Importing the Libraries

In [1]:
```
#We start off this project by importing all the necessary libraries that will be required for the process
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the Data

In [2]:
```
#Loading the data and removing the irrelevant columns.
df = pd.read_csv('smoking.csv')
df.head()
```

Out[2]:

eight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	hearing(right)	...	hemoglobin	Urine protein	serum creatinine	AST	ALT	Gtp	oral	dental caries	tartar	smoking
60	81.3	1.2	1.0	1.0	1.0	...	12.9	1.0	0.7	18.0	19.0	27.0	Y	0	Y	0
60	81.0	0.8	0.6	1.0	1.0	...	12.7	1.0	0.6	22.0	19.0	18.0	Y	0	Y	0
60	80.0	0.8	0.8	1.0	1.0	...	15.8	1.0	1.0	21.0	16.0	22.0	Y	0	N	1
70	88.0	1.5	1.5	1.0	1.0	...	14.7	1.0	1.0	19.0	26.0	18.0	Y	0	Y	0
60	86.0	1.0	1.0	1.0	1.0	...	12.5	1.0	0.6	16.0	14.0	22.0	Y	0	N	0

In [3]:
```
df = df.drop(columns=['ID','oral'])
df.head()
```

Out[3]:

waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	hearing(right)	systolic	...	LDL	hemoglobin	Urine protein	serum creatinine	AST	ALT	Gtp	dental caries	tartar	smoking
81.3	1.2	1.0	1.0	1.0	114.0	...	126.0	12.9	1.0	0.7	18.0	19.0	27.0	0	Y	0
81.0	0.8	0.6	1.0	1.0	119.0	...	127.0	12.7	1.0	0.6	22.0	19.0	18.0	0	Y	0
80.0	0.8	0.8	1.0	1.0	138.0	...	151.0	15.8	1.0	1.0	21.0	16.0	22.0	0	N	1
88.0	1.5	1.5	1.0	1.0	100.0	...	226.0	14.7	1.0	1.0	19.0	26.0	18.0	0	Y	0
86.0	1.0	1.0	1.0	1.0	120.0	...	107.0	12.5	1.0	0.6	16.0	14.0	22.0	0	N	0

In [4]:
```
#Checking the shape of a dataframe and datatypes of all columns along with calculating the statistical data.
df.shape
```

Out[4]: (55692, 25)

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55692 entries, 0 to 55691
Data columns (total 25 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   gender               55692 non-null  object 
 1   age                  55692 non-null  int64  
 2   height(cm)           55692 non-null  int64  
 3   weight(kg)           55692 non-null  int64  
 4   waist(cm)            55692 non-null  float64
 5   eyesight(left)       55692 non-null  float64
 6   eyesight(right)      55692 non-null  float64
 7   hearing(left)        55692 non-null  float64
 8   hearing(right)       55692 non-null  float64
 9   systolic             55692 non-null  float64
 10  relaxation           55692 non-null  float64
 11  fasting blood sugar  55692 non-null  float64
 12  Cholesterol          55692 non-null  float64
 13  triglyceride         55692 non-null  float64
 14  HDL                  55692 non-null  float64
 15  LDL                  55692 non-null  float64
 16  hemoglobin           55692 non-null  float64
 17  Urine protein        55692 non-null  float64
 18  serum creatinine     55692 non-null  float64
 19  AST                  55692 non-null  float64
 20  ALT                  55692 non-null  float64
 21  Gtp                  55692 non-null  float64
 22  dental caries        55692 non-null  int64  
 23  tartar               55692 non-null  object 
 24  smoking              55692 non-null  int64  
dtypes: float64(18), int64(5), object(2)
memory usage: 10.6+ MB
```

```
In [6]: df.describe()
```

Out[6]:

	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	hearing(right)	systolic	relaxation	...	
count	55692.000000	55692.000000	55692.000000	55692.000000	55692.000000	55692.000000	55692.000000	55692.000000	55692.000000	55692.000000	...	556
mean	44.182917	164.649321	65.864936	82.046418	1.012623	1.007443	1.025587	1.026144	121.494218	76.004830	...	
std	12.071418	9.194597	12.820306	9.274223	0.486873	0.485964	0.157902	0.159564	13.675989	9.679278	...	
min	20.000000	130.000000	30.000000	51.000000	0.100000	0.100000	1.000000	1.000000	71.000000	40.000000	...	
25%	40.000000	160.000000	55.000000	76.000000	0.800000	0.800000	1.000000	1.000000	112.000000	70.000000	...	
50%	40.000000	165.000000	65.000000	82.000000	1.000000	1.000000	1.000000	1.000000	120.000000	76.000000	...	
75%	55.000000	170.000000	75.000000	88.000000	1.200000	1.200000	1.000000	1.000000	130.000000	82.000000	...	
max	85.000000	190.000000	135.000000	129.000000	9.900000	9.900000	2.000000	2.000000	240.000000	146.000000	...	6

8 rows × 23 columns

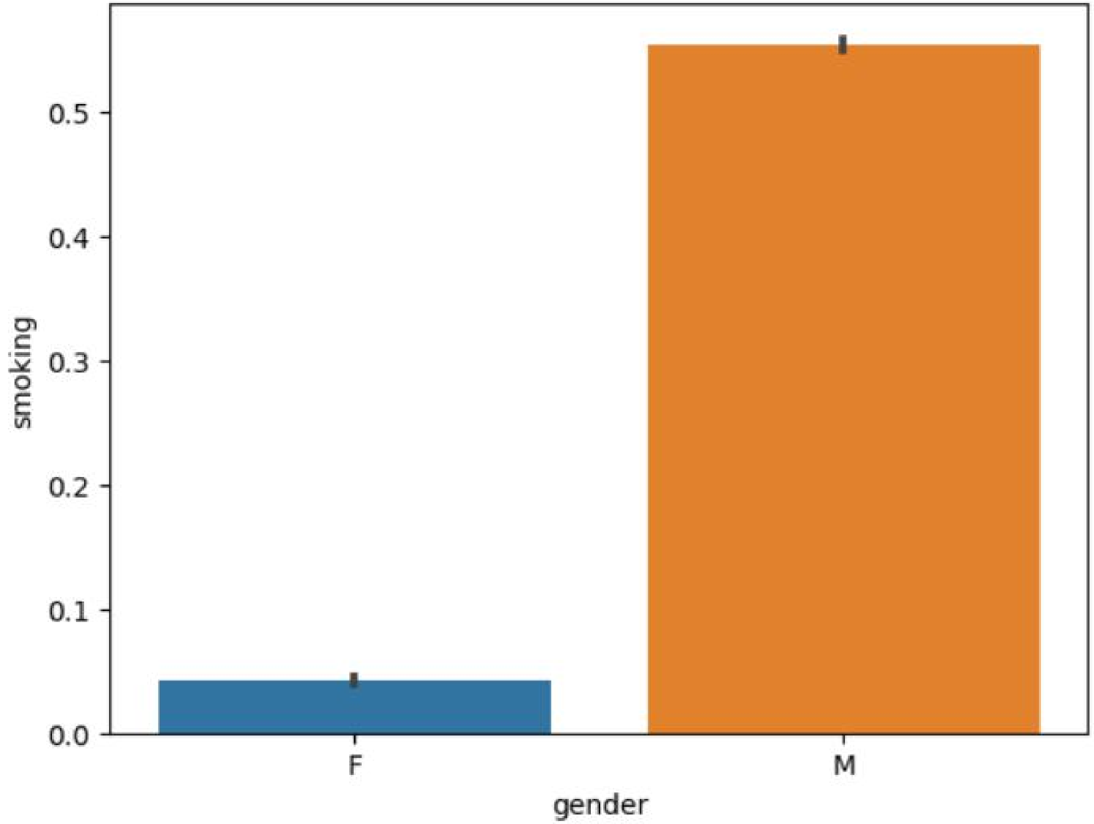
Missing Values

In [7]:
```
#Checking out the missing values in a dataframe
df.isnull().sum()
```

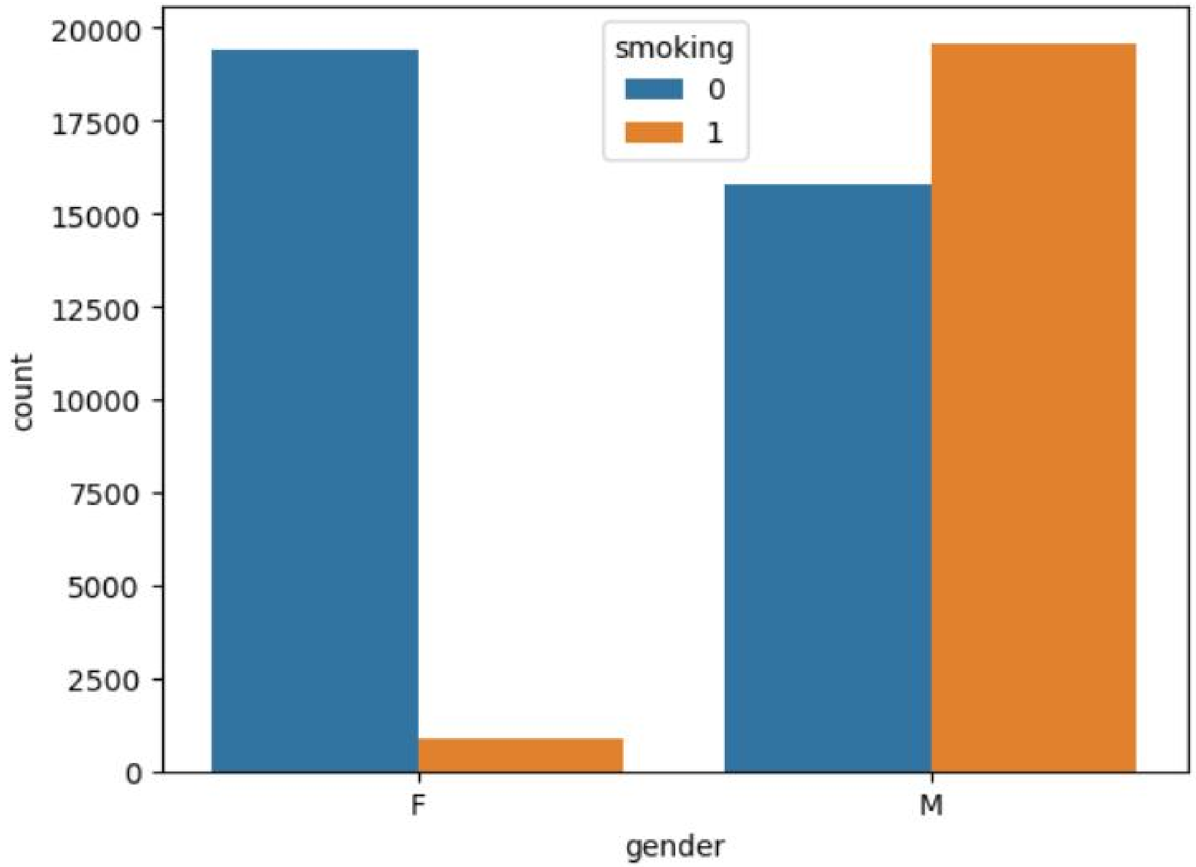
Out[7]:
```
gender                 0
age                    0
height(cm)             0
weight(kg)             0
waist(cm)              0
eyesight(left)         0
eyesight(right)        0
hearing(left)          0
hearing(right)         0
systolic               0
relaxation             0
fasting blood sugar    0
Cholesterol            0
triglyceride           0
HDL                    0
LDL                    0
hemoglobin             0
Urine protein          0
serum creatinine       0
AST                    0
ALT                    0
Gtp                    0
dental caries          0
tartar                 0
smoking                0
dtype: int64
```

Data Visualization ✓

```
In [12]: #We can clearly see from the below graph that most smokers are men
         sns.barplot(x=df['gender'],y=df['smoking'])
         plt.show()
```



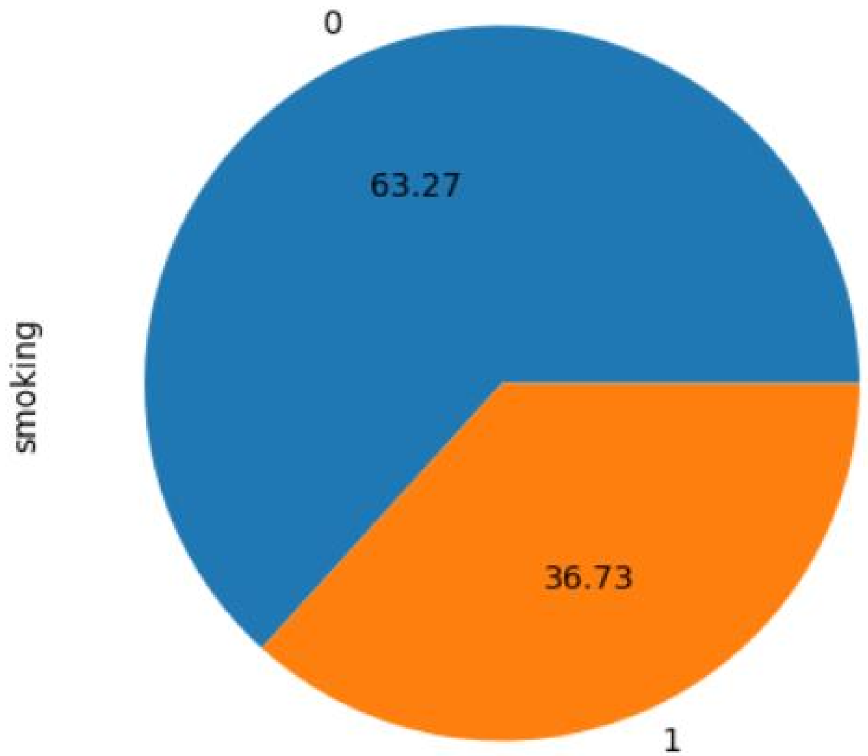
In [14]:
```
sns.countplot(x='gender', hue='smoking', data=df)
plt.show()
```



In [15]:
```
#There are 36.73 percent of the people who are smoking ciggarette.
```

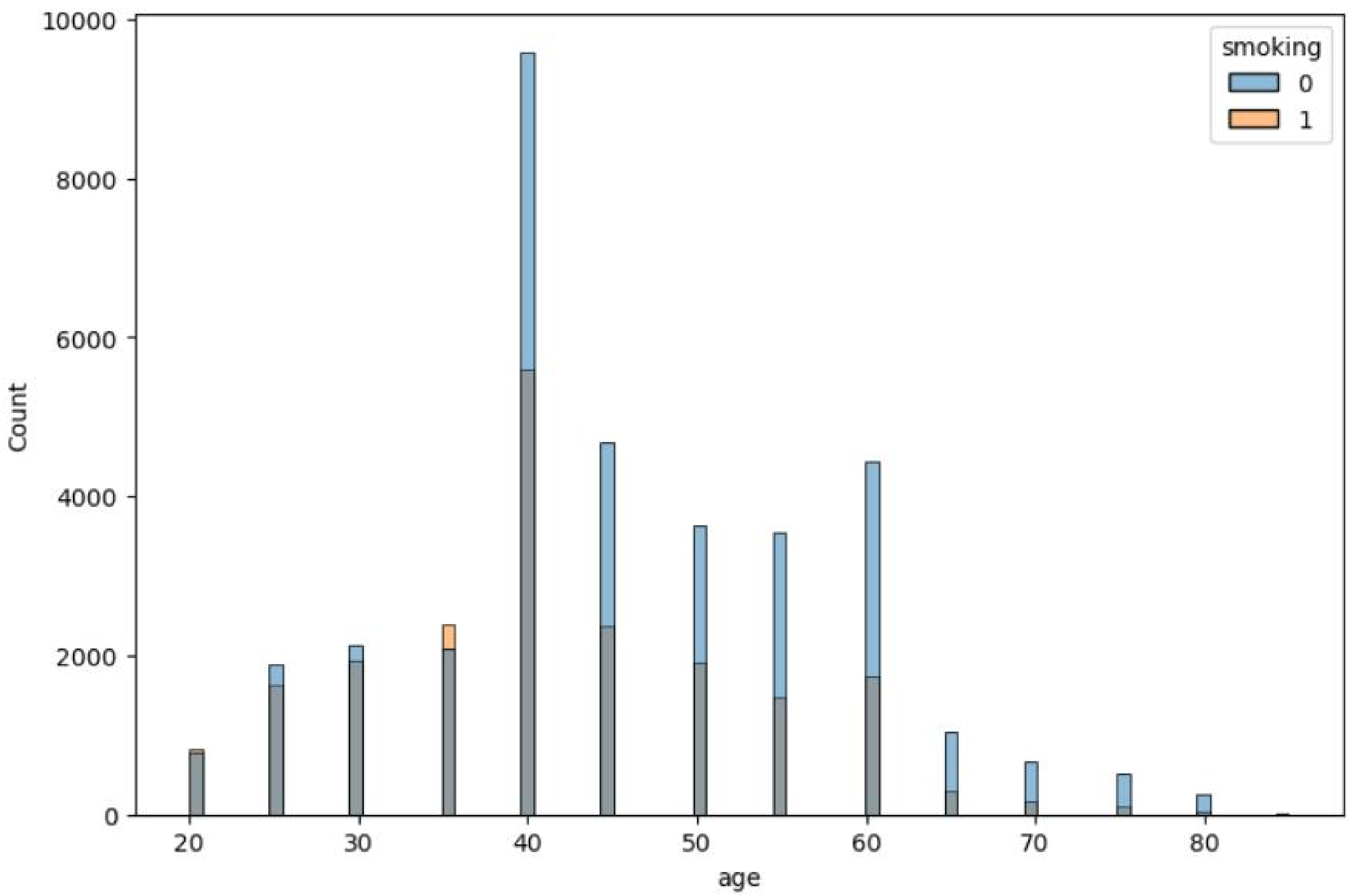
In [15]:
```
#There are 36.73 percent of the people who are smoking ciggarette.
plt.figure(figsize=(10,5))
df['smoking'].value_counts().plot.pie(autopct='%0.2f')
```

Out[15]: <Axes: ylabel='smoking'>



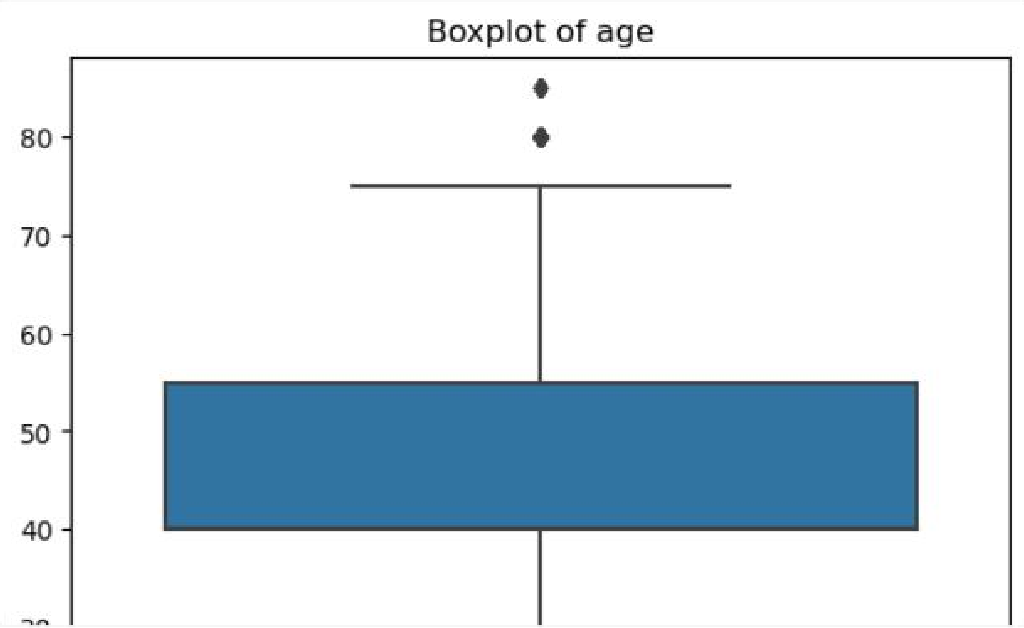
In [16]:
```
#Most number of smokers are having the age 40
plt.figure(figsize=(9,6))
sns.histplot(x='age', hue='smoking', data=df)
```

Out[16]: <Axes: xlabel='age', ylabel='Count'>

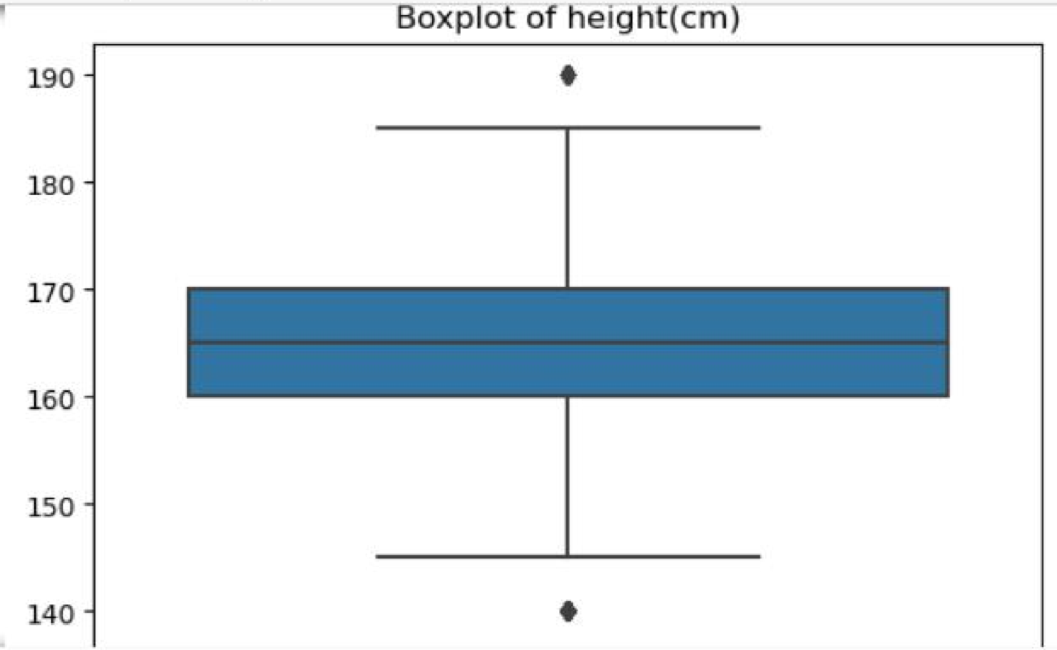


In [21]:

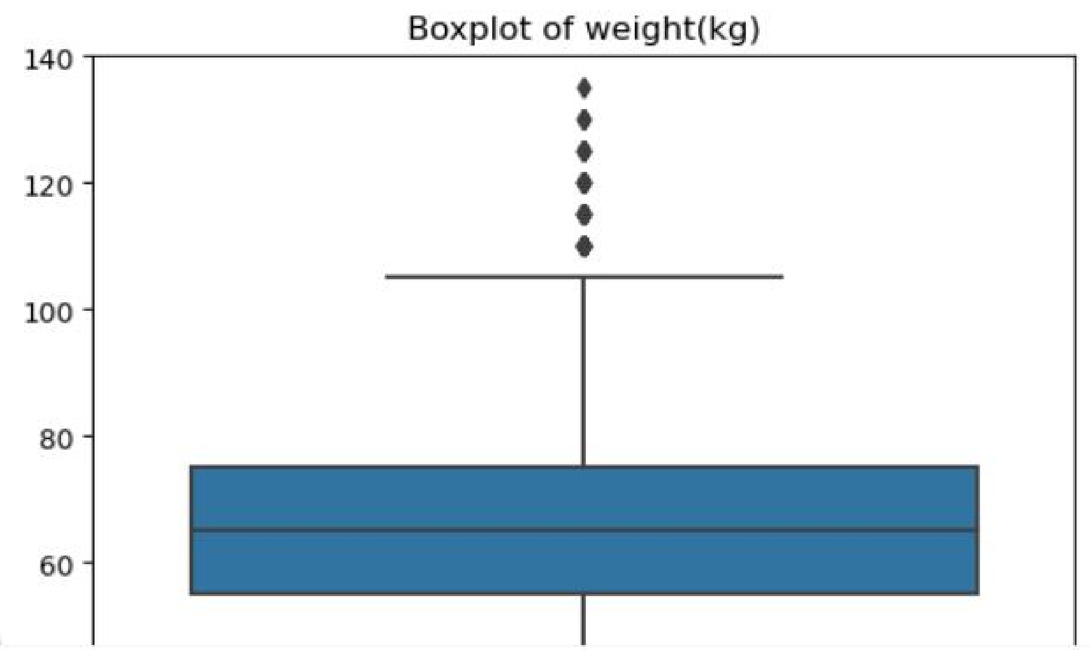
```
#Representation of columns using boxplot to detect outliers. Here outliers represent natural
#variations in the population, and they should be left as is in the dataset. These are called true
#outliers. Therefore for this dataset we will not remove outlier.
for i in df.columns:
    if(df[i].dtypes=="int64" or df[i].dtypes=='float64'):
        sns.boxplot(df[i]).set(title=f'Boxplot of {i}')
        plt.show()
```



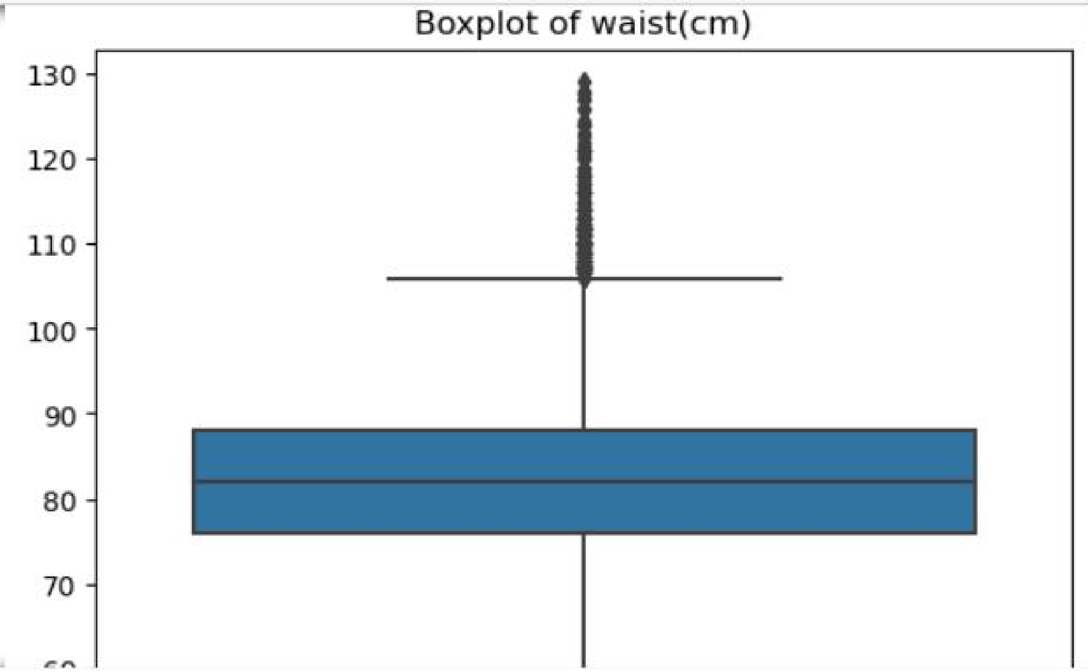
In [21]:
```
#Representation of columns using boxplot to detect outliers. Here outliers represent natural
#variations in the population, and they should be left as is in the dataset. These are called true
#outliers. Therefore for this dataset we will not remove outlier.
for i in df.columns:
    if(df[i].dtypes=="int64" or df[i].dtypes=='float64'):
        sns.boxplot(df[i]).set(title=f'Boxplot of {i}')
        plt.show()
```



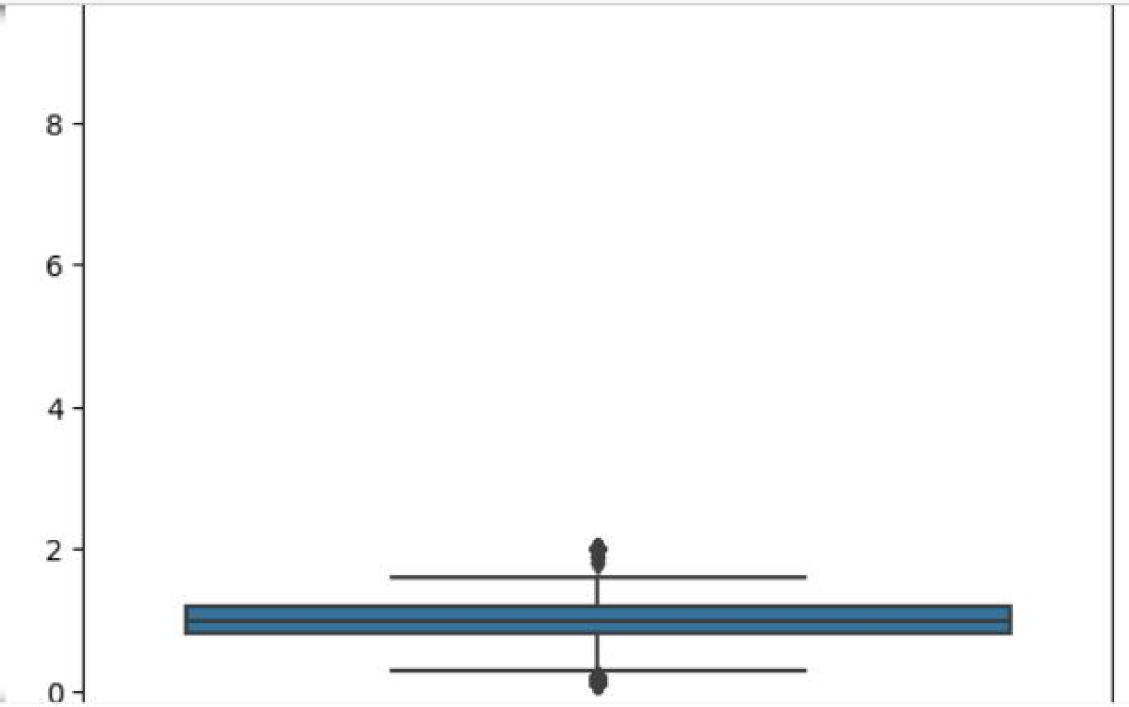
In [21]:
```
#Representation of columns using boxplot to detect outliers. Here outliers represent natural
#variations in the population, and they should be left as is in the dataset. These are called true
#outliers. Therefore for this dataset we will not remove outlier.
for i in df.columns:
    if(df[i].dtypes=="int64" or df[i].dtypes=='float64'):
        sns.boxplot(df[i]).set(title=f'Boxplot of {i}')
        plt.show()
```



```
In [21]: #Representation of columns using boxplot to detect outliers. Here outliers represent natural
         #variations in the population, and they should be left as is in the dataset. These are called true
         #outliers. Therefore for this dataset we will not remove outlier.
         for i in df.columns:
             if(df[i].dtypes=="int64" or df[i].dtypes=='float64'):
                 sns.boxplot(df[i]).set(title=f'Boxplot of {i}')
                 plt.show()
```



In [21]:
```
#Representation of columns using boxplot to detect outliers. Here outliers represent natural
#variations in the population, and they should be left as is in the dataset. These are called true
#outliers. Therefore for this dataset we will not remove outlier.
for i in df.columns:
    if(df[i].dtypes=="int64" or df[i].dtypes=='float64'):
        sns.boxplot(df[i]).set(title=f'Boxplot of {i}')
        plt.show()
```



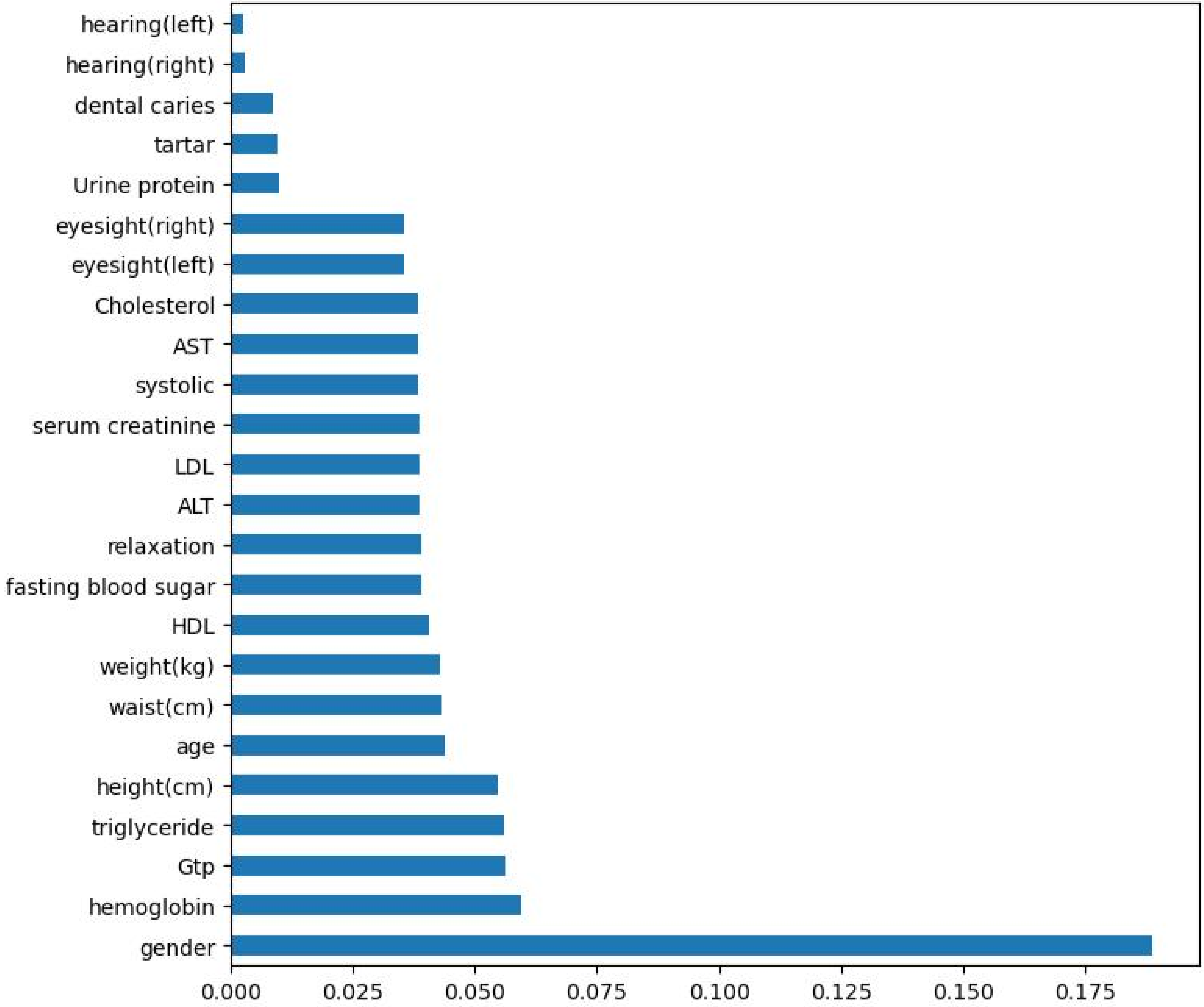
Data Cleaning

In [23]:
```
#Performing One Hot Encoding for categorical features of a dataframe

from sklearn.preprocessing import LabelEncoder

le= LabelEncoder()
df["gender"]=le.fit_transform(df["gender"])
df["tartar"]=le.fit_transform(df["tartar"])

df["dental caries"]=le.fit_transform(df["dental caries"])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55692 entries, 0 to 55691
Data columns (total 25 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   gender               55692 non-null  int64
 1   age                  55692 non-null  int64
 2   height(cm)           55692 non-null  int64
 3   weight(kg)           55692 non-null  int64
 4   waist(cm)            55692 non-null  float64
 5   eyesight(left)       55692 non-null  float64
 6   eyesight(right)      55692 non-null  float64
 7   hearing(left)        55692 non-null  float64
 8   hearing(right)       55692 non-null  float64
 9   systolic             55692 non-null  float64
 10  relaxation           55692 non-null  float64
 11  fasting blood sugar  55692 non-null  float64
 12  Cholesterol          55692 non-null  float64
 13  triglyceride         55692 non-null  float64
 14  HDL                  55692 non-null  float64
 15  LDL                  55692 non-null  float64
 16  hemoglobin           55692 non-null  float64
 17  Urine protein        55692 non-null  float64
 18  serum creatinine     55692 non-null  float64
 19  AST                  55692 non-null  float64
 20  ALT                  55692 non-null  float64
 21  Gtp                  55692 non-null  float64
 22  dental caries        55692 non-null  int64
 23  tartar               55692 non-null  int64
 24  smoking              55692 non-null  int64
dtypes: float64(18), int64(7)
memory usage: 10.6 MB
```

Feature selection using feature importance ✓

```
In [25]: #Feature importance is a technique that calculate a score for all the input features for a given model.
         #So out of 24 features we will select the top 15 features based on the score.

         X=df.iloc[:,:-1]

         y=df["smoking"]

         from sklearn.ensemble import ExtraTreesClassifier
         model=ExtraTreesClassifier()

         model. fit(X,y)
         df1=pd.Series(model.feature_importances_,index= X.columns)
         plt.figure(figsize=(8,8))
         df1.nlargest(24).plot(kind="barh")

         plt.show()
```


hearing(left)
hearing(right)
dental caries
tartar
Urine protein
eyesight(right)
eyesight(left)
Cholesterol
AST
systolic
serum creatinine
LDL
ALT
relaxation
fasting blood sugar
HDL
weight(kg)
waist(cm)
age
height(cm)
triglyceride
Gtp
hemoglobin
gender
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

Logistic Regression

n [30]:

```
#Calculating accuracy and generating the classification report of Logistic Regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Define X and y
X = df[['gender', 'height(cm)', 'Gtp', 'hemoglobin', 'triglyceride', 'age', 'weight(kg)', 'waist(cm)', 'HDL', 'serum creatinine',
y = df['smoking']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the data using StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

# Train a logistic regression model
lr = LogisticRegression()
lr.fit(x_train, y_train)

# Make predictions on the test set
y_pred = lr.predict(x_test)

# Evaluate the model's accuracy and generate a classification report
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(report)
```

Accuracy: 0.73

	precision	recall	f1-score	support
0	0.81	0.76	0.78	7027
1	0.63	0.69	0.66	4112
accuracy			0.73	11139
macro avg	0.72	0.73	0.72	11139
weighted avg	0.74	0.73	0.74	11139

Decision Tree ¶

In [33]:
```
#The accuracy of the logistic regression model is 78 percentage

from sklearn.tree import DecisionTreeClassifier
# Train a decision tree classifier
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)

# Make predictions on the test set
y_pred = dt.predict(x_test)

# Generate a classification report
report = classification_report(y_test, y_pred)

print(report)
```

```
              precision    recall  f1-score   support

           0       0.83      0.82      0.83      7027
           1       0.70      0.71      0.70      4112

    accuracy                           0.78     11139
   macro avg       0.76      0.76      0.76     11139
weighted avg       0.78      0.78      0.78     11139
```

Bagging Algorithm – Bagging Classifier

n [35]:
```
#Bootstrap Aggregation or bagging involves taking multiple samples from the training dataset
#(with replacement) and training a model for each sample
from sklearn.ensemble import BaggingClassifier

# Train a bagging classifier with decision tree base estimator
bagging_clf = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=1000)
bagging_clf.fit(x_train, y_train)

# Make predictions on the test set
y_pred = bagging_clf.predict(x_test)

# Generate a classification report
report = classification_report(y_test, y_pred)

print(report)
```

```
              precision    recall  f1-score   support

           0       0.88      0.85      0.86      7027
           1       0.75      0.80      0.77      4112

    accuracy                           0.83     11139
   macro avg       0.82      0.82      0.82     11139
weighted avg       0.83      0.83      0.83     11139
```

Bagging Algorithm – Extra Trees

In [36]:
```
from sklearn.ensemble import ExtraTreesClassifier
# Train an Extra Trees classifier
et = ExtraTreesClassifier(n_estimators=1000, random_state=42)
et.fit(x_train, y_train)

# Make predictions on the test set
y_pred = et.predict(x_test)

# Generate a classification report
report = classification_report(y_test, y_pred)

print(report)
```

```
              precision    recall  f1-score   support

           0       0.89      0.84      0.86      7027
           1       0.75      0.82      0.78      4112

    accuracy                           0.83     11139
   macro avg       0.82      0.83      0.82     11139
weighted avg       0.83      0.83      0.83     11139
```

Bagging Algorithm – Random Forest ✓

In [37]:

```
from sklearn.ensemble import RandomForestClassifier

# Train a random forest classifier
rfc = RandomForestClassifier(n_estimators=1060)
rfc.fit(x_train, y_train)

# Make predictions on the test set
y_pred = rfc.predict(x_test)

# Generate a classification report
report = classification_report(y_test, y_pred)

print(report)
```

```
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      7027
           1       0.75      0.80      0.78      4112

    accuracy                           0.83     11139
   macro avg       0.82      0.82      0.82     11139
weighted avg       0.83      0.83      0.83     11139
```