In [1]:
```
import os
os.getcwd()
```

Out[1]: 'C:\\Users\\chitt'

In [3]:
```
import numpy as np

import pandas as pd #excellent for dataset manupalation

# for data visulization
import matplotlib.pyplot as plt

#stats visualization
import seaborn as sns

#Labelencoding to convert categorical data into lowlevel language
from sklearn.preprocessing import LabelEncoder

#scaling data
from sklearn.preprocessing import StandardScaler

#data partions
from sklearn.model_selection import train_test_split

#algorithams
from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier


#accuracy confusion matric and classification report
from sklearn.metrics import accuracy_score,confusion_matrix,classification_repor


import warnings

# To ignore all warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
df=pd.read_csv(r"D:\NIT Daily Task\Oct\21st DIBETIC PREDICTION\DIBETIC PREDICTIO
```

In [7]:
```
df.head()
```

Out[7]:

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	bloo
0	Female	80.0	0	1	never	25.19	6.6	
1	Female	54.0	0	0	No Info	27.32	6.6	
2	Male	28.0	0	0	never	27.32	5.7	
3	Female	36.0	0	0	current	23.45	5.0	
4	Male	76.0	1	1	current	20.14	4.8	

In [9]:
```
df.isna().any()
```

Out[9]:
```
gender                 False
age                    False
hypertension           False
heart_disease          False
smoking_history        False
bmi                    False
HbA1c_level            False
blood_glucose_level    False
diabetes               False
dtype: bool
```

In [11]:
```
df.corr(numeric_only=True)
```

Out[11]:

	age	hypertension	heart_disease	bmi	HbA1c_level	bloo
age	1.000000	0.251171	0.233354	0.337396	0.101354	
hypertension	0.251171	1.000000	0.121262	0.147666	0.080939	
heart_disease	0.233354	0.121262	1.000000	0.061198	0.067589	
bmi	0.337396	0.147666	0.061198	1.000000	0.082997	
HbA1c_level	0.101354	0.080939	0.067589	0.082997	1.000000	
blood_glucose_level	0.110672	0.084429	0.070066	0.091261	0.166733	
diabetes	0.258008	0.197823	0.171727	0.214357	0.400660	

In [15]:
```
df.shape
```

Out[15]:
```
(100000, 9)
```

In [17]:
```
for column in df.columns: # itreating each column in df.columns
    unique_values = df[column].unique() #finding unique values of each column

    #printing unique values
    print('Column "{}" has unique values: {}'.format(column, unique_values))
```

```
Column "gender" has unique values: ['Female' 'Male' 'Other']
Column "age" has unique values: [80.   54.   28.   36.   76.   20.   44.   79.
42.   32.   53.   78.
 67.   15.   37.   40.    5.   69.   72.    4.   30.   45.   43.   50.
 41.   26.   34.   73.   77.   66.   29.   60.   38.    3.   57.   74.
 19.   46.   21.   59.   27.   13.   56.    2.    7.   11.    6.   55.
  9.   62.   47.   12.   68.   75.   22.   58.   18.   24.   17.   25.
  0.08 33.   16.   61.   31.    8.   49.   39.   65.   14.   70.    0.56
 48.   51.   71.    0.88 64.   63.   52.    0.16 10.   35.   23.    0.64
  1.16  1.64  0.72  1.88  1.32  0.8   1.24  1.    1.8   0.48  1.56  1.08
  0.24  1.4   0.4   0.32  1.72  1.48]
Column "hypertension" has unique values: [0 1]
Column "heart_disease" has unique values: [1 0]
Column "smoking_history" has unique values: ['never' 'No Info' 'current' 'former'
'ever' 'not current']
Column "bmi" has unique values: [25.19 27.32 23.45 ... 59.42 44.39 60.52]
Column "HbA1c_level" has unique values: [6.6 5.7 5.  4.8 6.5 6.1 6.  5.8 3.5 6.2
4.  4.5 9.  7.  8.8 8.2 7.5 6.8]
Column "blood_glucose_level" has unique values: [140  80 158 155  85 200 145 100
130 160 126 159  90 260 220 300 280 240]
Column "diabetes" has unique values: [0 1]
```

In [19]:

```
df["smoking_history"].value_counts() #Value count of smoking _history parameter
```

Out[19]:

```
smoking_history
No Info        35816
never          35095
former          9352
current         9286
not current     6447
ever            4004
Name: count, dtype: int64
```

In [21]:

```
df["smoking_history"].value_counts()/len(df) #finding the percentage
```

Out[21]:

```
smoking_history
No Info        0.35816
never          0.35095
former         0.09352
current        0.09286
not current    0.06447
ever           0.04004
Name: count, dtype: float64
```

In [23]:

```
# Replaceing No Info columns with pd.NA
df['smoking_history'] = df['smoking_history'].replace('No Info', pd.NA)

# Replace missing values with the mode it is string so we are using mode
mode_value = df['smoking_history'].mode()[0]
df['smoking_history'] = df['smoking_history'].fillna(mode_value) #filling no inf

# Printing the updated value counts
print(df['smoking_history'].value_counts())
```

```
smoking_history
never          70911
former          9352
current         9286
not current     6447
ever            4004
Name: count, dtype: int64
```

