# Name Kapil Uniyara ML Lab Work

Objective- The goal is to make some predictive models on a FHS dataset, and reviewing some exploratory and modelling techniques.

In [30]:
```
#Reading the file
import pandas as pd
data = pd.read_csv("Kapil Uniyara - framingham - Kapil Uniyara - framingham.csv")
```

In [31]:
```
#Lets to undersand the data perform Some initial exploration
data.head()
```

Out[31]:

| | male | age | education | currentSmoker | cigsPerDay | BPMeds | prevalentStroke | prevalentHyp | dia |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 39 | 4.0 | 0 | 0.0 | 0.0 | 0 | 0 | |
| **1** | 0 | 46 | 2.0 | 0 | 0.0 | 0.0 | 0 | 0 | |
| **2** | 1 | 48 | 1.0 | 1 | 20.0 | 0.0 | 0 | 0 | |
| **3** | 0 | 61 | 3.0 | 1 | 30.0 | 0.0 | 0 | 1 | |
| **4** | 0 | 46 | 3.0 | 1 | 23.0 | 0.0 | 0 | 0 | |

In [32]:
```
data.describe()
```

Out[32]:

| | male | age | education | currentSmoker | cigsPerDay | BPMeds | prevalent |
|---|---|---|---|---|---|---|---|
| **count** | 4240.000000 | 4240.000000 | 4135.000000 | 4240.000000 | 4211.000000 | 4187.000000 | 4240.0 |
| **mean** | 0.429245 | 49.580189 | 1.979444 | 0.494104 | 9.005937 | 0.029615 | 0.0 |
| **std** | 0.495027 | 8.572942 | 1.019791 | 0.500024 | 11.922462 | 0.169544 | 0.0 |
| **min** | 0.000000 | 32.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **25%** | 0.000000 | 42.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **50%** | 0.000000 | 49.000000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **75%** | 1.000000 | 56.000000 | 3.000000 | 1.000000 | 20.000000 | 0.000000 | 0.0 |
| **max** | 1.000000 | 70.000000 | 4.000000 | 1.000000 | 70.000000 | 1.000000 | 1.0 |

In [33]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4240 entries, 0 to 4239
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   male             4240 non-null   int64  
 1   age              4240 non-null   int64  
 2   education        4135 non-null   float64
 3   currentSmoker    4240 non-null   int64  
 4   cigsPerDay       4211 non-null   float64
 5   BPMeds           4187 non-null   float64
 6   prevalentStroke  4240 non-null   int64  
 7   prevalentHyp     4240 non-null   int64  
 8   diabetes         4240 non-null   int64  
 9   totChol          4190 non-null   float64
 10  sysBP            4240 non-null   float64
 11  diaBP            4240 non-null   float64
 12  BMI              4221 non-null   float64
 13  heartRate        4239 non-null   float64
 14  glucose          3852 non-null   float64
 15  TenYearCHD       4240 non-null   int64  
dtypes: float64(9), int64(7)
memory usage: 530.1 KB
```

In [34]:

```
data.isnull().sum()
```

Out[34]:

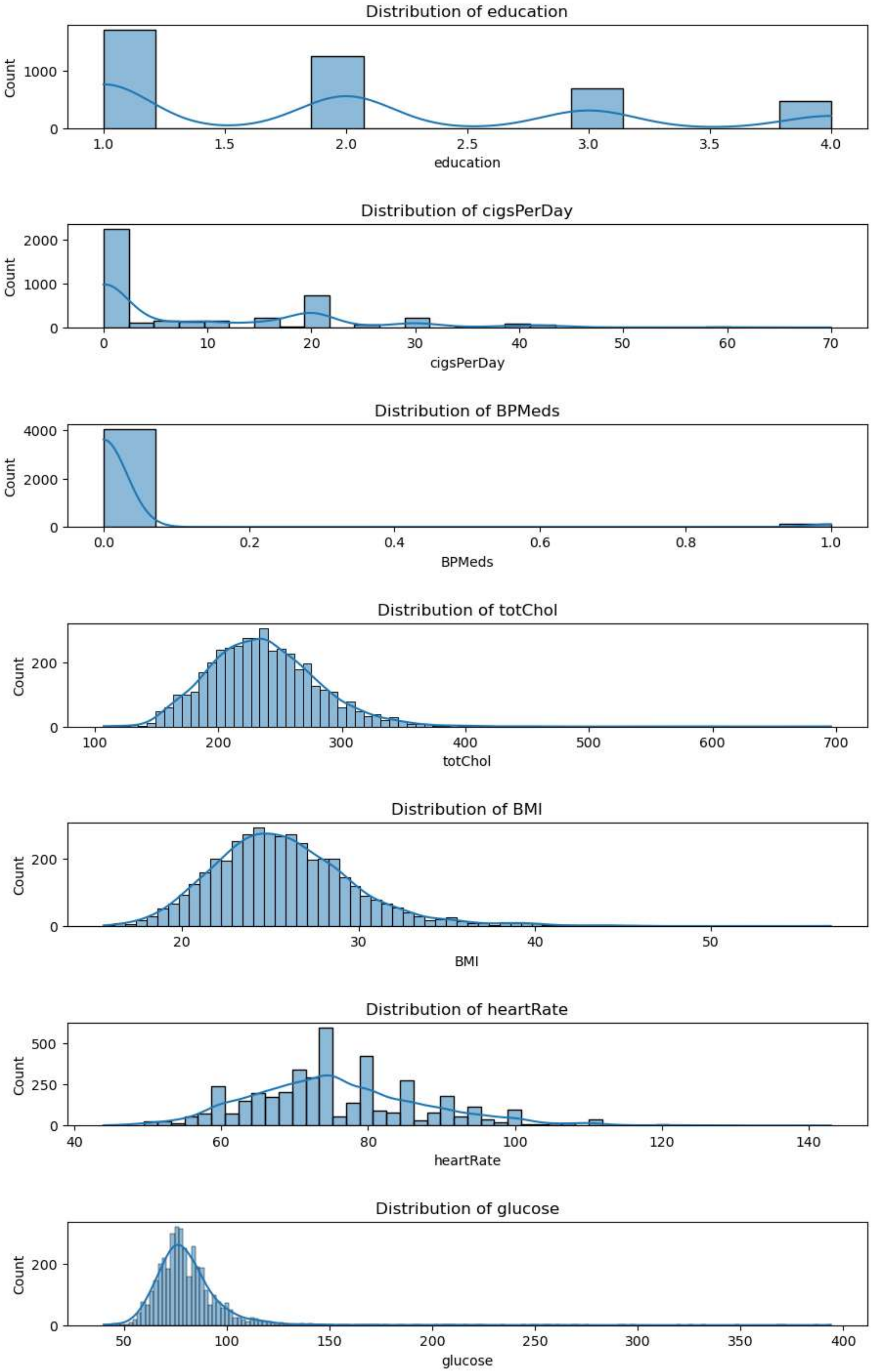
```
male                 0
age                  0
education          105
currentSmoker        0
cigsPerDay          29
BPMeds              53
prevalentStroke      0
prevalentHyp         0
diabetes             0
totChol             50
sysBP                0
diaBP                0
BMI                 19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

# Now we have some missing values, So lets make some visualization of it

In [35]:

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

cols_with_missing_values = ['education', 'cigsPerDay', 'BPMeds', 'totChol', 'BMI',

# Plotting the distributions of columns with missing values
fig, axes = plt.subplots(len(cols_with_missing_values), 1, figsize=(10, 15))
fig.tight_layout(pad=5.0)

for i, col in enumerate(cols_with_missing_values):
    sns.histplot(data[col], kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Count')
```

```
plt.show()
```




# Now lets fill missng values

In [36]:
```
from sklearn.impute import SimpleImputer
# For normally distributed or symmetric continuous variables, we'll use the mean
# For skewed continuous variables and categorical variables, we'll use the median

# Imputer for mean
mean_imputer = SimpleImputer(strategy='mean')

# Columns to impute with mean (normally distributed/symmetric)
mean_cols = ['heartRate']

# Impute the columns with mean
data[mean_cols] = mean_imputer.fit_transform(data[mean_cols])
```

In [37]:
```
# Imputer for median
median_imputer = SimpleImputer(strategy='median')

# Columns to impute with median (skewed distributions or categorical)
median_cols = ['education', 'cigsPerDay', 'BPMeds', 'totChol', 'BMI', 'glucose']

# Impute the columns with median
data[median_cols] = median_imputer.fit_transform(data[median_cols])
```

In [38]:
```
# Checking if there are any missing values left
data.isnull().sum()
```

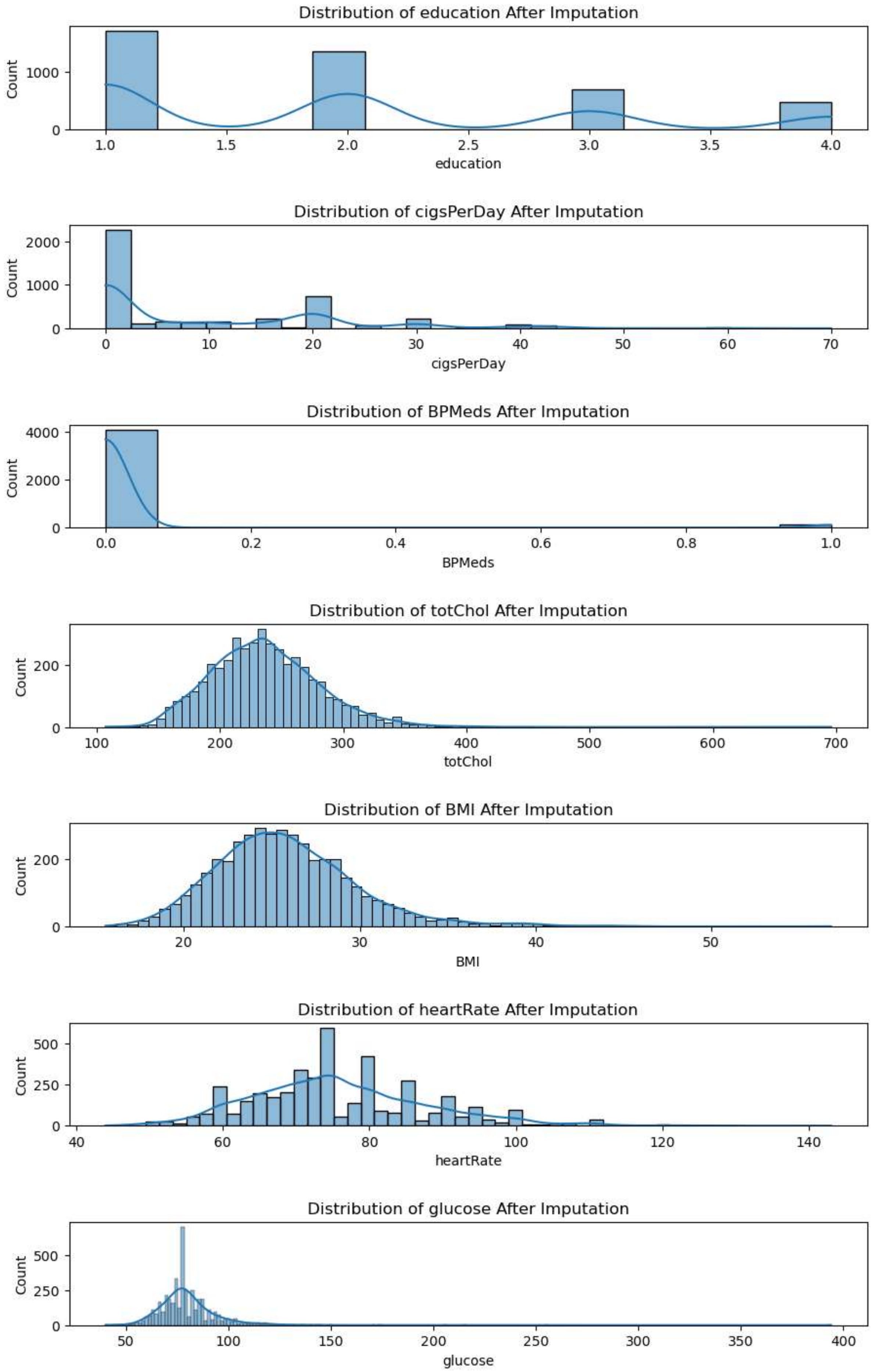
Out[38]:
```
male               0
age                0
education          0
currentSmoker      0
cigsPerDay         0
BPMeds             0
prevalentStroke    0
prevalentHyp       0
diabetes           0
totChol            0
sysBP              0
diaBP              0
BMI                0
heartRate          0
glucose            0
TenYearCHD         0
dtype: int64
```

In [39]:
```
data.describe()
```

Out[39]:

| | male | age | education | currentSmoker | cigsPerDay | BPMeds | prevalent |
|---|---|---|---|---|---|---|---|
| **count** | 4240.000000 | 4240.000000 | 4240.000000 | 4240.000000 | 4240.000000 | 4240.000000 | 4240.( |
| **mean** | 0.429245 | 49.580189 | 1.979953 | 0.494104 | 8.944340 | 0.029245 | 0.( |
| **std** | 0.495027 | 8.572942 | 1.007087 | 0.500024 | 11.904777 | 0.168513 | 0.( |
| **min** | 0.000000 | 32.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.( |
| **25%** | 0.000000 | 42.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.( |
| **50%** | 0.000000 | 49.000000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 | 0.( |
| **75%** | 1.000000 | 56.000000 | 3.000000 | 1.000000 | 20.000000 | 0.000000 | 0.( |
| **max** | 1.000000 | 70.000000 | 4.000000 | 1.000000 | 70.000000 | 1.000000 | 1.( |

In [40]:
```
# Plotting the distributions after imputation
fig, axes = plt.subplots(len(cols_with_missing_values), 1, figsize=(10, 15))
fig.tight_layout(pad=5.0)

for i, col in enumerate(cols_with_missing_values):
    sns.histplot(data[col], kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {col} After Imputation')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Count')

plt.show()
```

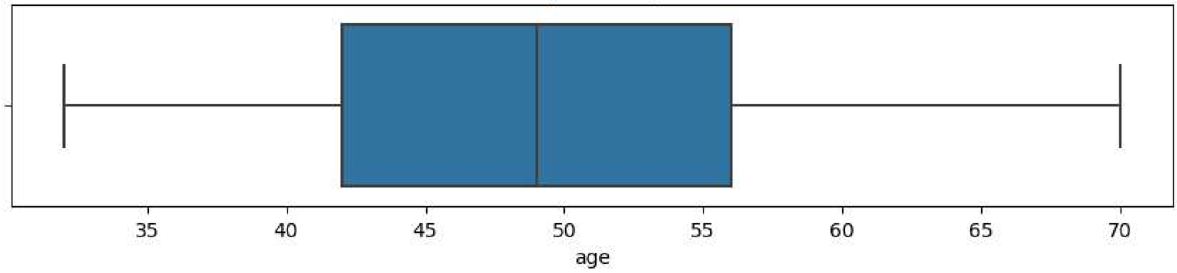



# Now lets see the outliers in the data

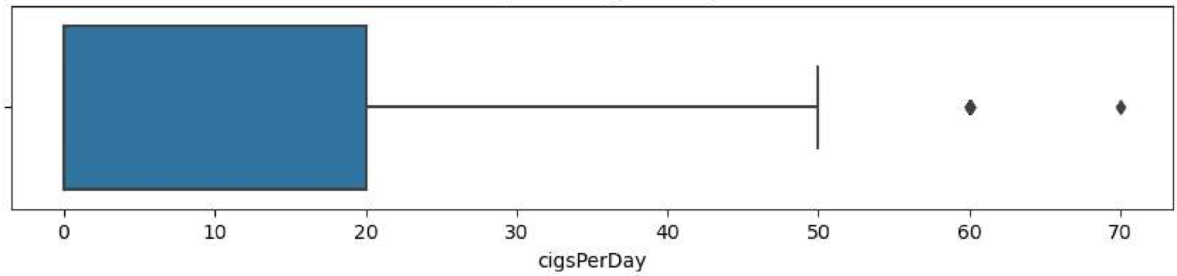
In [41]:
```
# Selecting continuous variables for outlier detection
continuous_cols = ['age', 'cigsPerDay', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartR
```

```
# Plotting the boxplots for continuous variables
fig, axes = plt.subplots(len(continuous_cols), 1, figsize=(10, 20))
fig.tight_layout(pad=5.0)

for i, col in enumerate(continuous_cols):
    sns.boxplot(x=data[col], ax=axes[i])
    axes[i].set_title(f'Boxplot of {col}')
    axes[i].set_xlabel(col)

plt.show()
```

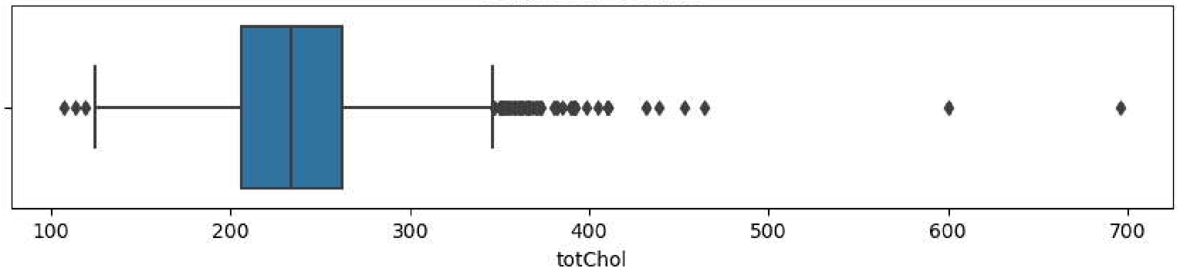
Boxplot of age



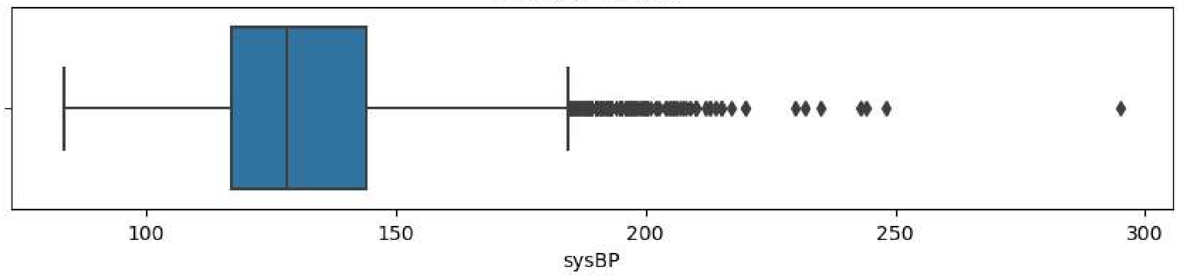

Boxplot of cigsPerDay



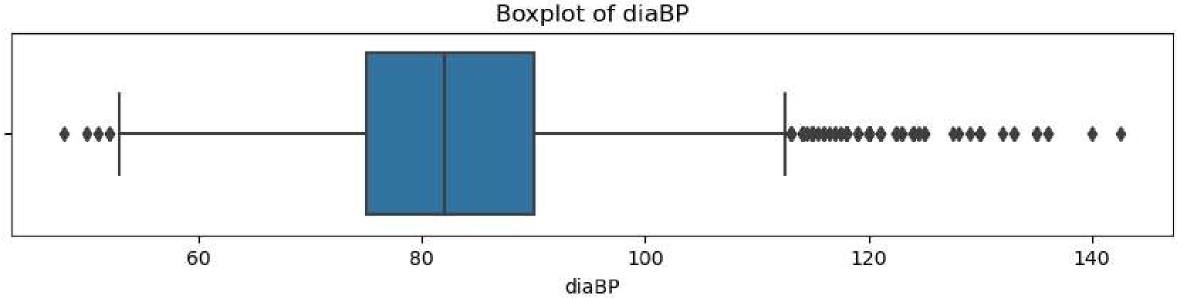

Boxplot of totChol



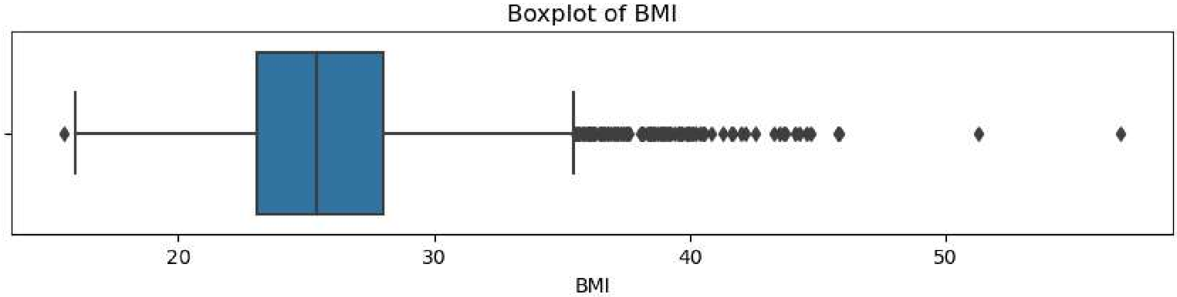

Boxplot of sysBP



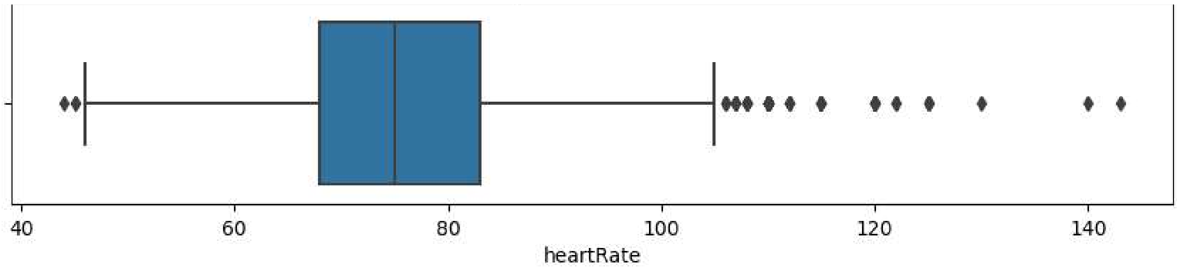

Boxplot of diaBP

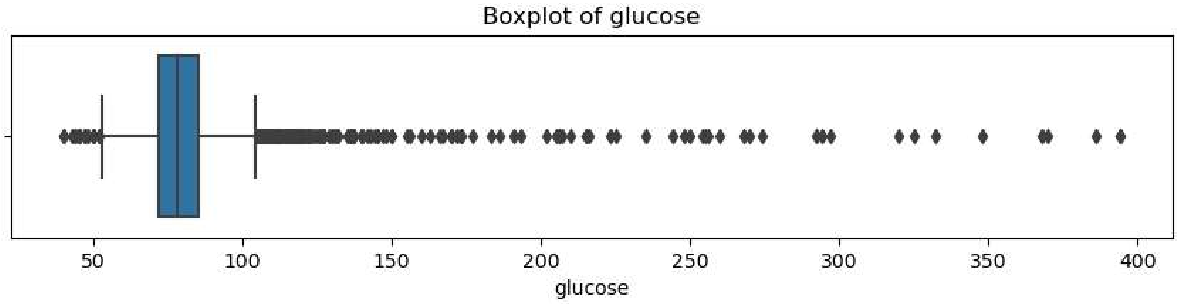



Boxplot of BMI




Boxplot of heartRate

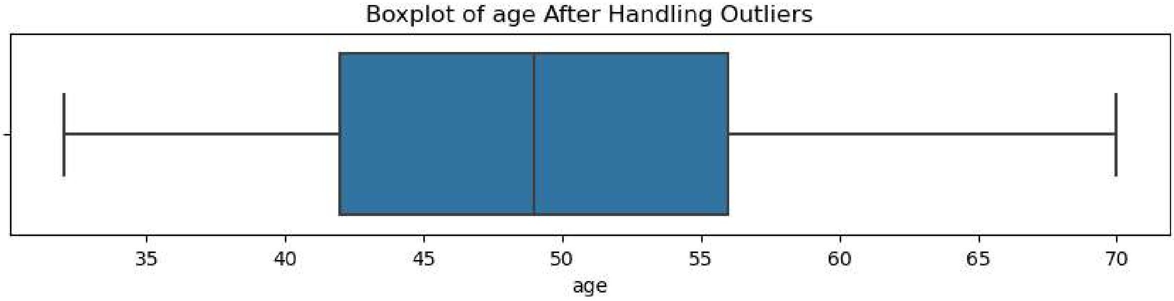


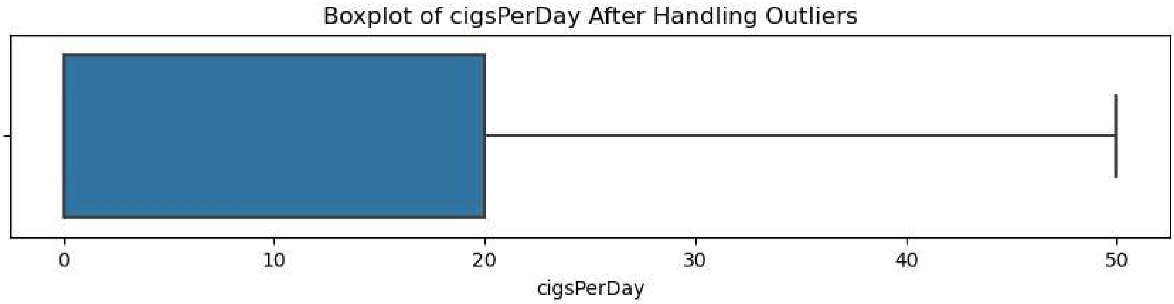



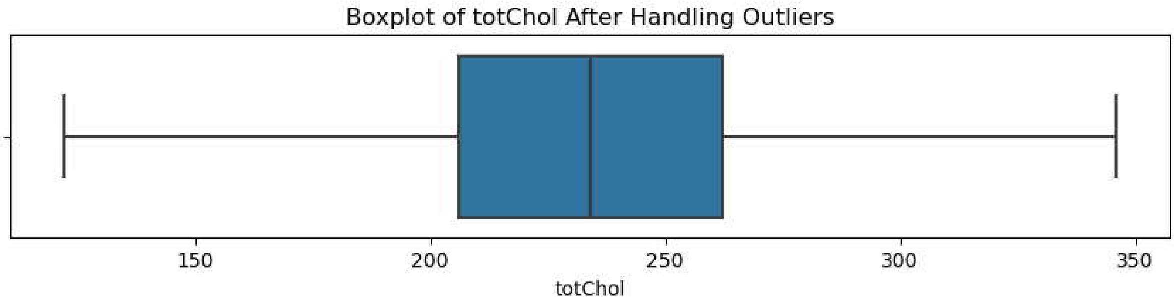

# We will be using interquntile range (IQR) method to handle outliers

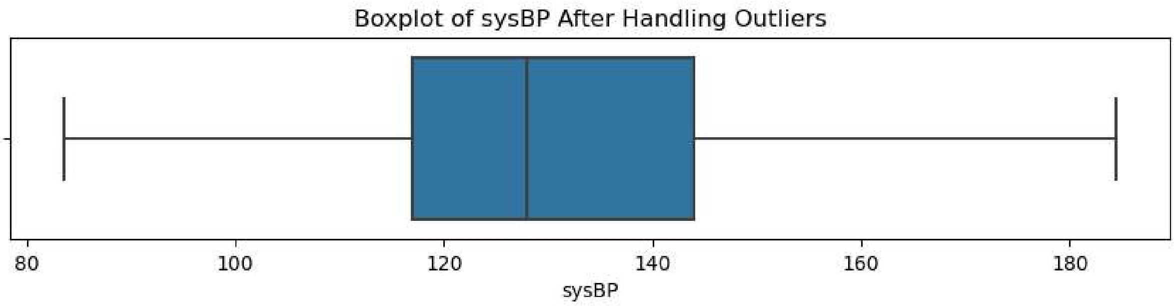
In [42]:
```
def handle_outliers_with_IQR(df, column):
    """ Handle outliers in a dataframe column using the IQR method.
        Values outside 1.5 * IQR from the Q1 and Q3 quartiles are considered outlie
    """
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)
```

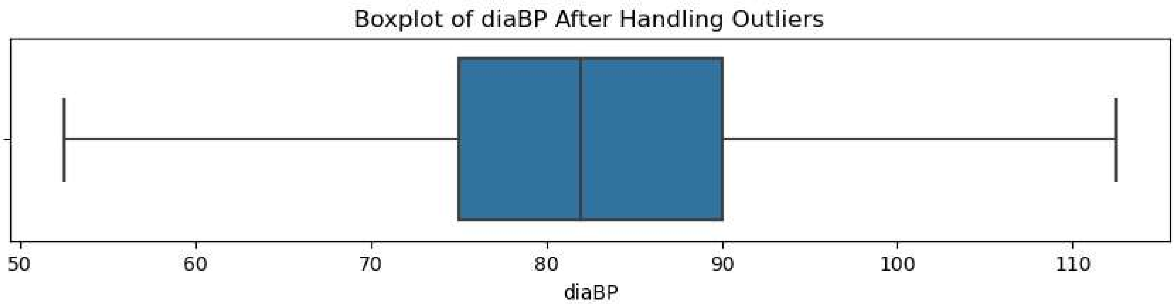
In [43]:
```
# Applying the IQR method to each continuous column
for col in continuous_cols:
    handle_outliers_with_IQR(data, col)

# Plotting the boxplots for continuous variables after handling outliers
fig, axes = plt.subplots(len(continuous_cols), 1, figsize=(10, 20))
fig.tight_layout(pad=5.0)

for i, col in enumerate(continuous_cols):
    sns.boxplot(x=data[col], ax=axes[i])
    axes[i].set_title(f'Boxplot of {col} After Handling Outliers')
    axes[i].set_xlabel(col)

plt.show()
```

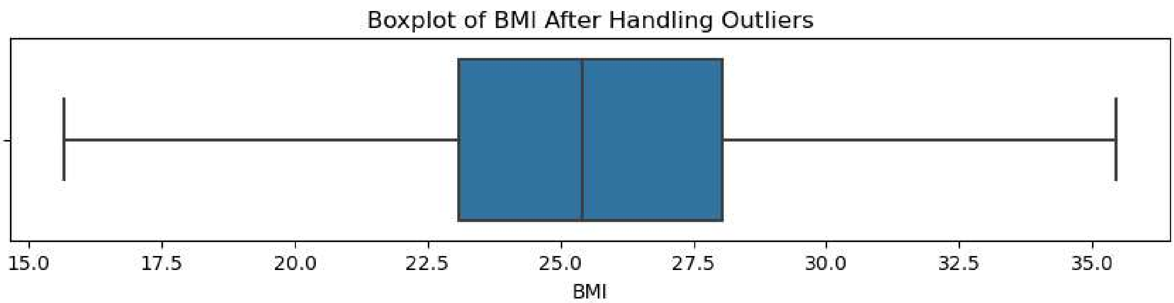

Boxplot of age After Handling Outliers
35
40
45
50
55
60
65
70
age


Boxplot of cigsPerDay After Handling Outliers
0
10
20
30
40
50
cigsPerDay

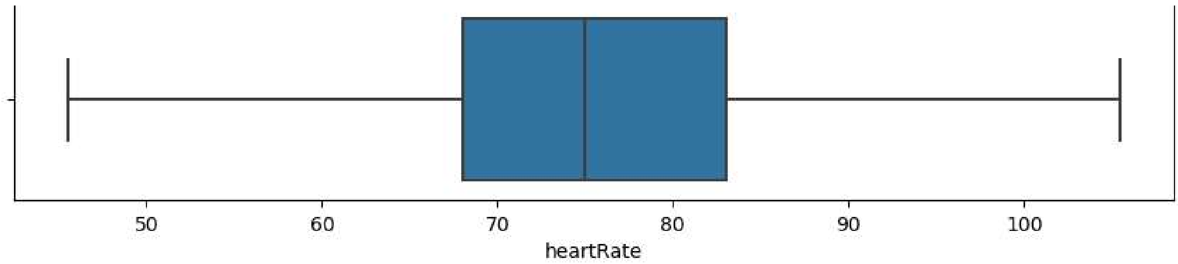

Boxplot of totChol After Handling Outliers
150
200
250
300
350
totChol

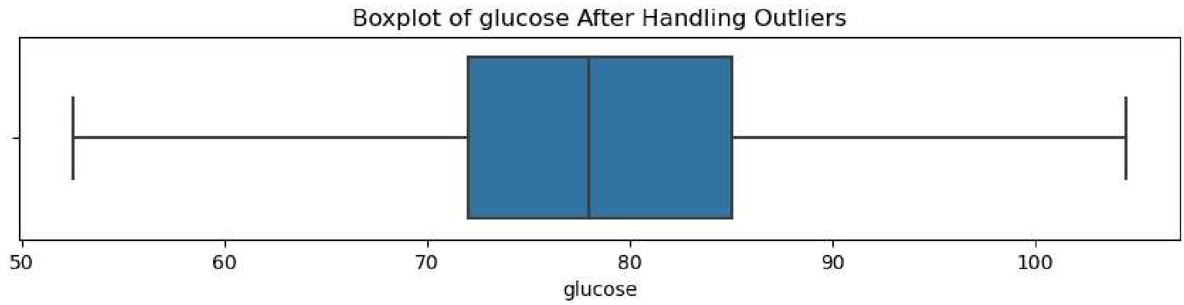

Boxplot of sysBP After Handling Outliers
80
100
120
140
160
180
sysBP


Boxplot of diaBP After Handling Outliers
50
60
70
80
90
100
110
diaBP


Boxplot of BMI After Handling Outliers
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
BMI


Boxplot of heartRate After Handling Outliers

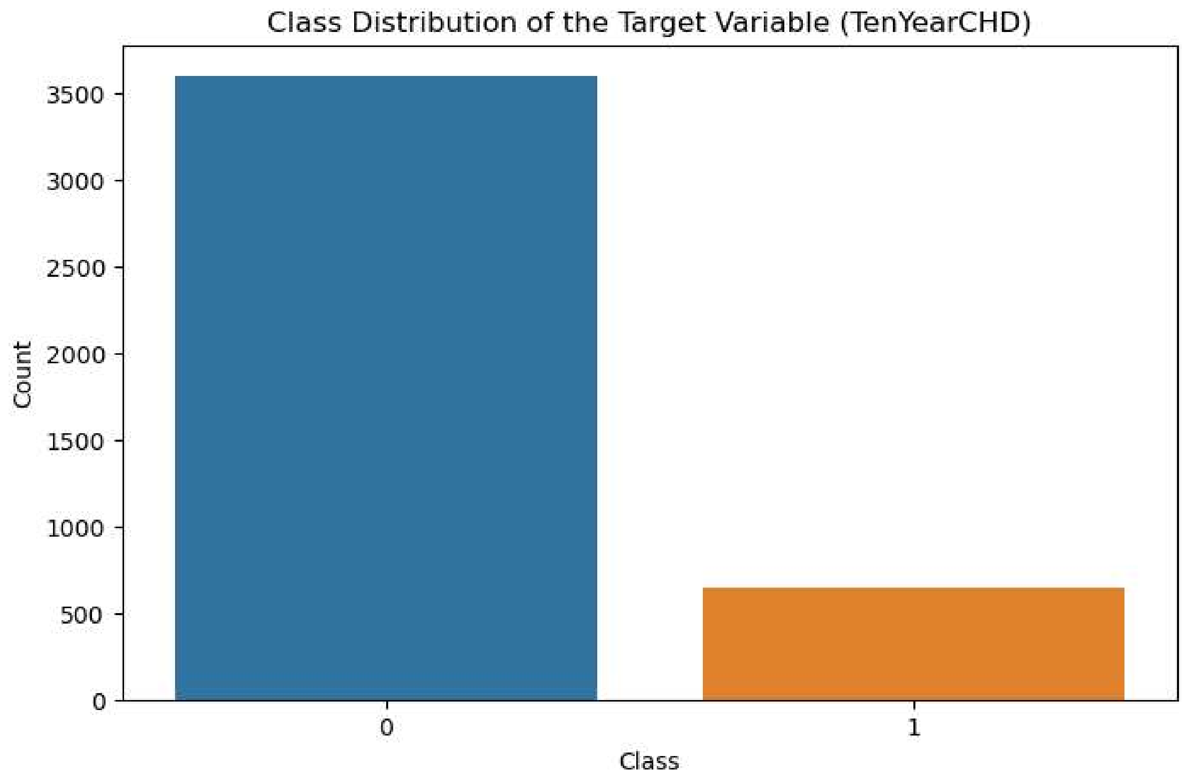



# Now lets check the data is imbalanced or not

In [44]:

```
# Checking if the data is imbalanced
target_column = 'TenYearCHD'
class_counts = data[target_column].value_counts()

# Plotting the class distribution
plt.figure(figsize=(8, 5))
sns.barplot(x=class_counts.index, y=class_counts.values)
plt.title('Class Distribution of the Target Variable (TenYearCHD)')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



In [45]:
```
class_counts
```

Out[45]:
```
TenYearCHD
0    3596
1     644
Name: count, dtype: int64
```

# We will be using oversampling to balance the dataset

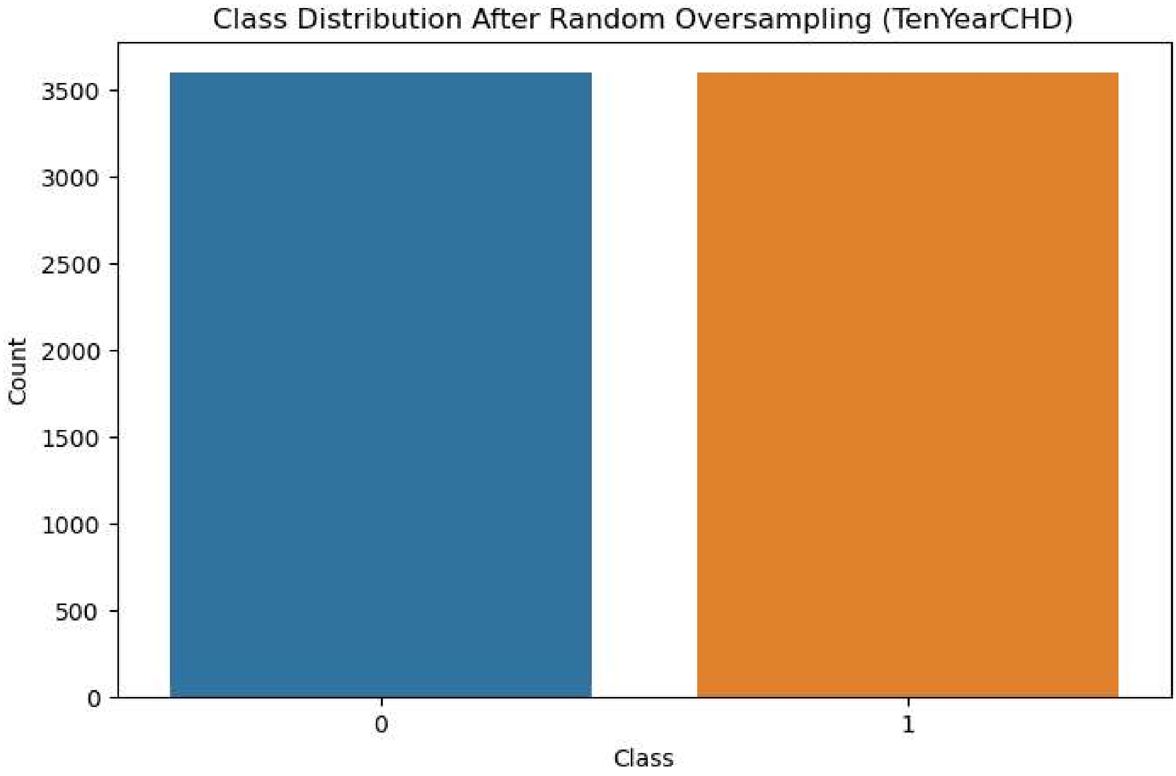
In [46]:
```
from sklearn.utils import resample

# Separate the minority and majority classes
data_majority = data[data[target_column] == 0]
data_minority = data[data[target_column] == 1]
```

In [47]:
```
# Upsample minority class
data_minority_upsampled = resample(data_minority,
                                   replace=True,     # sample with replacement
                                   n_samples=len(data_majority),    # to match majc
                                   random_state=123) # reproducible results

# Combine majority class with upsampled minority class
data_upsampled = pd.concat([data_majority, data_minority_upsampled])

# Display new class counts
upsampled_class_counts = data_upsampled[target_column].value_counts()
```

In [48]:
```
upsampled_class_counts
```

Out[48]:
```
TenYearCHD
0    3596
1    3596
Name: count, dtype: int64
```

In [49]:
```
# Plotting the class distribution after resampling
plt.figure(figsize=(8, 5))
sns.barplot(x=upsampled_class_counts.index, y=upsampled_class_counts.values)
plt.title('Class Distribution After Random Oversampling (TenYearCHD)')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```




# Now lets make predictions

In [50]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.utils import resample

# Splitting the dataset into features and target variable
X = data_upsampled.drop('TenYearCHD', axis=1)
y = data_upsampled['TenYearCHD']

# Splitting the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [51]:
```
# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the model
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)
```

In [52]:
```
accuracy_score(y_test, y_pred)
```

Out[52]:
```
0.9694232105628909
```

In [55]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       735
           1       0.95      0.99      0.97       704

    accuracy                           0.97      1439
   macro avg       0.97      0.97      0.97      1439
weighted avg       0.97      0.97      0.97      1439
```

# Saving the pickle file to make webui for this problem

In [56]:

```
import pickle

# Save the model to a pickle file
pickle_filename = 'random_forest_model.pkl'
with open(pickle_filename, 'wb') as file:
    pickle.dump(rf_classifier, file)

print(f"Model saved as {pickle_filename}")
```

```
Model saved as random_forest_model.pkl
```

In [ ]: