DATASET - DIABETES PREDICTION DATASET

https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset/data

INDEX

INTRODUCTION

The aim of this analysis is to investigate a range of health-related factors and their interconnections **to classify diabetes accurately**. These factors include aspects such as **age**, **gender**, **body mass index (BMI)**, **hypertension**, **heart disease**, **smoking history**, **HbA1c level**, and **blood glucose level**.

This comprehensive examination will not only provide insights into the patterns and trends in diabetes risk but will also create a solid base for further research. Specifically, research can be built on how these variables interact and influence diabetes occurrence and progression, crucial knowledge for improving patient care and outcomes in this increasingly critical area of healthcare.

DOMAIN KNOWLEDGE

- **AGE** - Age is an important factor in predicting diabetes risk. As individuals get older, their risk of developing diabetes increases. This is partly due to factors such as reduced physical activity, changes in hormone levels, and a higher likelihood of developing other health conditions that can contribute to diabetes.

- **GENDER** - Gender can play a role in diabetes risk, although the effect may vary. For example, women with a history of gestational diabetes (diabetes during pregnancy) have a higher risk of developing type 2 diabetes later in life. Additionally, some studies have suggested that men may have a slightly higher risk of diabetes compared to women.

- **BODY MASS INDEX (BMI)** - BMI is a measure of body fat based on a person's height and weight. It is commonly used as an indicator of overall weight status and can be helpful in predicting diabetes risk. Higher BMI is associated with a greater likelihood of developing type 2 diabetes. Excess body fat, particularly around the waist, can lead to insulin resistance and impair the body's ability to regulate blood sugar levels.

- **HYPERTENSION** - Hypertension, or high blood pressure, is a condition that often coexists with diabetes. The two conditions share common risk factors and can contribute to each other's development. Having hypertension increases the risk of developing type 2 diabetes and vice versa. Both conditions can have detrimental effects on cardiovascular health.

- **HEART DISEASE** - Heart Disease, including conditions such as coronary artery disease and heart failure, is associated with an increased risk of diabetes. The relationship between heart disease and diabetes is bidirectional, meaning that having one condition increases the risk of developing the other. This is because they share many common risk factors, such as obesity, high blood pressure, and high cholesterol.

- **SMOKING HISTORY** - Smoking is a modifiable risk factor for diabetes. Cigarette smoking has been found to increase the risk of developing type 2 diabetes. Smoking can contribute to insulin resistance and impair glucose metabolism. Quitting smoking can significantly reduce the risk of developing diabetes and its complications.

- **HbA1c LEVEL -** HbA1c (glycated hemoglobin) is a measure of the average blood glucose level over the past 2-3 months. It provides information about long-term blood sugar control. Higher HbA1c levels indicate poorer glycemic control and are associated with an increased risk of developing diabetes and its complications.

- **BLOOD GLUCOSE LEVEL -** Blood glucose level refers to the amount of glucose (sugar) present in the blood at a given time. Elevated blood glucose levels, particularly in the fasting state or after consuming carbohydrates, can indicate impaired glucose regulation and increase the risk of developing diabetes. Regular monitoring of blood glucose levels is important in the diagnosis and management of diabetes.

➢ **These features, when combined and analyzed with appropriate statistical and machine learning techniques, can help in predicting an individual's risk of developing diabetes.**

CLASSIFICATION ALGORITHMS

- **NAÏVE BAYES ALGORITHM** - In machine learning, Naïve Bayes classification is a straightforward and powerful algorithm for the classification task. Naïve Bayes classification is based on applying Bayes' theorem with strong independence assumption between the features. Naïve Bayes classification produces good results when we use it for textual data analysis such as Natural Language Processing.

  Naïve Bayes models are also known as simple Bayes or independent Bayes. All these names refer to the application of Bayes' theorem in the classifier's decision rule. Naïve Bayes classifier applies the Bayes' theorem in practice. This classifier brings the power of Bayes' theorem to machine learning.

- **K – NEAREST NEIGHBORS ALGORITHM** - In machine learning, k Nearest Neighbors or kNN is the simplest of all machine learning algorithms. It is a non-parametric algorithm used for classification and regression tasks. Non-parametric means there is no assumption required for data distribution. So, kNN does not require any underlying assumption to be made. In both classification and regression tasks, the input consists of the k closest training examples in the feature space. The output depends upon whether kNN is used for classification or regression purposes.

  kNN is a type of instance-based learning or lazy learning. Lazy learning means it does not require any training data points for model generation. All training data will be used in the testing phase. This makes training faster and testing slower and costlier. So, the testing phase requires more time and memory resources.

- **DECISION – TREE CLASSIFIER** - A Decision Tree algorithm is one of the most popular machine learning algorithms. It uses a tree like structure and their possible combinations to solve a particular problem. It belongs to the class of supervised learning algorithms where it can be used for both classification and regression purposes.

[1]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import  train_test_split, StratifiedKFold,␣
 ↪cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay)
from sklearn.preprocessing import StandardScaler
import category_encoders as ce
```

2.2 IMPORT DATASET

[2]:
```
df=pd.read_csv('diabetes_prediction_dataset.csv')
```

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

[3]:
```
   gender   age  hypertension  heart_disease smoking_history    bmi  \
0  Female  80.0             0              1           never  25.19
```

```
1  Female  54.0                0               0         No Info  27.32
2    Male  28.0                0               0           never  27.32
3  Female  36.0                0               0         current  23.45
4    Male  76.0                1               1         current  20.14

   HbA1c_level  blood_glucose_level  diabetes
0          6.6                  140         0
1          6.6                   80         0
2          5.7                  158         0
3          5.0                  155         0
4          4.8                  155         0
```

[4]:
```
df.shape
```

[4]:
```
(100000, 9)
```

[5]:
```
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
```

UNIQUE VALUES

[6]:
```
d = []
u = []
t = []
for col in df:
    d.append(col)
    u.append(df[col].nunique())
    t.append(df[col].dtype)
pd.DataFrame({'column':d,'type': t ,'unique value' : u})
```

[6]:
```
                column     type  unique value
0               gender   object             3
1                  age  float64           102
```

```
2         hypertension    int64              2
3        heart_disease    int64              2
4      smoking_history   object              6
5                  bmi  float64           4247
6          HbA1c_level  float64             18
7  blood_glucose_level    int64             18
8             diabetes    int64              2
```

[7]:
```
df['gender'].unique()
```

[7]:
```
array(['Female', 'Male', 'Other'], dtype=object)
```

[8]:
```
df['smoking_history'].unique()
```

[8]:
```
array(['never', 'No Info', 'current', 'former', 'ever', 'not current'],
      dtype=object)
```

2.3 EXPLORATORY DATA ANALYSIS (PREPROCESSING AND DATASET CLEANING)

[9]:
```
# MISSING VALUES

df.isnull().sum()
```

[9]:
```
gender                 0
age                    0
hypertension           0
heart_disease          0
smoking_history        0
bmi                    0
HbA1c_level            0
blood_glucose_level    0
diabetes               0
dtype: int64
```

[10]:
```
# HANDLING DUPLICATES

df.duplicated().sum()
```

[10]:
```
3854
```

[11]:
```
df.drop_duplicates(inplace=True)
```

[12]:
```
df.duplicated().sum()
```

[12]:
```
0
```

SUMMARY STATISTICS

[13]:
```
df.describe(include='object')
```

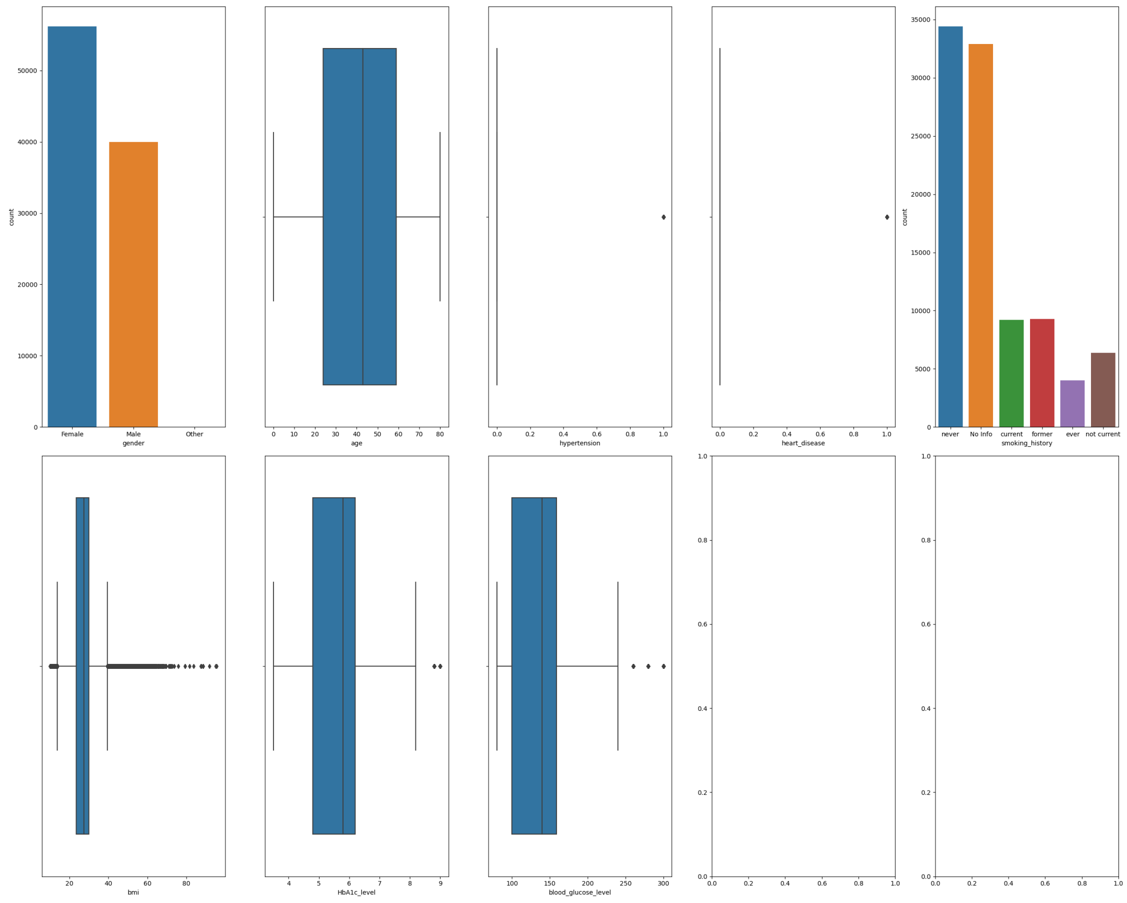
[13]:
```
        gender smoking_history
count    96146           96146
unique       3               6
top     Female           never
freq     56161           34398
```

[14]:
```
df.describe()
```

[14]:
```
                age  hypertension  heart_disease           bmi   HbA1c_level  \
count  96146.000000  96146.000000   96146.000000  96146.000000  96146.000000
mean      41.794326      0.077601       0.040803     27.321461      5.532609
std       22.462948      0.267544       0.197833      6.767716      1.073232
min        0.080000      0.000000       0.000000     10.010000      3.500000
25%       24.000000      0.000000       0.000000     23.400000      4.800000
50%       43.000000      0.000000       0.000000     27.320000      5.800000
75%       59.000000      0.000000       0.000000     29.860000      6.200000
max       80.000000      1.000000       1.000000     95.690000      9.000000

