```
NAME: YASH AWARE

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# importing libraries and magic functions

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
%config InlineBackend.figure_format ='retina'
%matplotlib inline

os.chdir("C:\\Users\\PC-6\\Desktop")
# read data
df = pd.read_csv('framingham.csv')

# first glimpse at data
df.head(20)

# data shape
df.shape
```

```
# data types
```

```
    male  age  education  currentSmoker  cigsPerDay  BPMeds
prevalentStroke  \
0      1   39        4.0              0         0.0     0.0
0
1      0   46        2.0              0         0.0     0.0
0
2      1   48        1.0              1        20.0     0.0
0
3      0   61        3.0              1        30.0     0.0
0
4      0   46        3.0              1        23.0     0.0
0
5      0   43        2.0              0         0.0     0.0
0
6      0   63        1.0              0         0.0     0.0
0
7      0   45        2.0              1        20.0     0.0
0
8      1   52        1.0              0         0.0     0.0
0
9      1   43        1.0              1        30.0     0.0
0
10     0   50        1.0              0         0.0     0.0
0
11     0   43        2.0              0         0.0     0.0
0
12     1   46        1.0              1        15.0     0.0
0
13     0   41        3.0              0         0.0     1.0
0
14     0   39        2.0              1         9.0     0.0
0
15     0   38        2.0              1        20.0     0.0
0
16     1   48        3.0              1        10.0     0.0
0
17     0   46        2.0              1        20.0     0.0
0
18     0   38        2.0              1         5.0     0.0
0
19     1   41        2.0              0         0.0     0.0
0

    prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  heartRate
glucose  \
0              0         0    195.0  106.0   70.0  26.97       80.0
77.0
```

```
1               0          0    250.0  121.0   81.0  28.73        95.0
76.0
2               0          0    245.0  127.5   80.0  25.34        75.0
70.0
3               1          0    225.0  150.0   95.0  28.58        65.0
103.0
4               0          0    285.0  130.0   84.0  23.10        85.0
85.0
5               1          0    228.0  180.0  110.0  30.30        77.0
99.0
6               0          0    205.0  138.0   71.0  33.11        60.0
85.0
7               0          0    313.0  100.0   71.0  21.68        79.0
78.0
8               1          0    260.0  141.5   89.0  26.36        76.0
79.0
9               1          0    225.0  162.0  107.0  23.61        93.0
88.0
10              0          0    254.0  133.0   76.0  22.91        75.0
76.0
11              0          0    247.0  131.0   88.0  27.64        72.0
61.0
12              1          0    294.0  142.0   94.0  26.31        98.0
64.0
13              1          0    332.0  124.0   88.0  31.31        65.0
84.0
14              0          0    226.0  114.0   64.0  22.35        85.0
NaN
15              1          0    221.0  140.0   90.0  21.35        95.0
70.0
16              1          0    232.0  138.0   90.0  22.37        64.0
72.0
17              0          0    291.0  112.0   78.0  23.38        80.0
89.0
18              0          0    195.0  122.0   84.5  23.24        75.0
78.0
19              0          0    195.0  139.0   88.0  26.88        85.0
65.0

    TenYearCHD
0            0
1            0
2            0
3            1
4            0
5            0
6            1
7            0
8            0
```

```
9             0
10            0
11            0
12            0
13            0
14            0
15            1
16            0
17            1
18            0
19            0
```

```
(4240, 16)
```

```
# check for dupicates
duplicate_df = df[df.duplicated()]
duplicate_df
```

```
Empty DataFrame
Columns: [male, age, education, currentSmoker, cigsPerDay, BPMeds,
prevalentStroke, prevalentHyp, diabetes, totChol, sysBP, diaBP, BMI,
heartRate, glucose, TenYearCHD]
Index: []
```

```
# checking for missing values
df.isna().sum()
null = df[df.isna().any(axis=1)]
null
```

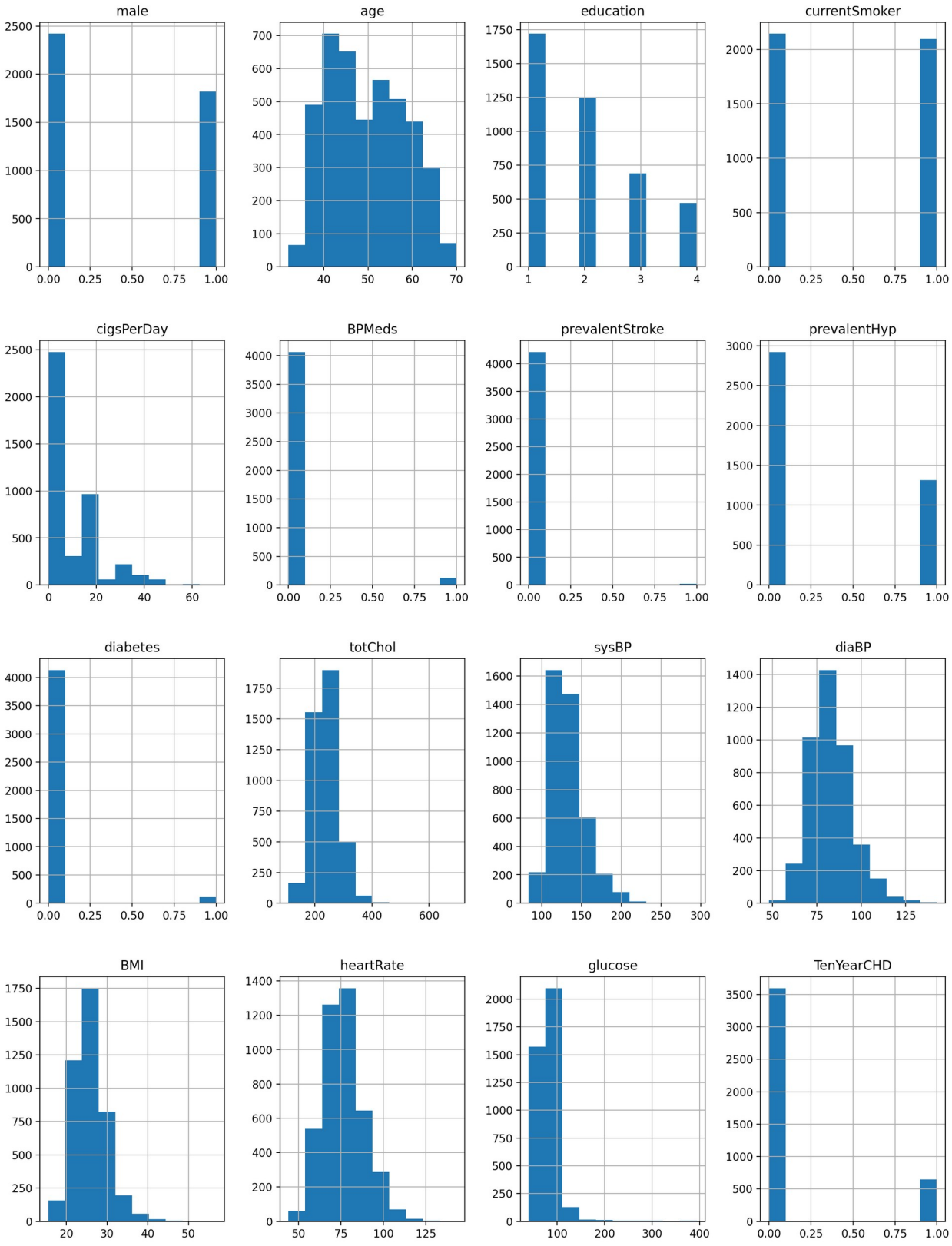
```
male                0
age                 0
education         105
currentSmoker       0
cigsPerDay         29
BPMeds             53
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            50
sysBP               0
diaBP               0
BMI                19
heartRate           1
glucose           388
TenYearCHD          0
dtype: int64
```

```
      male  age  education  currentSmoker  cigsPerDay  BPMeds  \
14       0   39        2.0              1         9.0     0.0
21       0   43        1.0              0         0.0     0.0
26       0   60        1.0              0         0.0     0.0
```

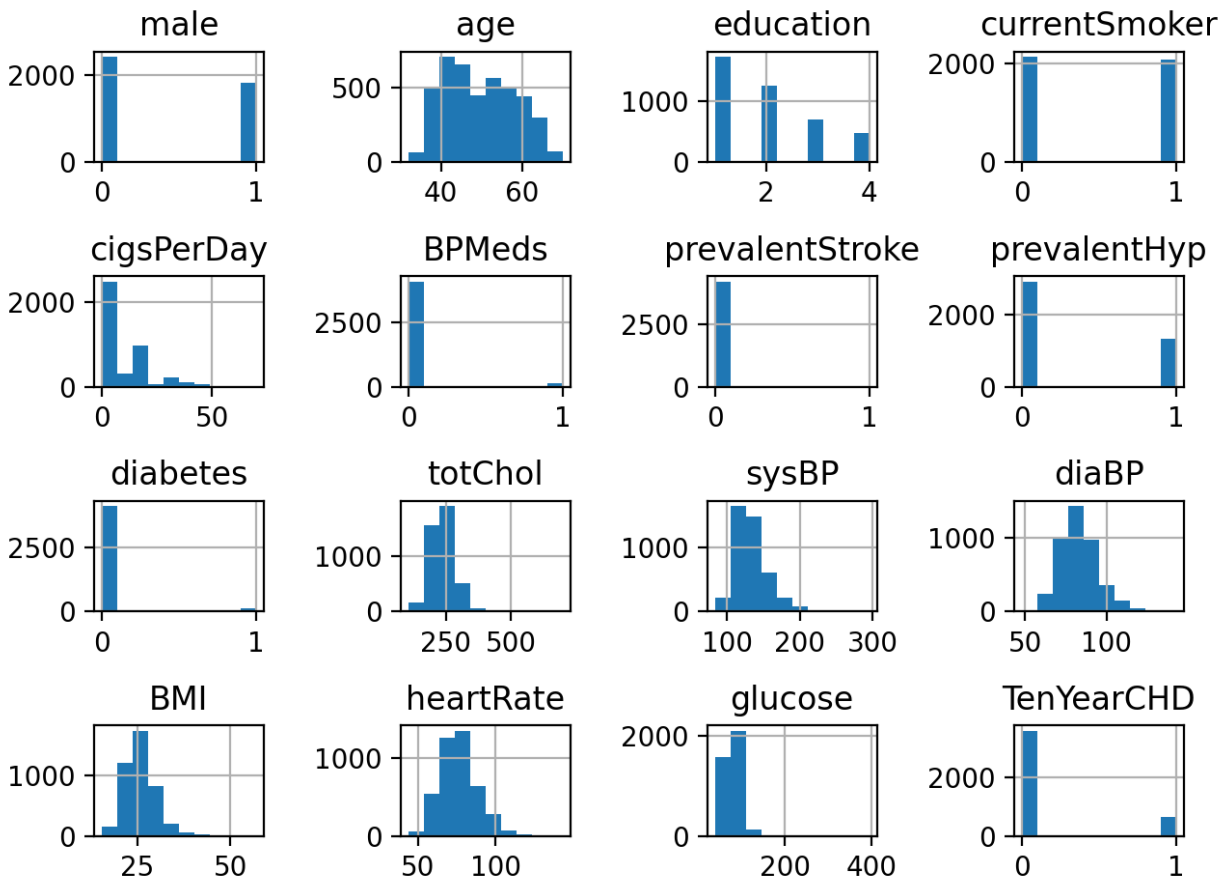
```
33       1   61        NaN              1         5.0     0.0
36       1   56        NaN              0         0.0     0.0
...    ...  ...        ...            ...         ...     ...
4208     0   51        1.0              1         9.0     0.0
4229     0   51        3.0              1        20.0     0.0
4230     0   56        1.0              1         3.0     0.0
4235     0   48        2.0              1        20.0     NaN
4236     0   44        1.0              1        15.0     0.0

      prevalentStroke  prevalentHyp  diabetes  totChol  sysBP  diaBP
BMI  \
14                  0             0         0    226.0  114.0   64.0
22.35
21                  0             0         0    185.0  123.5   77.5
29.89
26                  0             0         0    260.0  110.0   72.5
26.59
33                  0             0         0    175.0  134.0   82.5
18.59
36                  0             0         0    257.0  153.5  102.0
28.09
...               ...           ...       ...      ...    ...    ...
...
4208                0             0         0    340.0  152.0   76.0
25.74
4229                0             1         0    251.0  140.0   80.0
25.60
4230                0             1         0    268.0  170.0  102.0
22.89
4235                0             0         0    248.0  131.0   72.0
22.00
4236                0             0         0    210.0  126.5   87.0
19.16

      heartRate  glucose  TenYearCHD
14         85.0      NaN           0
21         70.0      NaN           0
26         65.0      NaN           0
33         72.0     75.0           1
36         72.0     75.0           0
...         ...      ...         ...
4208       70.0      NaN           0
4229       75.0      NaN           0
4230       57.0      NaN           0
4235       84.0     86.0           0
4236       86.0      NaN           0

[582 rows x 16 columns]
```

```python
import matplotlib.pyplot as plt

# Assuming df is your dataframe
fig = plt.figure(figsize=(15, 20))
df.hist()  # Automatically handles subplot layout
plt.tight_layout()
plt.show()
```

```
array([[<Axes: title={'center': 'male'}>,
        <Axes: title={'center': 'age'}>,
        <Axes: title={'center': 'education'}>,
        <Axes: title={'center': 'currentSmoker'}>],
       [<Axes: title={'center': 'cigsPerDay'}>,
        <Axes: title={'center': 'BPMeds'}>,
        <Axes: title={'center': 'prevalentStroke'}>,
        <Axes: title={'center': 'prevalentHyp'}>],
       [<Axes: title={'center': 'diabetes'}>,
        <Axes: title={'center': 'totChol'}>,
        <Axes: title={'center': 'sysBP'}>,
        <Axes: title={'center': 'diaBP'}>],
       [<Axes: title={'center': 'BMI'}>,
        <Axes: title={'center': 'heartRate'}>,
        <Axes: title={'center': 'glucose'}>,
        <Axes: title={'center': 'TenYearCHD'}>]], dtype=object)
```




```
<Figure size 1500x2000 with 0 Axes>
```

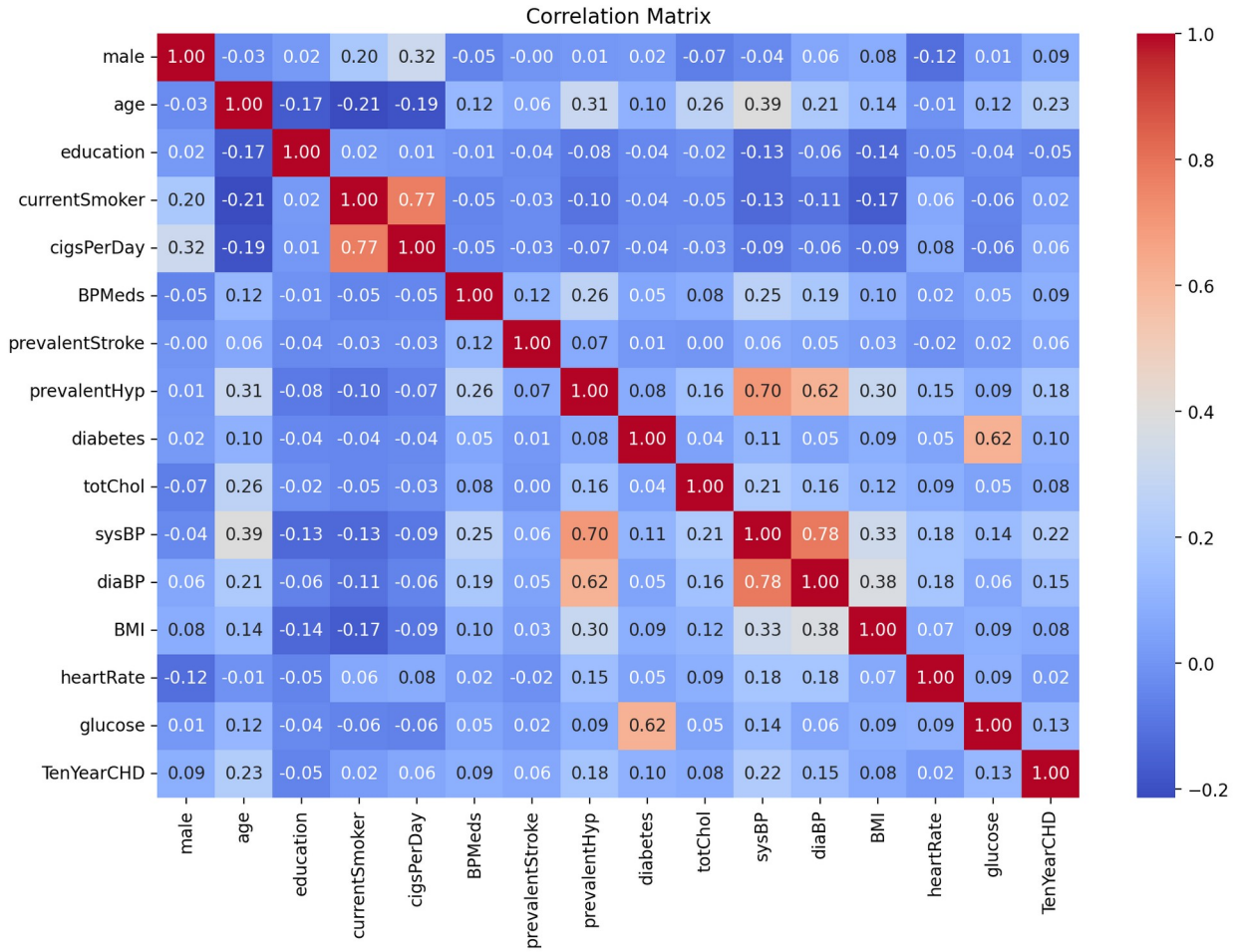



```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your dataframe
df_corr = df.corr()

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df_corr, annot=True, cmap='coolwarm', fmt='.2f',
cbar=True)
plt.title('Correlation Matrix')
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```

```
<Axes: >
```

```
Text(0.5, 1.0, 'Correlation Matrix')
```




```
# Dropping columns education and glucose
df = df.drop(['education'], axis=1)

# Checking for more missing data
df.isna().sum()
```

```
male                0
age                 0
currentSmoker       0
cigsPerDay         29
BPMeds             53
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            50
sysBP               0
diaBP               0
BMI                19
heartRate           1
glucose           388
```

```
TenYearCHD            0
dtype: int64
```

```
# Dropping all rows with missing data
df = df.dropna()
df.isna().sum()
df.columns
```

```
male               0
age                0
currentSmoker      0
cigsPerDay         0
BPMeds             0
prevalentStroke    0
prevalentHyp       0
diabetes           0
totChol            0
sysBP              0
diaBP              0
BMI                0
heartRate          0
glucose            0
TenYearCHD         0
dtype: int64
```

```
Index(['male', 'age', 'currentSmoker', 'cigsPerDay', 'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol',
'sysBP',
       'diaBP', 'BMI', 'heartRate', 'glucose', 'TenYearCHD'],
      dtype='object')
```

```
# Identify the features with the most importance for the outcome
variable Heart Disease

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# separate independent & dependent variables
X = df.iloc[:,0:14]  #independent columns
y = df.iloc[:,-1]    #target column i.e price range

# apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe
```

```
columns
print(featureScores.nlargest(11,'Score'))  #print 10 best features
```

```
           Specs       Score
9          sysBP  667.109932
13       glucose  402.409837
1            age  297.974980
8        totChol  252.958627
3     cigsPerDay  185.115196
10         diaBP  142.920436
6   prevalentHyp   82.342164
7       diabetes   31.711253
4         BPMeds   26.116583
0           male   19.178560
11           BMI   17.108210
```

```
featureScores = featureScores.sort_values(by='Score', ascending=False)
featureScores
```

```
              Specs       Score
9             sysBP  667.109932
13          glucose  402.409837
1               age  297.974980
8           totChol  252.958627
3         cigsPerDay  185.115196
10             diaBP  142.920436
6      prevalentHyp   82.342164
7          diabetes   31.711253
4            BPMeds   26.116583
0              male   19.178560
11              BMI   17.108210
5   prevalentStroke    8.480982
12        heartRate    3.635480
2     currentSmoker    0.904429
```

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,5))

# Assuming featureScores is a DataFrame
sns.barplot(x='Specs', y='Score', data=featureScores)

# Rotate the x-axis labels to prevent overlap
plt.xticks(rotation=45, ha='right', fontsize=12)

# Remove the warning by removing the palette argument
plt.box(False)
plt.title('Feature importance', fontsize=16)
plt.xlabel('\n Features', fontsize=14)
plt.ylabel('Importance \n', fontsize=14)
```

```
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```

```
<Figure size 2000x500 with 0 Axes>
```

```
<Axes: xlabel='Specs', ylabel='Score'>
```

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13],
 [Text(0, 0, 'sysBP'),
  Text(1, 0, 'glucose'),
  Text(2, 0, 'age'),
  Text(3, 0, 'totChol'),
  Text(4, 0, 'cigsPerDay'),
  Text(5, 0, 'diaBP'),
  Text(6, 0, 'prevalentHyp'),
  Text(7, 0, 'diabetes'),
  Text(8, 0, 'BPMeds'),
  Text(9, 0, 'male'),
  Text(10, 0, 'BMI'),
  Text(11, 0, 'prevalentStroke'),
  Text(12, 0, 'heartRate'),
  Text(13, 0, 'currentSmoker')])
```

```
Text(0.5, 1.0, 'Feature importance')
```

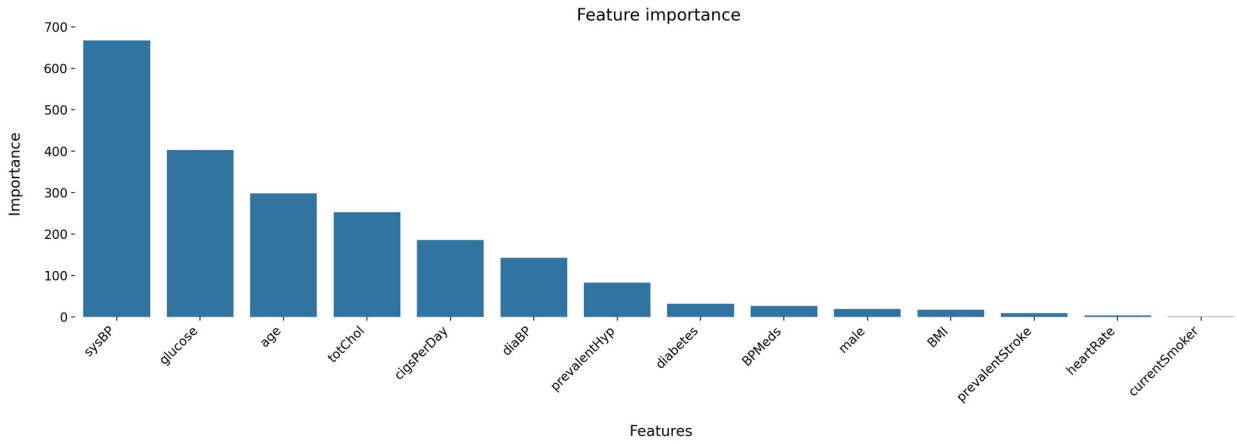
```
Text(0.5, 0, '\n Features')
```

```
Text(0, 0.5, 'Importance \n')
```

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13],
 [Text(0, 0, 'sysBP'),
  Text(1, 0, 'glucose'),
  Text(2, 0, 'age'),
  Text(3, 0, 'totChol'),
  Text(4, 0, 'cigsPerDay'),
  Text(5, 0, 'diaBP'),
  Text(6, 0, 'prevalentHyp'),
  Text(7, 0, 'diabetes'),
  Text(8, 0, 'BPMeds'),
  Text(9, 0, 'male'),
  Text(10, 0, 'BMI'),
  Text(11, 0, 'prevalentStroke'),
  Text(12, 0, 'heartRate'),
  Text(13, 0, 'currentSmoker')])
```

```
(array([  0., 100., 200., 300., 400., 500., 600., 700., 800.]),
 [Text(0, 0.0, '0'),
  Text(0, 100.0, '100'),
  Text(0, 200.0, '200'),
```

```
  Text(0, 300.0, '300'),
  Text(0, 400.0, '400'),
  Text(0, 500.0, '500'),
  Text(0, 600.0, '600'),
  Text(0, 700.0, '700'),
  Text(0, 800.0, '800')])
```




```
# selecting the 10 most impactful features for the target variable
features_list = featureScores["Specs"].tolist()[:10]
features_list
```

```
['sysBP',
 'glucose',
 'age',
 'totChol',
 'cigsPerDay',
 'diaBP',
 'prevalentHyp',
 'diabetes',
 'BPMeds',
 'male']
```

```
# Create new dataframe with selected features

df = df[['sysBP',
'glucose','age','totChol','cigsPerDay','diaBP','prevalentHyp','diabete
s','BPMeds','male','TenYearCHD']]
df.head()
```

```
   sysBP  glucose  age  totChol  cigsPerDay  diaBP  prevalentHyp
diabetes  \
0  106.0     77.0   39    195.0         0.0   70.0             0
0
1  121.0     76.0   46    250.0         0.0   81.0             0
0
2  127.5     70.0   48    245.0        20.0   80.0             0
```

```
0
3  150.0    103.0   61    225.0         30.0   95.0              1
0
4  130.0     85.0   46    285.0         23.0   84.0              0
0
```

```
   BPMeds  male  TenYearCHD
0     0.0     1           0
1     0.0     0           0
2     0.0     1           0
3     0.0     0           1
4     0.0     0           0
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate correlation matrix
df_corr = df.corr()

# Plot heatmap
plt.figure(figsize=(12, 8))  # Adjust figure size for better
readability
sns.heatmap(df_corr, annot=True, cmap='coolwarm', fmt=".2f",
linewidths=0.5, cbar_kws={'shrink': 0.8})

# Optional: Title and labels
plt.title('Correlation Heatmap', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```

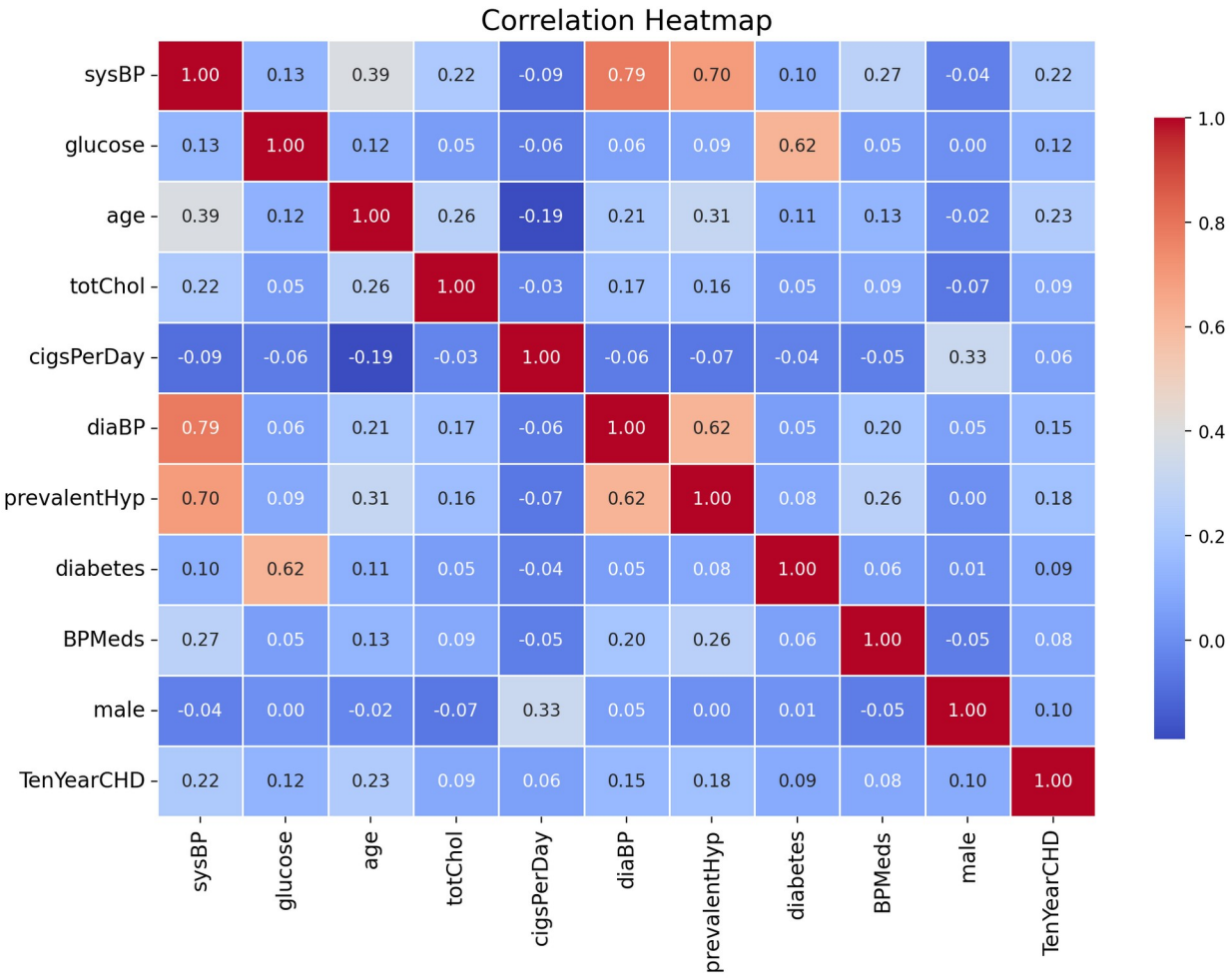
```
<Figure size 1200x800 with 0 Axes>
```

```
<Axes: >
```

```
Text(0.5, 1.0, 'Correlation Heatmap')
```

```
(array([ 0.5,  1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,  8.5,  9.5,
10.5]),
 [Text(0.5, 0, 'sysBP'),
  Text(1.5, 0, 'glucose'),
  Text(2.5, 0, 'age'),
  Text(3.5, 0, 'totChol'),
  Text(4.5, 0, 'cigsPerDay'),
  Text(5.5, 0, 'diaBP'),
  Text(6.5, 0, 'prevalentHyp'),
  Text(7.5, 0, 'diabetes'),
  Text(8.5, 0, 'BPMeds'),
  Text(9.5, 0, 'male'),
  Text(10.5, 0, 'TenYearCHD')])
```

```
(array([ 0.5,  1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,  8.5,  9.5,
10.5]),
 [Text(0, 0.5, 'sysBP'),
  Text(0, 1.5, 'glucose'),
  Text(0, 2.5, 'age'),
  Text(0, 3.5, 'totChol'),
  Text(0, 4.5, 'cigsPerDay'),
  Text(0, 5.5, 'diaBP'),
  Text(0, 6.5, 'prevalentHyp'),
  Text(0, 7.5, 'diabetes'),
  Text(0, 8.5, 'BPMeds'),
  Text(0, 9.5, 'male'),
  Text(0, 10.5, 'TenYearCHD')])
```



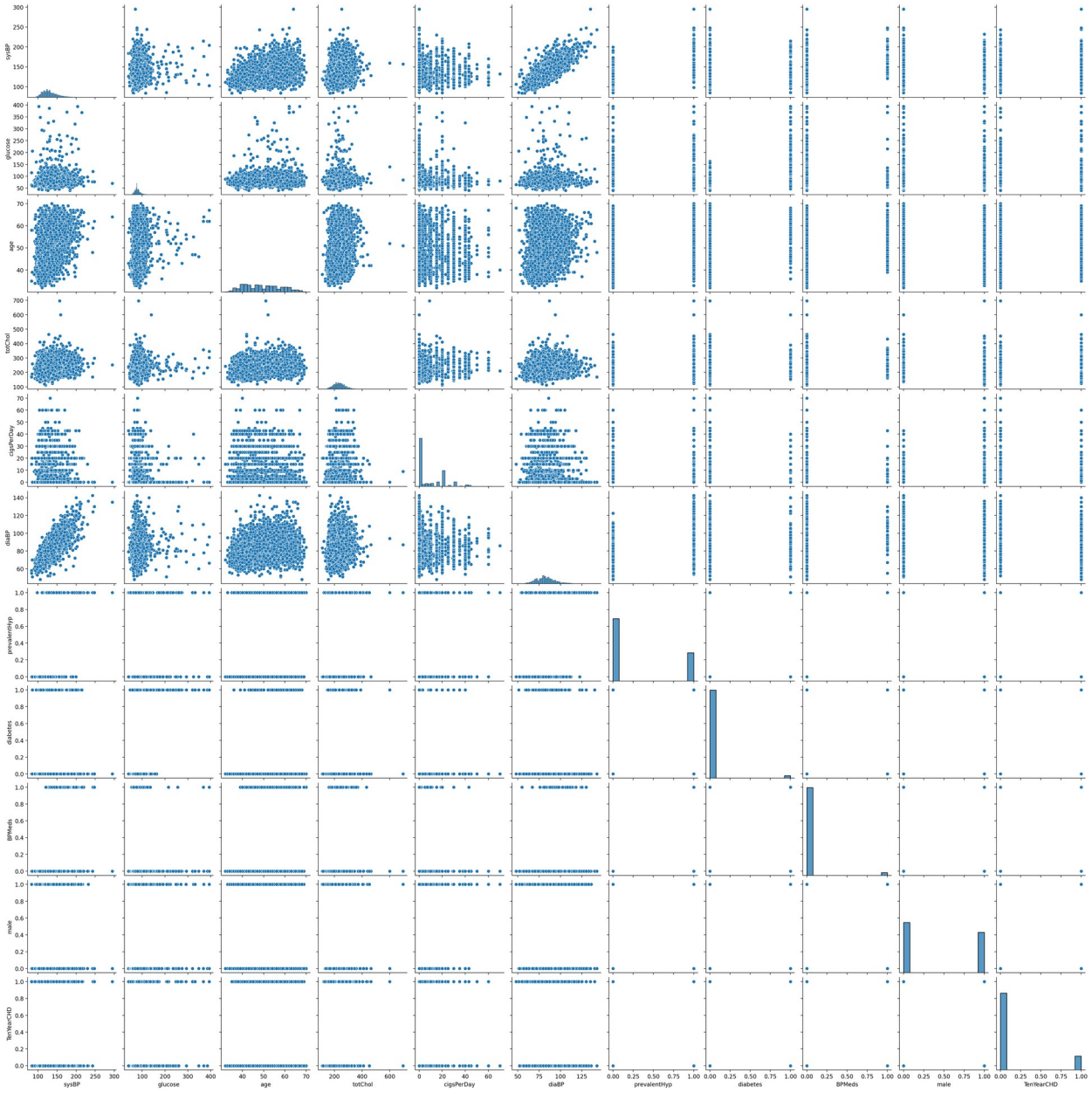

```
# Checking for outliers
df.describe()
sns.pairplot(df)
```

```
              sysBP      glucose          age      totChol   cigsPerDay
\
count  3751.000000  3751.000000  3751.000000  3751.000000  3751.000000

mean    132.368435    81.880032    49.573447   236.928019     9.008531

std      22.046522    23.882233     8.570204    44.611594    11.925097

min      83.500000    40.000000    32.000000   113.000000     0.000000

25%     117.000000    71.000000    42.000000   206.000000     0.000000

50%     128.000000    78.000000    49.000000   234.000000     0.000000

75%     144.000000    87.000000    56.000000   264.000000    20.000000

max     295.000000   394.000000    70.000000   696.000000    70.000000


              diaBP  prevalentHyp     diabetes       BPMeds
male  \
count  3751.000000   3751.000000  3751.000000  3751.000000
3751.000000
mean     82.938550      0.311917     0.027193     0.030392
0.445215
std      11.932779      0.463338     0.162666     0.171686
0.497056
min      48.000000      0.000000     0.000000     0.000000
0.000000
25%      75.000000      0.000000     0.000000     0.000000
0.000000
50%      82.000000      0.000000     0.000000     0.000000
0.000000
75%      90.000000      1.000000     0.000000     0.000000
1.000000
max     142.500000      1.000000     1.000000     1.000000
1.000000

        TenYearCHD
count  3751.000000
mean      0.152493
std       0.359546
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000

<seaborn.axisgrid.PairGrid at 0x239006469f0>
```

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Describe the dataset for summary statistics
print(df.describe())

# Pairplot to visualize pairwise relationships
sns.pairplot(df)

# Optional: Show the plot
plt.show()
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Detecting outliers
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()
print("Outliers detected:\n", outliers)
```

```
             sysBP      glucose          age      totChol   cigsPerDay
\
count  3751.000000  3751.000000  3751.000000  3751.000000  3751.000000

mean    132.368435    81.880032    49.573447   236.928019     9.008531

std      22.046522    23.882233     8.570204    44.611594    11.925097

min      83.500000    40.000000    32.000000   113.000000     0.000000

25%     117.000000    71.000000    42.000000   206.000000     0.000000

50%     128.000000    78.000000    49.000000   234.000000     0.000000

75%     144.000000    87.000000    56.000000   264.000000    20.000000

max     295.000000   394.000000    70.000000   696.000000    70.000000


             diaBP  prevalentHyp     diabetes       BPMeds
male  \
count  3751.000000   3751.000000  3751.000000  3751.000000
3751.000000
mean     82.938550      0.311917     0.027193     0.030392
0.445215
std      11.932779      0.463338     0.162666     0.171686
0.497056
min      48.000000      0.000000     0.000000     0.000000
0.000000
25%      75.000000      0.000000     0.000000     0.000000
0.000000
50%      82.000000      0.000000     0.000000     0.000000
```

```
0.000000
75%      90.000000      1.000000     0.000000     0.000000
1.000000
max     142.500000      1.000000     1.000000     1.000000
1.000000

        TenYearCHD
count  3751.000000
mean      0.152493
std       0.359546
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000

<seaborn.axisgrid.PairGrid at 0x23918af2240>
```



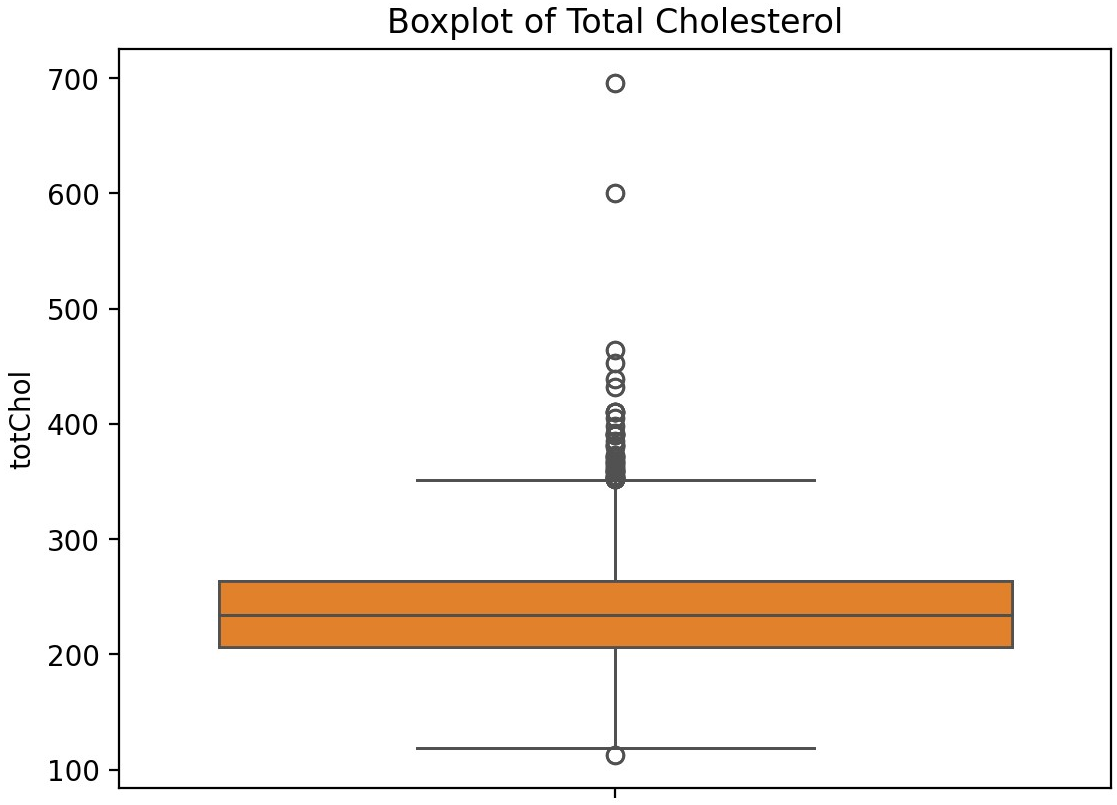

```
Outliers detected:
 sysBP           110
glucose          181
age                0
totChol           42
cigsPerDay        11
diaBP             69
prevalentHyp       0
diabetes         102
BPMeds           114
male               0
```

```
TenYearCHD      572
dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a boxplot to visualize the distribution of 'totChol'
sns.boxplot(df['totChol'])
plt.title('Boxplot of Total Cholesterol')
plt.show()

# Identify outliers in the 'totChol' column (values greater than 500
in this case)
outliers = df[df['totChol'] > 500]

# Display the outliers
print(outliers)
```

```
<Axes: ylabel='totChol'>
```

```
Text(0.5, 1.0, 'Boxplot of Total Cholesterol')
```

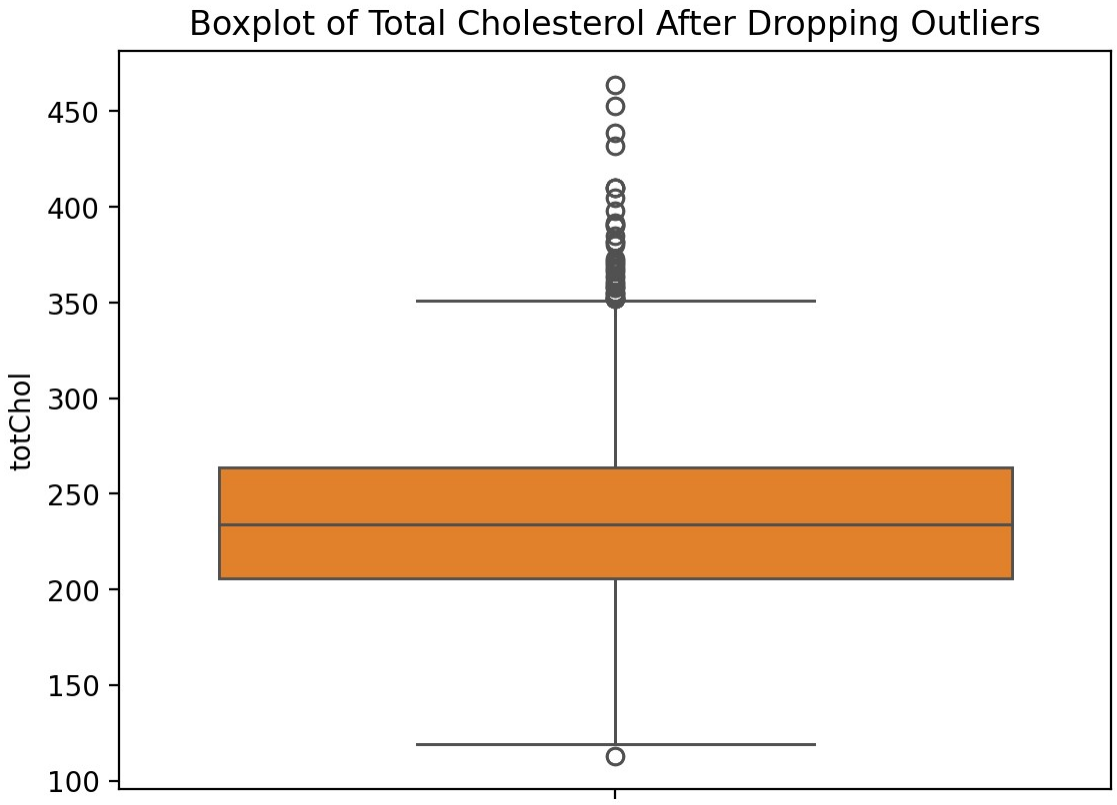



```
      sysBP  glucose  age  totChol  cigsPerDay  diaBP  prevalentHyp
diabetes  \
1111  159.5    140.0   52    600.0         0.0   94.0             1
```

```
1
3160  157.0     84.0   51    696.0          9.0   87.0                1
0

      BPMeds  male  TenYearCHD
1111     0.0     0           1
3160     0.0     1           0
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Drop outliers where 'totChol' is greater than 599
df = df.drop(df[df['totChol'] > 599].index)

# Create a boxplot to visualize the distribution of 'totChol' after
dropping outliers
sns.boxplot(df['totChol'])
plt.title('Boxplot of Total Cholesterol After Dropping Outliers')
plt.show()
```

```
<Axes: ylabel='totChol'>

Text(0.5, 1.0, 'Boxplot of Total Cholesterol After Dropping Outliers')
```




```
df_clean = df
```

```
scaler = MinMaxScaler(feature_range=(0,1))

#assign scaler to column:
df_scaled = pd.DataFrame(scaler.fit_transform(df_clean),
columns=df_clean.columns)

df_scaled.describe()
df.describe()
```

```
             sysBP      glucose          age      totChol   cigsPerDay
\
count  3749.000000  3749.000000  3749.000000  3749.000000  3749.000000

mean      0.230991     0.118260     0.462432     0.352447     0.128728

std       0.104228     0.067429     0.225589     0.124179     0.170391

min       0.000000     0.000000     0.000000     0.000000     0.000000

25%       0.158392     0.087571     0.263158     0.264957     0.000000

50%       0.210402     0.107345     0.447368     0.344729     0.000000

75%       0.283688     0.132768     0.631579     0.430199     0.285714

max       1.000000     1.000000     1.000000     1.000000     1.000000


             diaBP  prevalentHyp     diabetes       BPMeds
male  \
count  3749.000000   3749.000000  3749.000000  3749.000000
3749.000000
mean      0.369677      0.311550     0.026941     0.030408
0.445185
std       0.126290      0.463189     0.161931     0.171730
0.497053
min       0.000000      0.000000     0.000000     0.000000
0.000000
25%       0.285714      0.000000     0.000000     0.000000
0.000000
50%       0.359788      0.000000     0.000000     0.000000
0.000000
75%       0.444444      1.000000     0.000000     0.000000
1.000000
max       1.000000      1.000000     1.000000     1.000000
1.000000

        TenYearCHD
count  3749.000000
mean      0.152307
std       0.359366
```

```
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000

             sysBP      glucose          age      totChol   cigsPerDay
\
count  3749.000000  3749.000000  3749.000000  3749.000000  3749.000000

mean    132.354628    81.863964    49.572419   236.708722     9.010936

std      22.044275    23.869703     8.572367    43.586786    11.927370

min      83.500000    40.000000    32.000000   113.000000     0.000000

25%     117.000000    71.000000    42.000000   206.000000     0.000000

50%     128.000000    78.000000    49.000000   234.000000     0.000000

75%     143.500000    87.000000    56.000000   264.000000    20.000000

max     295.000000   394.000000    70.000000   464.000000    70.000000


             diaBP  prevalentHyp     diabetes       BPMeds
male  \
count  3749.000000   3749.000000  3749.000000  3749.000000
3749.000000
mean     82.934516      0.311550     0.026941     0.030408
0.445185
std      11.934410      0.463189     0.161931     0.171730
0.497053
min      48.000000      0.000000     0.000000     0.000000
0.000000
25%      75.000000      0.000000     0.000000     0.000000
0.000000
50%      82.000000      0.000000     0.000000     0.000000
0.000000
75%      90.000000      1.000000     0.000000     0.000000
1.000000
max     142.500000      1.000000     1.000000     1.000000
1.000000

        TenYearCHD
count  3749.000000
mean      0.152307
std       0.359366
min       0.000000
25%       0.000000
```

```
50%       0.000000
75%       0.000000
max       1.000000
```

Test - Train Split

```
# clarify what is y and what is x label
y = df_scaled['TenYearCHD']
X = df_scaled.drop(['TenYearCHD'], axis = 1)

# divide train test: 80 % - 20 %
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=29)

len(X_train)
len(X_test)
```

```
2999
```

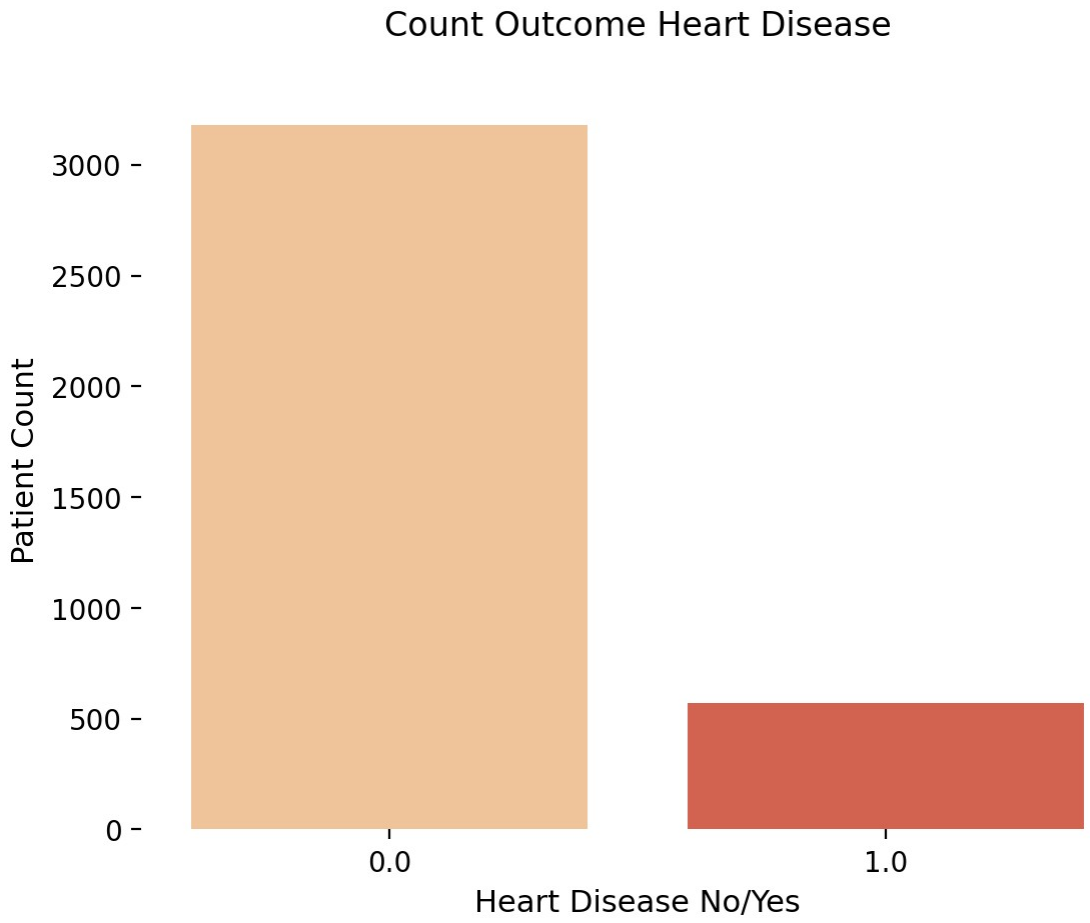
```
750
```

Resampling imbalanced Dataset

```
import seaborn as sns
import matplotlib.pyplot as plt

# Checking balance of outcome variable
target_count = df_scaled.TenYearCHD.value_counts()
print('Class 0:', target_count[0])
print('Class 1:', target_count[1])
print('Proportion:', round(target_count[0] / target_count[1], 2), ':
1')

# Create the countplot with the x argument explicitly set
sns.countplot(x='TenYearCHD', data=df_scaled, palette="OrRd",
hue=None)

# Customize the plot
plt.box(False)
plt.xlabel('Heart Disease No/Yes', fontsize=11)
plt.ylabel('Patient Count', fontsize=11)
plt.title('Count Outcome Heart Disease\n')
plt.savefig('Balance_Heart_Disease.png')
plt.show()
```

```
Class 0: 3178
Class 1: 571
Proportion: 5.57 : 1
```

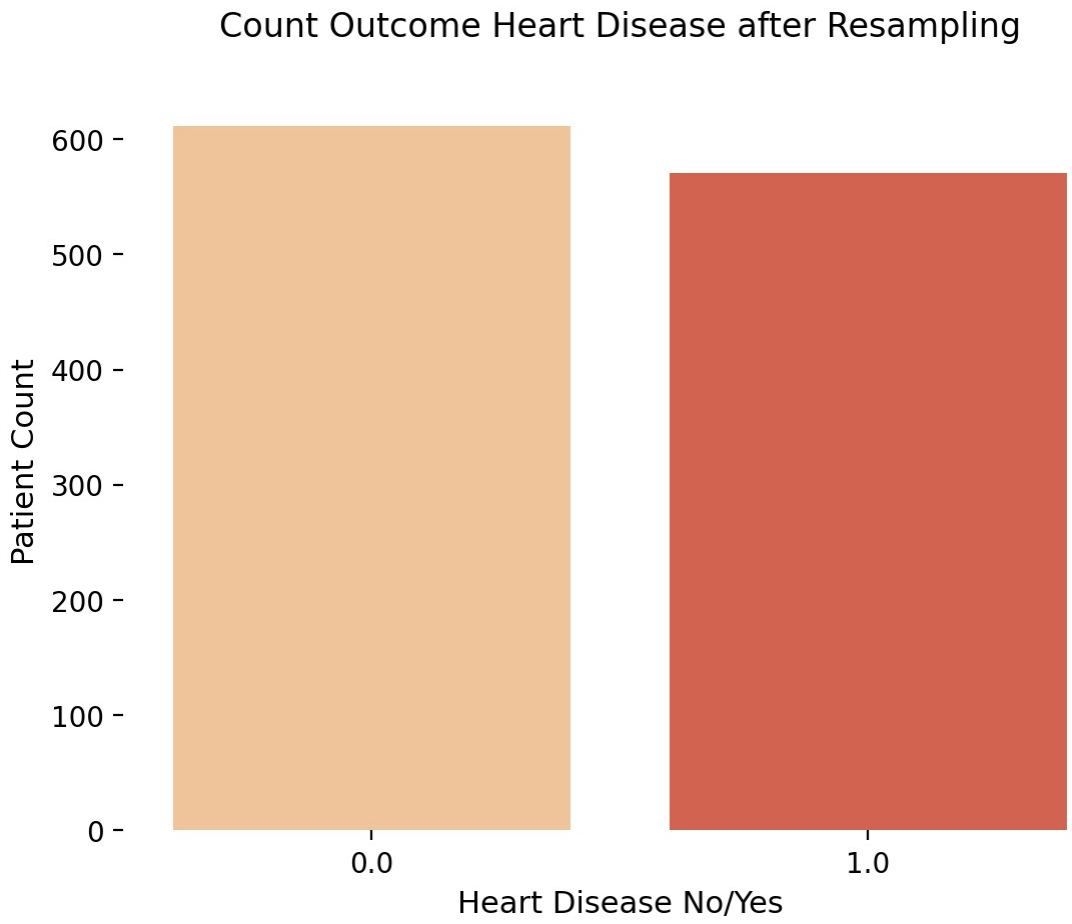
```
C:\Users\PC-6\AppData\Local\Temp\ipykernel_1176\3551037873.py:11:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='TenYearCHD', data=df_scaled, palette="OrRd",
hue=None)

<Axes: xlabel='TenYearCHD', ylabel='count'>

Text(0.5, 0, 'Heart Disease No/Yes')

Text(0, 0.5, 'Patient Count')

Text(0.5, 1.0, 'Count Outcome Heart Disease\n')
```




```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Shuffle df
shuffled_df = df_scaled.sample(frac=1, random_state=4)

# Put all the heart disease cases in a separate dataset
CHD_df = shuffled_df.loc[shuffled_df['TenYearCHD'] == 1]

# Randomly select observations from the non-heart disease (majority
class)
non_CHD_df = shuffled_df.loc[shuffled_df['TenYearCHD'] ==
0].sample(n=611, random_state=42)

# Concatenate both dataframes
normalized_df = pd.concat([CHD_df, non_CHD_df])

# Check new class counts
print(normalized_df.TenYearCHD.value_counts())

# Plot new count
sns.countplot(x='TenYearCHD', data=normalized_df, palette="OrRd",
hue=None)
plt.box(False)
plt.xlabel('Heart Disease No/Yes', fontsize=11)
plt.ylabel('Patient Count', fontsize=11)
plt.title('Count Outcome Heart Disease after Resampling\n')
# plt.savefig('Balance Heart Disease.png')
plt.show()
```

```
TenYearCHD
0.0    611
1.0    571
Name: count, dtype: int64

C:\Users\PC-6\AppData\Local\Temp\ipykernel_1176\2733195083.py:20:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='TenYearCHD', data=normalized_df, palette="OrRd",
hue=None)

<Axes: xlabel='TenYearCHD', ylabel='count'>

Text(0.5, 0, 'Heart Disease No/Yes')

Text(0, 0.5, 'Patient Count')

Text(0.5, 1.0, 'Count Outcome Heart Disease after Resampling\n')
```




```
y_train = normalized_df['TenYearCHD']
X_train = normalized_df.drop('TenYearCHD', axis=1)

from sklearn.pipeline import Pipeline

classifiers =
[LogisticRegression(),SVC(),DecisionTreeClassifier(),KNeighborsClassif
ier(2)]

for classifier in classifiers:
    pipe = Pipeline(steps=[('classifier', classifier)])
    pipe.fit(X_train, y_train)
    print("The accuracy score of {0} is: {1:.2f}%".format(classifier,
(pipe.score(X_test, y_test)*100)))


Pipeline(steps=[('classifier', LogisticRegression())])

The accuracy score of LogisticRegression() is: 65.73%

Pipeline(steps=[('classifier', SVC())])
```

```
The accuracy score of SVC() is: 64.80%

Pipeline(steps=[('classifier', DecisionTreeClassifier())])

The accuracy score of DecisionTreeClassifier() is: 70.13%

Pipeline(steps=[('classifier', KNeighborsClassifier(n_neighbors=2))])

The accuracy score of KNeighborsClassifier(n_neighbors=2) is: 80.00%
```

Modelling & Evaluation (without Pipeline)

```
# logistic regression again with the balanced dataset

normalized_df_reg = LogisticRegression().fit(X_train, y_train)

normalized_df_reg_pred = normalized_df_reg.predict(X_test)

# check accuracy: Accuracy: Overall, how often is the classifier
correct? Accuracy = (True Pos + True Negative)/total
acc = accuracy_score(y_test, normalized_df_reg_pred)
print(f"The accuracy score for LogReg is: {round(acc,3)*100}%")

# f1 score: The F1 score can be interpreted as a weighted average of
the precision and recall, where an F1 score reaches its best value at
1 and worst score at 0.
f1 = f1_score(y_test, normalized_df_reg_pred)
print(f"The f1 score for LogReg is: {round(f1,3)*100}%")

# Precision score: When it predicts yes, how often is it correct?
Precision=True Positive/predicted yes
precision = precision_score(y_test, normalized_df_reg_pred)
print(f"The precision score for LogReg is: {round(precision,3)*100}%")

# recall score: True Positive Rate(Sensitivity or Recall): When it's
actually yes, how often does it predict yes? True Positive Rate = True
Positive/actual yes
recall = recall_score(y_test, normalized_df_reg_pred)
print(f"The recall score for LogReg is: {round(recall,3)*100}%")

The accuracy score for LogReg is: 65.7%
The f1 score for LogReg is: 37.2%
The precision score for LogReg is: 26.0%
The recall score for LogReg is: 65.0%

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Ensure normalized_df_reg_pred is the prediction array
```

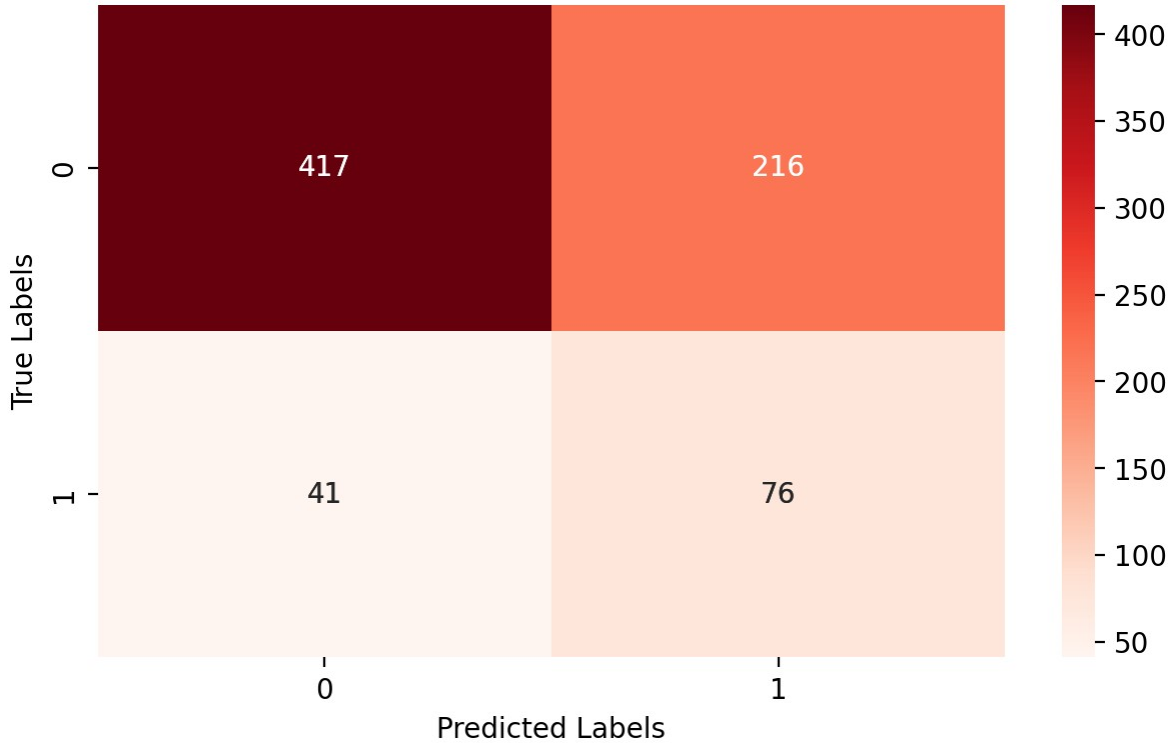
```
# cnf_matrix_log = confusion_matrix(y_test, normalized_df_reg_pred)  # Assuming it's already computed

# Calculate confusion matrix
cnf_matrix_log = confusion_matrix(y_test, normalized_df_reg_pred)

# Plotting the heatmap
sns.heatmap(pd.DataFrame(cnf_matrix_log), annot=True, cmap="Reds", fmt='g', cbar=False)

# Customizing the axes and labels
plt.title('Confusion Matrix Logistic Regression\n', y=1.1)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()

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

```
<Axes: title={'center': 'Confusion matrix Logistic Regression\n'}>
```

```
Text(0.5, 1.1, 'Confusion Matrix Logistic Regression\n')
```

```
Text(0.5, 9.444444444444459, 'Predicted Labels')
```

```
Text(9.444444444444452, 0.5, 'True Labels')
```




```
# Support Vector Machine

#initialize model
svm = SVC()

#fit model
svm.fit(X_train, y_train)

normalized_df_svm_pred = svm.predict(X_test)

# check accuracy: Accuracy: Overall, how often is the classifier
correct? Accuracy = (True Pos + True Negative)/total
acc = accuracy_score(y_test, normalized_df_svm_pred)
print(f"The accuracy score for SVM is: {round(acc,3)*100}%")

# f1 score: The F1 score can be interpreted as a weighted average of
the precision and recall, where an F1 score reaches its best value at
1 and worst score at 0.
f1 = f1_score(y_test, normalized_df_svm_pred)
print(f"The f1 score for SVM is: {round(f1,3)*100}%")

# Precision score: When it predicts yes, how often is it correct?
```

```
Precision=True Positive/predicted yes
precision = precision_score(y_test, normalized_df_svm_pred)
print(f"The precision score for SVM is: {round(precision,3)*100}%")

# recall score: True Positive Rate(Sensitivity or Recall): When it's
actually yes, how often does it predict yes? True Positive Rate = True
Positive/actual yes
recall = recall_score(y_test, normalized_df_svm_pred)
print(f"The recall score for SVM is: {round(recall,3)*100}%")
```

```
SVC()
```

```
The accuracy score for SVM is: 64.8%
The f1 score for SVM is: 37.7%
The precision score for SVM is: 26.1%
The recall score for SVM is: 68.4%
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix

# Assuming y_test and normalized_df_svm_pred are already defined

cnf_matrix_svm = confusion_matrix(y_test, normalized_df_svm_pred)

# Normalize confusion matrix if you want
# cnf_matrix_svm = cnf_matrix_svm.astype('float') /
cnf_matrix_svm.sum(axis=1)[:, np.newaxis]

# Create DataFrame from confusion matrix
cnf_matrix_df = pd.DataFrame(cnf_matrix_svm,
                             index=np.unique(y_test),
                             columns=np.unique(y_test))

# Plot the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_df, annot=True, cmap="Reds", fmt='g',
cbar=True)

# Labels and title
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - SVM', y=1.1)

# Adjust position of x-axis label
ax = plt.gca()
ax.xaxis.set_label_position("top")

# Tight layout to avoid clipping
```

```
plt.tight_layout()
plt.show()
```

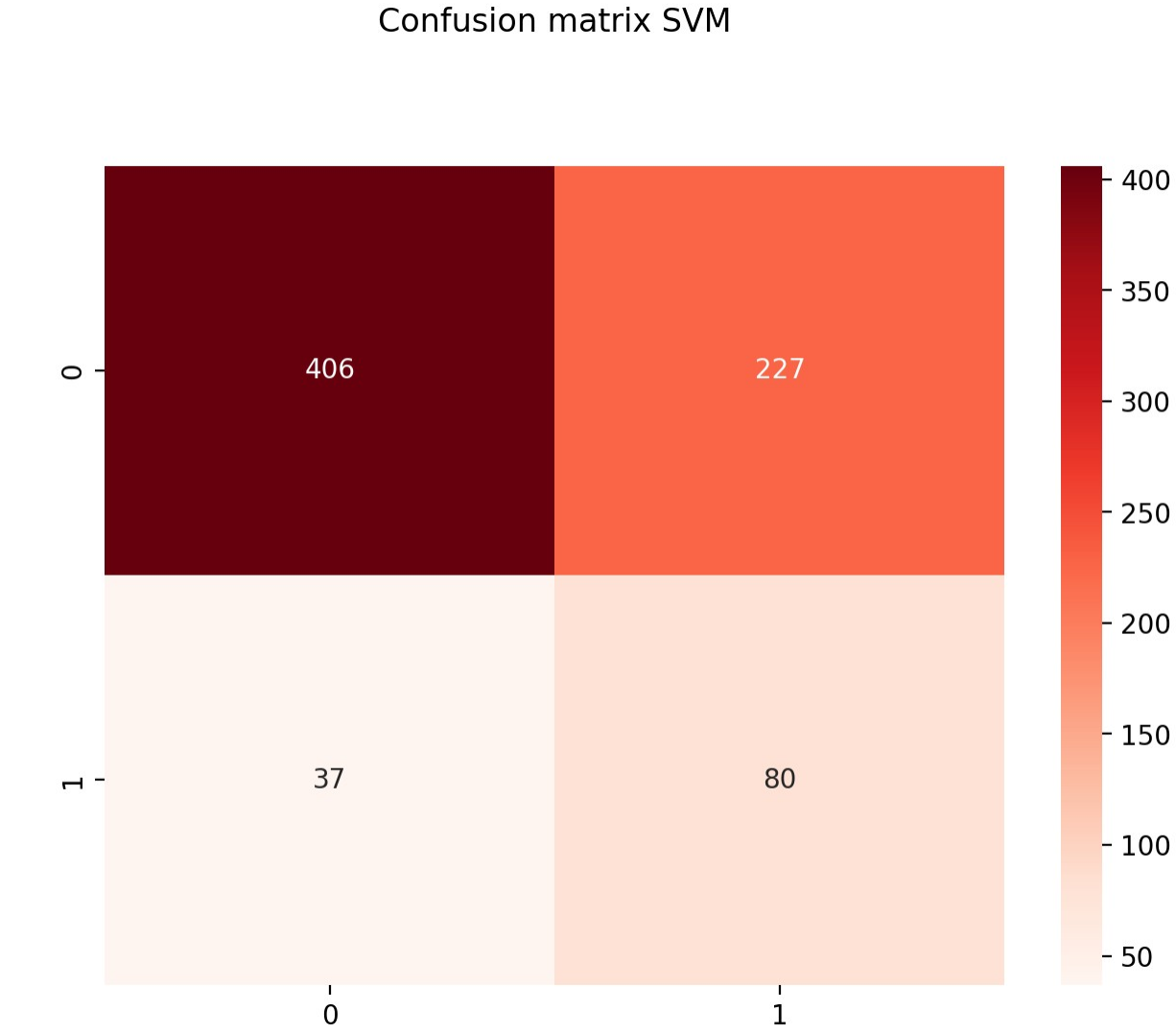
```
<Figure size 800x600 with 0 Axes>
```

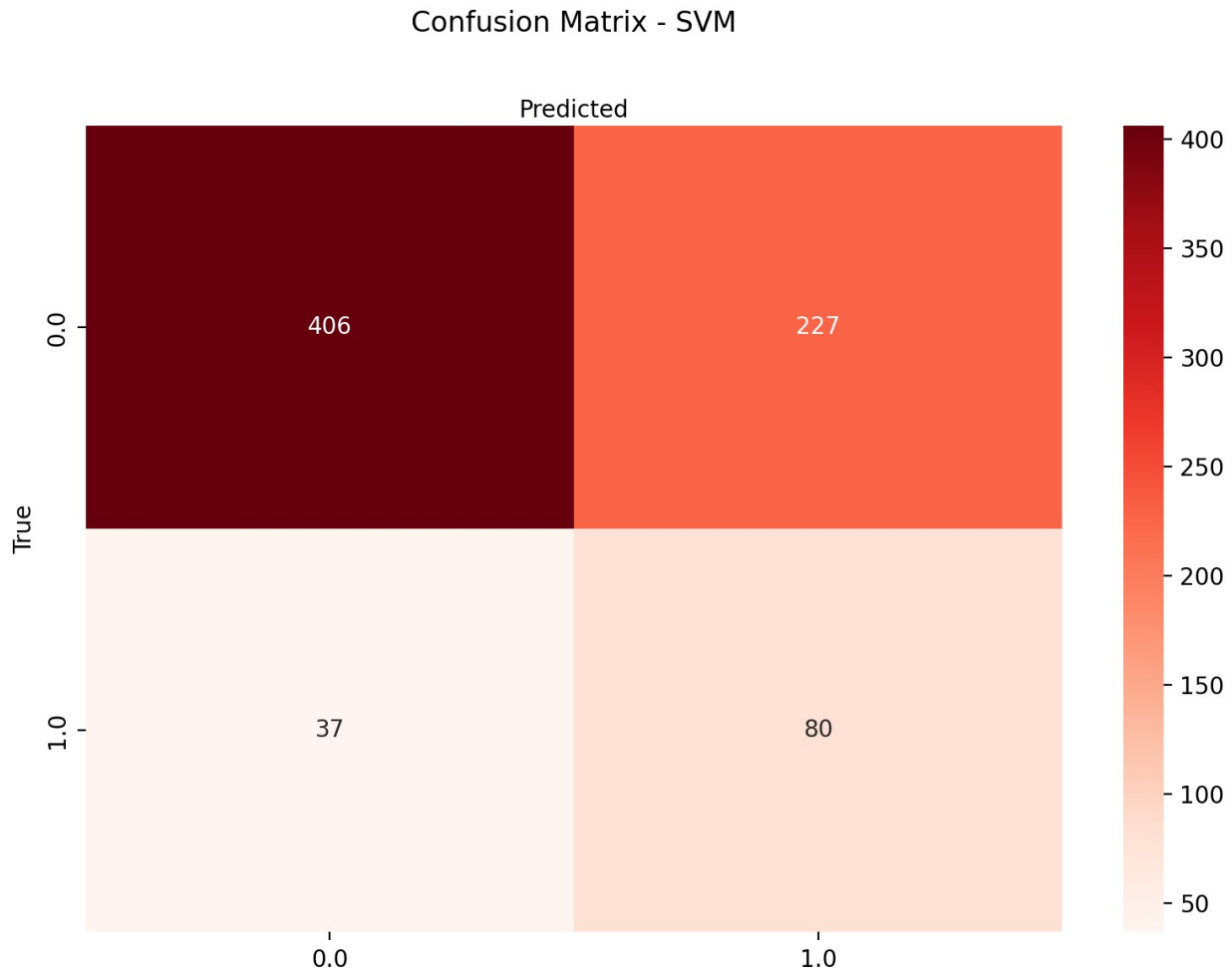
```
<Axes: >
```

```
Text(0.5, 36.72222222222221, 'Predicted')
```

```
Text(70.72222222222221, 0.5, 'True')
```

```
Text(0.5, 1.1, 'Confusion Matrix - SVM')
```






```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Initialize the Decision Tree Classifier model
dtc_up = DecisionTreeClassifier()

# Fit model
dtc_up.fit(X_train, y_train)

# Predict on test set
normalized_df_dtc_pred = dtc_up.predict(X_test)

# Accuracy: Overall, how often is the classifier correct?
acc = accuracy_score(y_test, normalized_df_dtc_pred)
print(f"The accuracy score for DTC is: {round(acc, 3) * 100}%")
```

```
# F1 score: The F1 score is the weighted average of precision and
recall
f1 = f1_score(y_test, normalized_df_dtc_pred)
print(f"The F1 score for DTC is: {round(f1, 3) * 100}%")

# Precision score: When it predicts yes, how often is it correct?
precision = precision_score(y_test, normalized_df_dtc_pred)
print(f"The precision score for DTC is: {round(precision, 3) * 100}%")

# Recall score: When it's actually yes, how often does it predict yes?
recall = recall_score(y_test, normalized_df_dtc_pred)
print(f"The recall score for DTC is: {round(recall, 3) * 100}%")

# Confusion Matrix for more insights
cnf_matrix_dtc = confusion_matrix(y_test, normalized_df_dtc_pred)
cnf_matrix_df = pd.DataFrame(cnf_matrix_dtc,
                             index=np.unique(y_test),
                             columns=np.unique(y_test))

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_df, annot=True, cmap="Blues", fmt='g',
cbar=True)

# Adding labels and title to the plot
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - Decision Tree', y=1.1)

# Adjust position of x-axis label
ax = plt.gca()
ax.xaxis.set_label_position("top")

# Tight layout to avoid clipping
plt.tight_layout()
plt.show()
```

```
DecisionTreeClassifier()
```

```
The accuracy score for DTC is: 70.5%
The F1 score for DTC is: 51.4%
The precision score for DTC is: 34.599999999999994%
The recall score for DTC is: 100.0%
```

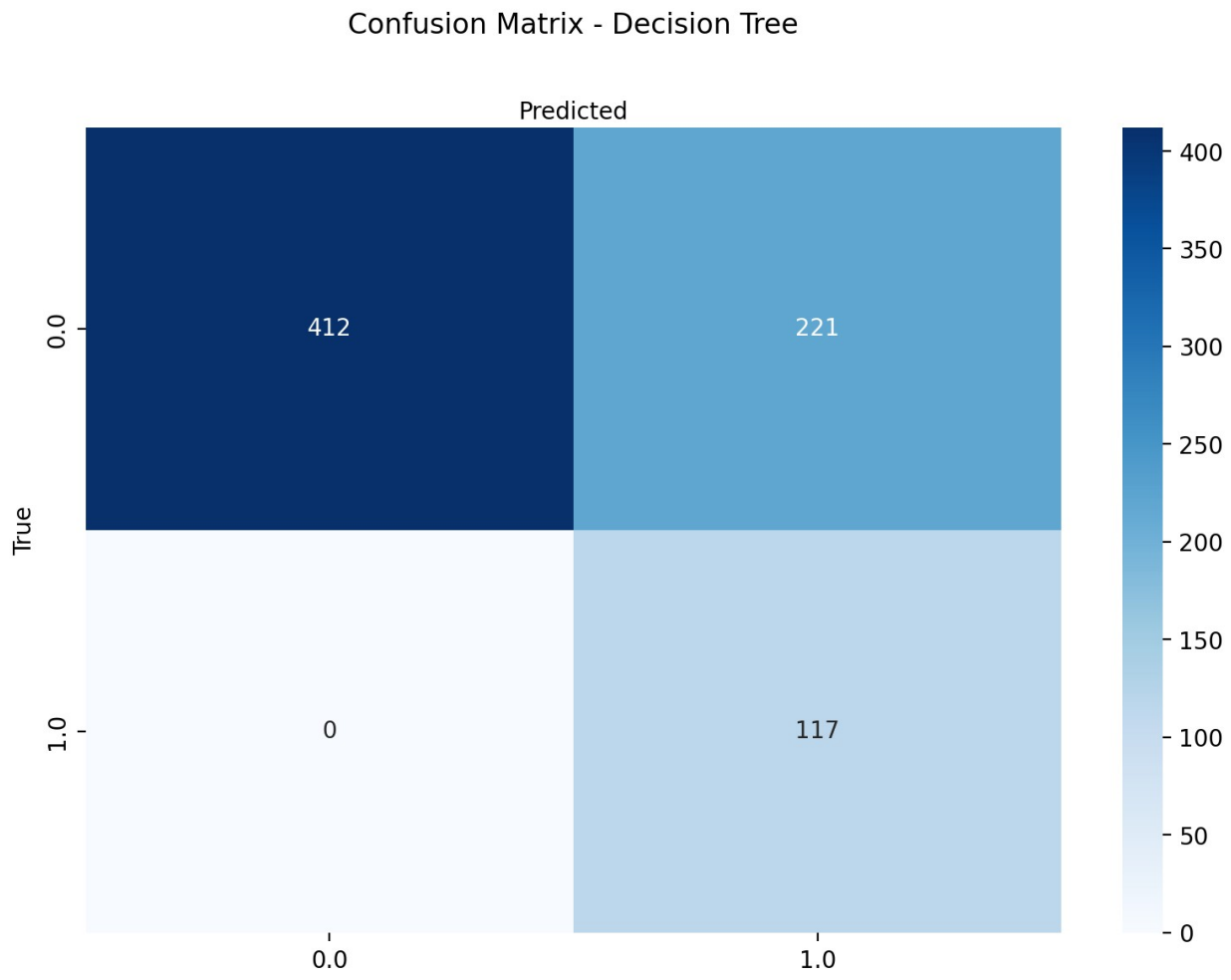
```
<Figure size 800x600 with 0 Axes>
```

```
<Axes: >
```

```
Text(0.5, 36.72222222222221, 'Predicted')
```

```
Text(70.72222222222221, 0.5, 'True')
```

```
Text(0.5, 1.1, 'Confusion Matrix - Decision Tree')
```




```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix

# Assuming y_test and normalized_df_dtc_pred are defined

# Compute confusion matrix
cnf_matrix_dtc = confusion_matrix(y_test, normalized_df_dtc_pred)

# Create DataFrame from confusion matrix for better clarity
cnf_matrix_df = pd.DataFrame(cnf_matrix_dtc,
                             index=np.unique(y_test),
                             columns=np.unique(y_test))

# Plot the confusion matrix
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cnf_matrix_df, annot=True, cmap="Reds", fmt='g',
cbar=True)

# Add axis labels
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - Decision Tree', y=1.1)

# Adjust position of x-axis label
ax = plt.gca()
ax.xaxis.set_label_position("top")

# Tight layout to avoid clipping
plt.tight_layout()
plt.show()
```

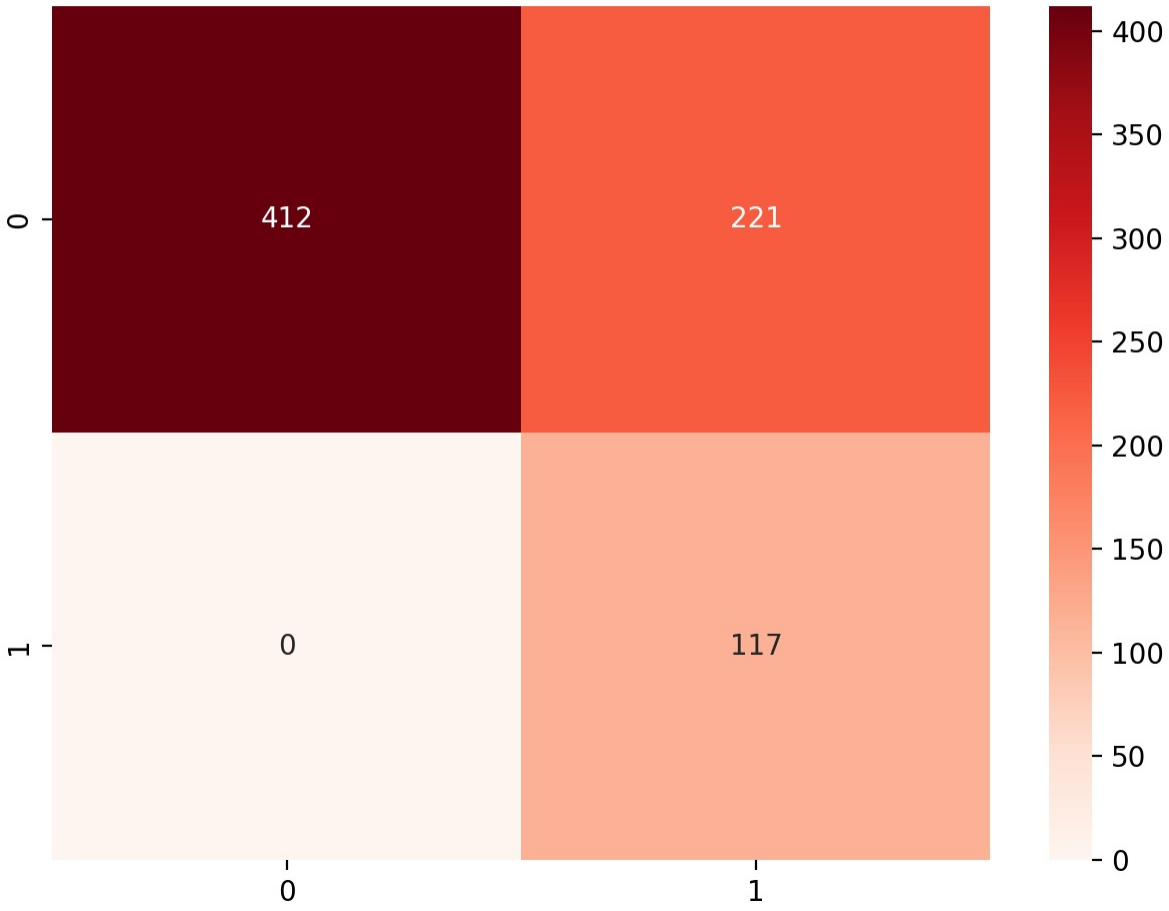
```
<Figure size 800x600 with 0 Axes>
```

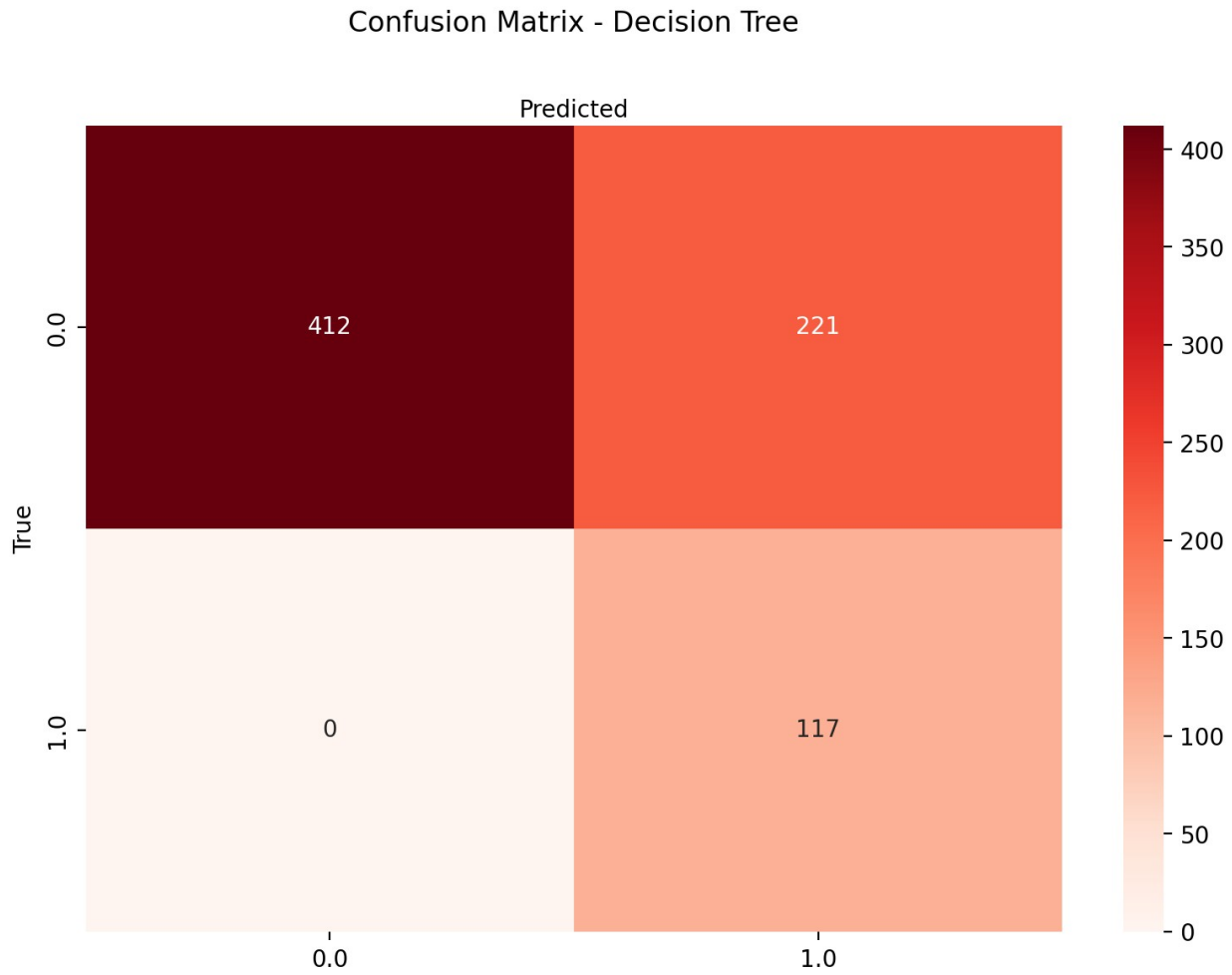
```
<Axes: >
```

```
Text(0.5, 36.72222222222221, 'Predicted')
```

```
Text(70.72222222222221, 0.5, 'True')
```

```
Text(0.5, 1.1, 'Confusion Matrix - Decision Tree')
```


Confusion matrix Decision Tree
412
221
0
117
0
1
0
1
400
350
300
250
200
150
100
50
0




```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for n_neighbors and metric
param_grid = {
    'n_neighbors': [1, 2, 3, 5, 7, 10],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

# Create a KNN classifier
knn = KNeighborsClassifier()

# Initialize GridSearchCV with 5-fold cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

# Fit the model on training data
grid_search.fit(X_train, y_train)

# Get the best parameters from grid search
print("Best parameters from grid search:", grid_search.best_params_)
```

```
# Train the KNN model using the best found parameters
best_knn = grid_search.best_estimator_

# Prediction on test data
normalized_df_knn_pred = best_knn.predict(X_test)

# Evaluate the model
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'metric': ['euclidean', 'manhattan',
'minkowski'],
                         'n_neighbors': [1, 2, 3, 5, 7, 10]},
             scoring='accuracy')
```

```
Best parameters from grid search: {'metric': 'euclidean',
'n_neighbors': 10}
```

```
# Assuming you have already defined best_knn after grid search
acc_test = best_knn.score(X_test, y_test)
print(f"The accuracy score of the test data is: {acc_test*100:.2f}%")

acc_train = best_knn.score(X_train, y_train)
print(f"The accuracy score of the training data is:
{acc_train*100:.2f}%")
```

```
The accuracy score of the test data is: 69.60%
The accuracy score of the training data is: 71.15%
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix

# Assuming y_test and normalized_df_knn_pred are defined
cnf_matrix_knn = confusion_matrix(y_test, normalized_df_knn_pred)

# Create the heatmap
plt.figure(figsize=(8, 6))  # Optional: adjust the figure size for
clarity
ax = plt.subplot()
sns.heatmap(pd.DataFrame(cnf_matrix_knn), annot=True, cmap="Reds",
fmt='g', cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class
0', 'Class 1'])

# Set labels and title
ax.set_xlabel('Predicted')
ax.set_ylabel('True')
ax.set_title('Confusion Matrix for KNN')
```

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

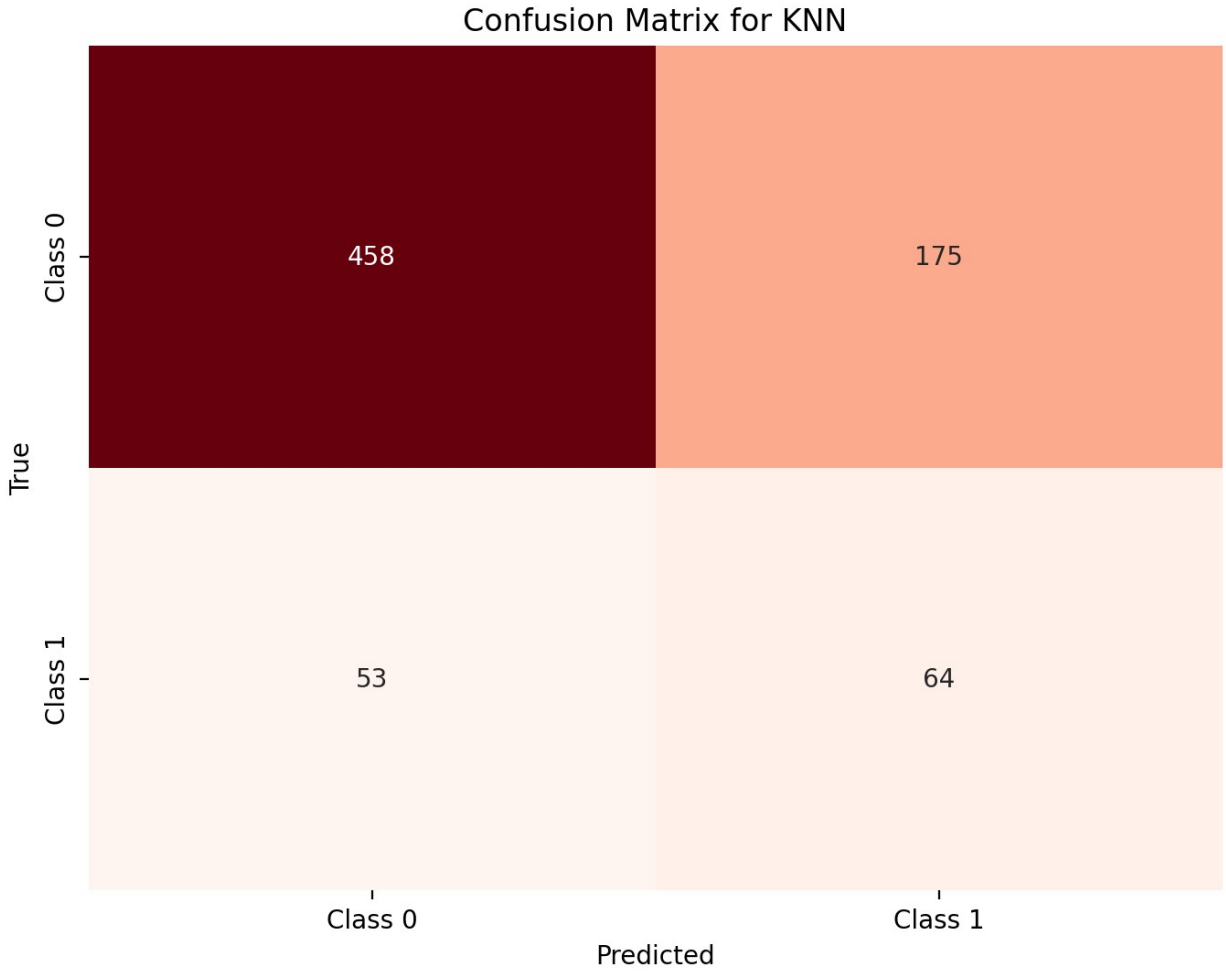
```
<Figure size 800x600 with 0 Axes>
```

```
<Axes: >
```

```
Text(0.5, 36.72222222222221, 'Predicted')
```

```
Text(70.72222222222221, 0.5, 'True')
```

```
Text(0.5, 1.0, 'Confusion Matrix for KNN')
```




```
# AU ROC CURVE KNN
'''the AUC ROC Curve is a measure of performance based on plotting the
true positive and false positive rate
and calculating the area under that curve.The closer the score to 1
the better the algorithm's ability to
distinguish between the two outcome classes.'''
```

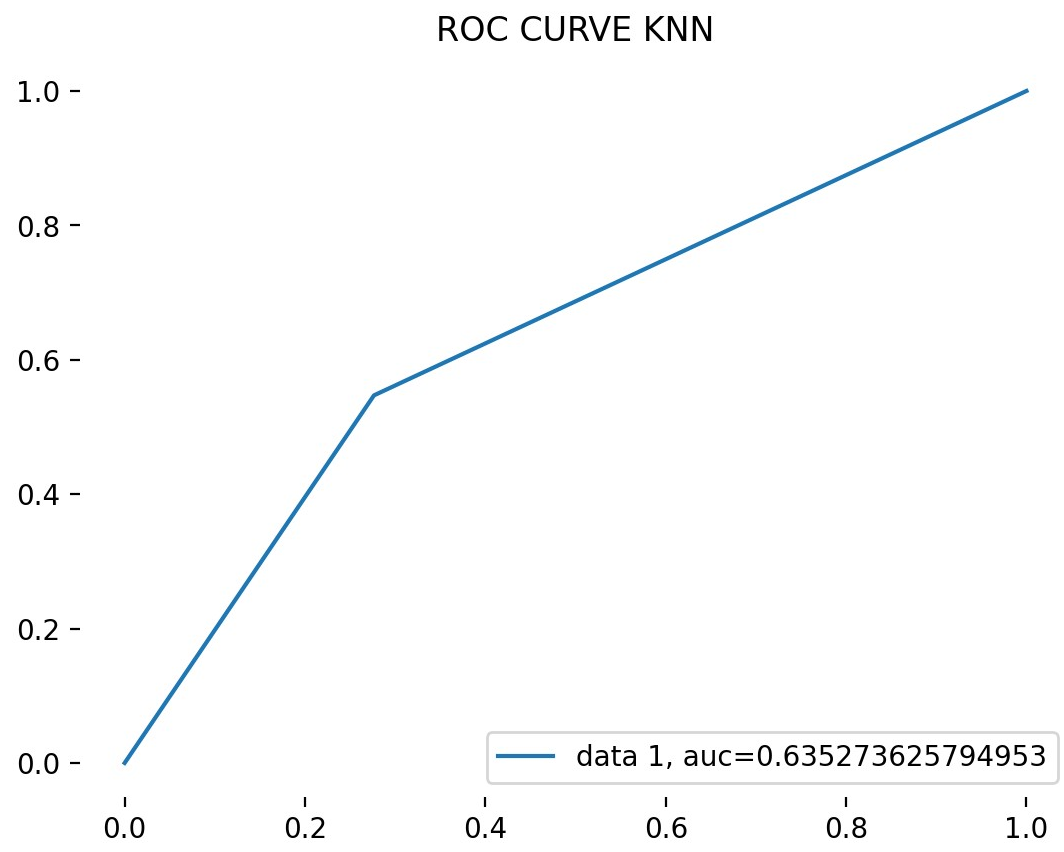
```
fpr, tpr, _ = roc_curve(y_test, normalized_df_knn_pred)
auc = roc_auc_score(y_test, normalized_df_knn_pred)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.box(False)
plt.title ('ROC CURVE KNN')
plt.show()

print(f"The score for the AUC ROC Curve is: {round(auc,3)*100}%")
```

```
"the AUC ROC Curve is a measure of performance based on plotting the
true positive and false positive rate \nand calculating the area under
that curve.The closer the score to 1 the better the algorithm's
ability to \ndistinguish between the two outcome classes."
```

```
[<matplotlib.lines.Line2D at 0x2394db81eb0>]
```

```
<matplotlib.legend.Legend at 0x2394b1b9100>
```

```
Text(0.5, 1.0, 'ROC CURVE KNN')
```




```
The score for the AUC ROC Curve is: 63.5%
```

```
def start_questionnaire():
    my_predictors = []
    parameters=['sysBP',
'glucose','age','totChol','cigsPerDay','diaBP','prevalentHyp','diabete
s','BPMeds','male']

    print('Input Patient Information:')

    age = input("Patient's age: >>> ")
    my_predictors.append(age)
    male = input("Patient's gender. male=1, female=0: >>> ")
    my_predictors.append(male)
    cigsPerDay = input("Patient's smoked cigarettes per day: >>> ")
    my_predictors.append(cigsPerDay)
    sysBP = input("Patient's systolic blood pressure: >>> ")
    my_predictors.append(sysBP)
    diaBP = input("Patient's diastolic blood pressure: >>> ")
    my_predictors.append(diaBP)
    totChol = input("Patient's cholesterin level: >>> ")
    my_predictors.append(totChol)
    prevalentHyp = input("Was Patient hypertensive? Yes=1, No=0 >>> ")

    my_predictors.append(prevalentHyp)
    diabetes = input("Did Patient have diabetes? Yes=1, No=0 >>> ")
    my_predictors.append(diabetes)
    glucose = input("What is the Patient's glucose level? >>> ")
    my_predictors.append(diabetes)
    BPMeds = input("Has Patient been on Blood Pressure Medication?
Yes=1, No=0 >>> ")
    my_predictors.append(BPMeds)

    my_data = dict(zip(parameters, my_predictors))
    my_df = pd.DataFrame(my_data, index=[0])
    scaler = MinMaxScaler(feature_range=(0,1))

    # assign scaler to column:
    my_df_scaled = pd.DataFrame(scaler.fit_transform(my_df),
columns=my_df.columns)
    my_y_pred = knn.predict(my_df)
    print('\n')
    print('Result:')
    if my_y_pred == 1:
        print("The patient will develop a Heart Disease.")
    if my_y_pred == 0:
        print("The patient will not develop a Heart Disease.")

start_questionnaire()
```

```
Input Patient Information:
```