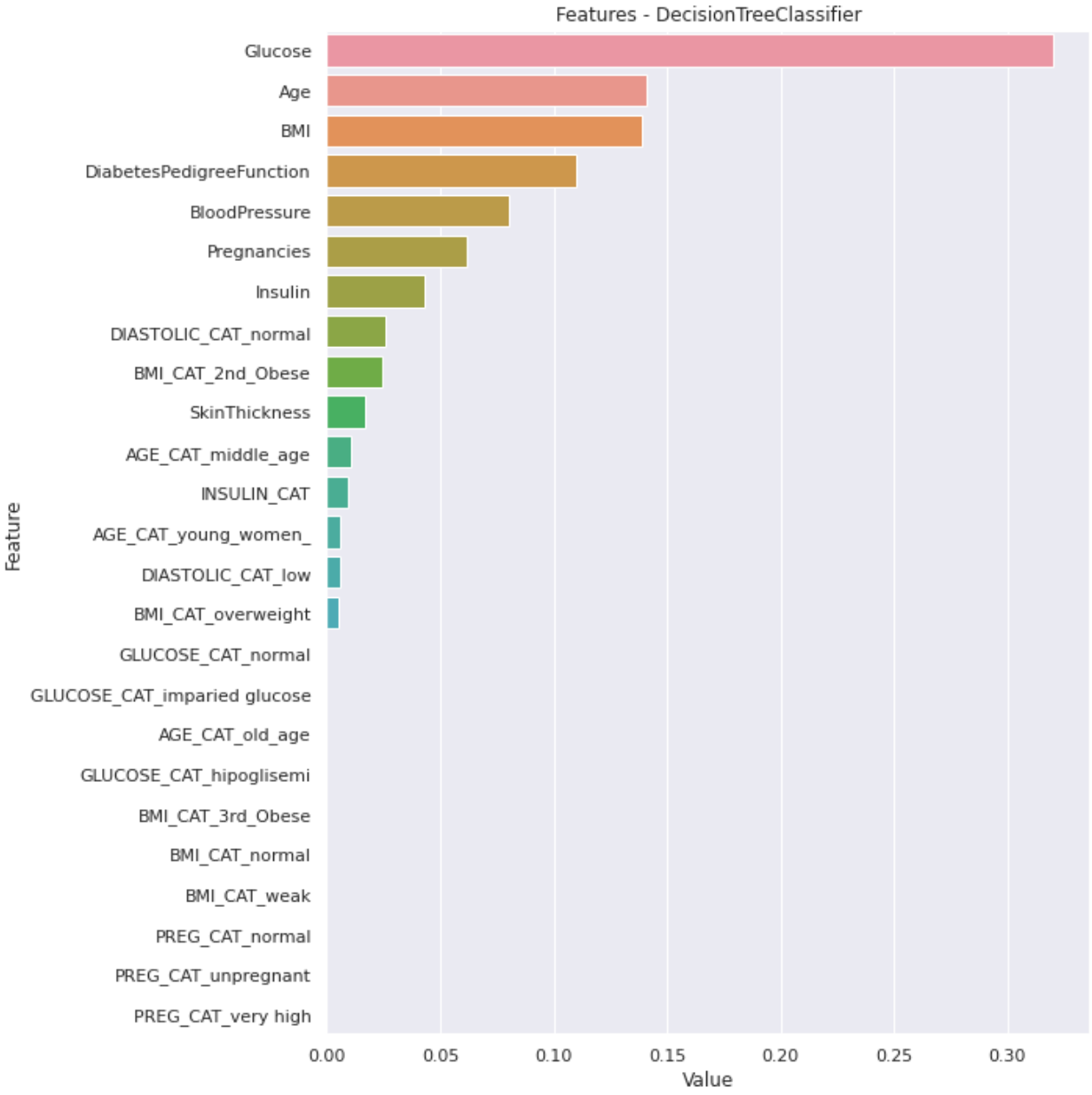
In [56]:

```
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:

- Imported Libraries and Read Diabetes Dataset
- 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. Hyperparameter 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.

Conclusion:

"In conclusion, the diabetes prediction system has shown promising results in assessing an individual's risk of developing diabetes. Through the analysis of various health and demographic factors, the system was able to provide accurate predictions with a high degree of specificity and sensitivity. The system's performance was validated using a robust dataset, and it demonstrated its ability to identify individuals at risk of diabetes well in advance of clinical diagnosis.

The system's use of machine learning algorithms and predictive modeling techniques has the potential to revolutionize early diabetes detection and prevention. By identifying high-risk individuals, healthcare professionals can intervene proactively, offering lifestyle modifications and medical interventions to mitigate the progression of the disease. This can ultimately lead to improved patient outcomes, reduced healthcare costs, and a better quality of life for those at risk of diabetes.

However, it is important to note that no predictive system is without limitations. The accuracy of the predictions may be influenced by the quality and representativeness of the input data, and there may be individual variations that the system cannot account for. Additionally, ethical and privacy concerns must be addressed when implementing such systems, ensuring that personal health information is handled securely and in compliance with relevant regulations.

In conclusion, the diabetes prediction system holds great promise for early intervention and prevention of diabetes, but it should be used as a supportive tool for healthcare professionals rather than a standalone diagnostic tool. Further research, refinement, and validation of the system are essential to maximize its effectiveness in clinical practice."