**AI BASED DIABETES PREDICTION SYSTEM**

**SUBMITTED BY**

**ASWIN S – 710021106006**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**ANNA UNIVERSITY REGIONAL CAMPUS COIMBATORE**

**ABSTRACT**

Diabetes is one of the most deadly and chronic diseases which cause an increase in blood sugar. If diabetes remains untreated and unidentified many difficulties may arise due to that. The tedious work is in identifying the process which results in visiting the clinic and consulting the doctor. But this tedious work has been solved with the rise in the approaches used by machine learning. This project gives a comprehensive prospect of work accomplished to develop a model that can predict the possibility of diabetes in patients with extreme accuracy. Therefore, various machine learning classification algorithms namely genetic algorithm, decision tree, random forest, Logistic regression, SVM and Naive Bayes are used for detecting diabetes. Further we have done the comparison among various performances of all the different algorithms. The performances are categories of various measures like Precision, Accuracy, F-Measure and Recall. The project helps in identifying the algorithm to classify the risk of diabetes. Different techniques were applied to the algorithms for improving the robustness. Additionally, the findings suggest that the best performance of disease risk classification is done with the help of a genetic algorithm.

**PROBLEM STATEMENT**

The problem we aim to address is the early prediction of diabetes using Artificial Intelligence (AI) to improve healthcare outcomes. Diabetes is a widespread chronic disease with severe health implications, and early detection is crucial for effective management and prevention of complications.

The AI-based diabetes prediction system is a cutting-edge application of artificial intelligence and machine learning techniques aimed at early detection and management of diabetes. This system leverages data from various sources, including medical records, wearable devices, and lifestyle information, to provide accurate predictions of an individual's risk of developing diabetes. By identifying at-risk individuals and offering personalized recommendations, the system can help in preventing or managing diabetes effectively. This abstract outlines the key components and design thinking behind this innovative solution.

**1)Feature Engineering**

Creating new variables is important for models. But you need to create a logical new variable. For this data set, some new variables were created according to BMI, Insulin and glucose variables.

In [40]:

*# According to BMI, some ranges were determined and categorical variables were assigned.*

NewBMI **=** pd**.**Series(["Underweight", "Normal", "Overweight", "Obesity 1", "Obesity 2", "Obesity 3"], dtype **=** "category")

df["NewBMI"] **=** NewBMI

df**.**loc[df["BMI"] **<** 18.5, "NewBMI"] **=** NewBMI[0]

df**.**loc[(df["BMI"] **>** 18.5) **&** (df["BMI"] **<=** 24.9), "NewBMI"] **=** NewBMI[1]

df**.**loc[(df["BMI"] **>** 24.9) **&** (df["BMI"] **<=** 29.9), "NewBMI"] **=** NewBMI[2]

df**.**loc[(df["BMI"] **>** 29.9) **&** (df["BMI"] **<=** 34.9), "NewBMI"] **=** NewBMI[3]

df**.**loc[(df["BMI"] **>** 34.9) **&** (df["BMI"] **<=** 39.9), "NewBMI"] **=** NewBMI[4]

df**.**loc[df["BMI"] **>** 39.9 ,"NewBMI"] **=** NewBMI[5]

In [41]:

df**.**head()

Out[41]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** | **NewBMI** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 6 | 148.0 | 72.0 | 35.0 | 169.5 | 33.6 | 0.627 | 50 | 1 | Obesity 1 |
| **1** | 1 | 85.0 | 66.0 | 29.0 | 102.5 | 26.6 | 0.351 | 31 | 0 | Overweight |
| **2** | 8 | 183.0 | 64.0 | 32.0 | 169.5 | 23.3 | 0.672 | 32 | 1 | Normal |
| **3** | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 | 0 | Overweight |
| **4** | 0 | 137.0 | 40.0 | 35.0 | 168.0 | 43.1 | 2.288 | 33 | 1 | Obesity 3 |

In [42]:

*# A categorical variable creation process is performed according to the insulin value.*

**def** set\_insulin(row):

**if** row["Insulin"] **>=** 16 **and** row["Insulin"] **<=** 166:

**return** "Normal"

**else**:

**return** "Abnormal"

In [43]:

*# The operation performed was added to the dataframe.*

df **=** df**.**assign(NewInsulinScore**=**df**.**apply(set\_insulin, axis**=**1))

df**.**head()