In [25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   gender               100000 non-null  object 
 1   age                  100000 non-null  float64
 2   hypertension         100000 non-null  int64  
 3   heart_disease        100000 non-null  int64  
 4   smoking_history      100000 non-null  object 
 5   bmi                  100000 non-null  float64
 6   HbA1c_level          100000 non-null  float64
 7   blood_glucose_level  100000 non-null  int64  
 8   diabetes             100000 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 6.9+ MB
```

In [31]:
```
df.gender.value_counts()
```

Out[31]:
```
gender
Female    58552
Male      41430
Other        18
Name: count, dtype: int64
```

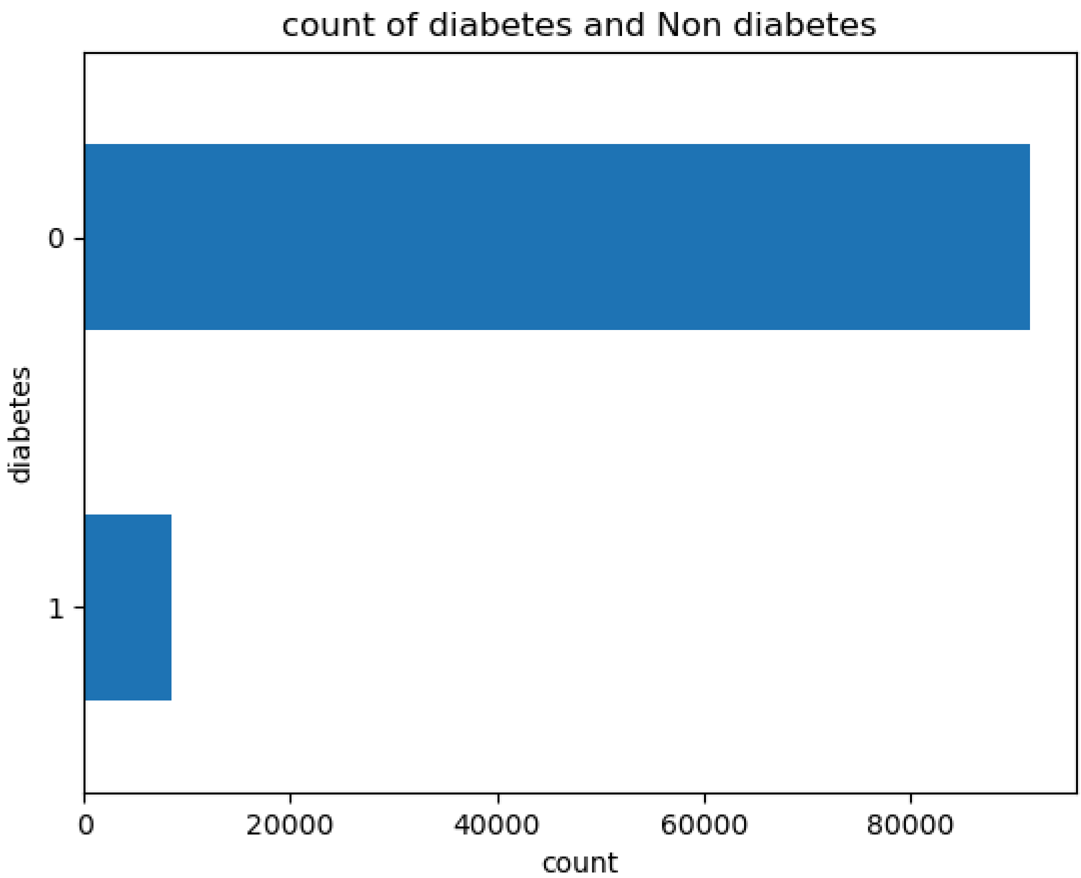
In [33]:
```
df.describe()
```

Out[33]:

	age	hypertension	heart_disease	bmi	HbA1c_level	bloo
count	100000.000000	100000.00000	100000.000000	100000.000000	100000.000000	
mean	41.885856	0.07485	0.039420	27.320767	5.527507	
std	22.516840	0.26315	0.194593	6.636783	1.070672	
min	0.080000	0.00000	0.000000	10.010000	3.500000	
25%	24.000000	0.00000	0.000000	23.630000	4.800000	
50%	43.000000	0.00000	0.000000	27.320000	5.800000	
75%	60.000000	0.00000	0.000000	29.580000	6.200000	
max	80.000000	1.00000	1.000000	95.690000	9.000000	

In [35]:
```
#removing , in bmi parameter
df["bmi"] = [float(str(i).replace(",", "")) for i in df["bmi"]]
```

In [37]:
```
#ploting value_counts of diabetes in graphical representation
df['diabetes'].value_counts().plot(kind='barh')

#Xlabel name
plt.xlabel('count')

#ylabel name
plt.ylabel('diabetes')

#title of the plot
plt.title('count of diabetes and Non diabetes')

#invert ylabes to no diabetes on top
plt.gca().invert_yaxis()

#printing the plot
plt.show()
```



In [39]:
```
df['diabetes'].value_counts()/len(df) #percentage of 1--diabetes and 2--no diabe
```

Out[39]:
```
diabetes
0    0.915
1    0.085
Name: count, dtype: float64
```

In [41]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   gender               100000 non-null  object 
 1   age                  100000 non-null  float64
 2   hypertension         100000 non-null  int64  
 3   heart_disease        100000 non-null  int64  
 4   smoking_history      100000 non-null  object 
 5   bmi                  100000 non-null  float64
 6   HbA1c_level          100000 non-null  float64
 7   blood_glucose_level  100000 non-null  int64  
 8   diabetes             100000 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 6.9+ MB
```

In [45]:
```
le=LabelEncoder()


le
```

Out[45]:

▾ LabelEncoder

```
LabelEncoder()
```

In [47]:
```
Label_encod_columns=['gender','smoking_history']  #selecting columns to apply La

df[Label_encod_columns]=df[Label_encod_columns].apply(le.fit_transform) #applyin
```