       blood_glucose_level      diabetes
count         96146.000000  96146.000000
mean            138.218231      0.088220
std              40.909771      0.283616
min              80.000000      0.000000
25%             100.000000      0.000000
50%             140.000000      0.000000
75%             159.000000      0.000000
max             300.000000      1.000000
```

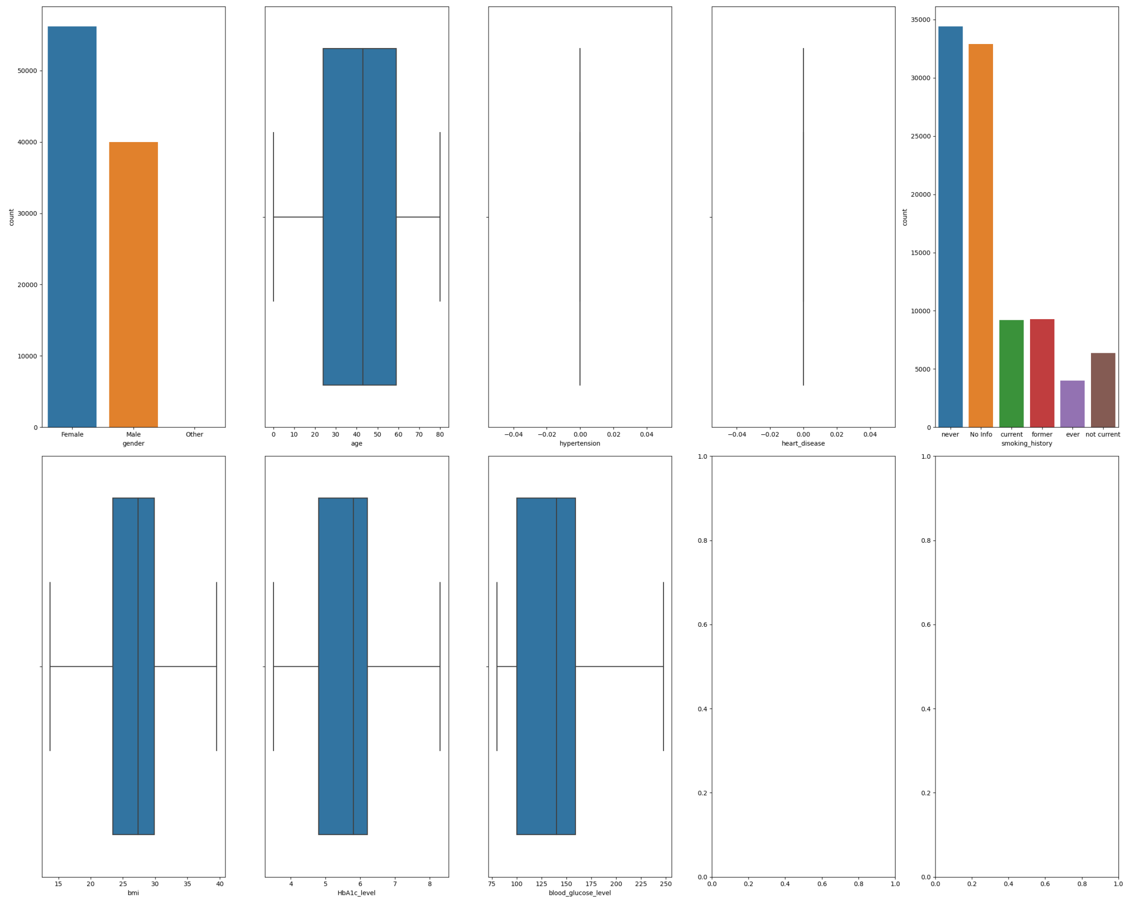
[15]:
```
# DETECTING OUTLIERS

def boxplots(df):
    cols = df.columns[:-1]
    n = 2
    m = 5
    fig, axes = plt.subplots(nrows=n, ncols=m, figsize=(25,20))
    for idx, col in enumerate(cols):
        i = idx // m
        j = idx % m
        if df[col].dtype == 'object':
            sns.countplot(data=df, x=col, ax=axes[i][j])
        else:
            sns.boxplot(data=df, x=col, ax=axes[i][j])
    plt.tight_layout()
    plt.show()
```

```
boxplots(df)
```



[16]:
```
# FIXING OUTLIERS

def outliers_removal(df, col):
    if df[col].dtype != 'object':
        perc = np.percentile(df[col], [0, 25, 50, 75, 100])
        iqr = perc[3] - perc[1]
        _min = perc[1] - 1.5*iqr
        _max = perc[3] + 1.5*iqr
        df.loc[df[col] > _max, col] = _max
        df.loc[df[col] < _min, col] = _min
    return df

for col in df.columns[:-1]:
    df = outliers_removal(df, col)
```

[17]:
```
boxplots(df)
```



```
[18]: df.shape
```

```
[18]: (96146, 9)
```

EXPLORING CATEGORICAL FEATURES

```
[19]: categorical = [var for var in df.columns if df[var].dtype=='O']
      print('There are {} categorical variable(s)\n'.format(len(categorical)))
      print('The categorical variable(s) are :\n\n', categorical)
```

```
There are 2 categorical variable(s)

The categorical variable(s) are :

 ['gender', 'smoking_history']
```

EXPLORING NUMERICAL FEATURES

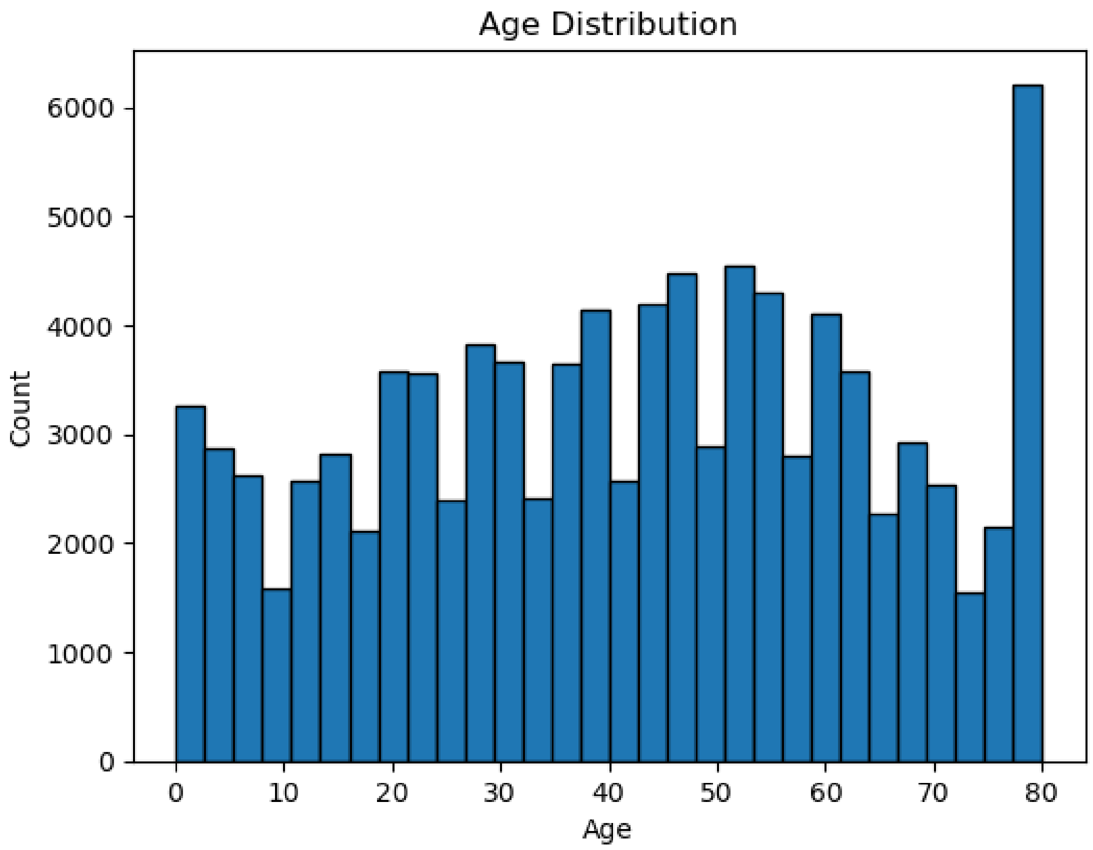
```
[20]: numerical=[var for var in df.columns if df[var].dtype!='O']
      print('There are {} numerical variable(s)\n'.format(len(numerical)))
      print('The numerical variable(s) are :\n\n', numerical)
```

```
There are 7 numerical variable(s)

The numerical variable(s) are :

 ['age', 'hypertension', 'heart_disease', 'bmi', 'HbA1c_level',
'blood_glucose_level', 'diabetes']
```

2.4 VISUALIZATION

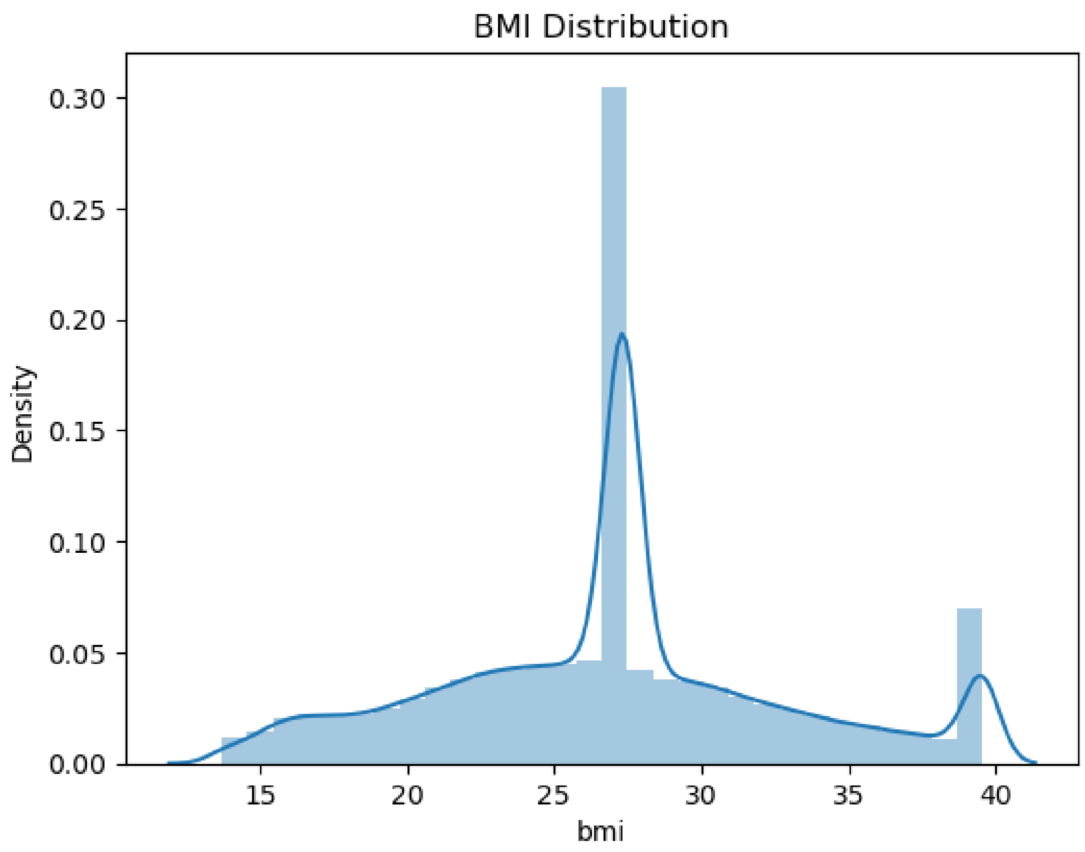
UNIVARIATE ANALYSIS

[21]:
```python
# HISTOGRAM FOR 'age'

plt.hist(df['age'], bins=30, edgecolor='black')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```



[22]:
```python
# DISTRIBUTION PLOT FOR 'bmi'
```

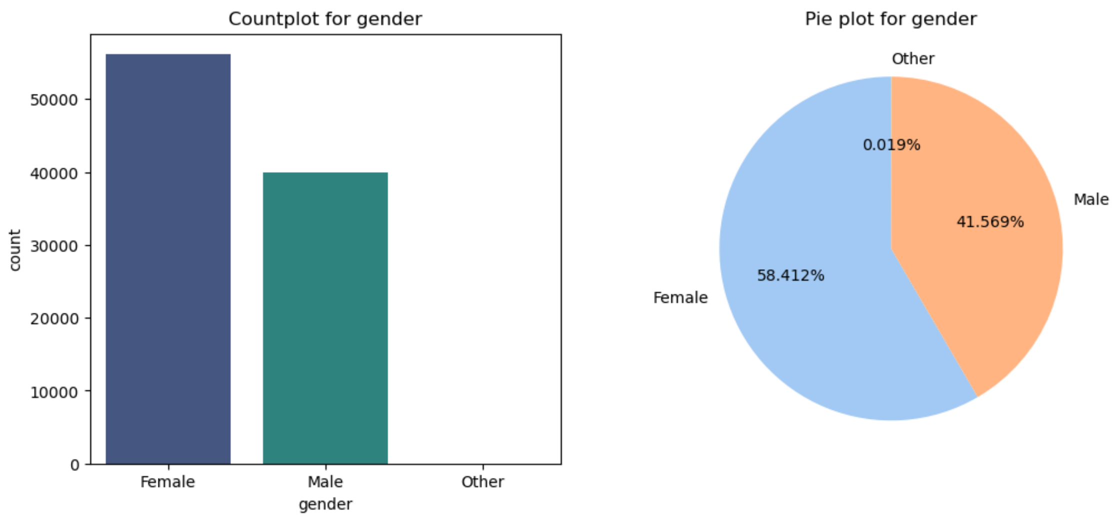
```
sns.distplot(df['bmi'], bins=30)
plt.title('BMI Distribution')
plt.show()
```



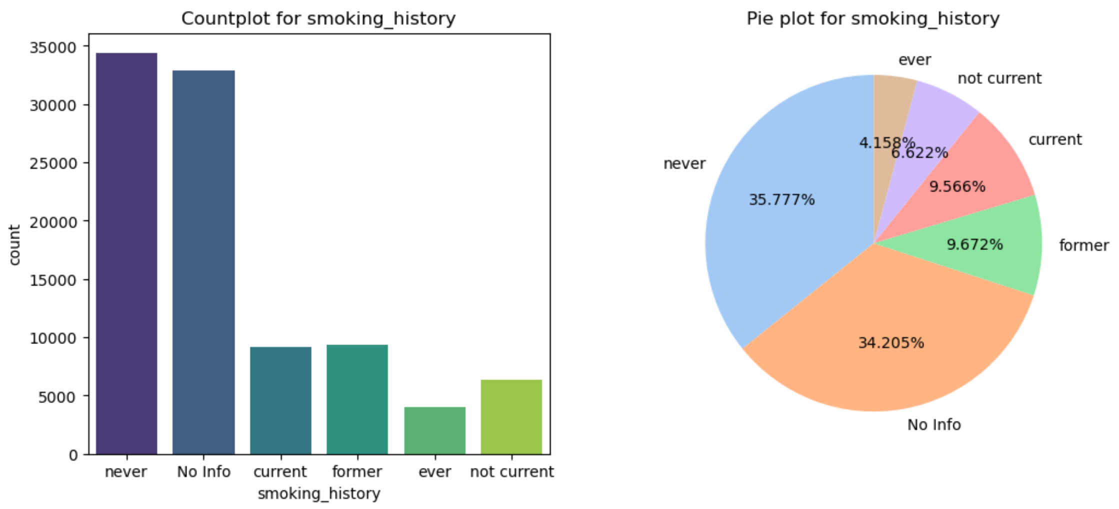
[23]:
```
def univariate_analysis_cat(col):
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))

    # COUNT PLOT
    sns.countplot(x=df[col],data=df ,palette='viridis', ax=ax[0])
    ax[0].set_title(f'Countplot for {col}')

    # PIE PLOT
    data_counts = df[col].value_counts()
    ax[1].pie(data_counts, labels=data_counts.index, autopct='%1.3f%%',␣
 ↪startangle=90, colors=sns.color_palette('pastel'))
    ax[1].set_title(f'Pie plot for {col}')

    plt.show()
```

[24]:
```
for col in categorical:
    print(f' Univariate Analysis for {col} feature:')
    univariate_analysis_cat(col)
```

```
 Univariate Analysis for gender feature:
```

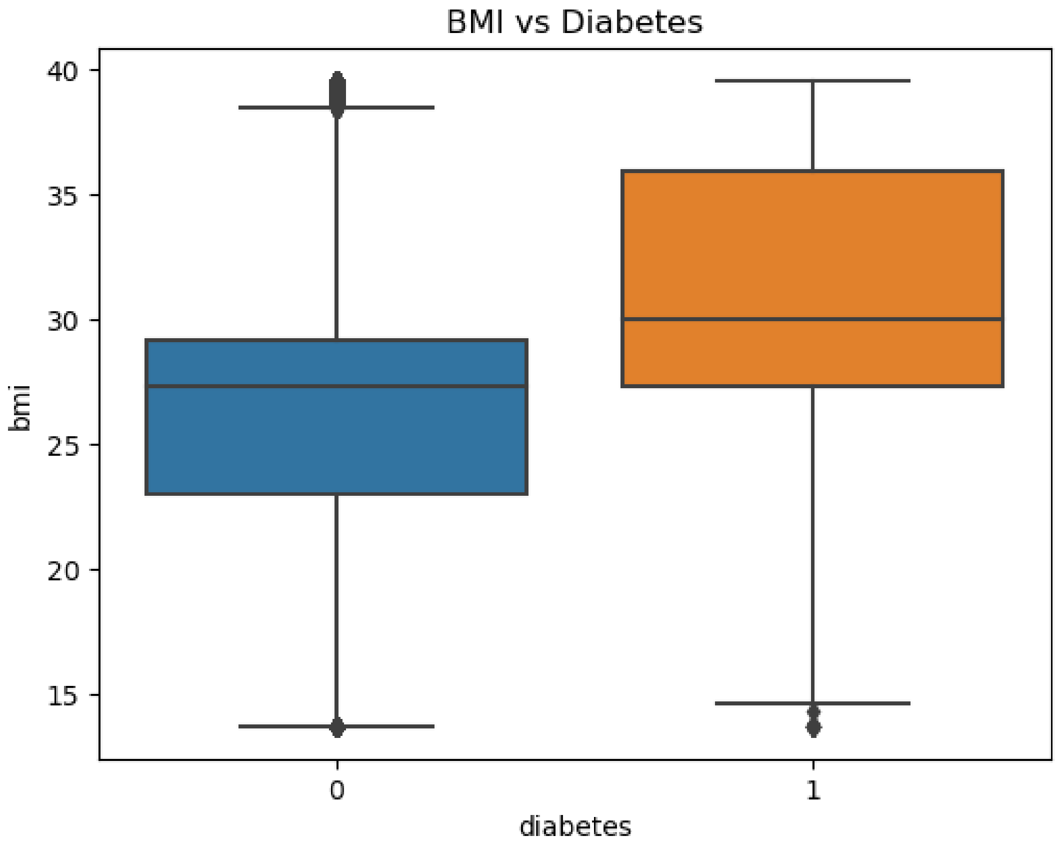


```
 Univariate Analysis for smoking_history feature:
```

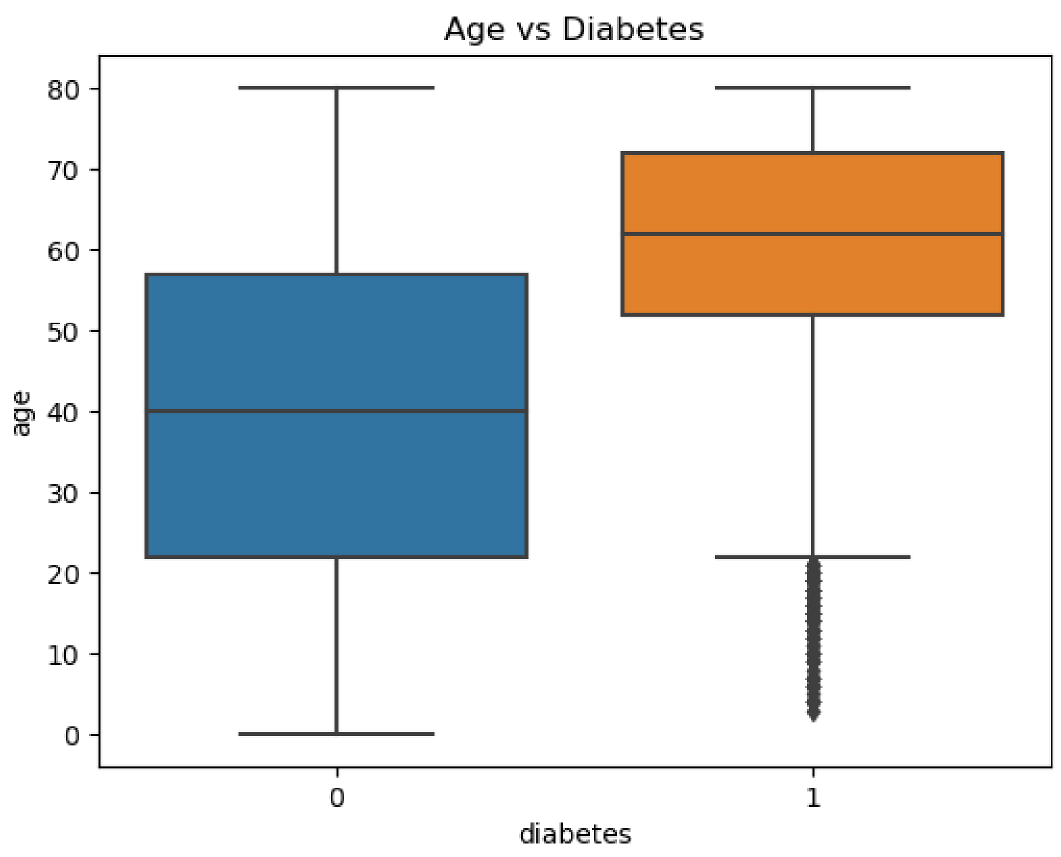


BIVARIATE ANALYSIS

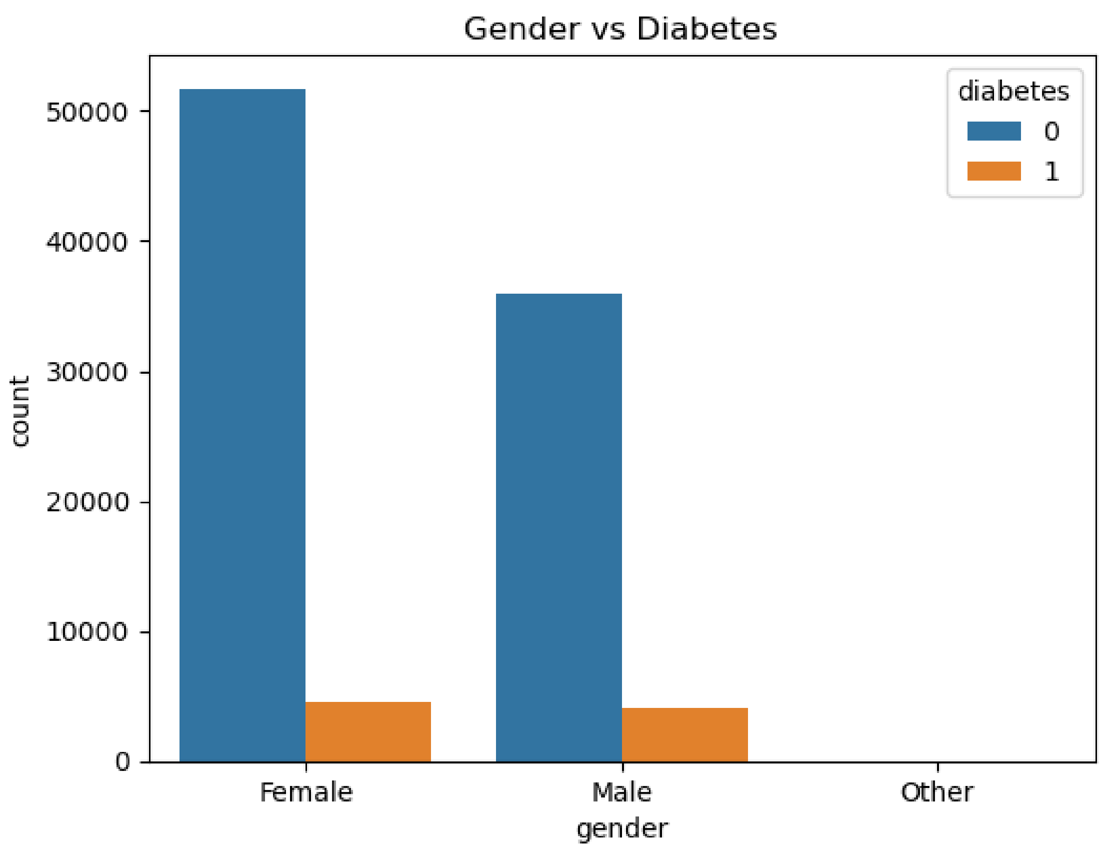
[25]:
```
# BOXPLOT FOR 'bmi' vs 'diabetes' CLASSIFICATION

sns.boxplot(x='diabetes', y='bmi', data=df)
plt.title('BMI vs Diabetes')
plt.show()
```



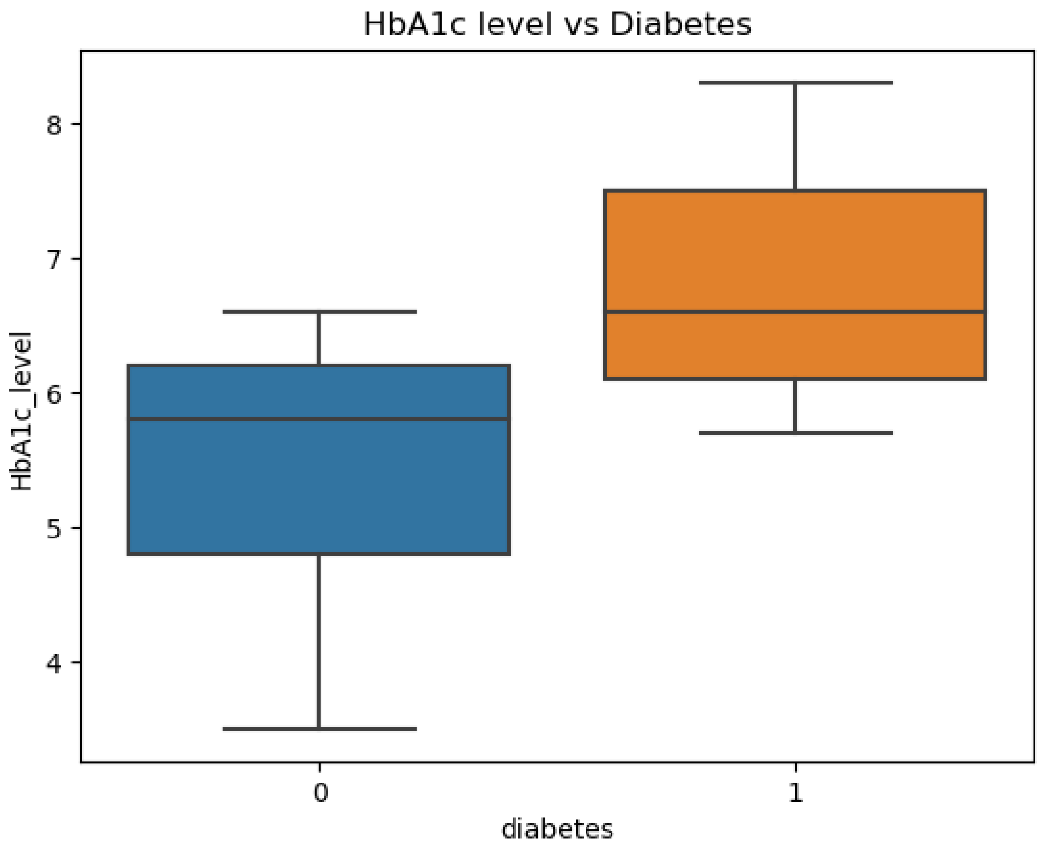
[26]:
```
# BOXPLOT FOR 'age' vs 'diabetes' CLASSIFICATION

sns.boxplot(x='diabetes', y='age', data=df)
plt.title('Age vs Diabetes')
plt.show()
```



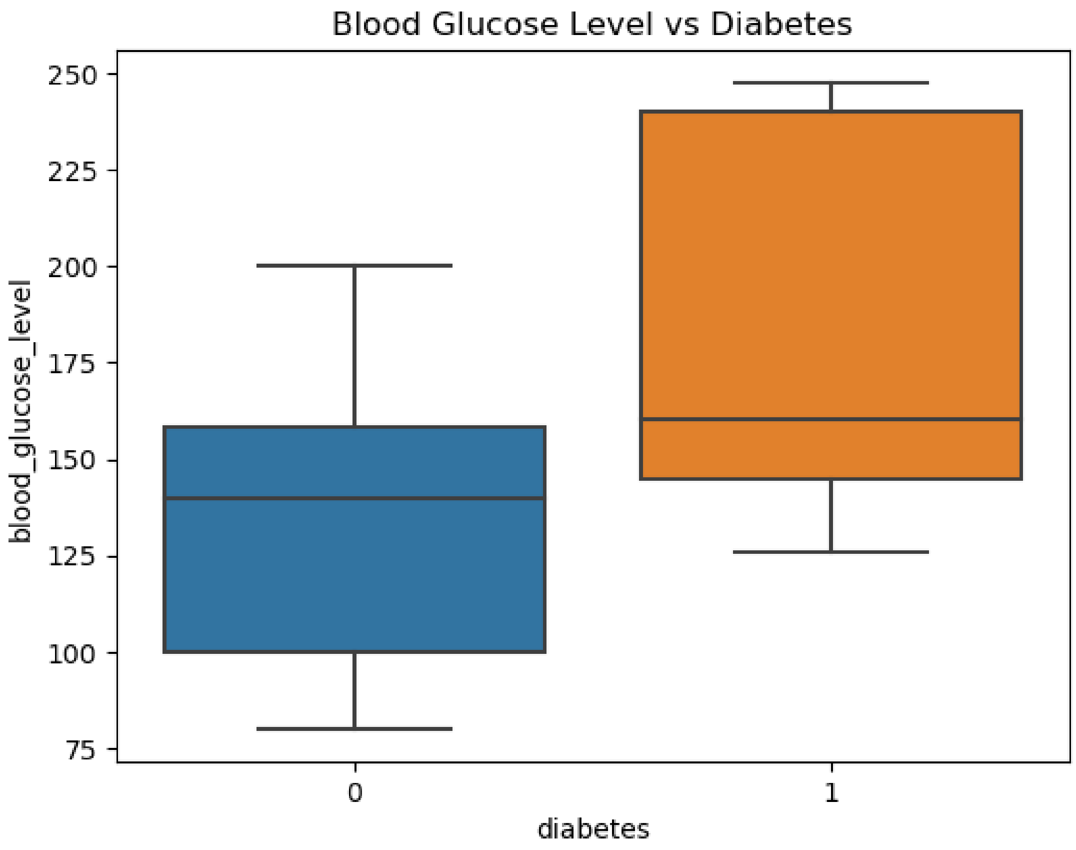
[27]:
```
# COUNT PLOT FOR 'gender' vs 'diabetes'

sns.countplot(x='gender', hue='diabetes', data=df)
plt.title('Gender vs Diabetes')
plt.show()
```



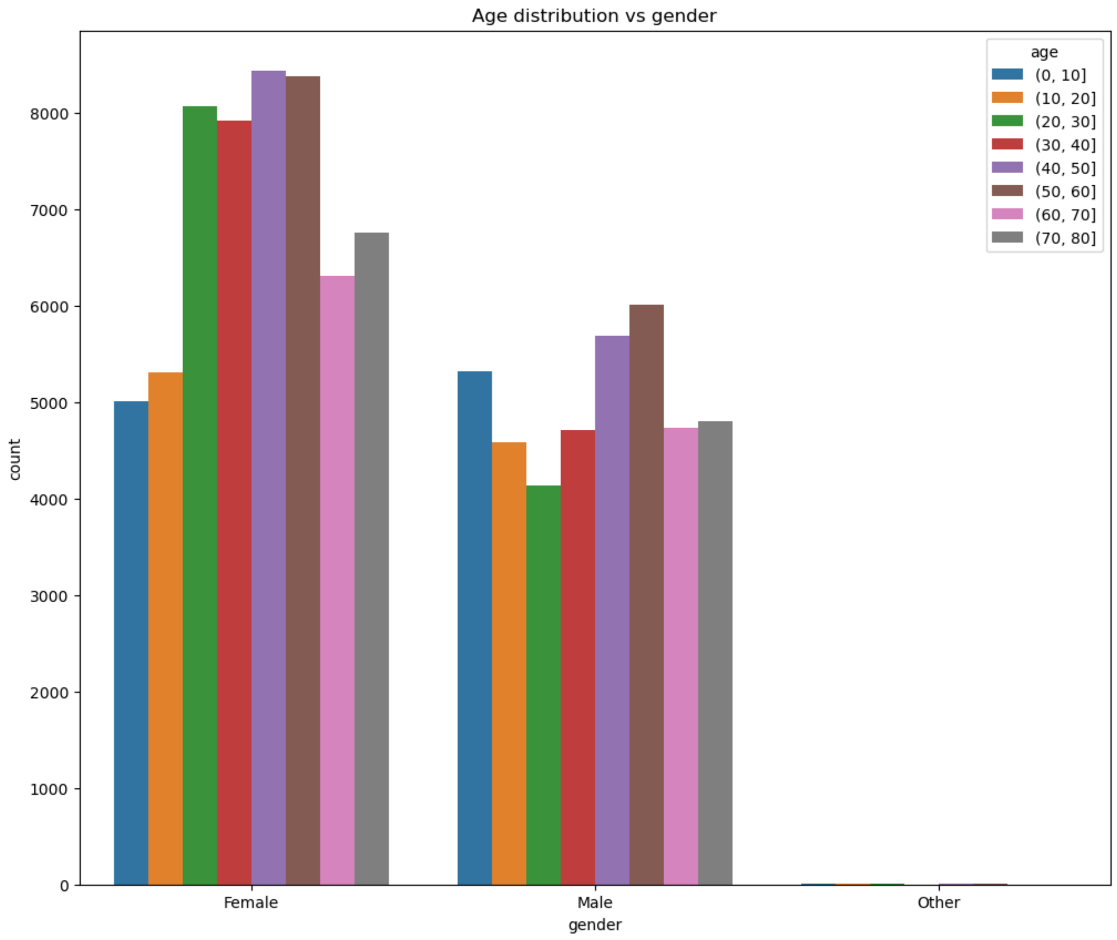
[28]:
```
# BOXPLOT FOR 'HbA1c_level' vs 'diabetes' CLASSIFICATION

sns.boxplot(x='diabetes', y='HbA1c_level', data=df)
plt.title('HbA1c level vs Diabetes')
plt.show()
```

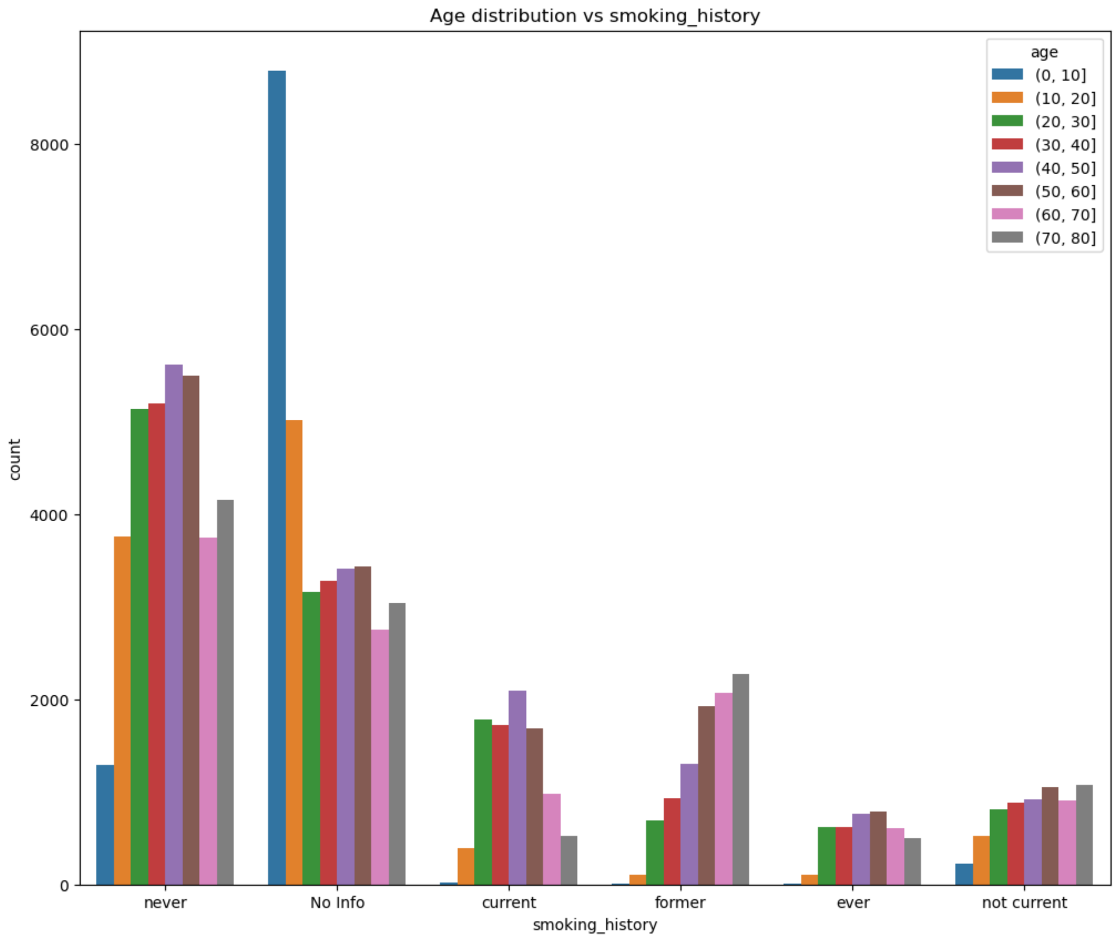


[29]:
```
# BOXPLOT FOR 'blood_glucose_level' vs 'diabetes' CLASSIFICATION

sns.boxplot(x='diabetes', y='blood_glucose_level', data=df)
plt.title('Blood Glucose Level vs Diabetes')
plt.show()
```



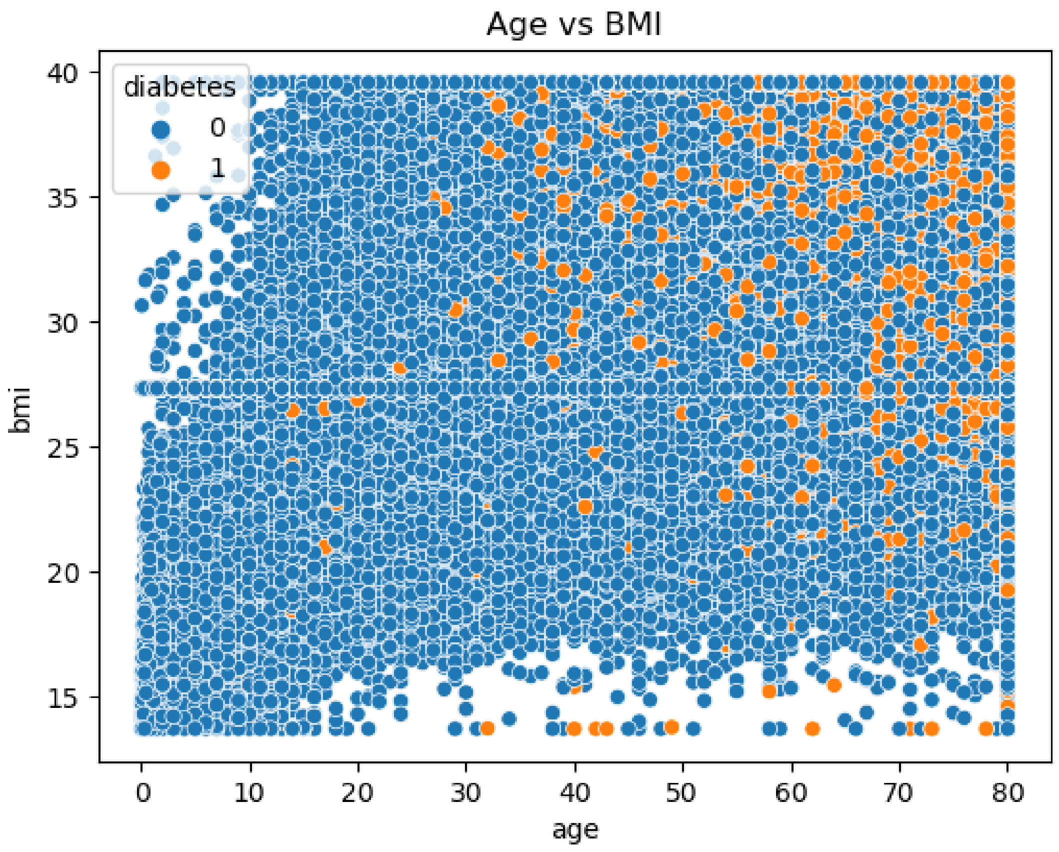
[30]:
```
age_group = pd.cut(df['age'], bins=[0, 10, 20, 30, 40, 50, 60, 70, 80])
for i in categorical:
    plt.figure(figsize=(12,10))

    # COUNT PLOT
    sns.countplot(x=df[i],data=df,hue=age_group)
    plt.title(f'Age distribution vs {i}')
    plt.show()
```


Age distribution vs gender
age
(0, 10]
(10, 20]
(20, 30]
(30, 40]
(40, 50]
(50, 60]
(60, 70]
(70, 80]
count
0
1000
2000
3000
4000
5000
6000
7000
8000
Female
Male
Other
gender



MULTIVARIATE ANALYSIS

[31]:
```
# SCATTER PLOT FOR 'age' vs 'bmi' COLORED BY 'diabetes' CLASSIFICATION

sns.scatterplot(x='age', y='bmi', hue='diabetes', data=df)
plt.title('Age vs BMI')
plt.show()
```



2.5 FUNCTION MAPPING

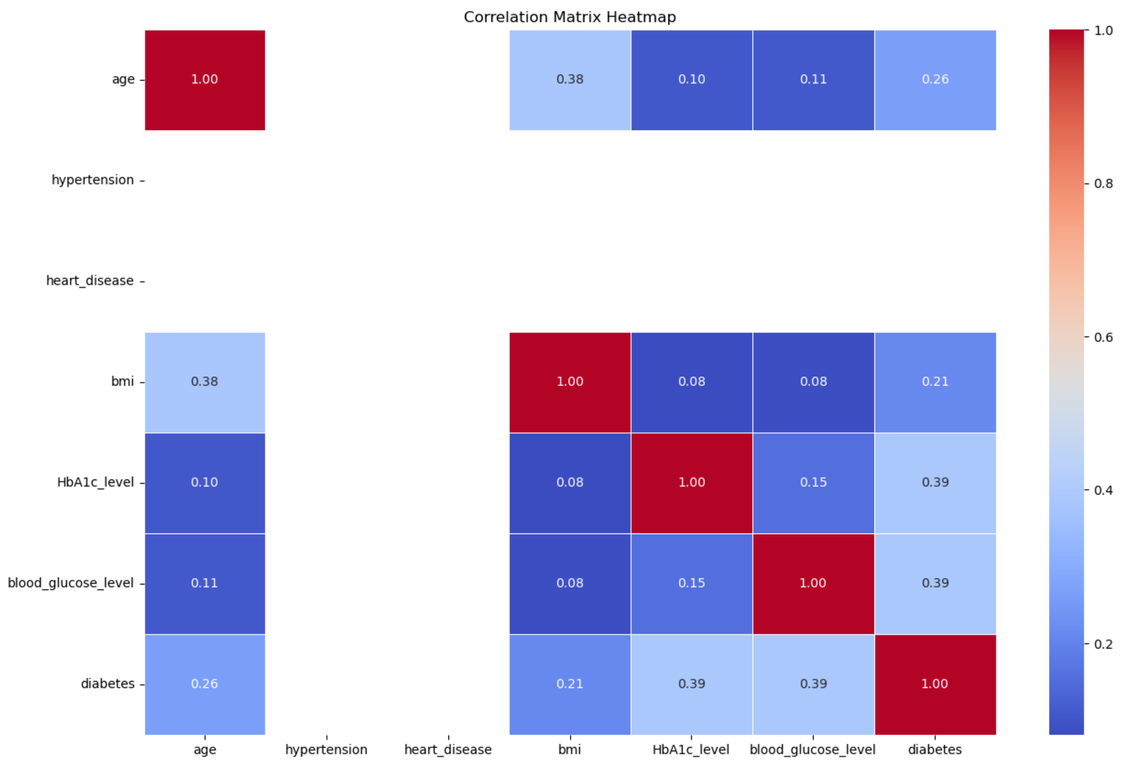
[32]:
```
# FUNCTION TO MAP THE EXISTING CATEGORIES TO NEW ONES

def recategorize_smoking(smoking_status):
    if smoking_status in ['never', 'No Info']:
        return 'non-smoker'
    elif smoking_status == 'current':
        return 'current'
    elif smoking_status in ['ever', 'former', 'not current']:
        return 'past_smoker'

# APPLY THE FUNCTION TO THE 'smoking_history' COLUMN

df['smoking_history'] = df['smoking_history'].apply(recategorize_smoking)

# NEW VALUE COUNTS
print(df['smoking_history'].value_counts())
```

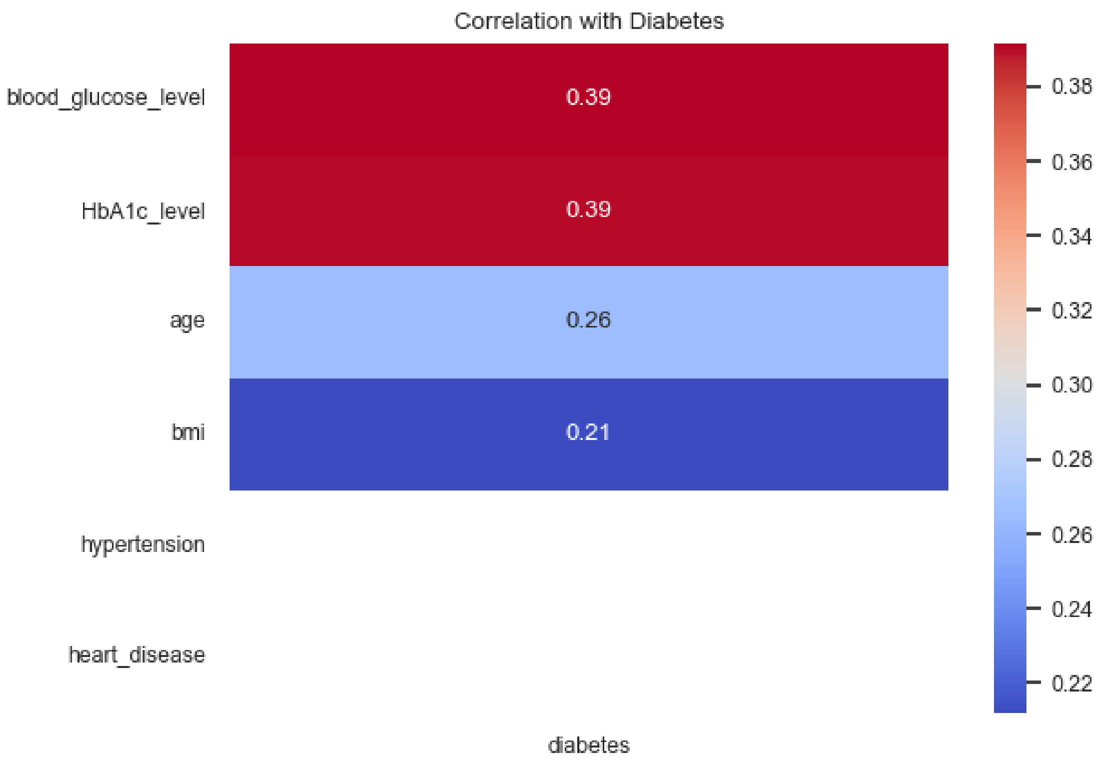
```
non-smoker     67285
```

```
past_smoker    19664
current         9197
Name: smoking_history, dtype: int64
```

2.6 CORRELATION BETWEEN FEATURES

[33]:
```
# COMPUTE THE CORRELATION MATRIX

correlation_matrix = df.corr()

# GRAPH 1

plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5,␣
  ↪fmt='.2f')
plt.title("Correlation Matrix Heatmap")
plt.show()



# GRAPH 2
# CREATE A HEATMAP OF THE CORRELATIONS WITH THE TARGET COLUMN

corr = df.corr()
target_corr = corr['diabetes'].drop('diabetes')

# SORT CORRELATION VALUES IN DESCENDING ORDER

target_corr_sorted = target_corr.sort_values(ascending=False)

sns.set(font_scale=0.8)
sns.set_style("white")
sns.set_palette("PuBuGn_d")
sns.heatmap(target_corr_sorted.to_frame(), cmap="coolwarm", annot=True, fmt='.
  ↪2f')
plt.title('Correlation with Diabetes')
plt.show()
```


Correlation Matrix Heatmap
age
hypertension
heart_disease
bmi
HbA1c_level
blood_glucose_level
diabetes
1.00
0.38
0.10
0.11
0.26
0.38
1.00
0.08
0.08
0.21
0.10
0.08
1.00
0.15
0.39
0.11
0.08
0.15
1.00
0.39
0.26
0.21
0.39
0.39
1.00
1.0
0.8
0.6
0.4
0.2


Correlation with Diabetes
blood_glucose_level
HbA1c_level
age
bmi
hypertension
heart_disease
0.39
0.39
0.26
0.21
diabetes
0.38
0.36
0.34
0.32
0.30
0.28
0.26
0.24
0.22

2.7 ENCODING

```
[34]: encoder=ce.OneHotEncoder(cols=categorical)
      df=encoder.fit_transform(df)

      print('All features are now Numerical!')
```

```
All features are now Numerical!
```

```
[35]: df.head()
```

```
[35]:    gender_1  gender_2  gender_3   age  hypertension  heart_disease  \
      0         1         0         0  80.0             0              0
      1         1         0         0  54.0             0              0
      2         0         1         0  28.0             0              0
      3         1         0         0  36.0             0              0
      4         0         1         0  76.0             0              0

         smoking_history_1  smoking_history_2  smoking_history_3    bmi  \
      0                  1                  0                  0  25.19
      1                  1                  0                  0  27.32
      2                  1                  0                  0  27.32
      3                  0                  1                  0  23.45
      4                  0                  1                  0  20.14

         HbA1c_level  blood_glucose_level  diabetes
      0          6.6                140.0         0
      1          6.6                 80.0         0
      2          5.7                158.0         0
      3          5.0                155.0         0
      4          4.8                155.0         0
```

2.8 NAIVE BAYES CLASSIFICATION

2.9 DECLARE FEATURE VECTOR AND TARGET VARIABLE

```
[36]: X=df.drop(['diabetes'],axis=1)
      y=df['diabetes']
```

Training set = 66.6% (2/3rd of total), Test set = 33.3%

```
[37]: # SPLITTING DATA INTO SEPARATE TRAINING AND TEST SET

      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.333,␣
       ↪random_state=42)
```

[38]:
```
# SHAPE OF X_train AND X_test

print(X.shape,X_train.shape, X_test.shape)
```

```
(96146, 12) (64129, 12) (32017, 12)
```

FEATURE SCALING

[39]:
```
cols=X_train.columns
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=cols)
X_test = pd.DataFrame(X_test, columns=cols)
```

[40]:
```
X_train.head()
```

[40]:
```
   gender_1  gender_2  gender_3       age  hypertension  heart_disease  \
0 -1.182549  1.183080 -0.014777 -1.502189           0.0            0.0
1 -1.182549  1.183080 -0.014777  0.455693           0.0            0.0
2  0.845631 -0.845251 -0.014777  0.411196           0.0            0.0
3  0.845631 -0.845251 -0.014777  0.322201           0.0            0.0
4  0.845631 -0.845251 -0.014777 -1.279703           0.0            0.0

   smoking_history_1  smoking_history_2  smoking_history_3       bmi  \
0           0.652474          -0.323605          -0.505943 -1.661402
1           0.652474          -0.323605          -0.505943 -0.101500
2           0.652474          -0.323605          -0.505943  0.053647
3           0.652474          -0.323605          -0.505943  0.411160
4           0.652474          -0.323605          -0.505943 -0.211114

   HbA1c_level  blood_glucose_level
0     0.930914             0.531649
1     0.262353             1.619390
2     0.930914            -0.193511
3     0.930914            -0.193511
4    -0.979261            -0.193511
```

MODEL TRAINING

[41]:
```
# APPLYING NAIVE BAYES CLASSIFIER

model=GaussianNB()
model.fit(X_train,y_train)
```

[41]:
```
GaussianNB()
```

[42]:
```
y_pred=model.predict(X_test)
df1=pd.DataFrame({'Actual Class':y_test,'Predicted Class':y_pred})
```

[43]:
```
df1
```

[43]:
```
       Actual Class  Predicted Class
2547              0                0
34774             0                1
71084             1                1
50584             0                1
80788             0                1
…               …                …
24564             0                0
65024             0                0
52869             0                1
17216             0                0
22362             0                1

[32017 rows x 2 columns]
```

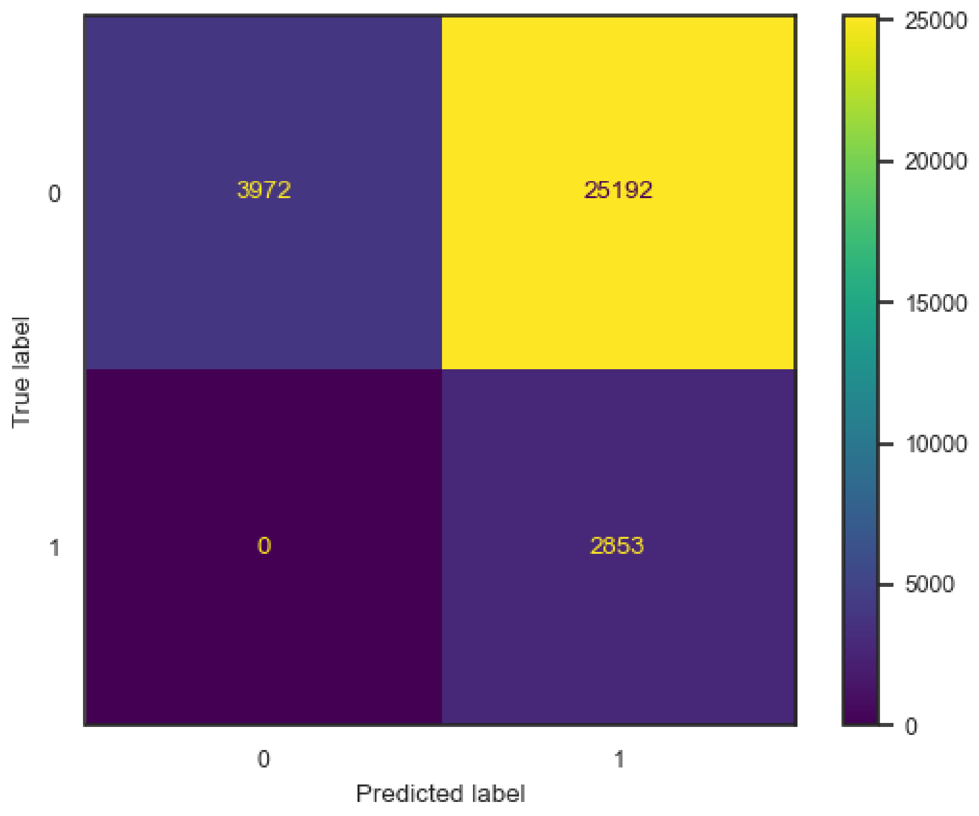
[44]:
```
# PRINT REPORT

print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.14      0.24     29164
           1       0.10      1.00      0.18      2853

    accuracy                           0.21     32017
   macro avg       0.55      0.57      0.21     32017
weighted avg       0.92      0.21      0.23     32017
```

[45]:
```
# ACCURACY

print('Accuracy : ',accuracy_score(y_test, y_pred)*100)
```

```
Accuracy :  21.316800449761065
```

[46]:
```
# MAKING CONFUSION MATRIX

labels=df['diabetes'].unique()
cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot();
```



Random subsampling

```
[47]: X=df.drop(['diabetes'],axis=1)
      y=df['diabetes']
```

```
[48]: accuracies=[]
      iterations=int(input('Enter the number of iterations :'))
      for i in range(iterations):
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
       ↪333, random_state=42)
          cols=X_train.columns
          scaler = StandardScaler()
          X_train = scaler.fit_transform(X_train)
          X_test = scaler.transform(X_test)
          X_train = pd.DataFrame(X_train, columns=cols)
          X_test = pd.DataFrame(X_test, columns=cols)
          model=GaussianNB()
          model.fit(X_train,y_train)
          y_pred=model.predict(X_test)
          accuracies.append(accuracy_score(y_test, y_pred)*100)
      avg_accuracy=sum(accuracies)/len(accuracies)
```

```
print('Average Accuracy :',avg_accuracy)
```

```
Enter the number of iterations :5
Average Accuracy : 21.316800449761065
```

Cross-Validation (Stratified - K Fold)

[49]:
```
X=df.drop(['diabetes'],axis=1)
y=df['diabetes']
```

[50]:
```
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

cnt = 1

for train_index, test_index in kf.split(X, y):
    print(f'Fold:{cnt}, Train set: {len(train_index)}, Test set:
 ↪{len(test_index)}')
    cnt+=1
```

```
Fold:1, Train set: 86531, Test set:9615
Fold:2, Train set: 86531, Test set:9615
Fold:3, Train set: 86531, Test set:9615
Fold:4, Train set: 86531, Test set:9615
Fold:5, Train set: 86531, Test set:9615
Fold:6, Train set: 86531, Test set:9615
Fold:7, Train set: 86532, Test set:9614
Fold:8, Train set: 86532, Test set:9614
Fold:9, Train set: 86532, Test set:9614
Fold:10, Train set: 86532, Test set:9614
```

[51]:
```
clf=GaussianNB()
score = cross_val_score(clf, X, y, cv= kf, scoring="accuracy")
print(f'Scores for each fold are: {score*100}')
print(f'Average score: {"{:.2f}".format(score.mean()*100)}')
```

```
Scores for each fold are: [85.13780551 84.04576183 83.9625585  85.50182007
84.2849714  84.81539262
 84.7410027  84.52257125 83.75286041 83.04555856]
Average score: 84.38
```

2.10 K-NEAREST CLASSIFICATION

2.11 DECLARE FEATURE VECTOR AND TARGET VARIABLE

[52]:
```
X=df.drop(['diabetes'],axis=1)
y=df['diabetes']
```

Training set = 66.6% (2/3rd of total), Test set = 33.3%

```
[53]: X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.333,␣
        ↪random_state=42)
```

```
[54]: print(X.shape, X_train.shape, X_test.shape)
```

```
(96146, 12) (64129, 12) (32017, 12)
```

FEATURE SCALING

```
[55]: scaler = StandardScaler()
      X_train = scaler.fit_transform(X_train)
      X_test = scaler.transform(X_test)
```

MODEL TRAINING

```
[56]: classifier = KNeighborsClassifier(n_neighbors = 7, metric = 'minkowski', p = 2)
      classifier.fit(X_train, y_train)
```

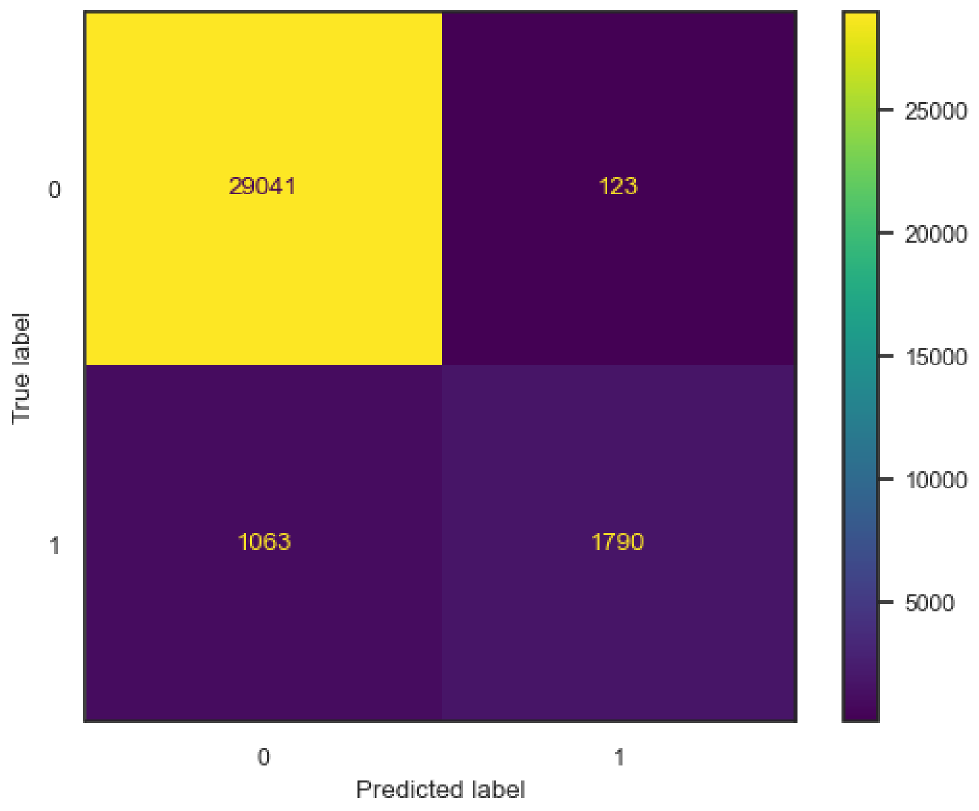
```
[56]: KNeighborsClassifier(n_neighbors=7)
```

```
[57]: # PREDICTING THE TEST RESULTS

      y_pred = classifier.predict(X_test)
```

```
[58]: df2=pd.DataFrame({'Actual Class':y_test,'Predicted Class':y_pred})
      df2
```

```
[58]:        Actual Class  Predicted Class
      2547              0                0
      34774             0                0
      71084             1                1
      50584             0                0
      80788             0                0
      …               …                …
      24564             0                0
      65024             0                0
      52869             0                0
      17216             0                0
      22362             0                0

      [32017 rows x 2 columns]
```

```
[59]: # MAKING THE CONFUSION MATRIX

      labels=df['diabetes'].unique()
      cm = confusion_matrix(y_test, y_pred, labels=labels)
      disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
      disp.plot();
```



```
[60]: # PRINTING ACCURACY

      acc= accuracy_score(y_test, y_pred)
      print('Accuracy : ',acc*100)
```

```
Accuracy :  96.29571789986569
```

Random subsampling

```
[61]: X=df.drop(['diabetes'],axis=1)
      y=df['diabetes']
```

```
[62]: accuracies=[]
      iterations=int(input('Enter the number of iterations :'))
      for i in range(iterations):
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
       ↪333, random_state=42)
          scaler = StandardScaler()
          X_train = scaler.fit_transform(X_train)
          X_test = scaler.transform(X_test)
          classifier = KNeighborsClassifier(n_neighbors = 7, metric = 'minkowski', p␣
       ↪= 2)
```

```
    classifier.fit(X_train, y_train)
    y_pred=classifier.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred)*100)
avg_accuracy=sum(accuracies)/len(accuracies)
print('Average Accuracy :',avg_accuracy)
```

```
Enter the number of iterations :5
Average Accuracy : 96.29571789986569
```

Cross-Validation (Stratified - K Fold)

[63]:
```
X=df.drop(['diabetes'],axis=1)
y=df['diabetes']
```

[64]:
```
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

cnt = 1

for train_index, test_index in kf.split(X, y):
    print(f'Fold:{cnt}, Train set: {len(train_index)}, Test set:
 ↪{len(test_index)}')
    cnt+=1
```

```
Fold:1, Train set: 86531, Test set:9615
Fold:2, Train set: 86531, Test set:9615
Fold:3, Train set: 86531, Test set:9615
Fold:4, Train set: 86531, Test set:9615
Fold:5, Train set: 86531, Test set:9615
Fold:6, Train set: 86531, Test set:9615
Fold:7, Train set: 86532, Test set:9614
Fold:8, Train set: 86532, Test set:9614
Fold:9, Train set: 86532, Test set:9614
Fold:10, Train set: 86532, Test set:9614
```

[65]:
```
clf=KNeighborsClassifier(n_neighbors = 7, metric = 'minkowski', p = 2)
score = cross_val_score(clf, X, y, cv= kf, scoring="accuracy")
print(f'Scores for each fold are: {score*100}')
print(f'Average score: {"{:.2f}".format(score.mean()*100)}')
```

```
Scores for each fold are: [95.31981279 95.55902236 95.27821113 95.7774311
95.43421737 95.1326053
 95.13209902 95.14250052 95.215311   95.236114  ]
Average score: 95.32
```

2.12 DECISION TREE CLASSIFICATION

2.13 DECLARE FEATURE VECTOR AND TARGET VARIABLE

[66]:
```
X=df.drop(['diabetes'],axis=1)
y=df['diabetes']
```

Training set = 66.6% (2/3rd of total), Test set = 33.3%

[67]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.333,␣
↪random_state=42)
```

[68]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(96146, 12) (64129, 12) (32017, 12)
```

FEATURE SCALING

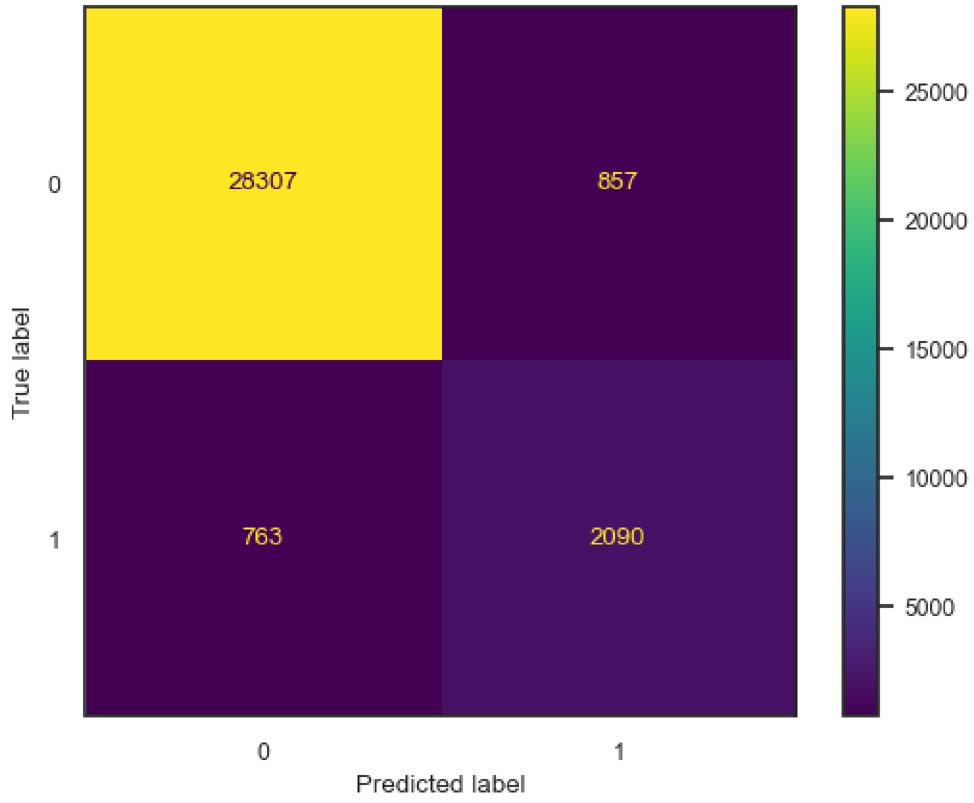
[69]:
```
cols=X_train.columns
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=cols)
X_test = pd.DataFrame(X_test, columns=cols)
```

MODEL TRAINING

[70]:
```
clf = DecisionTreeClassifier(criterion='gini')

clf = clf.fit(X_train,y_train)

# PREDICT THE RESPONSE FOR TEST DATASET
y_pred = clf.predict(X_test)

df3=pd.DataFrame({'Actual Class':y_test,'Predicted Class':y_pred})
df3
```

[70]:
```
       Actual Class  Predicted Class
2547              0                0
34774             0                0
71084             1                1
50584             0                0
80788             0                0
…               …              …
24564             0                0
65024             0                0
52869             0                0
17216             0                0
22362             0                0

[32017 rows x 2 columns]
```

[71]:
```
# MAKING THE CONFUSION MATRIX

labels=df['diabetes'].unique()
cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot();
```



[72]:
```
print("Accuracy : ",accuracy_score(y_test, y_pred)*100)
```

```
Accuracy :  94.94018802511167
```

Random subsampling

[73]:
```
X=df.drop(['diabetes'],axis=1)
y=df['diabetes']
```

[74]:
```
accuracies=[]
iterations=int(input('Enter the number of iterations :'))
for i in range(iterations):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
 ↪333, random_state=42)
    cols=X_train.columns
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    X_train = pd.DataFrame(X_train, columns=cols)
    X_test = pd.DataFrame(X_test, columns=cols)
    clf=DecisionTreeClassifier(criterion='gini')
    clf=clf.fit(X_train,y_train)
    y_pred=clf.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred)*100)
avg_accuracy=sum(accuracies)/len(accuracies)
print('Average Accuracy :',avg_accuracy)
```

```
Enter the number of iterations :5
Average Accuracy : 94.96142674204329
```

Cross-Validation (Stratified - K Fold)

[75]:
```
X=df.drop(['diabetes'],axis=1)
y=df['diabetes']
```

[76]:
```
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

cnt = 1

for train_index, test_index in kf.split(X, y):
    print(f'Fold:{cnt}, Train set: {len(train_index)}, Test set:
 ↪{len(test_index)}')
    cnt+=1
```

```
Fold:1, Train set: 86531, Test set:9615
Fold:2, Train set: 86531, Test set:9615
Fold:3, Train set: 86531, Test set:9615
Fold:4, Train set: 86531, Test set:9615
Fold:5, Train set: 86531, Test set:9615
Fold:6, Train set: 86531, Test set:9615
Fold:7, Train set: 86532, Test set:9614
Fold:8, Train set: 86532, Test set:9614
Fold:9, Train set: 86532, Test set:9614
Fold:10, Train set: 86532, Test set:9614
```

[77]:
```
clf=DecisionTreeClassifier(criterion='gini')
score = cross_val_score(clf, X, y, cv= kf, scoring="accuracy")
print(f'Scores for each fold are: {score*100}')
print(f'Average score: {"{:.2f}".format(score.mean()*100)}')
```

```
Scores for each fold are: [94.71658866 94.74778991 94.88299532 95.17420697
94.95579823 94.89339574
 94.61202413 95.12169752 94.85125858 95.05928854]
Average score: 94.90
```

CONCLUSION

ACCURACIES

1. **Naïve Bayes Classification Model** Parameters [Training-set=66.6%, Test-set=33.3%, random_state=42, GaussianNB()] –

   a) **21.316800449761065**
   b) Random subsampling (5 iterations) = **21.316800449761065**
   c) Cross – Validation (Stratified – K Fold with n_splits=10, shuffle=True, random_state=42) = **84.38**

2. **kNN Classification Model** Parameters [Training-set=66.6%, Test-set=33.3%, random_state=42, n_neighbors = 7, metric = 'minkowski', p = 2] –

   a) **96.29571789986569**
   b) Random subsampling (5 iterations) = **96.29571789986569**
   c) Cross – Validation (Stratified – K Fold with n_splits=10, shuffle=True, random_state=42) = **95.32**

3. **Decision Tree Classification Model** Parameters [Training-set=66.6%, Test-set=33.3%, random_state=42, n_neighbors = 7, metric = 'minkowski', p = 2] –

   a) **94.94018802511167**
   b) Random subsampling (5 iterations) = **94.96142674204329**
   c) Cross – Validation (Stratified – K Fold with n_splits=10, shuffle=True, random _state=42) = **94.90**

Based on performance metrics of classification model parameters -

- **Both the kNN and Decision Tree classification models outperform the Naïve Bayes classification model in terms of accuracy (CROSS-VALIDATION).**
- **Therefore, for the given diabetes prediction dataset, either the kNN or Decision Tree model would be a better choice for classification tasks compared to the Naïve Bayes model.**

THANK YOU