Out[43]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** | **NewBMI** | **NewInsulinScore** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 6 | 148.0 | 72.0 | 35.0 | 169.5 | 33.6 | 0.627 | 50 | 1 | Obesity 1 | Abnormal |
| **1** | 1 | 85.0 | 66.0 | 29.0 | 102.5 | 26.6 | 0.351 | 31 | 0 | Overweight | Normal |
| **2** | 8 | 183.0 | 64.0 | 32.0 | 169.5 | 23.3 | 0.672 | 32 | 1 | Normal | Abnormal |
| **3** | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 | 0 | Overweight | Normal |
| **4** | 0 | 137.0 | 40.0 | 35.0 | 168.0 | 43.1 | 2.288 | 33 | 1 | Obesity 3 | Abnormal |

In [44]:

*# Some intervals were determined according to the glucose variable and these were assigned categorical variables.*

NewGlucose **=** pd**.**Series(["Low", "Normal", "Overweight", "Secret", "High"], dtype **=** "category")

df["NewGlucose"] **=** NewGlucose

df**.**loc[df["Glucose"] **<=** 70, "NewGlucose"] **=** NewGlucose[0]

df**.**loc[(df["Glucose"] **>** 70) **&** (df["Glucose"] **<=** 99), "NewGlucose"] **=** NewGlucose[1]

df**.**loc[(df["Glucose"] **>** 99) **&** (df["Glucose"] **<=** 126), "NewGlucose"] **=** NewGlucose[2]

df**.**loc[df["Glucose"] **>** 126 ,"NewGlucose"] **=** NewGlucose[3]

In [45]:

df**.**head()

Out[45]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** | **NewBMI** | **NewInsulinScore** | **NewGlucose** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 6 | 148.0 | 72.0 | 35.0 | 169.5 | 33.6 | 0.627 | 50 | 1 | Obesity 1 | Abnormal | Secret |
| **1** | 1 | 85.0 | 66.0 | 29.0 | 102.5 | 26.6 | 0.351 | 31 | 0 | Overweight | Normal | Normal |
| **2** | 8 | 183.0 | 64.0 | 32.0 | 169.5 | 23.3 | 0.672 | 32 | 1 | Normal | Abnormal | Secret |
| **3** | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 | 0 | Overweight | Normal | Normal |
| **4** | 0 | 137.0 | 40.0 | 35.0 | 168.0 | 43.1 | 2.288 | 33 | 1 | Obesity 3 | Abnormal | Secret |

**2) One Hot Encoding**

Categorical variables in the data set should be converted into numerical values. For this reason, these transformation processes are performed with Label Encoding and One Hot Encoding method.

In [46]:

*# Here, by making One Hot Encoding transformation, categorical variables were converted into numerical values. It is also protected from the Dummy variable trap.*

df **=** pd**.**get\_dummies(df, columns **=**["NewBMI","NewInsulinScore", "NewGlucose"], drop\_first **=** **True**)

In [47]:

df**.**head()

Out[47]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** | **NewBMI\_Obesity 1** | **NewBMI\_Obesity 2** | **NewBMI\_Obesity 3** | **NewBMI\_Overweight** | **NewBMI\_Underweight** | **NewInsulinScore\_Normal** | **NewGlucose\_Low** | **NewGlucose\_Normal** | **NewGlucose\_Overweight** | **NewGlucose\_Secret** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 6 | 148.0 | 72.0 | 35.0 | 169.5 | 33.6 | 0.627 | 50 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **1** | 1 | 85.0 | 66.0 | 29.0 | 102.5 | 26.6 | 0.351 | 31 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 8 | 183.0 | 64.0 | 32.0 | 169.5 | 23.3 | 0.672 | 32 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **3** | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **4** | 0 | 137.0 | 40.0 | 35.0 | 168.0 | 43.1 | 2.288 | 33 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

In [48]:

categorical\_df **=** df[['NewBMI\_Obesity 1','NewBMI\_Obesity 2', 'NewBMI\_Obesity 3', 'NewBMI\_Overweight','NewBMI\_Underweight',

'NewInsulinScore\_Normal','NewGlucose\_Low','NewGlucose\_Normal', 'NewGlucose\_Overweight', 'NewGlucose\_Secret']]

In [49]:

categorical\_df**.**head()

Out[49]:

|  | **NewBMI\_Obesity 1** | **NewBMI\_Obesity 2** | **NewBMI\_Obesity 3** | **NewBMI\_Overweight** | **NewBMI\_Underweight** | **NewInsulinScore\_Normal** | **NewGlucose\_Low** | **NewGlucose\_Normal** | **NewGlucose\_Overweight** | **NewGlucose\_Secret** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **1** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **3** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **4** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

In [50]:

y **=** df["Outcome"]

X **=** df**.**drop(["Outcome",'NewBMI\_Obesity 1','NewBMI\_Obesity 2', 'NewBMI\_Obesity 3', 'NewBMI\_Overweight','NewBMI\_Underweight',

'NewInsulinScore\_Normal','NewGlucose\_Low','NewGlucose\_Normal', 'NewGlucose\_Overweight', 'NewGlucose\_Secret'], axis **=** 1)

cols **=** X**.**columns

index **=** X**.**index

In [51]:

X**.**head()

Out[51]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 6 | 148.0 | 72.0 | 35.0 | 169.5 | 33.6 | 0.627 | 50 |
| **1** | 1 | 85.0 | 66.0 | 29.0 | 102.5 | 26.6 | 0.351 | 31 |
| **2** | 8 | 183.0 | 64.0 | 32.0 | 169.5 | 23.3 | 0.672 | 32 |
| **3** | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 |
| **4** | 0 | 137.0 | 40.0 | 35.0 | 168.0 | 43.1 | 2.288 | 33 |

In [52]:

*# The variables in the data set are an effective factor in increasing the performance of the models by standardization.*

*# There are multiple standardization methods. These are methods such as" Normalize"," MinMax"," Robust" and "Scale".*

**from** sklearn.preprocessing **import** RobustScaler

transformer **=** RobustScaler()**.**fit(X)

X **=** transformer**.**transform(X)

X **=** pd**.**DataFrame(X, columns **=** cols, index **=** index)

In [53]:

X**.**head()

Out[53]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.6 | 0.775 | 0.000 | 1.000000 | 1.000000 | 0.177778 | 0.669707 | 1.235294 |
| **1** | -0.4 | -0.800 | -0.375 | 0.142857 | 0.000000 | -0.600000 | -0.049511 | 0.117647 |
| **2** | 1.0 | 1.650 | -0.500 | 0.571429 | 1.000000 | -0.966667 | 0.786971 | 0.176471 |
| **3** | -0.4 | -0.700 | -0.375 | -0.714286 | -0.126866 | -0.433333 | -0.528990 | -0.470588 |
| **4** | -0.6 | 0.500 | -2.000 | 1.000000 | 0.977612 | 1.233333 | 4.998046 | 0.235294 |

In [54]:

X **=** pd**.**concat([X,categorical\_df], axis **=** 1)

In [55]:

X**.**head()

Out[55]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **NewBMI\_Obesity 1** | **NewBMI\_Obesity 2** | **NewBMI\_Obesity 3** | **NewBMI\_Overweight** | **NewBMI\_Underweight** | **NewInsulinScore\_Normal** | **NewGlucose\_Low** | **NewGlucose\_Normal** | **NewGlucose\_Overweight** | **NewGlucose\_Secret** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.6 | 0.775 | 0.000 | 1.000000 | 1.000000 | 0.177778 | 0.669707 | 1.235294 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **1** | -0.4 | -0.800 | -0.375 | 0.142857 | 0.000000 | -0.600000 | -0.049511 | 0.117647 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 1.0 | 1.650 | -0.500 | 0.571429 | 1.000000 | -0.966667 | 0.786971 | 0.176471 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **3** | -0.4 | -0.700 | -0.375 | -0.714286 | -0.126866 | -0.433333 | -0.528990 | -0.470588 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **4** | -0.6 | 0.500 | -2.000 | 1.000000 | 0.977612 | 1.233333 | 4.998046 | 0.235294 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

In [56]:

y**.**head()

Out[56]:

0 1

1 0

2 1

3 0

4 1

Name: Outcome, dtype: int64

**3) Base Models**

In [57]:

*# Validation scores of all base models*

models = []

models.append(('LR', LogisticRegression(random\_state = 12345)))

models.append(('KNN', KNeighborsClassifier()))

models.append(('CART', DecisionTreeClassifier(random\_state = 12345)))

models.append(('RF', RandomForestClassifier(random\_state = 12345)))

models.append(('SVM', SVC(gamma='auto', random\_state = 12345)))

models.append(('XGB', GradientBoostingClassifier(random\_state = 12345)))

models.append(("LightGBM", LGBMClassifier(random\_state = 12345)))

*# evaluate each model in turn*

results = []

names = []

In [58]:

**for** name, model **in** models:

kfold = KFold(n\_splits = 10, random\_state = 12345)

cv\_results = cross\_val\_score(model, X, y, cv = 10, scoring= "accuracy")

results.append(cv\_results)

names.append(name)

msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

print(msg)

*# boxplot algorithm comparison*

fig = plt.figure(figsize=(15,10))

fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results)

ax.set\_xticklabels(names)

plt.show()

LR: 0.848684 (0.036866)

KNN: 0.840789 (0.023866)

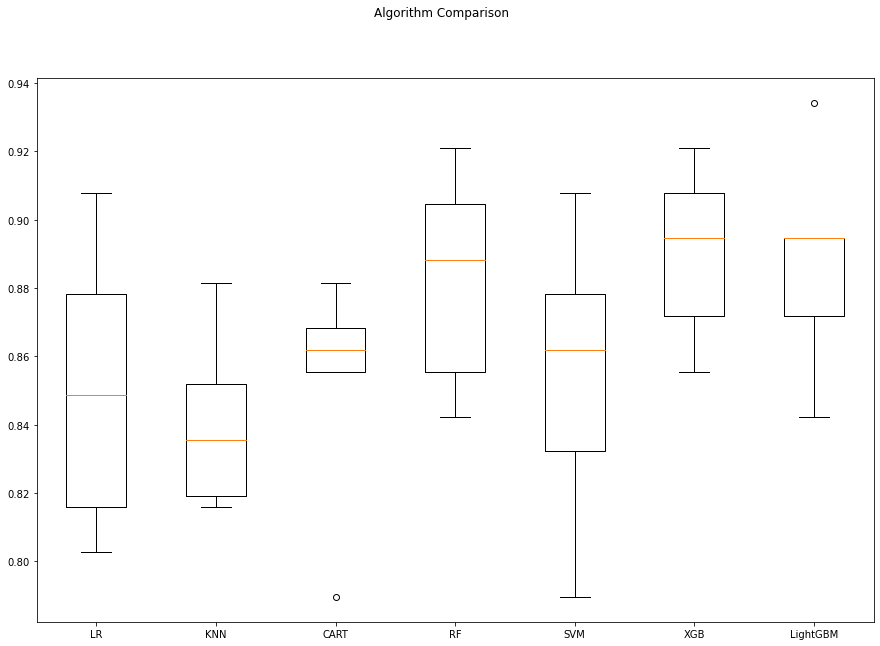
CART: 0.857895 (0.024826)

RF: 0.881579 (0.026316)

SVM: 0.853947 (0.036488)

XGB: 0.890789 (0.020427)

LightGBM: 0.885526 (0.024298)



**4) Model Tuning**

**1) Random Forests Tuning**

In [59]:

rf\_params **=** {"n\_estimators" :[100,200,500,1000],

"max\_features": [3,5,7],

"min\_samples\_split": [2,5,10,30],

"max\_depth": [3,5,8,**None**]}

In [60]:

rf\_model **=** RandomForestClassifier(random\_state **=** 12345)

In [61]:

gs\_cv **=** GridSearchCV(rf\_model,

rf\_params,

cv **=** 10,

n\_jobs **=** **-**1,

verbose **=** 2)**.**fit(X, y)

Fitting 10 folds for each of 192 candidates, totalling 1920 fits

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n\_jobs=-1)]: Done 33 tasks | elapsed: 10.4s

[Parallel(n\_jobs=-1)]: Done 154 tasks | elapsed: 48.2s

[Parallel(n\_jobs=-1)]: Done 357 tasks | elapsed: 1.9min

[Parallel(n\_jobs=-1)]: Done 640 tasks | elapsed: 3.5min

[Parallel(n\_jobs=-1)]: Done 1005 tasks | elapsed: 5.7min

[Parallel(n\_jobs=-1)]: Done 1450 tasks | elapsed: 8.4min

[Parallel(n\_jobs=-1)]: Done 1920 out of 1920 | elapsed: 11.4min finished

In [62]:

gs\_cv**.**best\_params\_

Out[62]:

{'max\_depth': 8,

'max\_features': 7,

'min\_samples\_split': 2,

'n\_estimators': 500}

**2) LightGBM Tuning**

In [63]:

lgbm **=** LGBMClassifier(random\_state **=** 12345)

In [64]:

lgbm\_params **=** {"learning\_rate": [0.01, 0.03, 0.05, 0.1, 0.5],

"n\_estimators": [500, 1000, 1500],

"max\_depth":[3,5,8]}

In [65]:

gs\_cv **=** GridSearchCV(lgbm,

lgbm\_params,

cv **=** 10,

n\_jobs **=** **-**1,

verbose **=** 2)**.**fit(X, y)

Fitting 10 folds for each of 45 candidates, totalling 450 fits

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n\_jobs=-1)]: Done 33 tasks | elapsed: 3.7s

[Parallel(n\_jobs=-1)]: Done 154 tasks | elapsed: 25.3s

[Parallel(n\_jobs=-1)]: Done 357 tasks | elapsed: 57.4s

[Parallel(n\_jobs=-1)]: Done 450 out of 450 | elapsed: 1.1min finished

In [66]:

gs\_cv**.**best\_params\_

Out[66]:

{'learning\_rate': 0.01, 'max\_depth': 3, 'n\_estimators': 1000}

**3) XGBoost Tuning**

In [67]:

xgb **=** GradientBoostingClassifier(random\_state **=** 12345)

In [68]:

xgb\_params **=** {

"learning\_rate": [0.01, 0.1, 0.2, 1],

"min\_samples\_split": np**.**linspace(0.1, 0.5, 10),

"max\_depth":[3,5,8],

"subsample":[0.5, 0.9, 1.0],

"n\_estimators": [100,1000]}

In [69]:

xgb\_cv\_model **=** GridSearchCV(xgb,xgb\_params, cv **=** 10, n\_jobs **=** **-**1, verbose **=** 2)**.**fit(X, y)

Fitting 10 folds for each of 720 candidates, totalling 7200 fits

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n\_jobs=-1)]: Done 33 tasks | elapsed: 4.0s

[Parallel(n\_jobs=-1)]: Done 154 tasks | elapsed: 42.3s

[Parallel(n\_jobs=-1)]: Done 357 tasks | elapsed: 1.9min

[Parallel(n\_jobs=-1)]: Done 640 tasks | elapsed: 3.1min

[Parallel(n\_jobs=-1)]: Done 1005 tasks | elapsed: 5.4min

[Parallel(n\_jobs=-1)]: Done 1450 tasks | elapsed: 8.4min

[Parallel(n\_jobs=-1)]: Done 1977 tasks | elapsed: 11.6min

[Parallel(n\_jobs=-1)]: Done 2584 tasks | elapsed: 14.8min

[Parallel(n\_jobs=-1)]: Done 3273 tasks | elapsed: 19.3min

[Parallel(n\_jobs=-1)]: Done 4042 tasks | elapsed: 23.6min

[Parallel(n\_jobs=-1)]: Done 4893 tasks | elapsed: 28.7min

[Parallel(n\_jobs=-1)]: Done 5824 tasks | elapsed: 34.5min

[Parallel(n\_jobs=-1)]: Done 6837 tasks | elapsed: 40.9min

[Parallel(n\_jobs=-1)]: Done 7200 out of 7200 | elapsed: 43.2min finished

In [70]:

xgb\_cv\_model**.**best\_params\_

Out[70]:

{'learning\_rate': 0.1,

'max\_depth': 5,

'min\_samples\_split': 0.1,

'n\_estimators': 100,

'subsample': 1.0}

**CONCLUSION**

The AI-based diabetes prediction system is a proactive solution to the problem of late diabetes diagnosis. By applying a design thinking approach, we have developed a system that leverages advanced AI techniques to predict diabetes risk early, enabling timely interventions and personalized healthcare. This system has the potential to improve the lives of individuals at risk of diabetes and enhance the efficiency of healthcare delivery. Early diagnosis is the key to better managing diabetes and reducing its long-term complications, and our AI system is a step forward in achieving this goal.