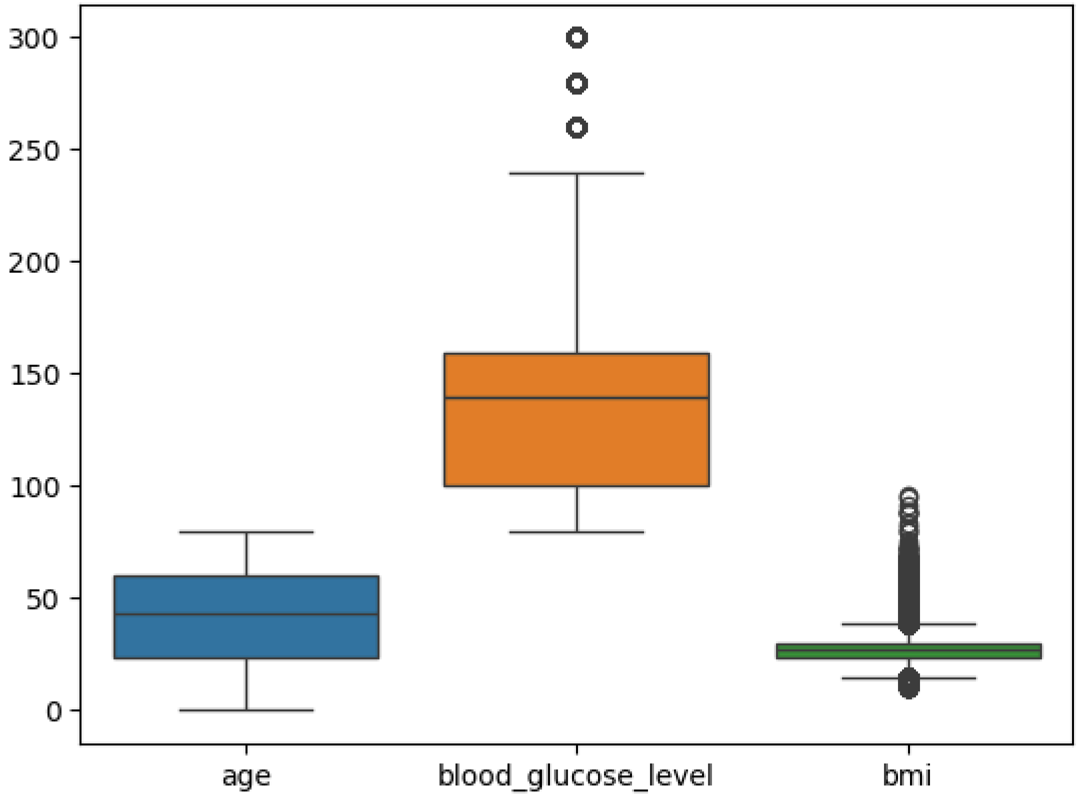
In [49]:
```
df.head(3)
```

Out[49]:

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blo
0	0	80.0	0	1	3	25.19	6.6	
1	0	54.0	0	0	3	27.32	6.6	
2	1	28.0	0	0	3	27.32	5.7	

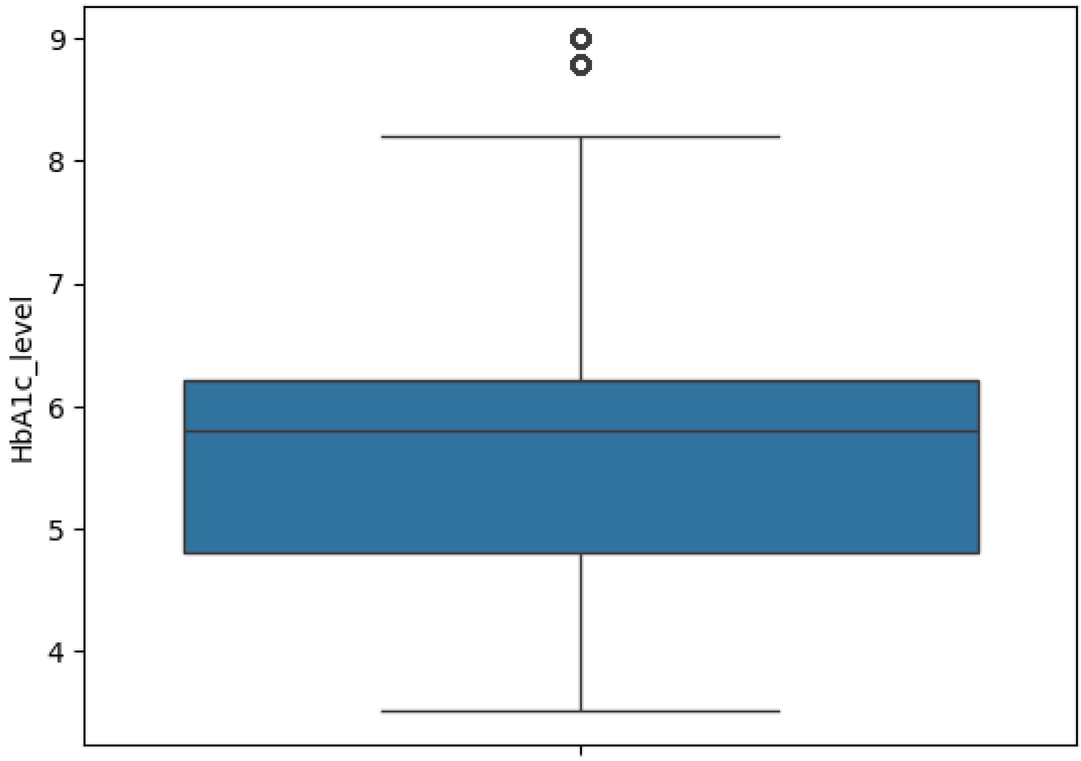
In [51]:
```
sns.boxplot(data=df[['age','blood_glucose_level','bmi']])
```

Out[51]: <Axes: >



In [53]:
```
sns.boxplot(data=df['HbA1c_level']) #checking outlayers using boxplot
```

Out[53]: <Axes: ylabel='HbA1c_level'>

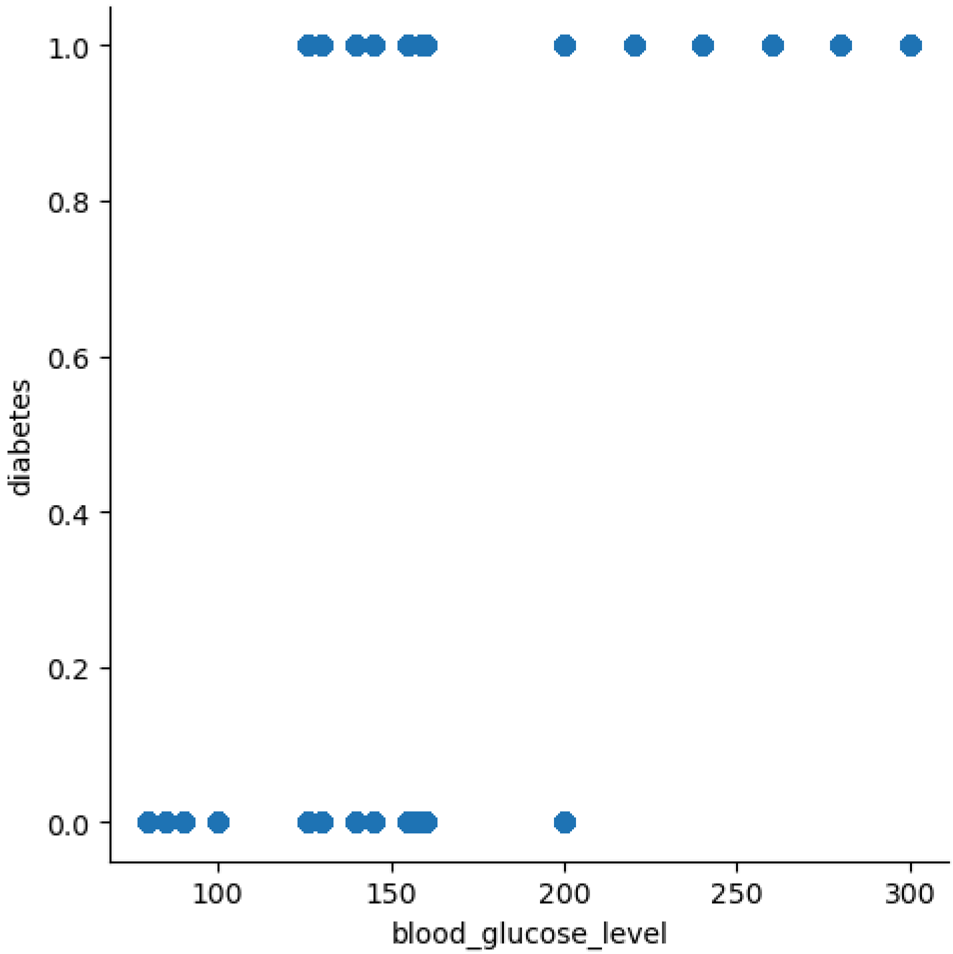


In [55]:
```
''' it is always good to ignore outliers in medical data '''
```

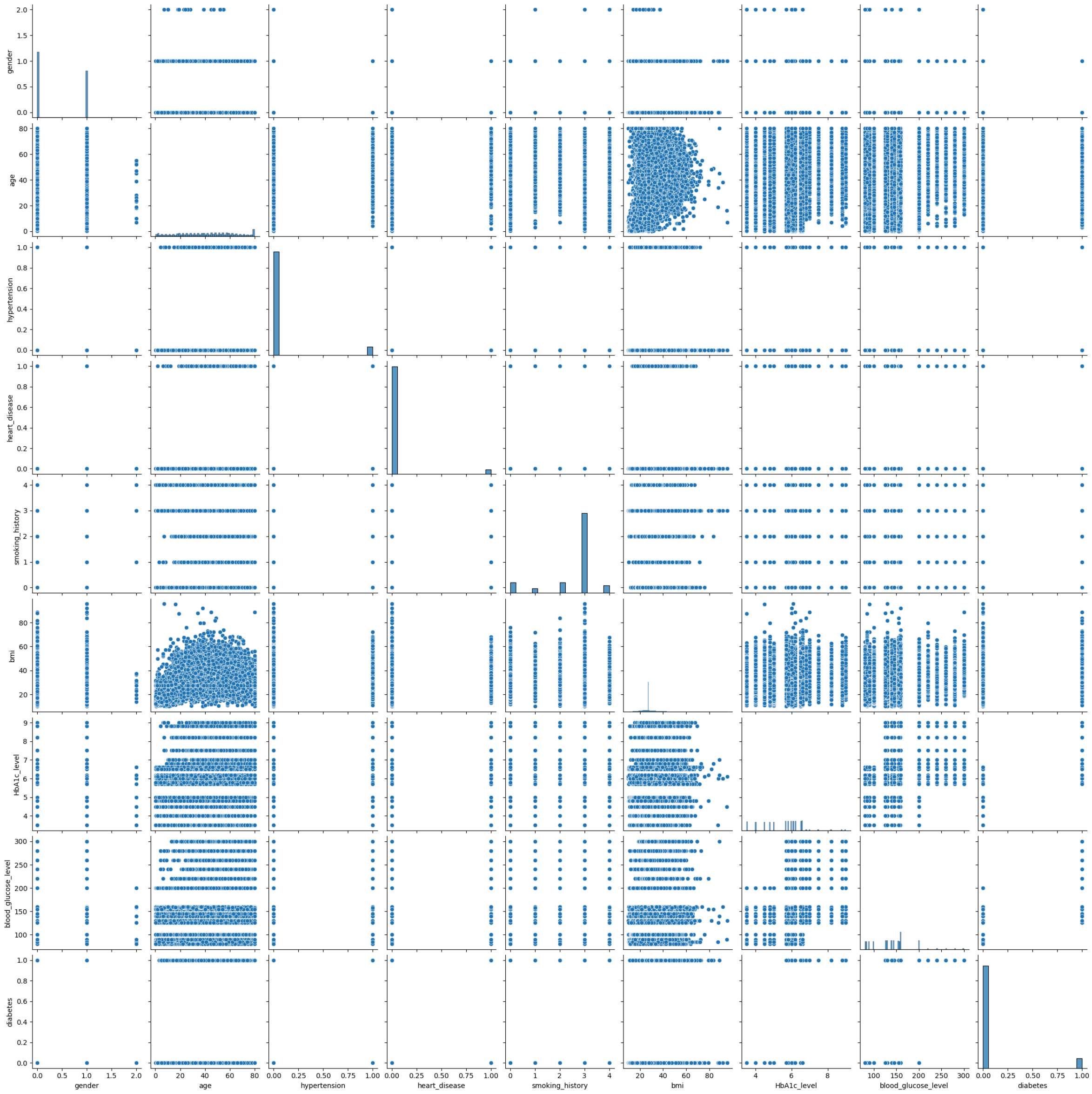
Out[55]: ' it is always good to ignore outliers in medical data '

In [57]:
```
sns.lmplot(data=df, x='blood_glucose_level', y='diabetes', fit_reg=False)#implot
```

Out[57]: <seaborn.axisgrid.FacetGrid at 0x24e8d3eb440>



In [59]:
```
sns.pairplot(df) #using pairplot to check relation between parameters

#print the pairplot
plt.show()
```



In [61]:
```
'''when age increase hypertension and hert disease ,blood_glucose_level and diab
   relationship between them

    *bmi

    *HbA1c_level

    *blood_glucose_level

    these four paramers have relationship between each other

    *gender and smokling history it doesnot effect on diabetes

'''
```

Out[61]:
```
'when age increase hypertension and hert disease ,blood_glucose_level and diabe
tes and age  and also the is a \n    relationship between them\n\n     *bmi\n
\n     *HbA1c_level\n     \n     *blood_glucose_level\n     \n     these four parame
rs have relationship between each other\n     \n     *gender and smokling history
it doesnot effect on diabetes\n\n'
```

In [63]:
```
df.corr()
```

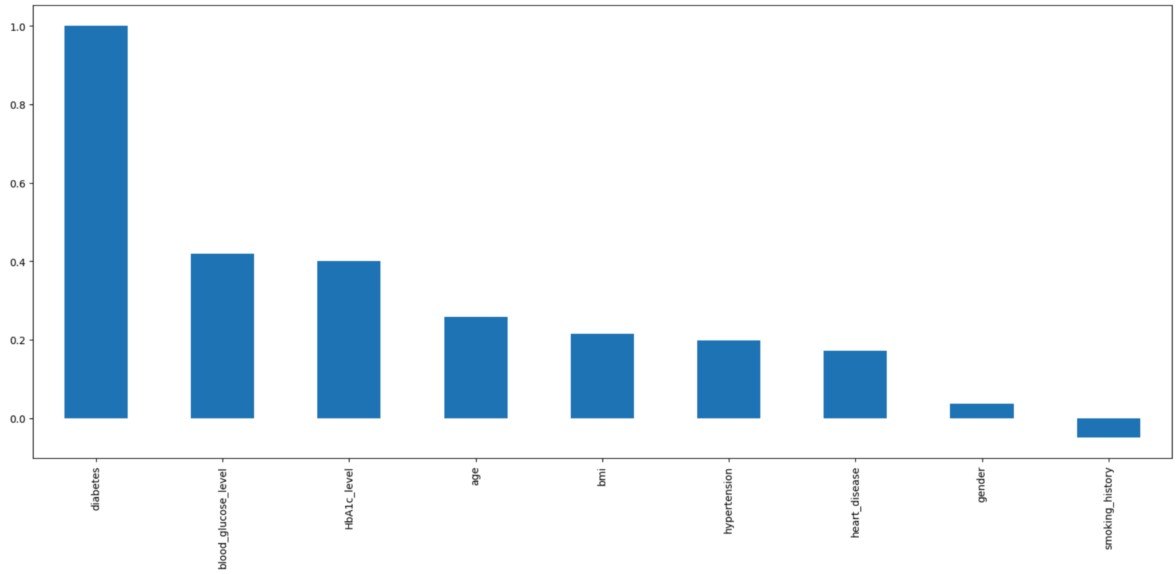
Out[63]:

	gender	age	hypertension	heart_disease	smoking_history
gender	1.000000	-0.030656	0.014203	0.077696	-0.044081
age	-0.030656	1.000000	0.251171	0.233354	-0.098969
hypertension	0.014203	0.251171	1.000000	0.121262	-0.048631
heart_disease	0.077696	0.233354	0.121262	1.000000	-0.048253
smoking_history	-0.044081	-0.098969	-0.048631	-0.048253	1.000000
bmi	-0.022994	0.337396	0.147666	0.061198	-0.087735
HbA1c_level	0.019957	0.101354	0.080939	0.067589	-0.017534
blood_glucose_level	0.017199	0.110672	0.084429	0.070066	-0.022985
diabetes	0.037411	0.258008	0.197823	0.171727	-0.049841

In [65]:

```
plt.figure(figsize=(20,8))

df.corr()['diabetes'].sort_values(ascending=False).plot(kind='bar')
```

Out[65]: <Axes: >



In [67]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   gender               100000 non-null  int32  
 1   age                  100000 non-null  float64
 2   hypertension         100000 non-null  int64  
 3   heart_disease        100000 non-null  int64  
 4   smoking_history      100000 non-null  int32  
 5   bmi                  100000 non-null  float64
 6   HbA1c_level          100000 non-null  float64
 7   blood_glucose_level  100000 non-null  int64  
 8   diabetes             100000 non-null  int64  
dtypes: float64(3), int32(2), int64(4)
memory usage: 6.1 MB
```

In [69]:
```
#selecting X variables
X = df.loc[:, 'age':'heart_disease'].join(df.loc[:, 'bmi':'blood_glucose_level']


X
```

Out[69]:

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level
0	80.0	0	1	25.19	6.6	140
1	54.0	0	0	27.32	6.6	80
2	28.0	0	0	27.32	5.7	158
3	36.0	0	0	23.45	5.0	155
4	76.0	1	1	20.14	4.8	155
...	...	...	...	...	...	...
99995	80.0	0	0	27.32	6.2	90
99996	2.0	0	0	17.37	6.5	100
99997	66.0	0	0	27.83	5.7	155
99998	24.0	0	0	35.42	4.0	100
99999	57.0	0	0	22.43	6.6	90

100000 rows × 6 columns

In [71]:
```
y=df.loc[:,'diabetes']
y
```

Out[71]:
```
0        0
1        0
2        0
3        0
4        0
        ..
99995    0
99996    0
99997    0
99998    0
99999    0
Name: diabetes, Length: 100000, dtype: int64
```

In [73]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

In [75]:
```
X_train.head()
```

Out[75]:

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level
10382	2.0	0	0	16.45	6.2	159
73171	55.0	0	0	24.59	6.0	130
30938	24.0	0	0	21.77	4.5	130
99310	30.0	0	0	27.32	6.2	159
58959	13.0	0	0	18.37	6.5	130

In [77]:
```
print('Shape of Train data')

print(X_train.shape)

print(y_train.shape)

print('Shape of Testing data')

print(X_test.shape)

print(y_test.shape)
```

```
Shape of Train data
(80000, 6)
(80000,)
Shape of Testing data
(20000, 6)
(20000,)
```

In [79]:
```
ss=StandardScaler() #activating StandardScaler()

ss
```

Out[79]:
```
▾ StandardScaler
StandardScaler()
```

In [81]:
```
X_train_scaled=ss.fit_transform(X_train) #scaling X_train data
```

In [83]:
```
if len(X_test.shape) == 1:   #if x is 1d array
    X_test = X_test.values.reshape(-1, 1) #converting to 2d array

X_test_scaled = ss.fit_transform(X_test) #scaling X_test data
```

In [85]:
```
model_lr=LogisticRegression()  #activating Logistic Regression
```

In [87]:
```
model_lr.fit(X_train_scaled,y_train) #training logistic regression model
```

Out[87]:
```
▾ LogisticRegression
LogisticRegression()
```

In [89]:
```
y_pred=model_lr.predict(X_test_scaled) #predecting y_test data
y_pred[:10]
```

Out[89]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [91]:
```
y_test[:10] # actual y_test data
```

Out[91]:
```
3582     0
60498    0
53227    0
21333    0
3885     0
51521    0
84261    0
10685    1
59948    0
41032    0
Name: diabetes, dtype: int64
```

In [95]:
```
accuracy_score(y_pred,y_test)
```

Out[95]:
```
0.95975
```

In [97]:
```
print(classification_report(y_pred,y_test)) #classifiaction_report
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98     18736
           1       0.63      0.86      0.73      1264

    accuracy                           0.96     20000
   macro avg       0.81      0.91      0.85     20000
weighted avg       0.97      0.96      0.96     20000
```

In [99]:
```
''' As you can see that the accuracy is quite low, and as it's an imbalanced dat
Hence, we need to check recall, precision & f1 score for the minority class, and
Hence, moving ahead to call SMOTEENN (UpSampling + ENN)'''


'''main advantage of using SMOTEENN is that it addresses both overfitting and un
```

Out[99]: 'main advantage of using SMOTEENN is that it addresses both overfitting and underfitting issues that can arise from class imbalance. By generating synthetic samples and removing noisy ones'

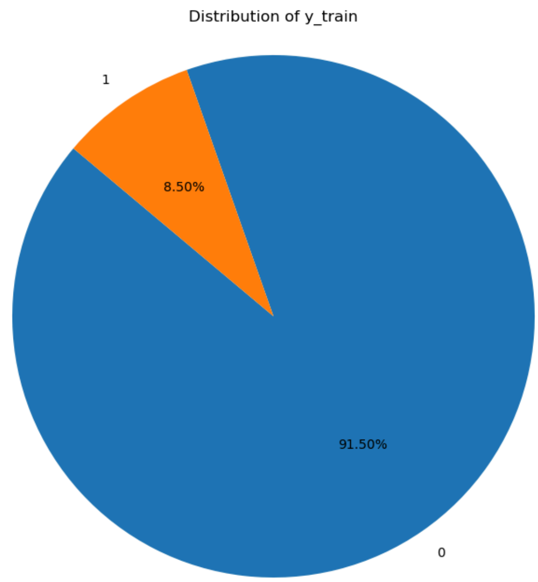
In [101…
```
confusion_matrix(y_pred,y_test)
```

Out[101…
```
array([[18114,   622],
       [  183,  1081]], dtype=int64)
```

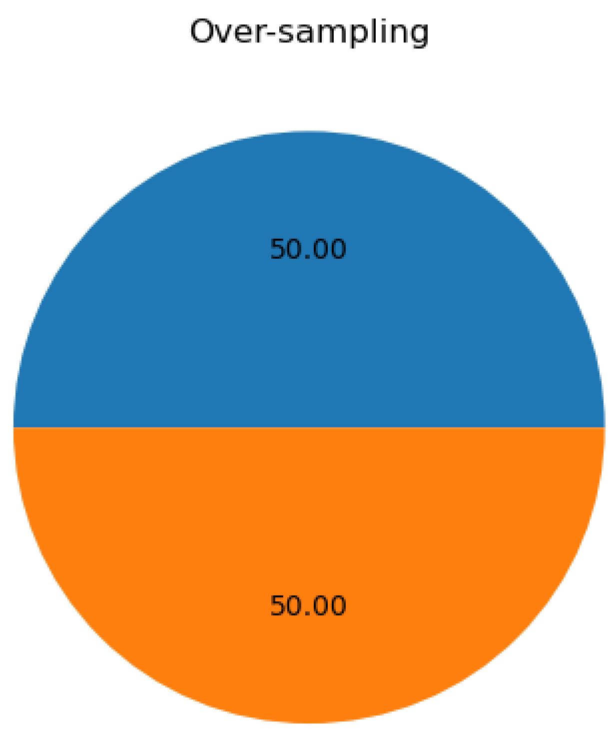
In [103…
```
y_train.value_counts()
```

Out[103…
```
diabetes
0    73203
1     6797
Name: count, dtype: int64
```

In [105…
```
value_counts=y_train.value_counts()

plt.figure(figsize=(16, 8))

plt.pie(value_counts, labels=value_counts.index, autopct='%1.2f%%', startangle=1

plt.title('Distribution of y_train')

plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



In [107…
```
from imblearn.over_sampling import SMOTE # using smote function to balance our s

smote=SMOTE()

X_ovs,y_ovs=smote.fit_resample(X,y) #passing X and y variables to it to balance

fig, oversp = plt.subplots()

oversp.pie( y_ovs.value_counts(), autopct='%.2f')

oversp.set_title("Over-sampling")

plt.show()
```



In [109…
```
Xr_train,Xr_test,yr_train,yr_test=train_test_split(X_ovs,y_ovs,train_size=0.7,ra
```

In [111…
```
print('train data shape')

print(Xr_train.shape)

print(yr_train.shape)

print('test data shape')

print(Xr_test.shape)

print(yr_test.shape)
```

```
train data shape
(128099, 6)
(128099,)
test data shape
(54901, 6)
(54901,)
```

In [113…
```
print('y_train and y_test value_count')
print(yr_train.value_counts())
print(yr_test.value_counts())
```

```
y_train and y_test value_count
diabetes
0    64131
1    63968
Name: count, dtype: int64
diabetes
1    27532
0    27369
Name: count, dtype: int64
```

In [115…
```
ss=StandardScaler()

ss
```

Out[115…
```
▾ StandardScaler
StandardScaler()
```

In [117…
```
data=Xr_train,Xr_test



xr_train_sc=ss.fit_transform(Xr_train) # scaling our resampling data xr train


Xr_test_sc=ss.fit_transform(Xr_test) # scaling our resamplig xr_test data
```

In [119…
```
Xr_train_scaled = pd.DataFrame(xr_train_sc) #Xr_train_scaled converting into the

print(Xr_train_scaled.shape)
Xr_train_scaled.head()
print(yr_train.shape)
```

```
(128099, 6)
(128099,)
```

In [121…
```
Xr_test_scaled=pd.DataFrame(Xr_test_sc) #Xr_test converting into the dataframe

print(Xr_test_scaled.shape)
Xr_test_scaled.head()
```

```
(54901, 6)
```

Out[121…

	0	1	2	3	4	5
0	0.648387	-0.293492	-0.206627	0.841287	-0.245012	-0.653366
1	-1.095183	-0.293492	-0.206627	-0.404988	0.369655	-0.057241
2	-1.466964	-0.293492	-0.206627	-0.288104	0.369655	-1.459888
3	-0.769875	3.407246	-0.206627	0.288161	0.369655	-1.372222
4	-1.374018	-0.293492	-0.206627	-0.288104	-2.152519	-1.109226

In [123…
```
model_lk=LogisticRegression()


model_lk.fit(Xr_train_scaled,yr_train)  #trining the model
```

Out[123…
```
▾ LogisticRegression
LogisticRegression()
```

In [125…
```
y_pred_lr=model_lk.predict(Xr_test_scaled) #predecting yr_test data
y_pred_lr[:10]
```

Out[125…
```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 1], dtype=int64)
```

In [127…
```
yr_test[:10]
```

Out[127…
```
180328    1
573       0
13494     0
93981     0
75389     0
180973    1
71021     0
19293     0
16393     0
121419    1
Name: diabetes, dtype: int64
```

In [129…
```
#classification_report for predict value and orginal value

print(classification_report(y_pred_lr,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88     27317
           1       0.89      0.88      0.88     27584

    accuracy                           0.88     54901
   macro avg       0.88      0.88      0.88     54901
weighted avg       0.88      0.88      0.88     54901
```

In [131…
```
#confusion_matrix for predict value and orginal value

confusion_matrix(y_pred_lr,yr_test)
```

