Diabetes_Predection

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df=pd.read_csv(r"C:\Users\JANHAVI\Desktop\diabetes_prediction_dataset.csv")
```

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

Out[3]:

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_
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 [4]:
```
df.isna().any()
```

Out[4]:
```
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 [5]:
```
df.corr(numeric_only=True)
```

Out[5]:

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_le
age	1.000000	0.251171	0.233354	0.337396	0.101354	0.110
hypertension	0.251171	1.000000	0.121262	0.147666	0.080939	0.084
heart_disease	0.233354	0.121262	1.000000	0.061198	0.067589	0.070
bmi	0.337396	0.147666	0.061198	1.000000	0.082997	0.091
HbA1c_level	0.101354	0.080939	0.067589	0.082997	1.000000	0.166
blood_glucose_level	0.110672	0.084429	0.070066	0.091261	0.166733	1.000
diabetes	0.258008	0.197823	0.171727	0.214357	0.400660	0.419

In [6]:

```
df.shape
```

Out[6]:

```
(100000, 9)
```

Unique Elements

In [7]:

```
for column in df.columns:
    unique_values = df[column].unique()
    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.   4
2.   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 1
30 160 126 159  90 260 220 300 280 240]
Column "diabetes" has unique values: [0 1]
```

In [8]:

```
df["smoking_history"].value_counts()
```

Out[8]:

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

In [9]:

```
df["smoking_history"].value_counts()/len(df)
```

Out[9]:
```
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 [10]:
```
# 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 info v

# 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 [11]:
```
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 [12]:
```
df.gender.value_counts()
```

Out[12]:
```
gender
Female    58552
Male      41430
Other        18
Name: count, dtype: int64
```

In [13]:
```
df.describe()
```

Out[13]:

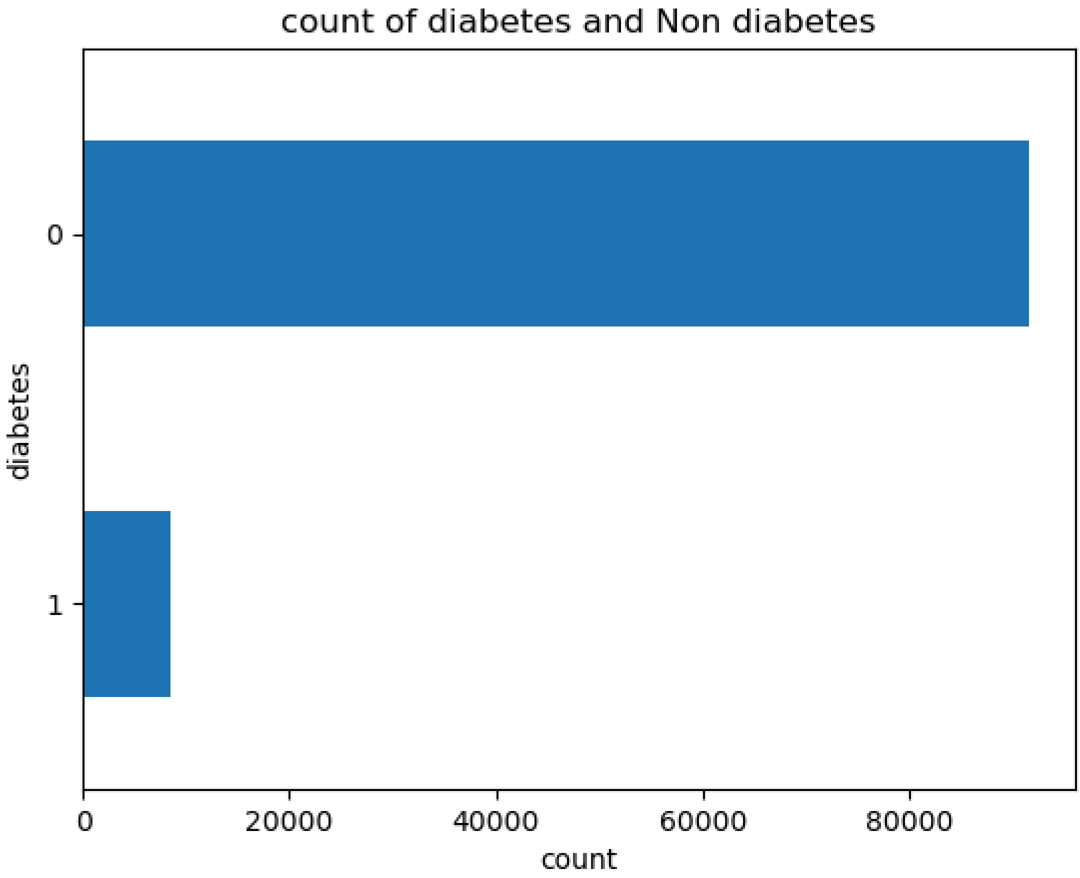
	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_l
count	100000.000000	100000.00000	100000.000000	100000.000000	100000.000000	100000.000
mean	41.885856	0.07485	0.039420	27.320767	5.527507	138.058
std	22.516840	0.26315	0.194593	6.636783	1.070672	40.708
min	0.080000	0.00000	0.000000	10.010000	3.500000	80.000
25%	24.000000	0.00000	0.000000	23.630000	4.800000	100.000
50%	43.000000	0.00000	0.000000	27.320000	5.800000	140.000
75%	60.000000	0.00000	0.000000	29.580000	6.200000	159.000
max	80.000000	1.00000	1.000000	95.690000	9.000000	300.000

In [14]:

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

In [15]:

```
#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 [16]:
```
df['diabetes'].value_counts()/len(df) #percentage of 1--diabetes and 2--no diabetes
```

Out[16]:
```
diabetes
0    0.915
1    0.085
Name: count, dtype: float64
```

In [17]:
```
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 [18]:
```
le=LabelEncoder() #activating label encoder function

le
```

Out[18]:
```
▾ LabelEncoder
▶ Parameters
```

In [19]:
```
Label_encod_columns=['gender','smoking_history']  #selecting columns to apply label

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

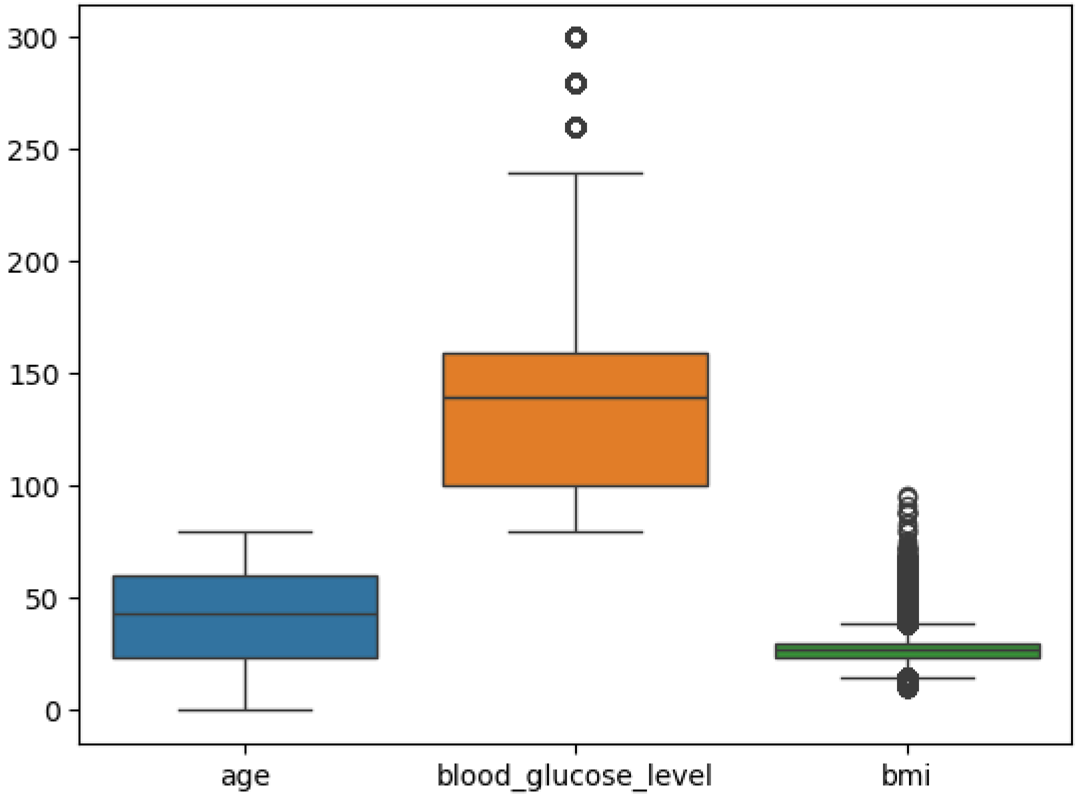
In [20]:
```
df.head(3)
```

Out[20]:

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_
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	

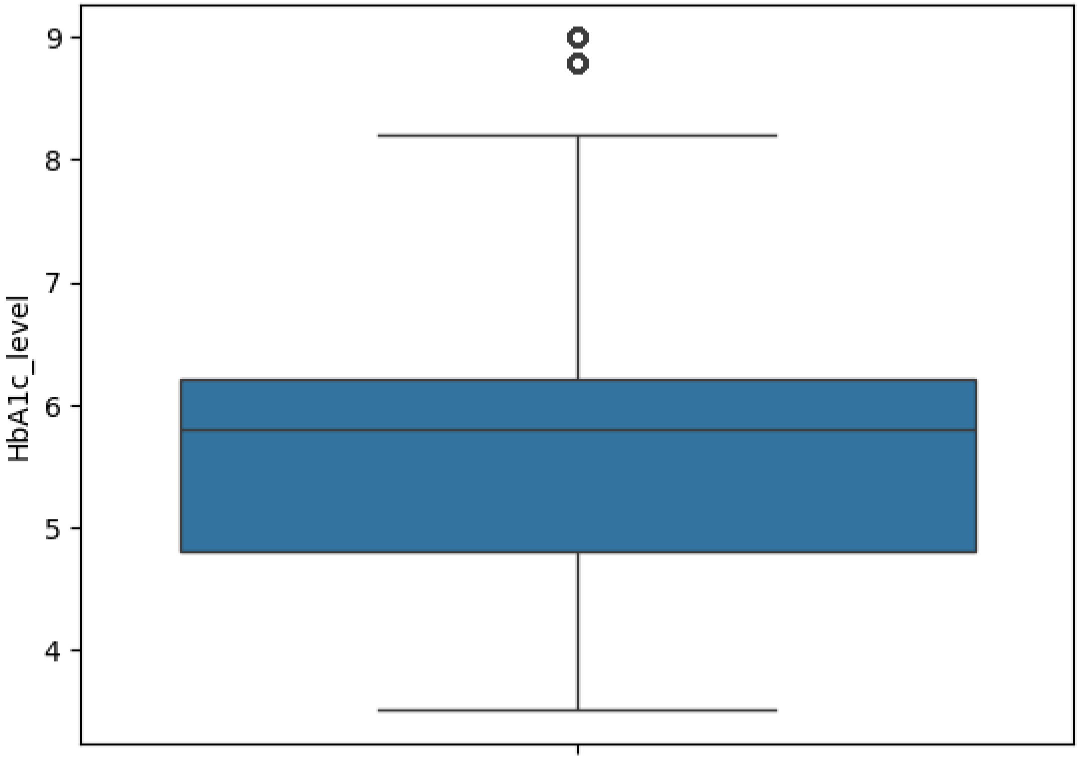
In [21]:
```
sns.boxplot(data=df[['age','blood_glucose_level','bmi']]) #checking outliers using
```

Out[21]:
```
<Axes: >
```



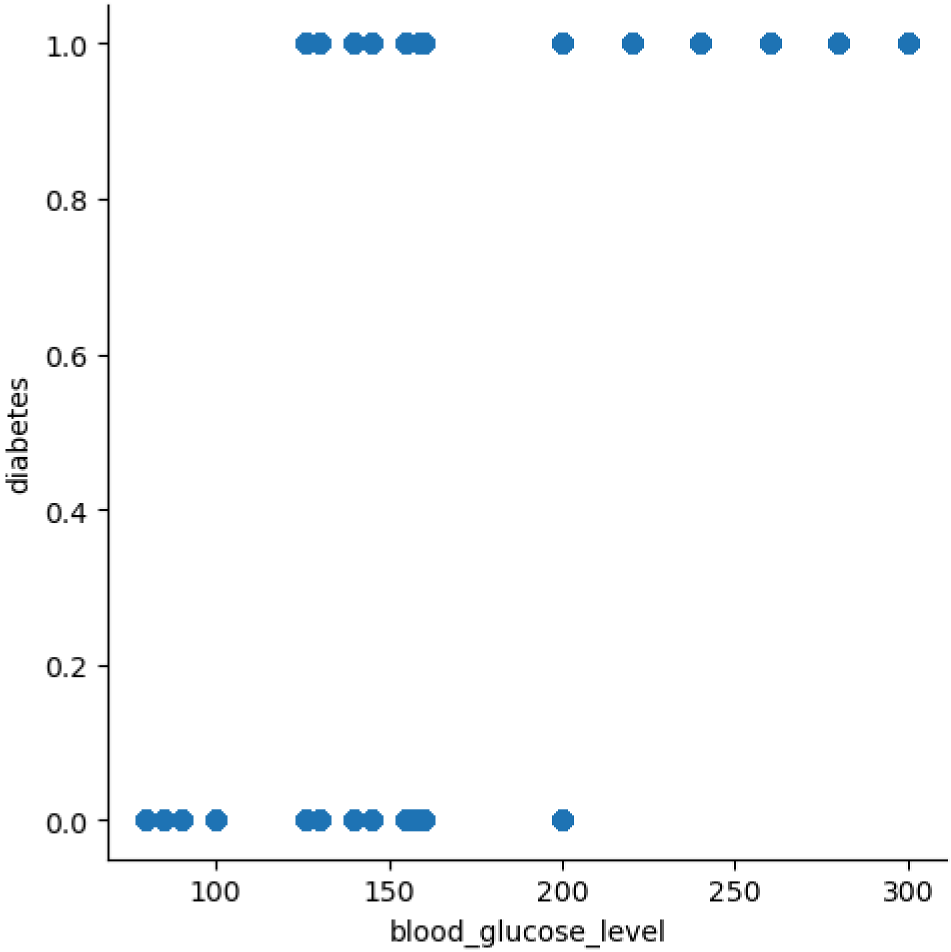
In [22]:
```
sns.boxplot(data=df['HbA1c_level'])
```

Out[22]:
```
<Axes: ylabel='HbA1c_level'>
```

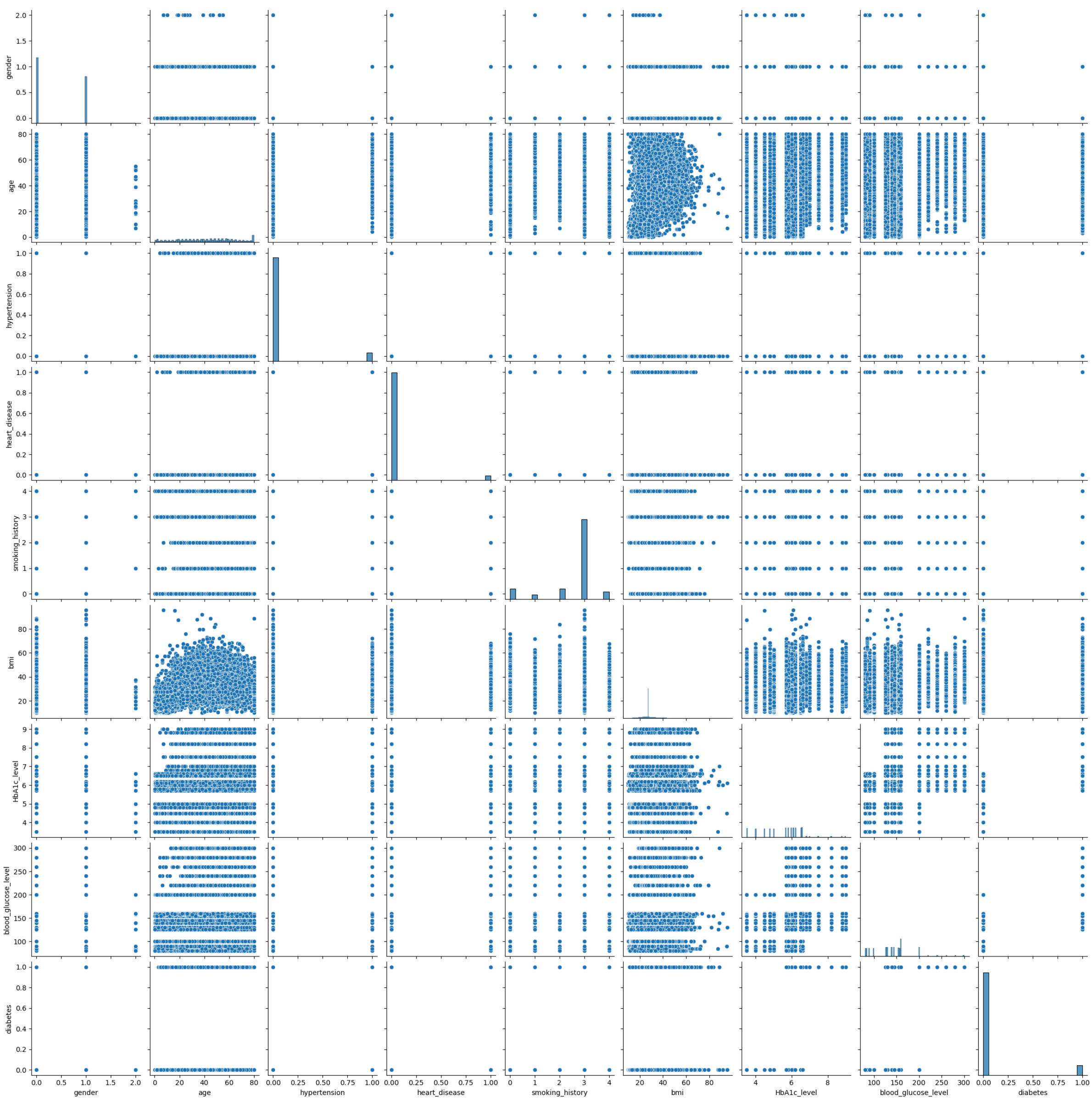


In [23]:
```
sns.lmplot(data=df, x='blood_glucose_level', y='diabetes', fit_reg=False)#implot pl
```

Out[23]: <seaborn.axisgrid.FacetGrid at 0x23078334c90>



In [24]:
```
sns.pairplot(df) #using pairplot to check relation between parameters

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



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

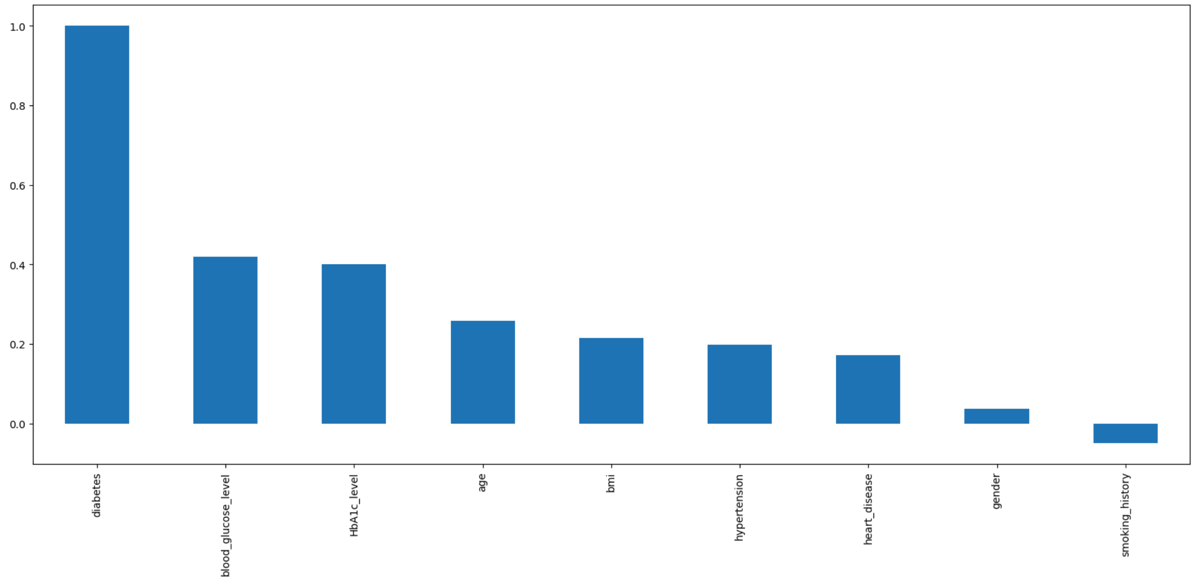
Out[25]:

	gender	age	hypertension	heart_disease	smoking_history	bmi
gender	1.000000	-0.030656	0.014203	0.077696	-0.044081	-0.022994
age	-0.030656	1.000000	0.251171	0.233354	-0.098969	0.337396
hypertension	0.014203	0.251171	1.000000	0.121262	-0.048631	0.147666
heart_disease	0.077696	0.233354	0.121262	1.000000	-0.048253	0.061198
smoking_history	-0.044081	-0.098969	-0.048631	-0.048253	1.000000	-0.087735
bmi	-0.022994	0.337396	0.147666	0.061198	-0.087735	1.000000
HbA1c_level	0.019957	0.101354	0.080939	0.067589	-0.017534	0.082997
blood_glucose_level	0.017199	0.110672	0.084429	0.070066	-0.022985	0.091261
diabetes	0.037411	0.258008	0.197823	0.171727	-0.049841	0.214357

In [26]:
```
plt.figure(figsize=(20,8)) #figsize
```

```
#printing graphical representations of
df.corr()['diabetes'].sort_values(ascending=False).plot(kind='bar')
```

Out[26]: <Axes: >



In [27]:
```
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 [28]:
```
#selecting X variables
X = df.loc[:, 'age':'heart_disease'].join(df.loc[:, 'bmi':'blood_glucose_level'])


X
```

Out[28]:

	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 [29]:
```
y=df.loc[:,'diabetes']  #y variable

y #printing y variable
```

Out[29]:
```
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 [30]:
```
# spliting trining and testing data in 70 30 rating testing size is 0.3 random_stat

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

In [31]:
```
X_train.head()
```

Out[31]:

	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 [32]:
```
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 [33]:
```
ss=StandardScaler() #activating StandardScaler()

ss
```

Out[33]:

```
StandardScaler
▶ Parameters
```

In [34]:
```
X_train_scaled=ss.fit_transform(X_train) #scaling X_train data
```

In [35]:
```
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 [36]:
```
model_lr=LogisticRegression()  #activating logistic Regression
```

In [37]:
```
model_lr.fit(X_train_scaled,y_train) #training logistic regression model
```

Out[37]:

```
LogisticRegression
▶ Parameters
```

In [38]:
```
y_pred=model_lr.predict(X_test_scaled) #predecting y_test data
y_pred[:10]
```

Out[38]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [39]:
```
y_test[:10] # actual y_test data
```

Out[39]:
```
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 [40]:
```
accuracy_score(y_pred,y_test) #accuracy_score
```

Out[40]: 0.95975

In [41]:
```
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
```

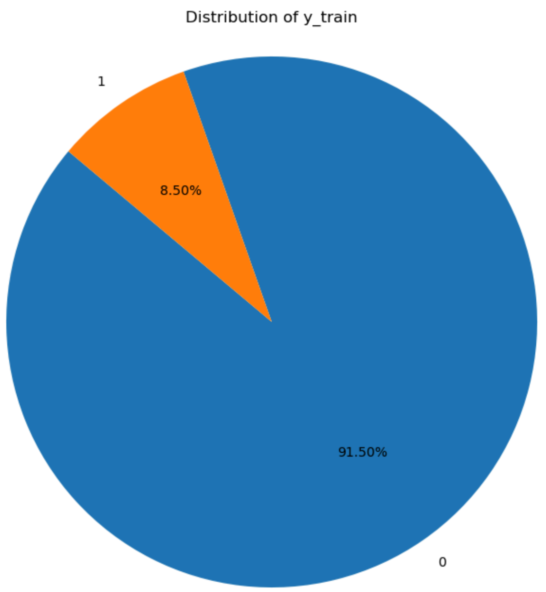
In [42]:
```
confusion_matrix(y_pred,y_test) #confusion_matrix
```

Out[42]:
```
array([[18114,   622],
       [  183,  1081]], dtype=int64)
```

In [43]:
```
y_train.value_counts() #data is highly imblancing
```

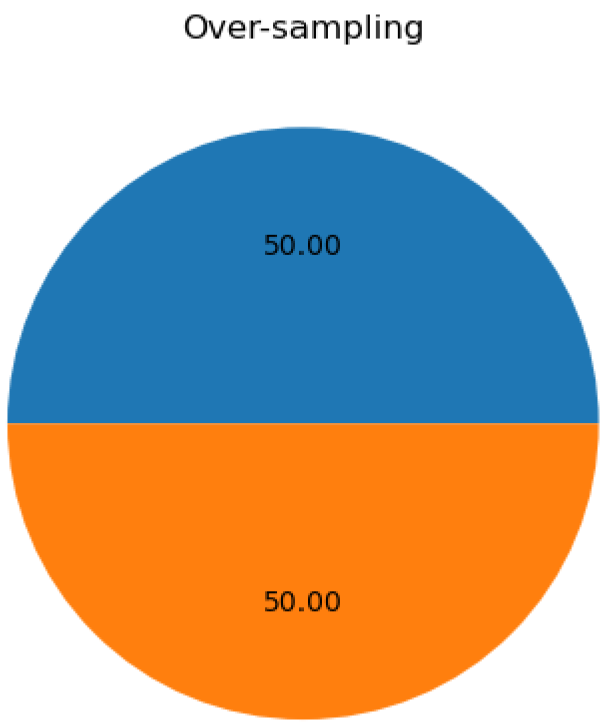
Out[43]:
```
diabetes
0    73203
1     6797
Name: count, dtype: int64
```

In [44]:
```
value_counts=y_train.value_counts()

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

plt.pie(value_counts, labels=value_counts.index, autopct='%1.2f%%', startangle=140)

plt.title('Distribution of y_train')

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

plt.show()
```



In [45]:
```
from imblearn.over_sampling import SMOTE # using smote function to balance our set

smote=SMOTE()

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

```
fig, oversp = plt.subplots()

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

oversp.set_title("Over-sampling")

plt.show()
```



In [46]:
```
# Dividing our resampling data into 70 30 ratio

Xr_train,Xr_test,yr_train,yr_test=train_test_split(X_ovs,y_ovs,train_size=0.7,rando
```

In [47]:
```
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 [48]:
```
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 [49]:
```
ss=StandardScaler()

ss
```

Out[49]:
```
▾ StandardScaler
▶ Parameters
```

In [50]:
```
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 [51]:
```
Xr_train_scaled = pd.DataFrame(xr_train_sc) #Xr_train_scaled converting into the da

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

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

In [52]:
```
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[52]:

	0	1	2	3	4	5
0	0.957108	-0.294735	-0.204364	0.539410	-0.111145	2.029416
1	-1.095860	-0.294735	-0.204364	-0.404865	0.370604	-0.064547
2	-1.467180	-0.294735	-0.204364	-0.287066	0.370604	-1.460522
3	-0.770954	3.392878	-0.204364	0.293712	0.370604	-1.373274
4	-1.374350	-0.294735	-0.204364	-0.287066	-2.150914	-1.111528

In [53]:
```
model_lk=LogisticRegression()


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

Out[53]:

```
▾ LogisticRegression

▶ Parameters
```

In [54]:

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

Out[54]:

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

In [55]:

```
yr_test[:10]
```

Out[55]:

```
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 [56]:

```
#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     27383
           1       0.88      0.88      0.88     27518

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

In [57]:

```
#confusion_matrix for predict value and orginal value

confusion_matrix(y_pred_lr,yr_test)
```

Out[57]:

```
array([[24174,  3209],
       [ 3195, 24323]], dtype=int64)
```

Decision Tree Classifier

In [58]:

```
# 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[58]:

```
▾ DecisionTreeClassifier

▶ Parameters
```

In [59]:
```
y_pred_dtc=model_dtc.predict(Xr_test_scaled) # predicting yr_test data
```

In [60]:
```
# classification report for decisionTreeclassifier

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

```
              precision    recall  f1-score   support

           0       0.64      1.00      0.78     17500
           1       1.00      0.73      0.85     37401

    accuracy                           0.82     54901
   macro avg       0.82      0.87      0.81     54901
weighted avg       0.88      0.82      0.82     54901
```

In [61]:
```
confusion_matrix(y_pred_dtc,yr_test)
```

Out[61]:
```
array([[17433,    67],
       [ 9936, 27465]], dtype=int64)
```

Random Forest Classifier

In [62]:
```
model_rfc=RandomForestClassifier() #activating the fuction

model_rfc.fit(Xr_train_scaled,yr_train)
```

Out[62]:
```
▾ RandomForestClassifier
▶ Parameters
```

In [63]:
```
y_pred_rfc=model_rfc.predict(Xr_test_scaled)
```

In [64]:
```
print(classification_report(y_pred_rfc,yr_test))
```

```
              precision    recall  f1-score   support

           0       0.78      0.99      0.87     21526
           1       0.99      0.82      0.90     33375

    accuracy                           0.89     54901
   macro avg       0.89      0.90      0.88     54901
weighted avg       0.91      0.89      0.89     54901
```

In [65]:
```
confusion_matrix(y_pred_rfc,yr_test)
```

Out[65]:
```
array([[21306,   220],
       [ 6063, 27312]], dtype=int64)
```

XGBOOST

In [66]:
```
model_xgb=XGBClassifier()

model_xgb.fit(Xr_train_scaled,yr_train)
```

Out[66]:

```
▾ XGBClassifier
▶ Parameters
```

In [67]:

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

In [68]:

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

```
              precision    recall  f1-score   support

           0       0.89      0.96      0.92     25342
           1       0.96      0.90      0.93     29559

    accuracy                           0.93     54901
   macro avg       0.93      0.93      0.93     54901
weighted avg       0.93      0.93      0.93     54901
```

In [69]:

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

Out[69]:

```
array([[24305,  1037],
       [ 3064, 26495]], dtype=int64)
```

Hyperparameter Tunning & Best Parameter GridSearchcv

In [ ]:

```
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_tr

    # 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 in a


# 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)
```

Final Model

In [74]:
```
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[74]:
```
▾ LogisticRegression
▶ Parameters
```

In [75]:
```
import pickle

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

In [76]:
```
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
```

In [ ]: