In [24]:

```
import pandas as pd
```

In [25]:

```
#check for importing that data from the csv file

df=pd.read_csv('./framingham.csv')
df.shape
```

Out[25]:

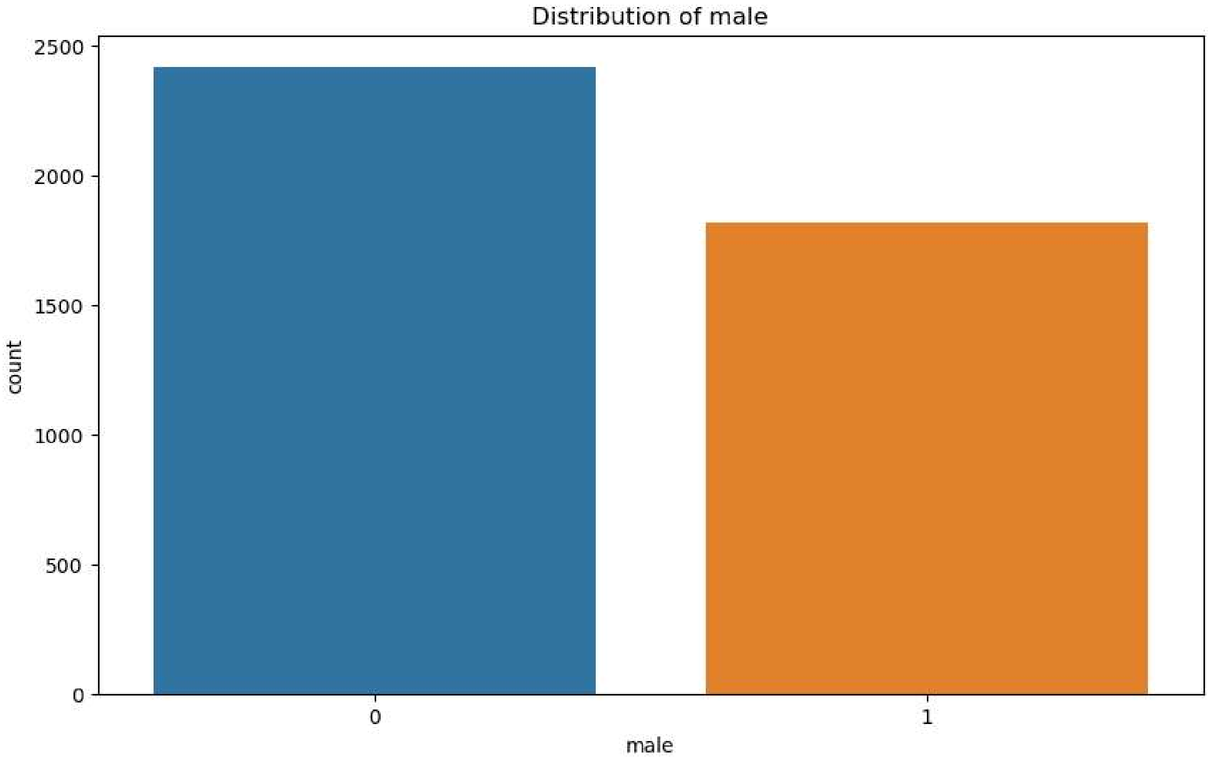
```
(4240, 16)
```

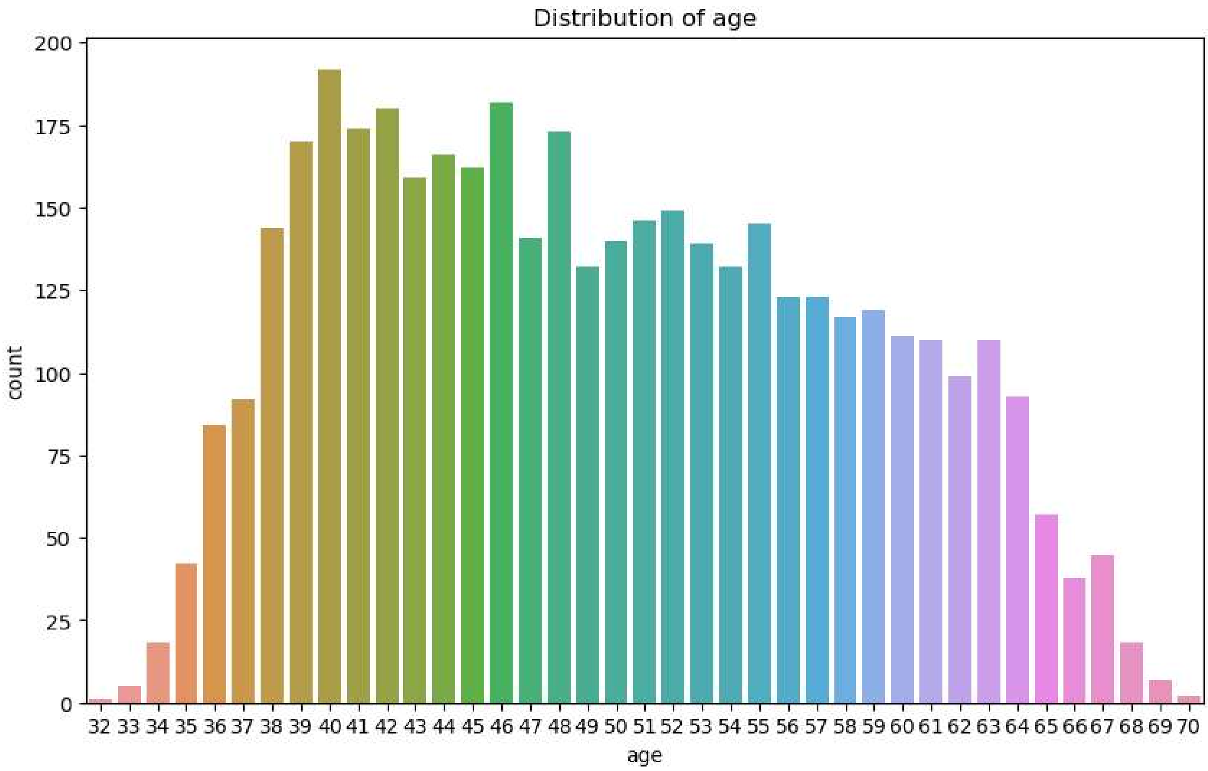
In [26]:

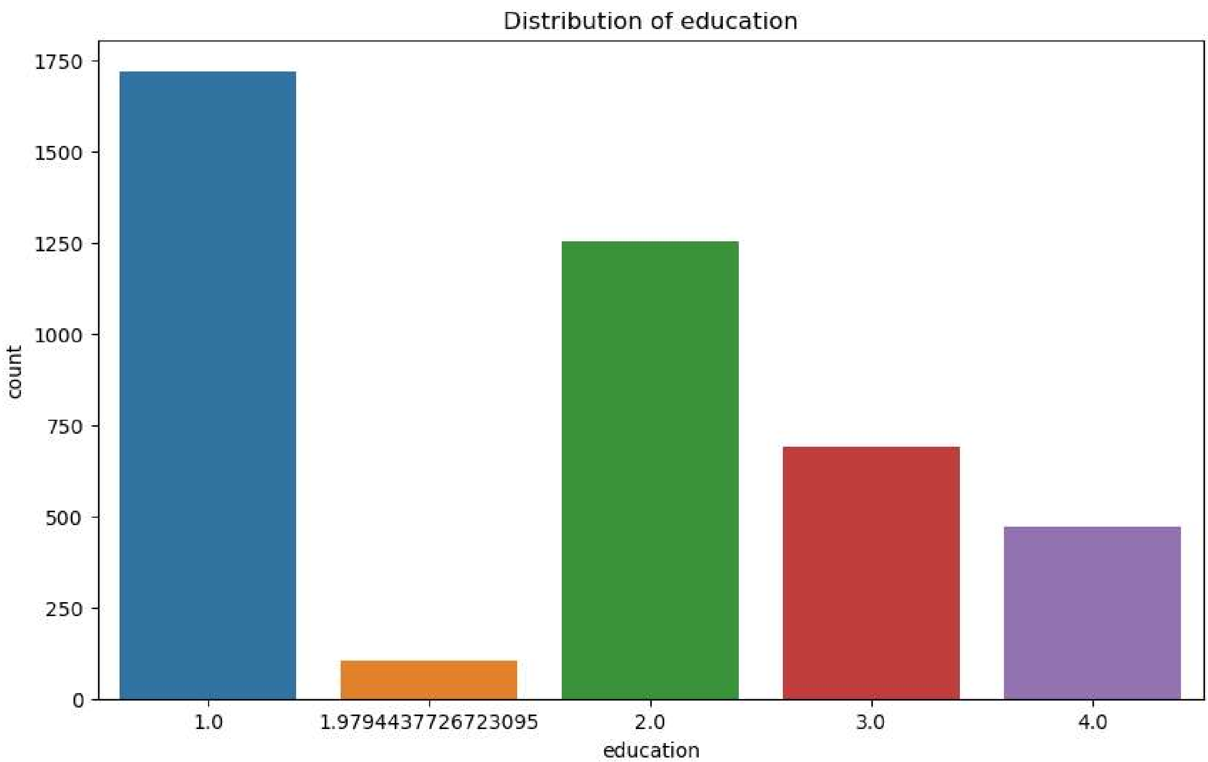
```
#importing data and visualizing it.

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('./framingham.csv')

# Display the first few rows of the dataset to understand its structure
print(data.head())

# Check for missing values
missing_values = data.isnull().sum()

# Print columns with missing values
columns_with_missing_values = missing_values[missing_values > 0].index
print("Columns with missing values:")
print(data.isnull().sum())

for column in columns_with_missing_values:
    if data[column].dtype == 'float64':
        data[column].fillna(data[column].mean(), inplace=True)

print("\nMissing values after handling:")
print(data.isnull().sum())

# Basic statistics of numeric columns
print(data.describe())

# visualising the distribution of each parameter.
# Loop through all columns (except the target 'TenYearCHD')
for column in data.columns:
    if column != 'TenYearCHD':
        plt.figure(figsize=(10, 6))
        sns.countplot(x=column, data=data)
        plt.title(f'Distribution of {column}')
        plt.show()

# Visualize the correlation matrix of numeric features
plt.figure(figsize=(12, 8))
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

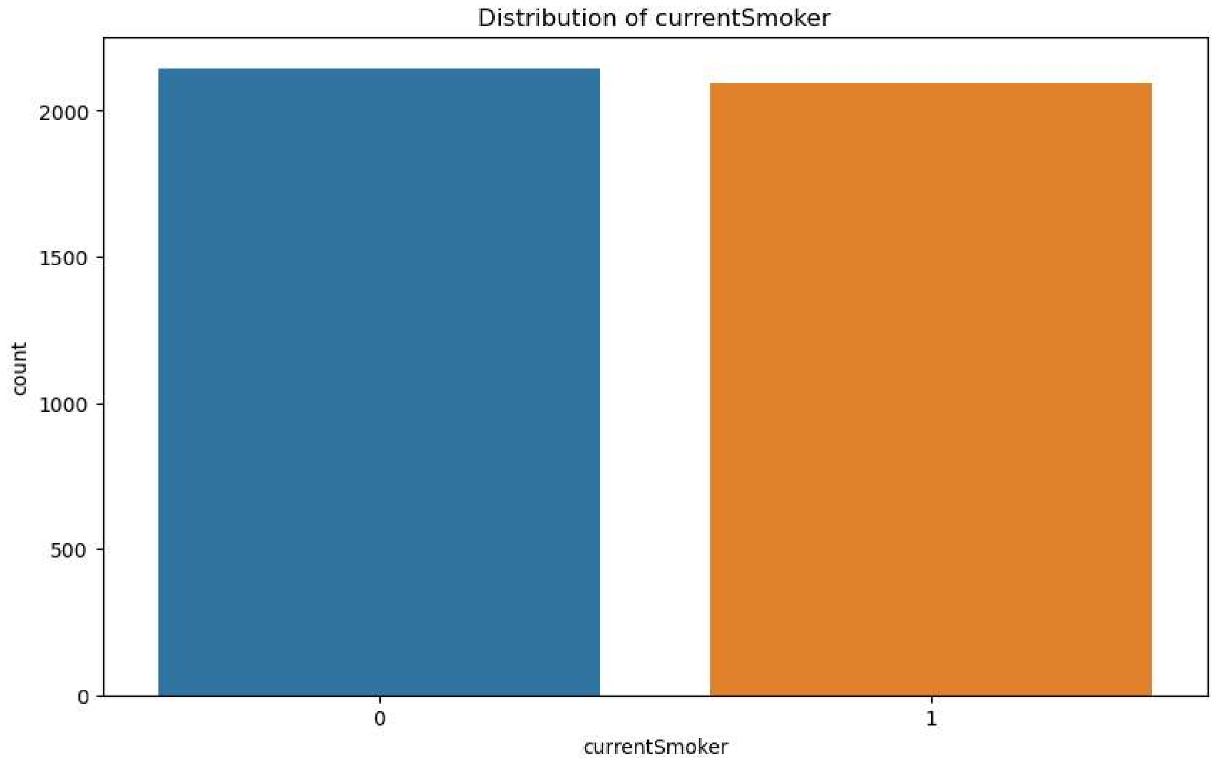
```
   male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
0     1   39        4.0              0         0.0     0.0                0
1     0   46        2.0              0         0.0     0.0                0
2     1   48        1.0              1        20.0     0.0                0
3     0   61        3.0              1        30.0     0.0                0
4     0   46        3.0              1        23.0     0.0                0

   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

   TenYearCHD
0           0
1           0
2           0
3           1
4           0
Columns with missing values:
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

Missing values after handling:
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
              male          age    education  currentSmoker   cigsPerDay  \
count  4240.000000  4240.000000  4240.000000    4240.000000  4240.000000
mean      0.429245    49.580189     1.979444       0.494104     9.005937
std       0.495027     8.572942     1.007082       0.500024    11.881610
min       0.000000    32.000000     1.000000       0.000000     0.000000
25%       0.000000    42.000000     1.000000       0.000000     0.000000
50%       0.000000    49.000000     2.000000       0.000000     0.000000
75%       1.000000    56.000000     3.000000       1.000000    20.000000
max       1.000000    70.000000     4.000000       1.000000    70.000000

            BPMeds  prevalentStroke  prevalentHyp     diabetes      totChol  \
count  4240.000000      4240.000000   4240.000000  4240.000000  4240.000000
mean      0.029615         0.005896      0.310613     0.025708   236.699523
std       0.168481         0.076569      0.462799     0.158280    44.327521
```

```
min       0.000000         0.000000      0.000000      0.000000   107.000000
25%       0.000000         0.000000      0.000000      0.000000   206.000000
50%       0.000000         0.000000      0.000000      0.000000   234.000000
75%       0.000000         0.000000      1.000000      0.000000   262.000000
max       1.000000         1.000000      1.000000      1.000000   696.000000

            sysBP        diaBP          BMI    heartRate      glucose  \
count  4240.000000  4240.000000  4240.000000  4240.000000  4240.000000
mean    132.354599    82.897759    25.800801    75.878981    81.963655
std      22.033300    11.910394     4.070687    12.023929    22.831748
min      83.500000    48.000000    15.540000    44.000000    40.000000
25%     117.000000    75.000000    23.077500    68.000000    72.000000
50%     128.000000    82.000000    25.410000    75.000000    80.000000
75%     144.000000    90.000000    28.032500    83.000000    85.000000
max     295.000000   142.500000    56.800000   143.000000   394.000000

         TenYearCHD
count  4240.000000
mean      0.151887
std       0.358953
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000
```

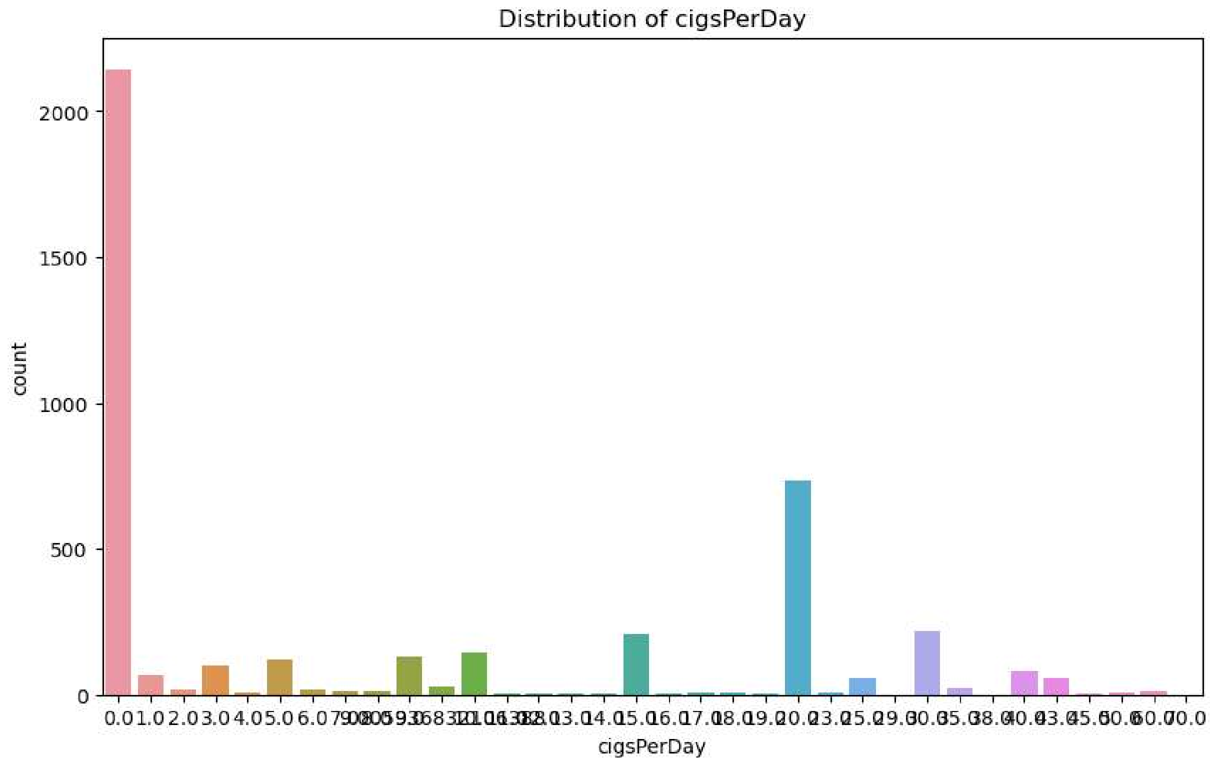



Distribution of age
count
age
0
25
50
75
100
125
150
175
200
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

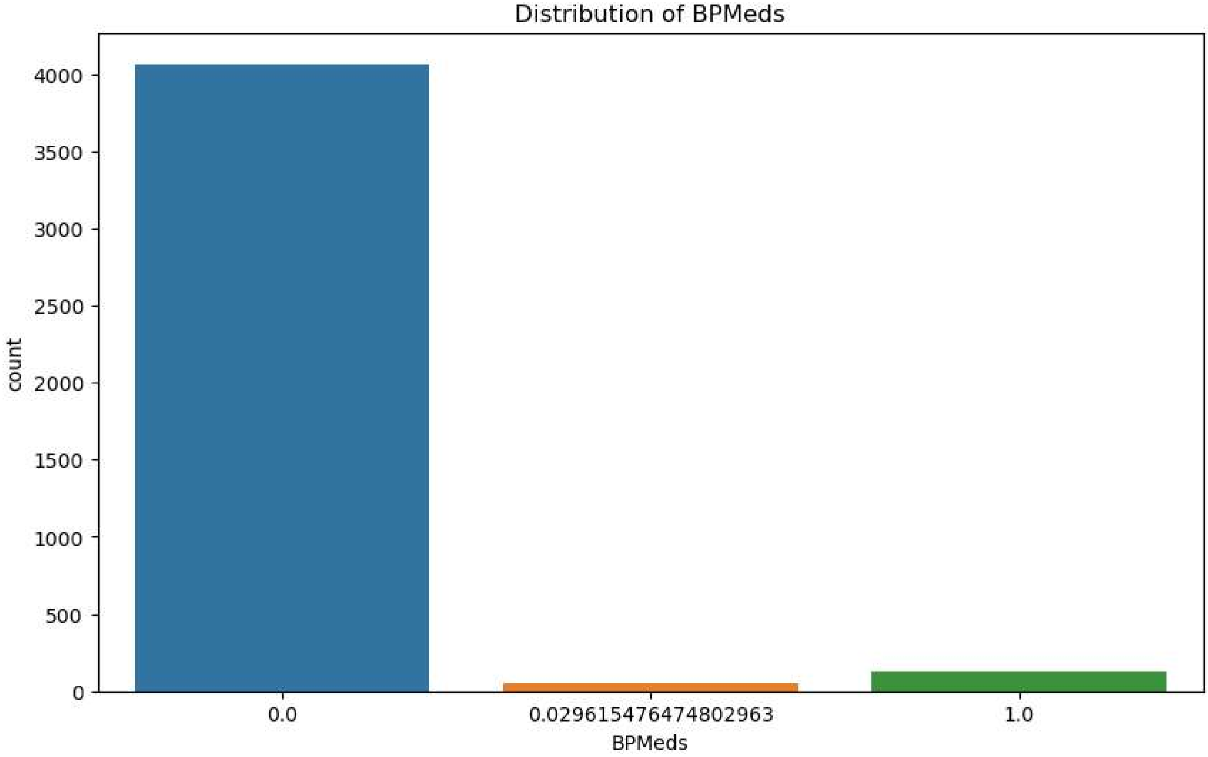

Distribution of education
count
education
0
250
500
750
1000
1250
1500
1750
1.0
1.9794437726723095
2.0
3.0
4.0

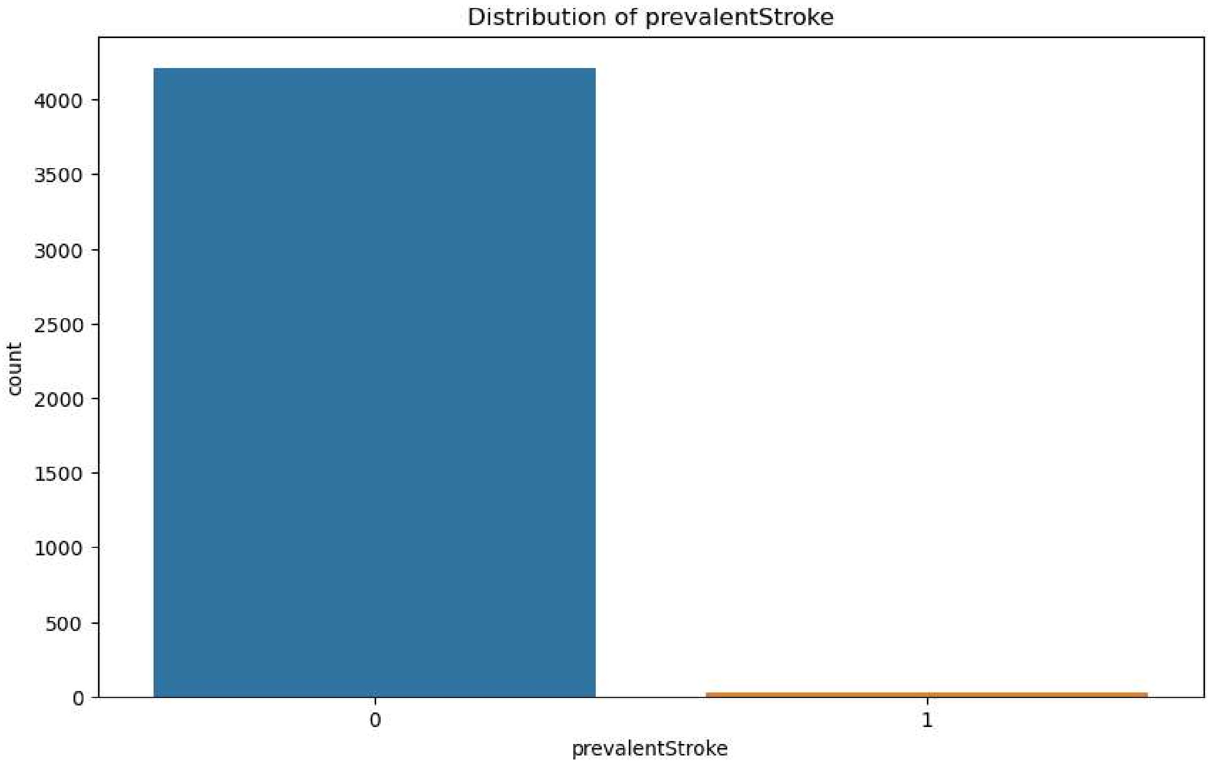


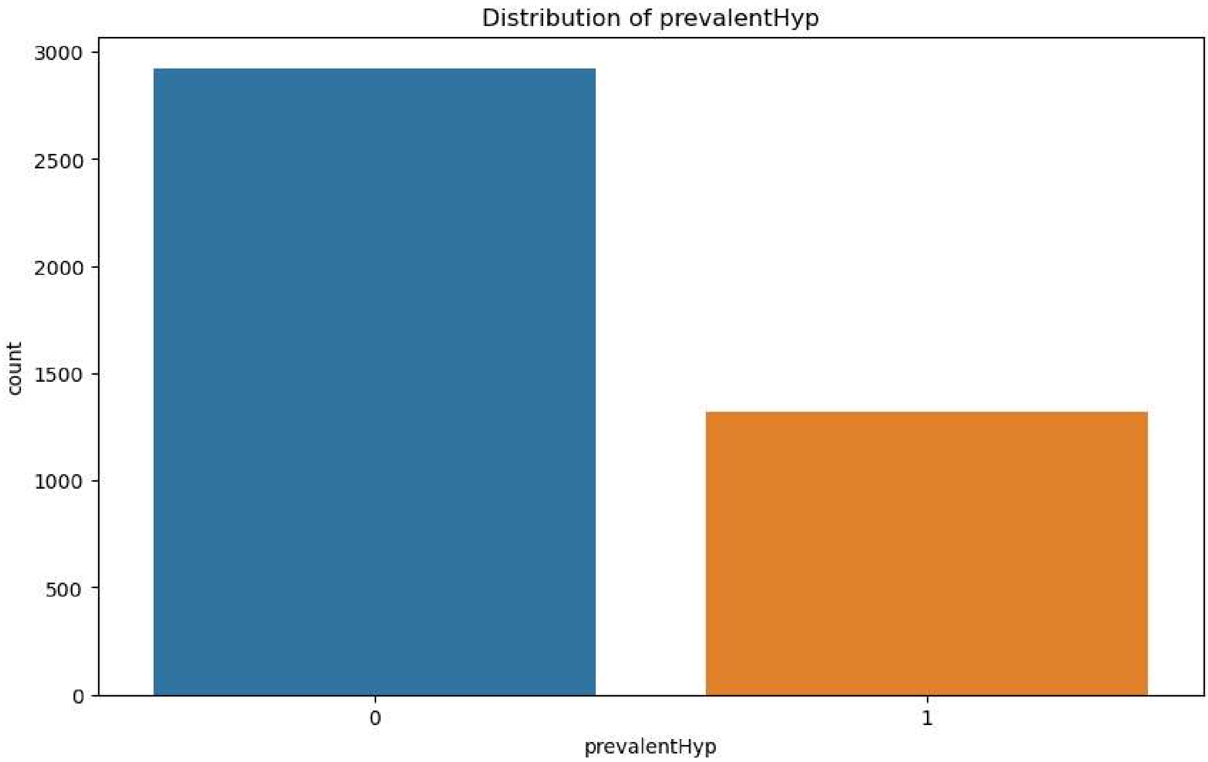
Distribution of currentSmoker
count
0
500
1000
1500
2000
0
1
currentSmoker

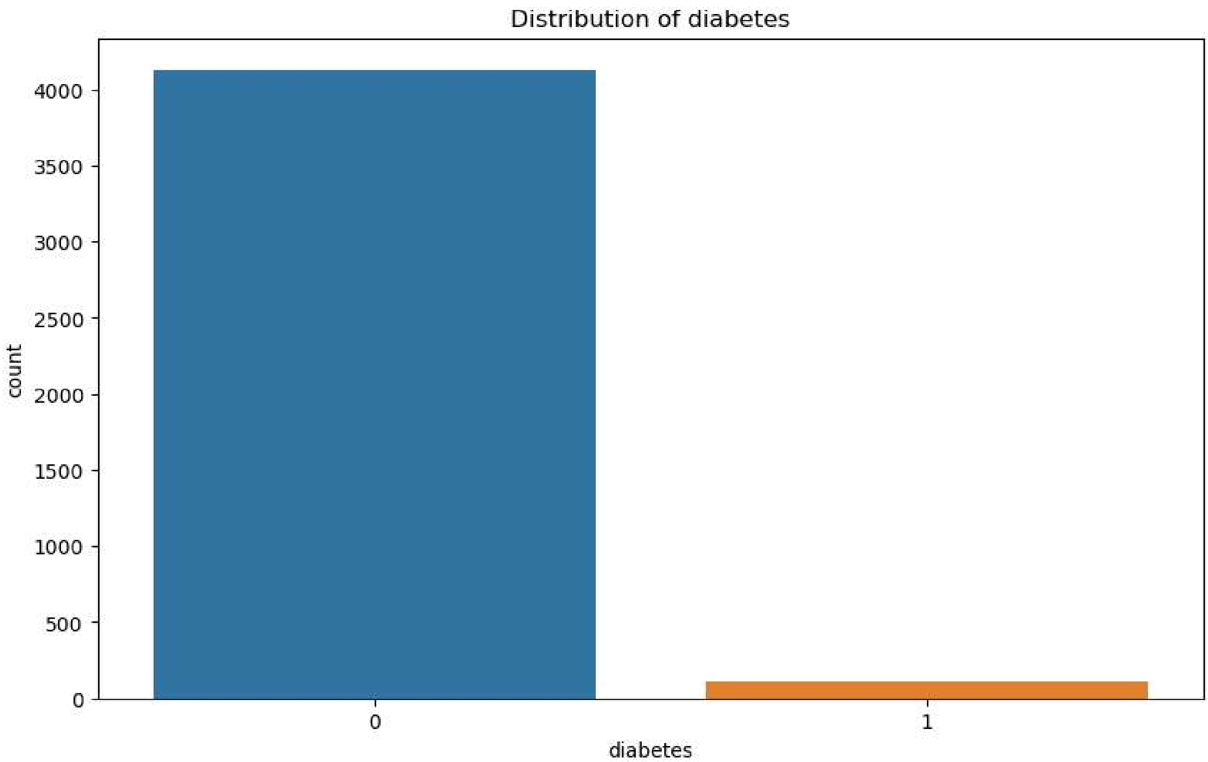


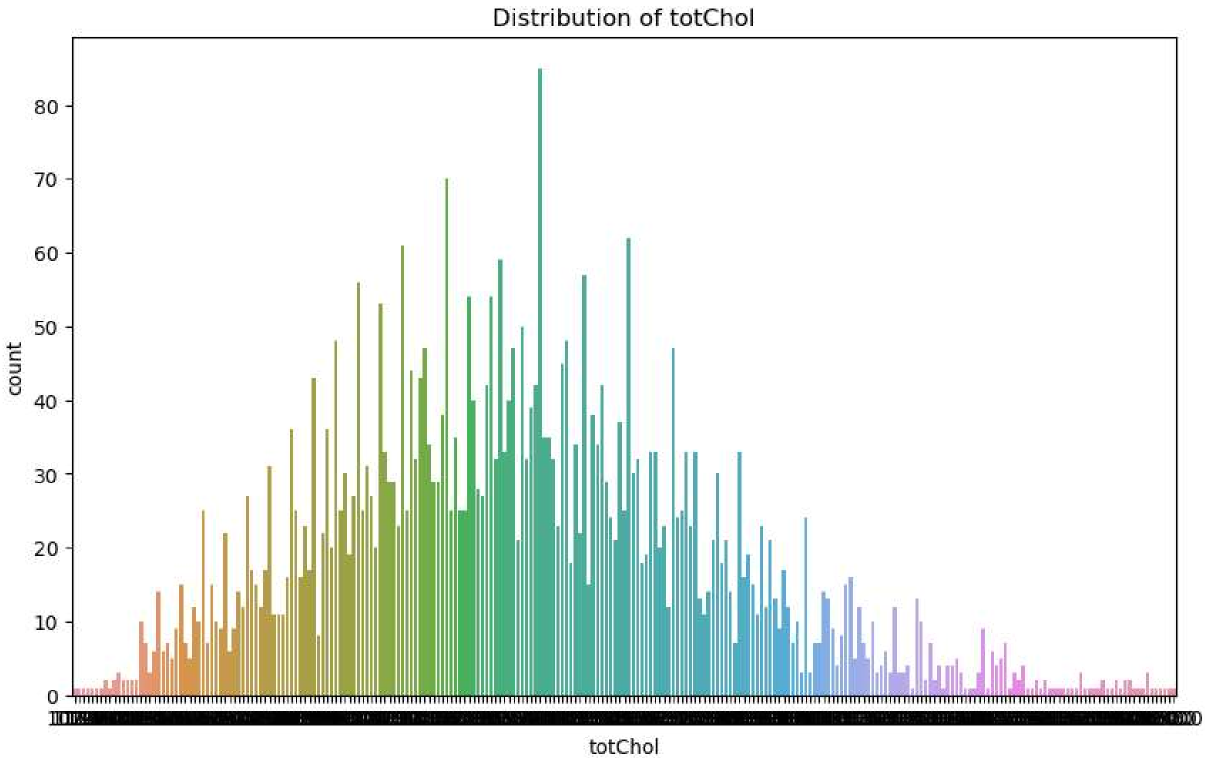
Distribution of cigsPerDay
count
0
500
1000
1500
2000
cigsPerDay

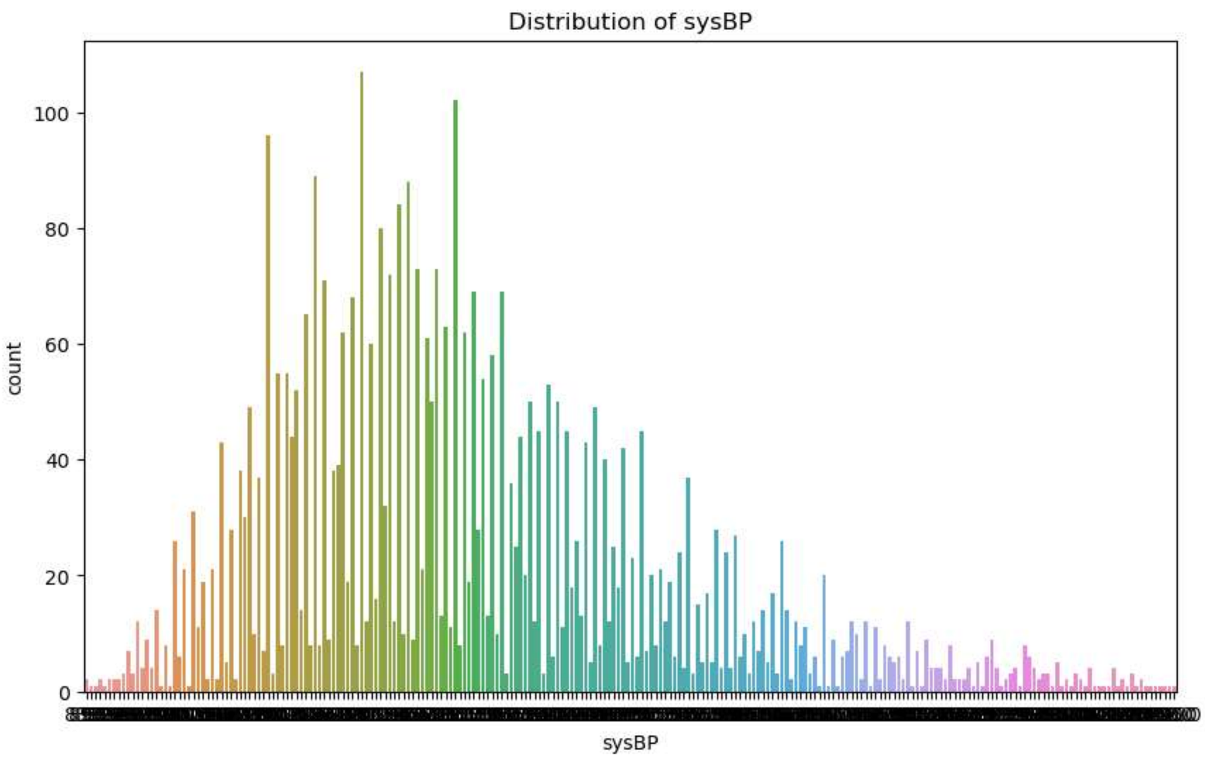

Distribution of BPMeds
count
4000
3500
3000
2500
2000
1500
1000
500
0
0.0
0.029615476474802963
1.0
BPMeds

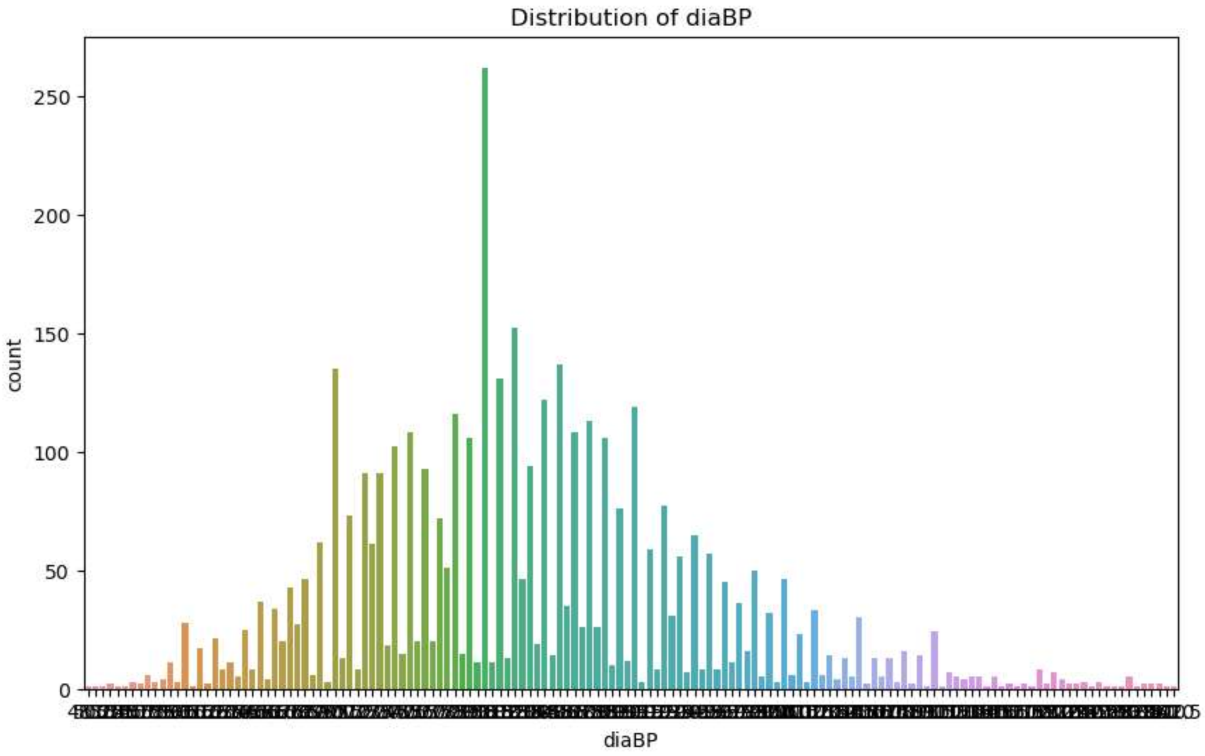

Distribution of prevalentStroke
count
4000
3500
3000
2500
2000
1500
1000
500
0
0
1
prevalentStroke

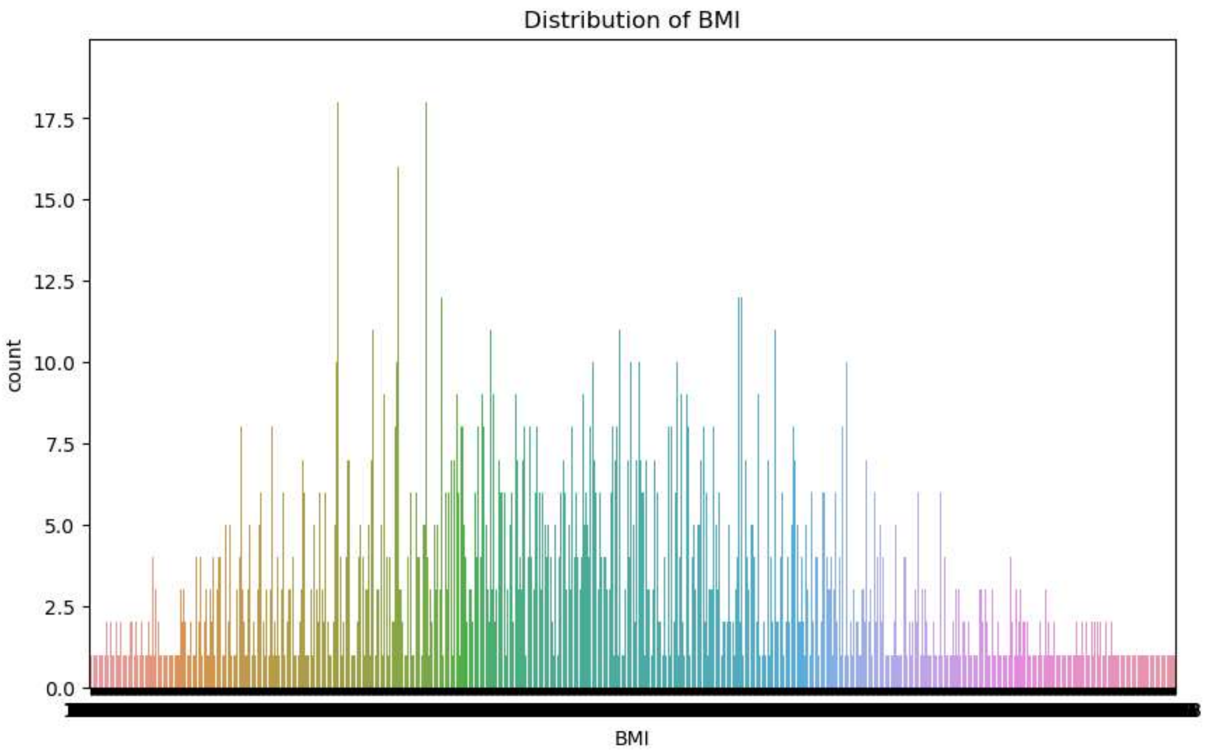

Distribution of prevalentHyp
count
3000
2500
2000
1500
1000
500
0
0
1
prevalentHyp

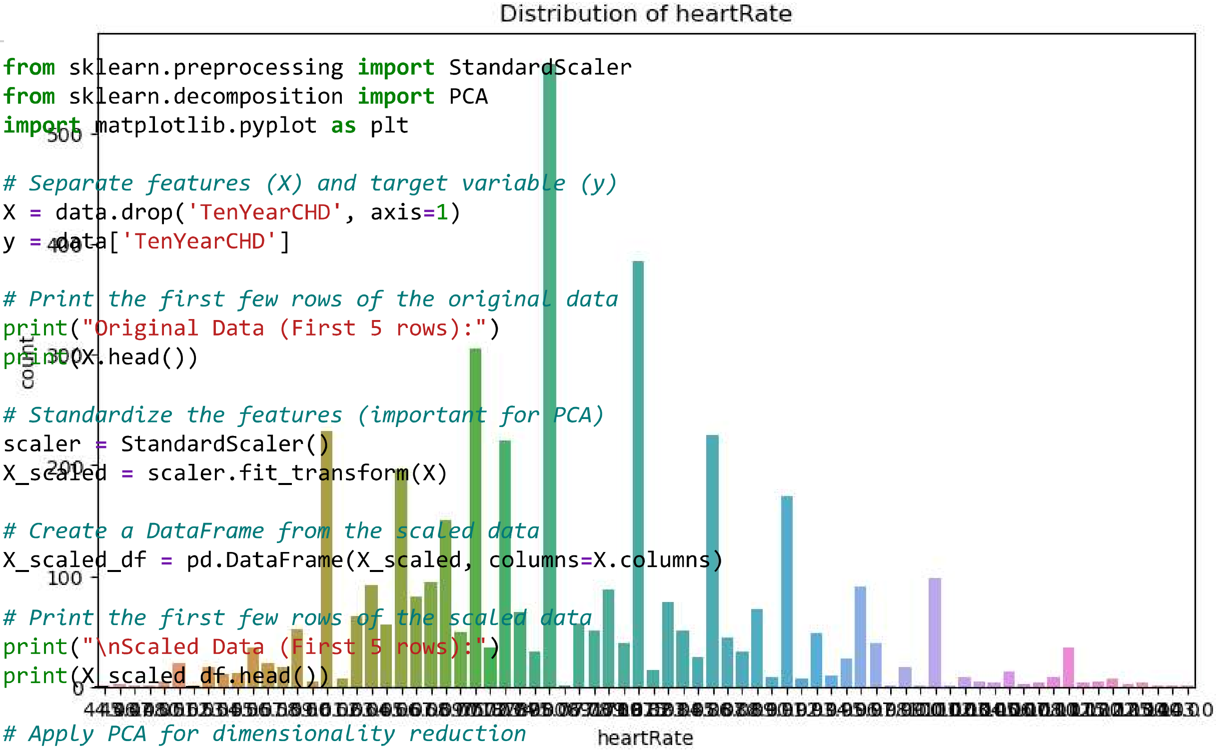

Distribution of diabetes
count
4000
3500
3000
2500
2000
1500
1000
500
0
0
1
diabetes


Distribution of totChol
count
totChol

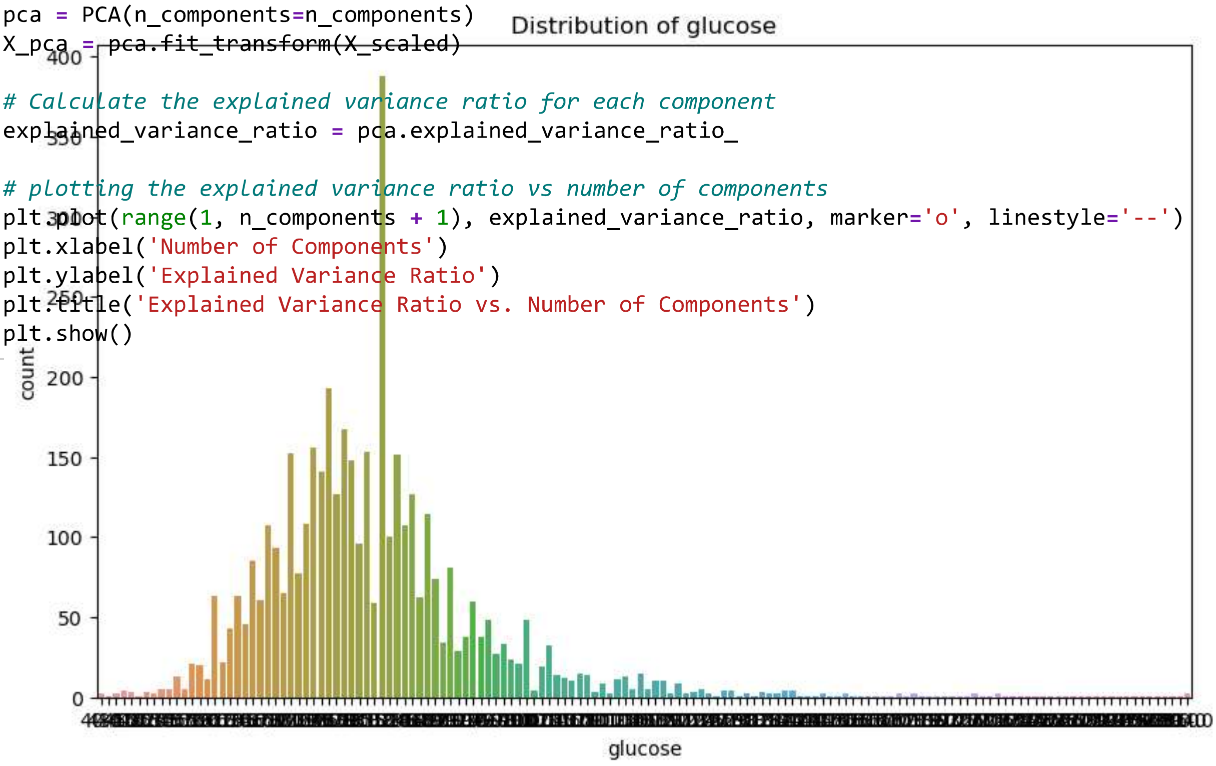

Distribution of sysBP
count
sysBP


Distribution of diaBP
count
diaBP

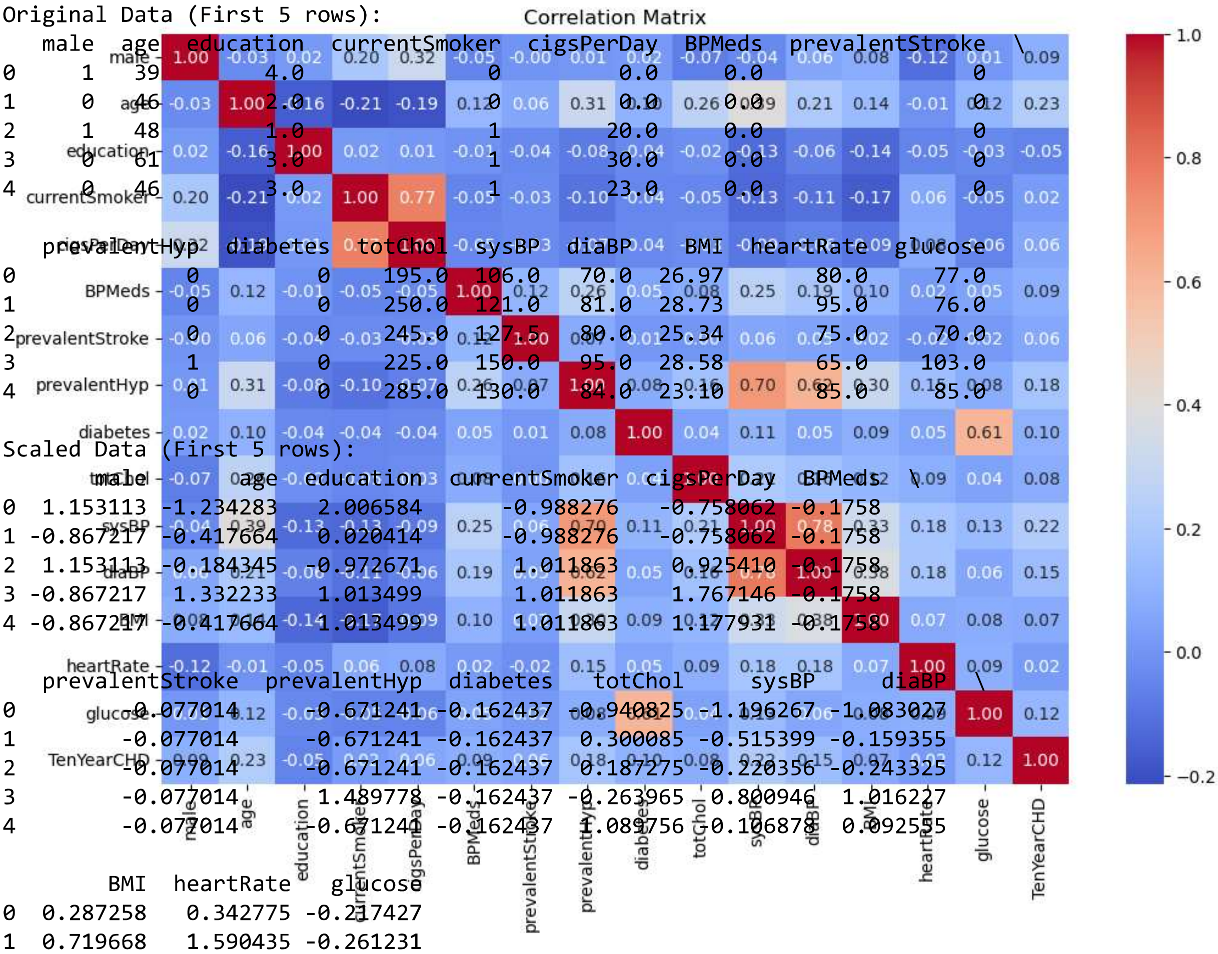

Distribution of BMI
count
BMI

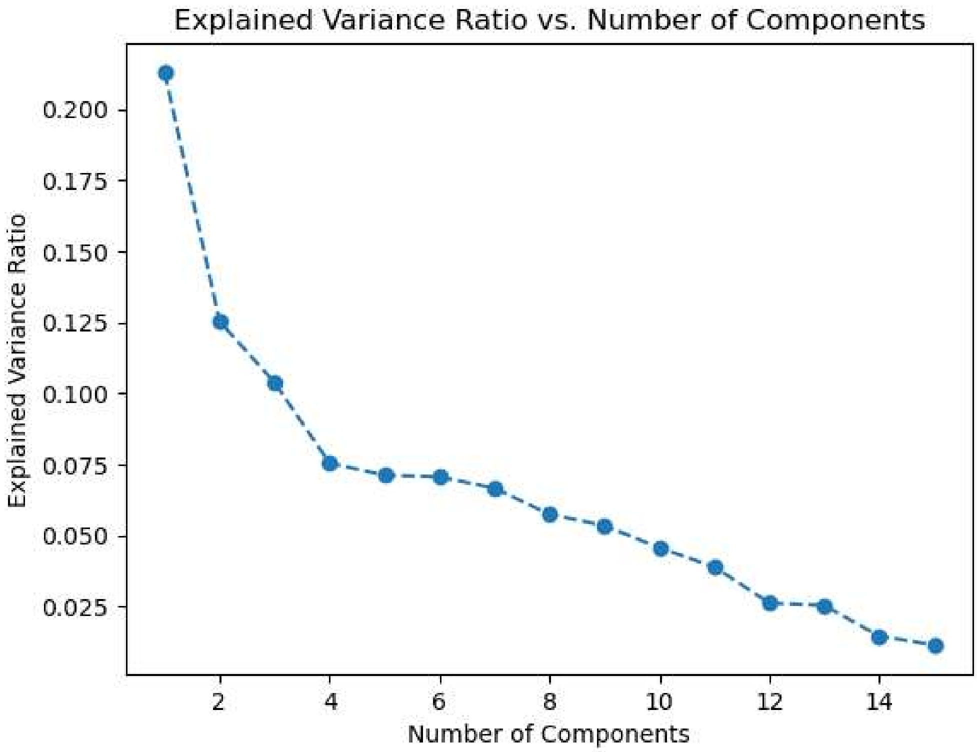



```python
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Separate features (X) and target variable (y)
X = data.drop('TenYearCHD', axis=1)
y = data['TenYearCHD']

# Print the first few rows of the original data
print("Original Data (First 5 rows):")
print(X.head())

# Standardize the features (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create a DataFrame from the scaled data
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Print the first few rows of the scaled data
print("\nScaled Data (First 5 rows):")
print(X_scaled_df.head())

# Apply PCA for dimensionality reduction
n_components = 15  # starting with all 15 attributes
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)

# Calculate the explained variance ratio for each component
explained_variance_ratio = pca.explained_variance_ratio_

# plotting the explained variance ratio vs number of components
plt.plot(range(1, n_components + 1), explained_variance_ratio, marker='o', linestyle='--')
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio vs. Number of Components')
plt.show()
```



```
Original Data (First 5 rows):
   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

   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

Scaled Data (First 5 rows):
       male       age  education  currentSmoker  cigsPerDay   BPMeds  \
0  1.153113 -1.234283   2.006584      -0.988276   -0.758062  -0.1758
1 -0.867217 -0.417664   0.020414      -0.988276   -0.758062  -0.1758
2  1.153113 -0.184345  -0.972671       1.011863    0.925410  -0.1758
3 -0.867217  1.332233   1.013499       1.011863    1.767146  -0.1758
4 -0.867217 -0.417664   1.013499       1.011863    1.177931  -0.1758

   prevalentStroke  prevalentHyp  diabetes   totChol     sysBP     diaBP  \
0        -0.077014     -0.671241 -0.162437 -0.940825 -1.196267 -1.083027
1        -0.077014     -0.671241 -0.162437  0.300085 -0.515399 -0.159355
2        -0.077014     -0.671241 -0.162437  0.187275 -0.220356 -0.243325
3        -0.077014      1.489778 -0.162437 -0.263965  0.800946  1.016227
4        -0.077014     -0.671241 -0.162437  1.089756 -0.106878  0.092555

        BMI  heartRate   glucose
0  0.287258   0.342775 -0.217427
1  0.719668   1.590435 -0.261231
2 -0.113213  -0.073111 -0.524054
3  0.682815  -0.904884  0.921472
4 -0.663554   0.758662  0.133004
```





In [28]:

```
# Update X with the selected number of components
# The explained variance ratio goes down steeply till 4 number of components and then decreases slowly
# so 4 components have most of the effect on the final output so let us update the number to 4.
n_components_to_keep = 4  # Updating based on the plot.
X_selected = X_pca[:, :n_components_to_keep]

# Get the loadings for the first few principal components
loadings = pca.components_[:n_components_to_keep]

# Create a DataFrame to display the loadings and their corresponding attributes
loadings_df = pd.DataFrame(loadings, columns=X.columns)

# Print the attributes used for each principal component
for i in range(n_components_to_keep):
    print(f"Attributes for Principal Component {i + 1}:")
    print(loadings_df.iloc[i].sort_values(ascending=False))
    print("\n")
```

```
Attributes for Principal Component 1:
sysBP              0.483876
diaBP              0.437357
prevalentHyp       0.433780
age                0.297353
BMI                0.285905
BPMeds             0.201591
totChol            0.187347
glucose            0.145971
diabetes           0.134675
heartRate          0.123631
prevalentStroke    0.065923
male              -0.047280
education         -0.110604
cigsPerDay        -0.167438
currentSmoker     -0.197331
Name: 0, dtype: float64


Attributes for Principal Component 2:
cigsPerDay         0.631560
currentSmoker      0.590094
male               0.350047
diaBP              0.197794
prevalentHyp       0.161615
heartRate          0.155022
sysBP              0.152936
BMI                0.050690
BPMeds             0.039056
totChol            0.017655
diabetes          -0.016920
prevalentStroke   -0.017044
education         -0.021605
glucose           -0.024434
age               -0.108061
Name: 1, dtype: float64


Attributes for Principal Component 3:
diabetes           0.686029
glucose            0.682774
currentSmoker      0.058432
heartRate          0.056922
male               0.054923
cigsPerDay         0.053104
age                0.017670
totChol           -0.017148
prevalentStroke   -0.022951
BMI               -0.026716
education         -0.029890
BPMeds            -0.046681
sysBP             -0.093008
prevalentHyp      -0.118592
diaBP             -0.151852
Name: 2, dtype: float64


Attributes for Principal Component 4:
male               0.564149
prevalentStroke    0.278847
BMI                0.214414
age                0.095735
BPMeds             0.089385
diabetes           0.038080
prevalentHyp       0.018720
diaBP              0.011063
glucose           -0.001716
sysBP             -0.032723
cigsPerDay        -0.034051
education         -0.041742
currentSmoker     -0.114240
```

```
totChol           -0.276844
heartRate         -0.667279
Name: 3, dtype: float64
```

In [37]:

```
# spliting the data

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# Split the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)

# Print the shapes of the resulting sets to verify the split
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (3392, 4)
X_test shape: (848, 4)
y_train shape: (3392,)
y_test shape: (848,)
```

In [38]:

```
# Resampling the data

from imblearn.over_sampling import SMOTE, ADASYN
from imblearn.under_sampling import RandomUnderSampler

# Define a list of resampling strategies to try
resampling_strategies = [
    ('SMOTE', SMOTE(random_state=42)),
    ('ADASYN', ADASYN(random_state=42)),
    ('RandomUnderSampler', RandomUnderSampler(random_state=42))
]

# Define a list of classifiers to try (Logistic Regression, Decision Tree, KNN, SVM)
classifiers = [
    ('LogisticRegression', LogisticRegression(random_state=42)),
    ('DecisionTree', DecisionTreeClassifier(random_state=42)),
    ('KNN', KNeighborsClassifier()),
    ('SVM', SVC(random_state=42))  # Include SVM
]
```

In [39]:

```
# Pipeling and finding the accuracies of different models in combination with different sampling techniques

from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report

# Initialize a dictionary to store results
results = {}

# Iterate through resampling strategies
for resampling_name, resampling_strategy in resampling_strategies:
    # Create a dictionary for each resampling strategy
    results[resampling_name] = {}

    # Create a pipeline for each classifier
    for classifier_name, classifier in classifiers:
        # Create a pipeline with resampling and classification steps
        model = Pipeline([
            ('resampling', resampling_strategy),
            ('classifier', classifier)
        ])

        # Define hyperparameters to tune (customize as needed)
        param_grid = {}

        if classifier_name == 'LogisticRegression':
            param_grid = {
                'classifier__C': [0.01, 0.1, 1, 10]  # Hyperparameters for Logistic Regression
            }
        elif classifier_name == 'DecisionTree':
            param_grid = {
                'classifier__max_depth': [None, 1, 3, 5, 10]  # Hyperparameters for Decision Tree
            }
        elif classifier_name == 'KNN':
            param_grid = {
                'classifier__n_neighbors': [3, 5, 7, 9, 13]  # Hyperparameters for KNN
            }
        elif classifier_name == 'SVM':
            param_grid = {
                'classifier__C': [0.0000000001,0.0001,0.01,1,10],  # Hyperparameters for SVM
                'classifier__kernel': ['linear', 'rbf']
            }

        # Perform hyperparameter tuning using GridSearchCV
        grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
        grid_search.fit(X_train, y_train)

        # Get the best classifier from the grid search
        best_classifier = grid_search.best_estimator_

        # Make predictions on the test data
        y_pred = best_classifier.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)

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

        # Store the results
        results[resampling_name][classifier_name] = {
            'best_classifier': best_classifier,
            'accuracy': accuracy,
            'classification_report': class_report
        }
```

In [40]:

```
# Print the results
for resampling_name, classifiers_dict in results.items():
    print(f"Resampling Strategy: {resampling_name}")
    for classifier_name, metrics in classifiers_dict.items():
        print(f"Classifier: {classifier_name}")
        print(f"Accuracy: {metrics['accuracy']:.4f}")
        print(f"Classification Report:\n{metrics['classification_report']}")
        print()
```

```
Resampling Strategy: SMOTE
Classifier: LogisticRegression
Accuracy: 0.6604
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.68      0.77       725
           1       0.23      0.56      0.32       123

    accuracy                           0.66       848
   macro avg       0.56      0.62      0.55       848
weighted avg       0.80      0.66      0.71       848


Classifier: DecisionTree
Accuracy: 0.7288
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.79      0.83       725
           1       0.23      0.38      0.29       123

    accuracy                           0.73       848
   macro avg       0.56      0.58      0.56       848
weighted avg       0.79      0.73      0.75       848


Classifier: KNN
Accuracy: 0.6781
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.72      0.79       725
           1       0.20      0.41      0.27       123

    accuracy                           0.68       848
   macro avg       0.54      0.57      0.53       848
weighted avg       0.78      0.68      0.72       848


Classifier: SVM
Accuracy: 0.8467
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.98      0.92       725
           1       0.32      0.05      0.08       123

    accuracy                           0.85       848
   macro avg       0.59      0.52      0.50       848
weighted avg       0.78      0.85      0.80       848


Resampling Strategy: ADASYN
Classifier: LogisticRegression
Accuracy: 0.6450
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.65      0.76       725
           1       0.23      0.61      0.33       123

    accuracy                           0.65       848
   macro avg       0.57      0.63      0.55       848
weighted avg       0.81      0.65      0.70       848


Classifier: DecisionTree
Accuracy: 0.7182
Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.89      0.77      0.82       725
           1       0.23      0.41      0.30       123

    accuracy                           0.72       848
   macro avg       0.56      0.59      0.56       848
weighted avg       0.79      0.72      0.75       848


Classifier: KNN
Accuracy: 0.6616
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.71      0.78       725
           1       0.18      0.37      0.24       123

    accuracy                           0.66       848
   macro avg       0.52      0.54      0.51       848
weighted avg       0.77      0.66      0.70       848


Classifier: SVM
Accuracy: 0.6627
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.68      0.77       725
           1       0.23      0.58      0.33       123

    accuracy                           0.66       848
   macro avg       0.57      0.63      0.55       848
weighted avg       0.81      0.66      0.71       848


Resampling Strategy: RandomUnderSampler
Classifier: LogisticRegression
Accuracy: 0.6757
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.70      0.79       725
           1       0.23      0.52      0.32       123

    accuracy                           0.68       848
   macro avg       0.56      0.61      0.55       848
weighted avg       0.80      0.68      0.72       848


Classifier: DecisionTree
Accuracy: 0.7241
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.78      0.83       725
           1       0.23      0.39      0.29       123

    accuracy                           0.72       848
   macro avg       0.56      0.59      0.56       848
weighted avg       0.79      0.72      0.75       848


Classifier: KNN
Accuracy: 0.6191
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.63      0.74       725
           1       0.20      0.56      0.30       123

    accuracy                           0.62       848
   macro avg       0.55      0.59      0.52       848
```

```
weighted avg       0.79      0.62      0.67       848


Classifier: SVM
Accuracy: 0.8514
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       725
           1       0.40      0.05      0.09       123

    accuracy                           0.85       848
   macro avg       0.63      0.52      0.50       848
weighted avg       0.79      0.85      0.80       848
```