Out[131…
```
array([[24165,  3152],
       [ 3204, 24380]], dtype=int64)
```

Decision Tree Classifier

In [134…
```
# activating DecisionTree Classifier
model_dtc=DecisionTreeClassifier()

# passing xr_train_scaled, yr_train to trining the model
model_dtc.fit(Xr_train_scaled,yr_train)

model_dtc
```

Out[134…
```
DecisionTreeClassifier

DecisionTreeClassifier()
```

In [136…
```
y_pred_dtc=model_dtc.predict(Xr_test_scaled) # predicting yr_test data
```

In [138…
```
# classification report for decisionTreeclassifier

print(classification_report(y_pred_dtc,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.79      0.99      0.88     21954
           1       0.99      0.83      0.90     32947

    accuracy                           0.89     54901
   macro avg       0.89      0.91      0.89     54901
weighted avg       0.91      0.89      0.89     54901
```

In [140…
```
confusion_matrix(y_pred_dtc,yr_test)
```

Out[140…
```
array([[21675,   279],
       [ 5694, 27253]], dtype=int64)
```

In [142…
```
model_rfc=RandomForestClassifier() #activating the fuction

model_rfc.fit(Xr_train_scaled,yr_train)
```

Out[142…
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [144…
```
y_pred_rfc=model_rfc.predict(Xr_test_scaled)
```

In [146…
```
print(classification_report(y_pred_rfc,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     25424
           1       0.99      0.92      0.95     29477

    accuracy                           0.95     54901
   macro avg       0.95      0.95      0.95     54901
weighted avg       0.95      0.95      0.95     54901
```

In [150…
```
confusion_matrix(y_pred_rfc,yr_test)
```

Out[150…
```
array([[25054,   370],
       [ 2315, 27162]], dtype=int64)
```

XGBOOST

In [154…
```
model_xgb=XGBClassifier()

model_xgb.fit(Xr_train_scaled,yr_train)
```

Out[154…

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rou
nds=None,
              enable_categorical=False, eval_metric=None, feature_ty
pes=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_
bin=None,
```

In [156…

```
y_pred_xgb=model_xgb.predict(Xr_test_scaled)
```

In [158…

```
print(classification_report(y_pred_xgb,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.76      0.98      0.86     21230
           1       0.99      0.81      0.89     33671

    accuracy                           0.88     54901
   macro avg       0.88      0.90      0.87     54901
weighted avg       0.90      0.88      0.88     54901
```

In [160…

```
confusion_matrix(y_pred_xgb,yr_test)
```

Out[160…

```
array([[20896,   334],
       [ 6473, 27198]], dtype=int64)
```

finding the hyperparameter tuning and best param grid

In [163…

```
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.linear_model import LogisticRegression

# Define the parameter grid to search over
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization parameter
    'penalty': ['l1', 'l2']               # Penalty type
}

# Create a Logistic Regression model
logistic = LogisticRegression()

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=logistic, param_grid=param_grid, cv=10)

# Initialize an empty list to store the accuracy scores
accuracy_scores = []

# Perform cross-validation 10 times
for _ in range(10):
```

```
    # Fit the GridSearchCV object to the training data
    grid_search.fit(Xr_train_scaled, yr_train)

    # Get the best parameters
    best_params = grid_search.best_params_

    # Perform cross-validation with the best model
    cv_scores = cross_val_score(grid_search.best_estimator_, Xr_train_scaled, yr
       # Store the mean accuracy score
    accuracy_scores.append(cv_scores.mean())

# Print the accuracy scores obtained over 10 iterations
#print("Accuracy scores over 10 iterations:", accuracy_scores)
print("Accuracy scores over 10 iterations:", ["{:.2f}".format(score) for score i


# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best parameters found:", best_params)
print("Best cross-validation score:", best_score)
```

```
Accuracy scores over 10 iterations: ['0.88', '0.88', '0.88', '0.88', '0.88', '0.8
8', '0.88', '0.88', '0.88', '0.88']
Best parameters found: {'C': 0.001, 'penalty': 'l2'}
Best cross-validation score: 0.8848390832485136
```

Final Model

In [166…

```
from sklearn.linear_model import LogisticRegression

# Create a Logistic Regression model with the best parameters
final_model = LogisticRegression(C=0.001, penalty='l2')

# Fit the final model to the entire training dataset
final_model.fit(Xr_train_scaled, yr_train)
```

Out[166…

```
▾ LogisticRegression
LogisticRegression(C=0.001)
```

In [168…

```
import pickle

# Save the final model to a pickle file
with open('final_model.pkl', 'wb') as file:
    pickle.dump(final_model, file)
```

In [170…

```
import pickle
import numpy as np

# Load the model from the pickle file
with open('final_model.pkl', 'rb') as file:
    loaded_model = pickle.load(file)

# Define the mean and standard deviation of the training data
mean_values = [41.885856, 0.07485, 0.03942, 27.320767, 5.527507, 138.058060]
```

```
std_values = [22.516840, 0.26315, 0.194593, 6.636783, 1.070672, 40.708136]

# Define the input features for prediction
age = 30
hypertension = 0
heart_disease = 0
bmi = 100.0
HbA1c_level = 5.0
blood_glucose_level = 90

# Scale the input features manually
scaled_features = [(x - mean) / std for x, mean, std in zip(
    [age, hypertension, heart_disease, bmi, HbA1c_level, blood_glucose_level],
    mean_values, std_values
)]

# Make predictions on the scaled data
prediction = loaded_model.predict([scaled_features])

# Print the prediction
if prediction[0] == 1:
    print("Diabetic")
else:
    print("Not Diabetic")
```

```
Diabetic
```

Comleted

In [ ]: