In [22]:
```
import pandas as pd
df = pd.read_excel("C:\\Users\\USER\\Desktop\\AI.ICA\\Health\\health care dataset.x
dt = pd.read_excel("C:\\Users\\USER\\Desktop\\AI.ICA\\Health\\health care dataset.x
```

In [2]:
```
print(dt)
```

```
         id  gender   age  hypertension  heart_disease ever_married  \
0      9046    Male  67.0             0              1          Yes
1     51676  Female  61.0             0              0          Yes
2     31112    Male  80.0             0              1          Yes
3     60182  Female  49.0             0              0          Yes
4      1665  Female  79.0             1              0          Yes
...     ...     ...   ...           ...            ...          ...
5105  18234  Female  80.0             1              0          Yes
5106  44873  Female  81.0             0              0          Yes
5107  19723  Female  35.0             0              0          Yes
5108  37544    Male  51.0             0              0          Yes
5109  44679  Female  44.0             0              0          Yes

          work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0           Private          Urban             228.69  36.6  formerly smoked
1     Self-employed          Rural             202.21   NaN     never smoked
2           Private          Rural             105.92  32.5     never smoked
3           Private          Urban             171.23  34.4           smokes
4     Self-employed          Rural             174.12  24.0     never smoked
...             ...            ...                ...   ...              ...
5105        Private          Urban              83.75   NaN     never smoked
5106  Self-employed          Urban             125.20  40.0     never smoked
5107  Self-employed          Rural              82.99  30.6     never smoked
5108        Private          Rural             166.29  25.6  formerly smoked
5109       Govt_job          Urban              85.28  26.2          Unknown

      stroke
0          1
1          1
2          1
3          1
4          1
...      ...
5105       0
5106       0
5107       0
5108       0
5109       0

[5110 rows x 12 columns]
```

In [3]:
```
#check datatype
print(dt.dtypes)
```

```
id                     int64
gender                object
age                  float64
hypertension           int64
heart_disease          int64
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
stroke                 int64
dtype: object
```

In [4]:
```
#age can not be float
#round it up
import pandas as pd



# Round the values in the 'age' column to the nearest integer
dt['age'] = dt['age'].round()

# Convert 'age' column to integer type
dt['age'] = dt['age'].astype(int)

# Display the DataFrame with rounded 'age' values
print(dt)
```

```
         id  gender  age  hypertension  heart_disease ever_married  \
0      9046    Male   67             0              1          Yes
1     51676  Female   61             0              0          Yes
2     31112    Male   80             0              1          Yes
3     60182  Female   49             0              0          Yes
4      1665  Female   79             1              0          Yes
...     ...     ...  ...           ...            ...          ...
5105  18234  Female   80             1              0          Yes
5106  44873  Female   81             0              0          Yes
5107  19723  Female   35             0              0          Yes
5108  37544    Male   51             0              0          Yes
5109  44679  Female   44             0              0          Yes

          work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0           Private          Urban             228.69  36.6  formerly smoked
1     Self-employed          Rural             202.21   NaN     never smoked
2           Private          Rural             105.92  32.5     never smoked
3           Private          Urban             171.23  34.4           smokes
4     Self-employed          Rural             174.12  24.0     never smoked
...             ...            ...                ...   ...              ...
5105        Private          Urban              83.75   NaN     never smoked
5106  Self-employed          Urban             125.20  40.0     never smoked
5107  Self-employed          Rural              82.99  30.6     never smoked
5108        Private          Rural             166.29  25.6  formerly smoked
5109       Govt_job          Urban              85.28  26.2          Unknown

      stroke
0          1
1          1
2          1
3          1
4          1
...      ...
5105       0
5106       0
5107       0
5108       0
5109       0

[5110 rows x 12 columns]
```

In [5]:
```
# confirm if age is in int.
print(dt.dtypes)
```

```
id                     int64
gender                object
age                    int32
hypertension           int64
heart_disease          int64
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
stroke                 int64
dtype: object
```

In [6]:
```
#check for duplicate rows
duplicate_rows = dt[dt.duplicated()]

# Print the duplicate rows
print("Duplicate Rows:")
print(duplicate_rows)
```

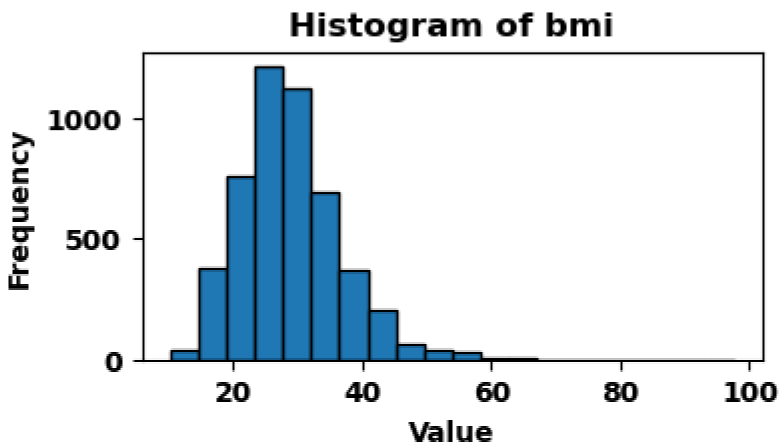
```
Duplicate Rows:
Empty DataFrame
Columns: [id, gender, age, hypertension, heart_disease, ever_married, work_type, Res
idence_type, avg_glucose_level, bmi, smoking_status, stroke]
Index: []
```

In [7]:
```
#check for missing values
import pandas as pd


# Check for missing values using isna() or isnull()
missing_values = dt.isna().sum()

# Display the count of missing values for each column
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

In [8]:
```
#handle missing values in bmi column
#bim is continuous
#check its distribution using histogram.
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(4, 2))
plt.hist(dt['bmi'], bins=20, edgecolor='k')  # Adjust the number of bins as needed
plt.title('Histogram of bmi')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



In [9]:

```
#bmi is not normally distributed. so replace missing values with median
#view bim statistical summary
summary_stats = dt['bmi'].describe()
print(summary_stats)
```

```
count    4909.000000
mean       28.893237
std         7.854067
min        10.300000
25%        23.500000
50%        28.100000
75%        33.100000
max        97.600000
Name: bmi, dtype: float64
```

In [10]:

```
#replace missing values
median_bmi = dt['bmi'].median()

# Replace missing values with the median
dt['bmi'].fillna(median_bmi, inplace=True)
```

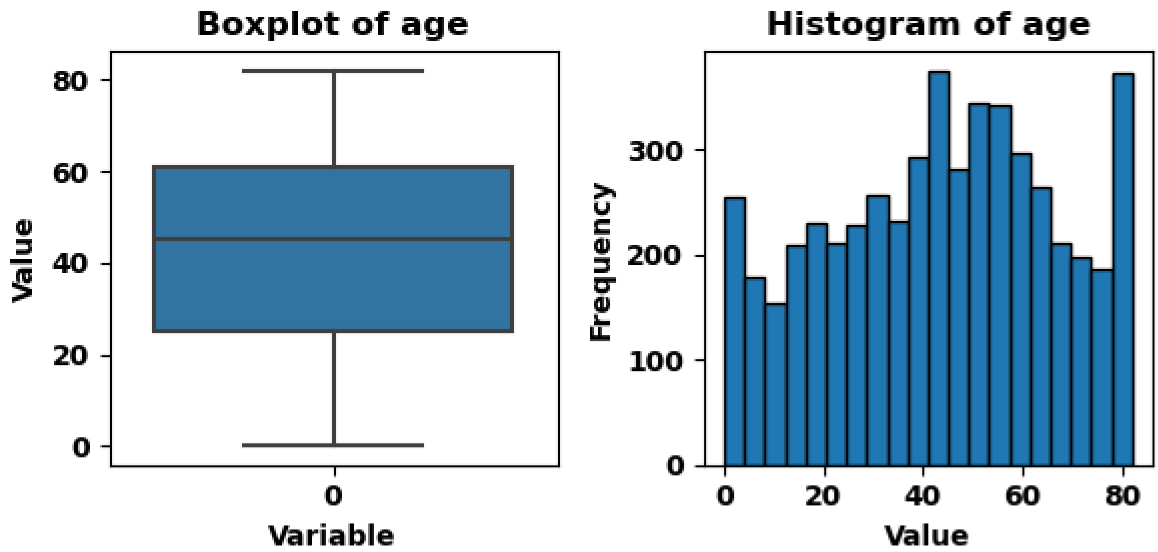
In [11]:

```
#check if the missing values have been reolaced
missing_values = dt.isna().sum()

# Display the count of missing values for each column
print("Missing Values:")
print(missing_values)
```

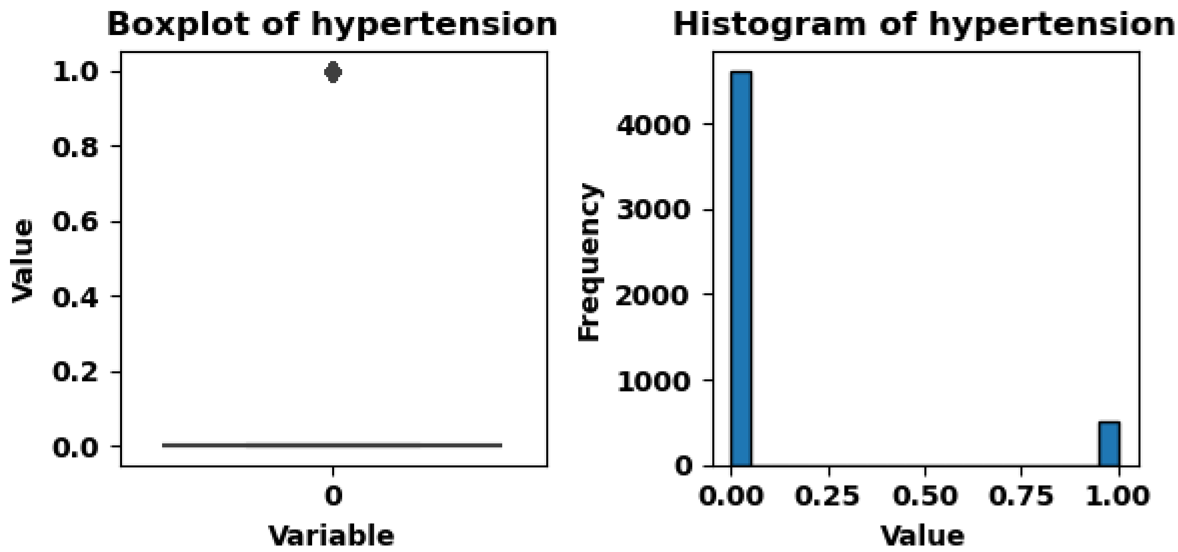
```
Missing Values:
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

In [12]:

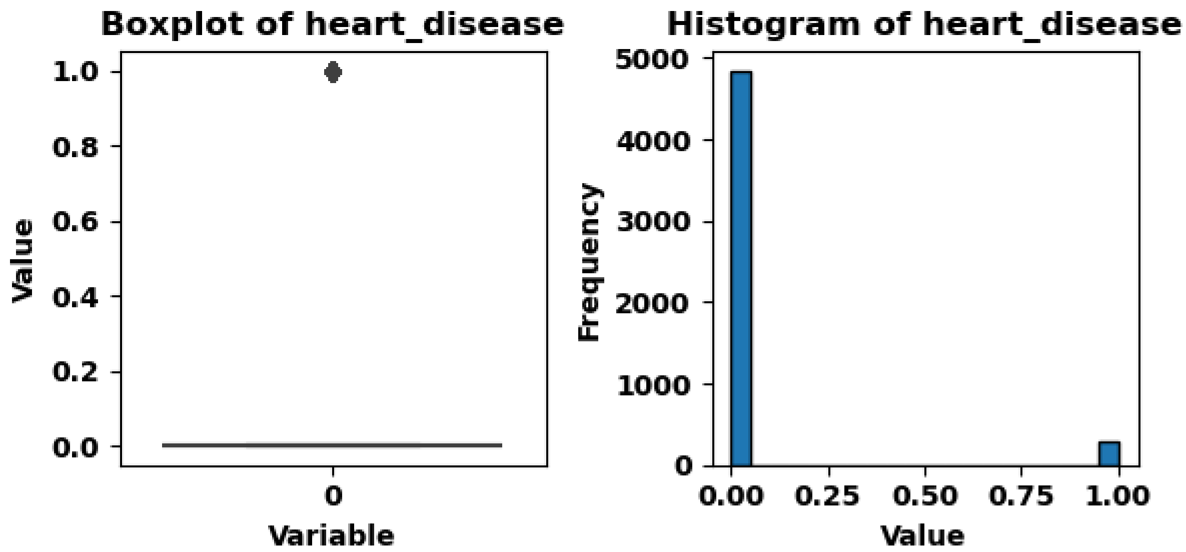
```
#examine the numerical variables for outliers
#for age column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['age'], ax=axs[0])
axs[0].set_title('Boxplot of age')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['age'], bins=20, edgecolor='k')  # Adjust the number of bins as need
axs[1].set_title('Histogram of age')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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



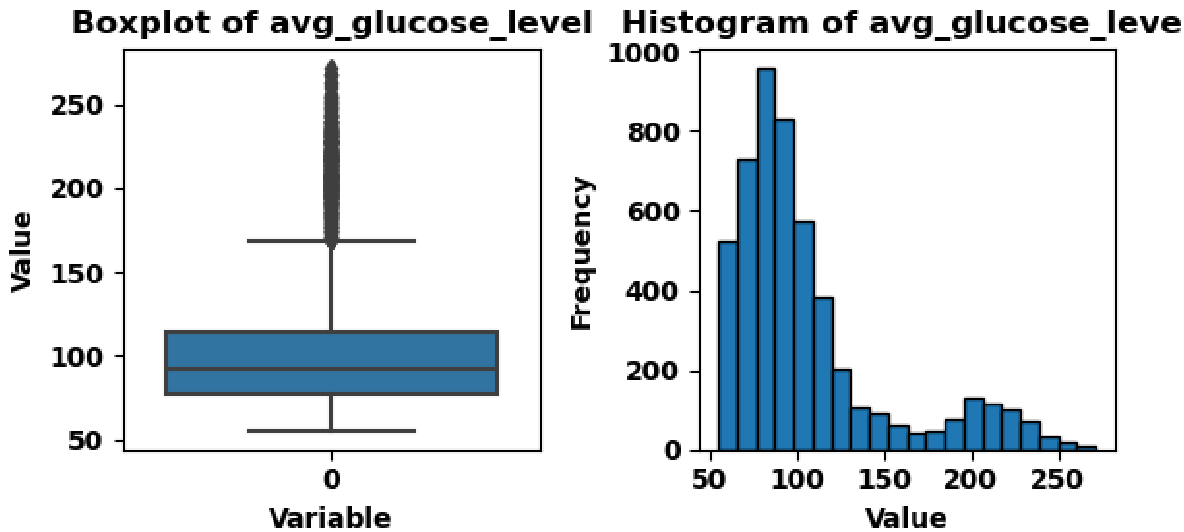
In [13]:
```
#check outliers in hypertension column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['hypertension'], ax=axs[0])
axs[0].set_title('Boxplot of hypertension')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['hypertension'], bins=20, edgecolor='k')  # Adjust the number of bin
axs[1].set_title('Histogram of hypertension')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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



In [14]:
```
#check for outliers in heart_disease column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['heart_disease'], ax=axs[0])
axs[0].set_title('Boxplot of heart_disease')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['heart_disease'], bins=20, edgecolor='k')  # Adjust the number of bi
axs[1].set_title('Histogram of heart_disease')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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

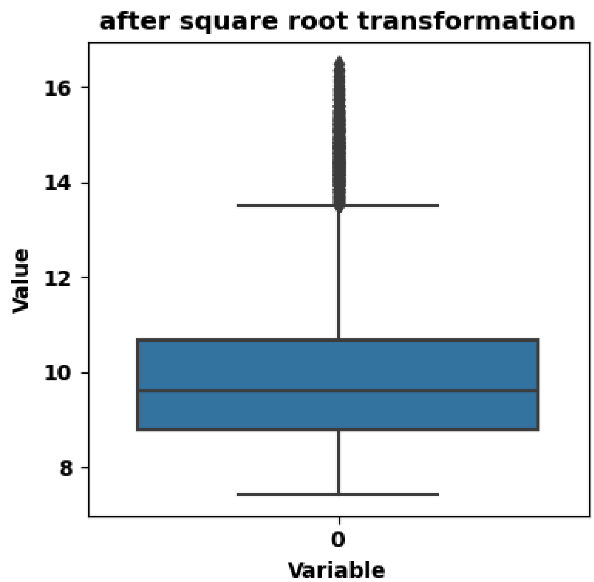


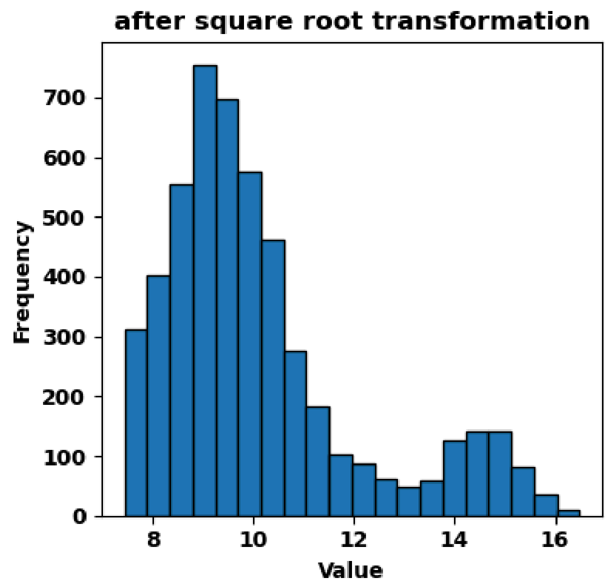
In [15]:
```
#check for outliers in avg_glucose_level
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['avg_glucose_level'], ax=axs[0])
axs[0].set_title('Boxplot of avg_glucose_level')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['avg_glucose_level'], bins=20, edgecolor='k')  # Adjust the number o
axs[1].set_title('Histogram of avg_glucose_level')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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



In [23]:
```
# transform avg_glucose_level using root square method
import numpy as np


dt['avg_glucose_level'] = np.sqrt(dt['avg_glucose_level'])
```

In [25]:
```
#confirm if the transformation
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['avg_glucose_level'], ax=axs[0])
axs[0].set_title('after square root transformation')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['avg_glucose_level'], bins=20, edgecolor='k')  # Adjust the number o
axs[1].set_title('after square root transformation')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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





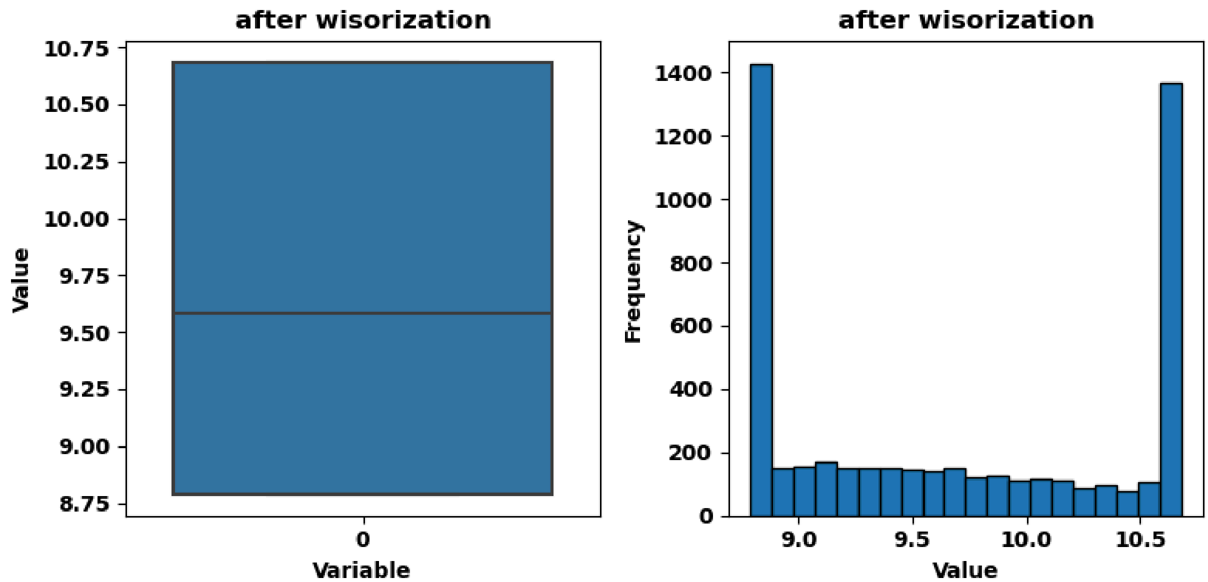
In [32]:

```
#avg_glucose_level contains outliers
#handle the outliers using winzorization method.
#handle outliers on the right side using winsorization method
import numpy as np



# Adjust the percentile thresholds
lower_threshold = np.percentile(dt['avg_glucose_level'],25)  # Lowering to 0.5th pe
upper_threshold = np.percentile(dt['avg_glucose_level'],75)  # Raising to 99.5th pe

# Replace outliers with threshold values
dt['avg_glucose_level'] = np.where(dt['avg_glucose_level'] < lower_threshold, lower
dt['avg_glucose_level'] = np.where(dt['avg_glucose_level'] > upper_threshold, upper
```

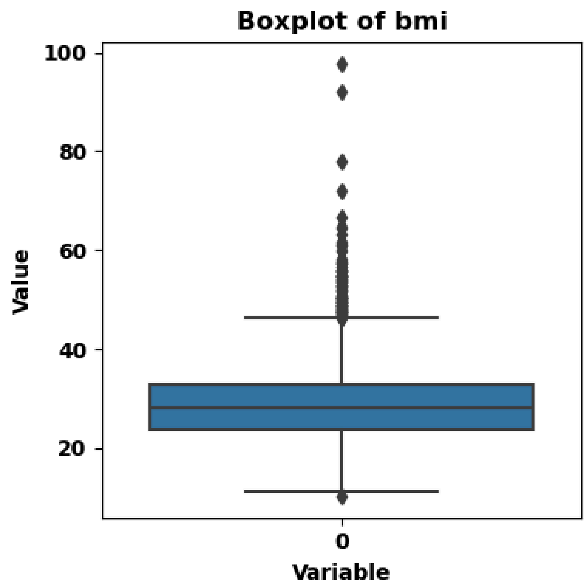
In [34]:

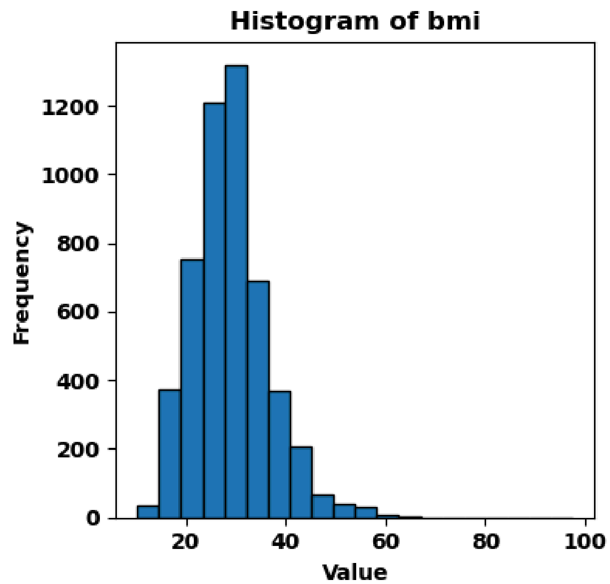
```
#confirm if the outliers has been handled
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['avg_glucose_level'], ax=axs[0])
axs[0].set_title('after wisorization')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['avg_glucose_level'], bins=20, edgecolor='k')  # Adjust the number o
axs[1].set_title('after wisorization')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()
```

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



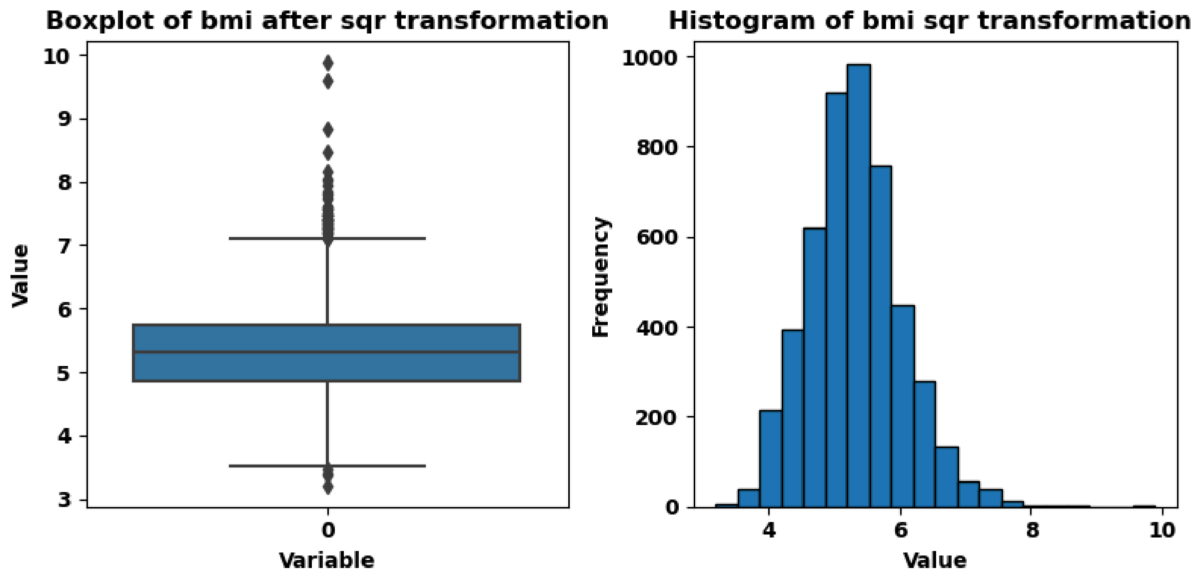
In [28]:
```
#check for outliers in bim column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['bmi'], ax=axs[0])
axs[0].set_title('Boxplot of bmi')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['bmi'], bins=20, edgecolor='k')  # Adjust the number of bins as need
axs[1].set_title('Histogram of bmi')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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





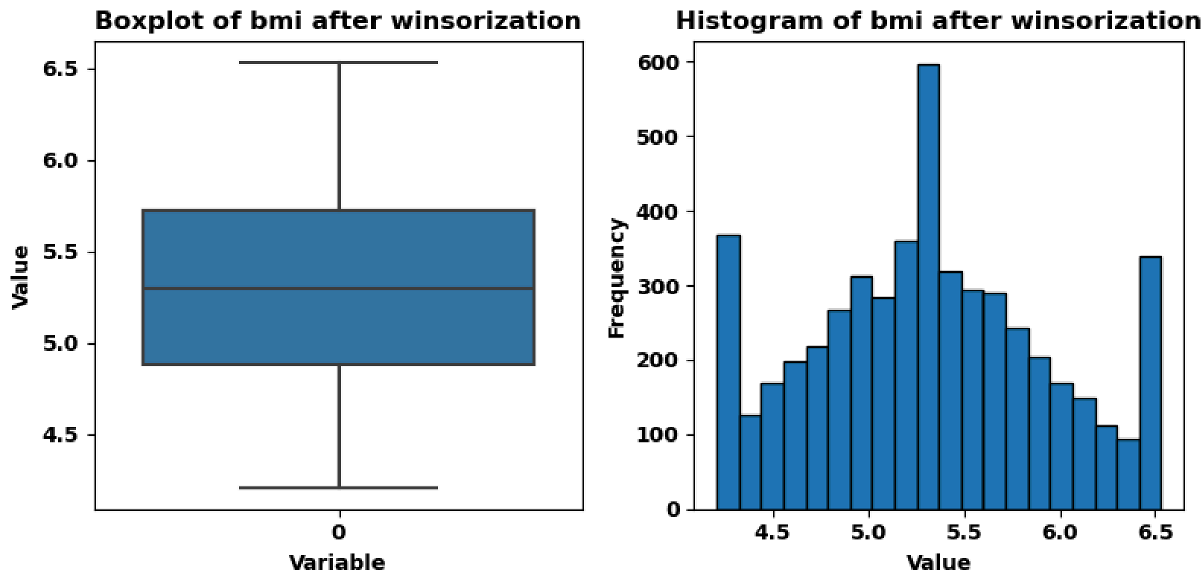
In [35]:
```
#transform bmi column using root square method
import numpy as np


dt['bmi'] = np.sqrt(dt['bmi'])
```

In [36]:
```
#confirm the transformation
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['bmi'], ax=axs[0])
axs[0].set_title('Boxplot of bmi after sqr transformation')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['bmi'], bins=20, edgecolor='k')  # Adjust the number of bins as need
axs[1].set_title('Histogram of bmi sqr transformation ')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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

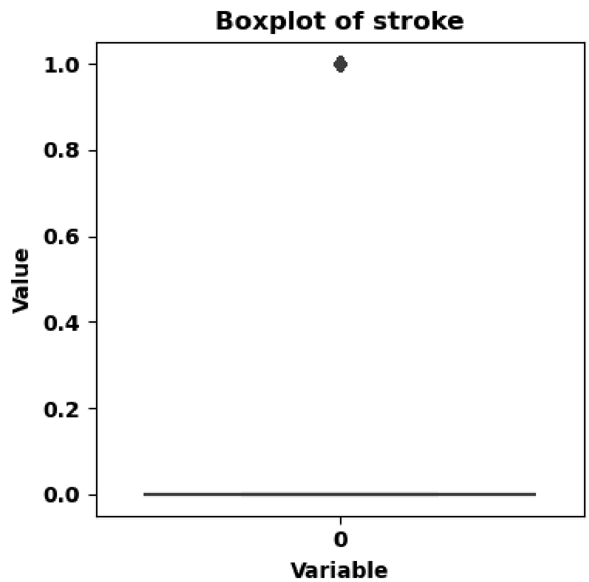


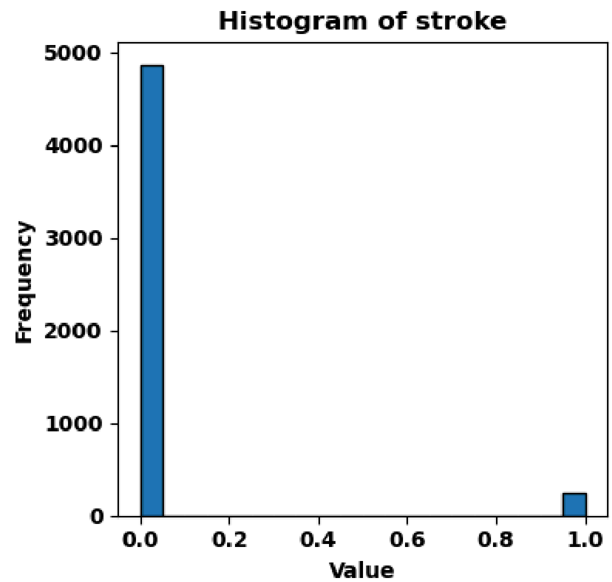
In [34]:
```python
#handle the outliers using winsorization method
import numpy as np



# Adjust the percentile thresholds
lower_threshold = np.percentile(dt['bmi'],5)  # Lowering to 0.5th percentile
upper_threshold = np.percentile(dt['bmi'],95)  # Raising to 99.5th percentile

# Replace outliers with threshold values
dt['bmi'] = np.where(dt['bmi'] < lower_threshold, lower_threshold, dt['bmi'])
dt['bmi'] = np.where(dt['bmi'] > upper_threshold, upper_threshold, dt['bmi'])
```

In [35]:
```python
# check to see if the outliers have been handled
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['bmi'], ax=axs[0])
axs[0].set_title('Boxplot of bmi after winsorization')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['bmi'], bins=20, edgecolor='k')  # Adjust the number of bins as need
axs[1].set_title('Histogram of bmi after winsorization')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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



In [36]:
```
# examine stroke column for outliers
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['stroke'], ax=axs[0])
axs[0].set_title('Boxplot of stroke')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['stroke'], bins=20, edgecolor='k')  # Adjust the number of bins as n
axs[1].set_title('Histogram of stroke')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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





In [37]:
```
#for exploratory data analysis, label encode the categorical variables

from sklearn.preprocessing import LabelEncoder

# Create a label encoder object
label_encoder = LabelEncoder()

# Iterate over each column in the DataFrame
for column in dt.columns:
    # Check if the column data type is 'object' (categorical)
    if dt[column].dtype == 'object':
        # Apply label encoding to the column
        dt[column] = label_encoder.fit_transform(dt[column].astype(str))

# Display the updated DataFrame with label encoded categorical variables
print(dt)
```

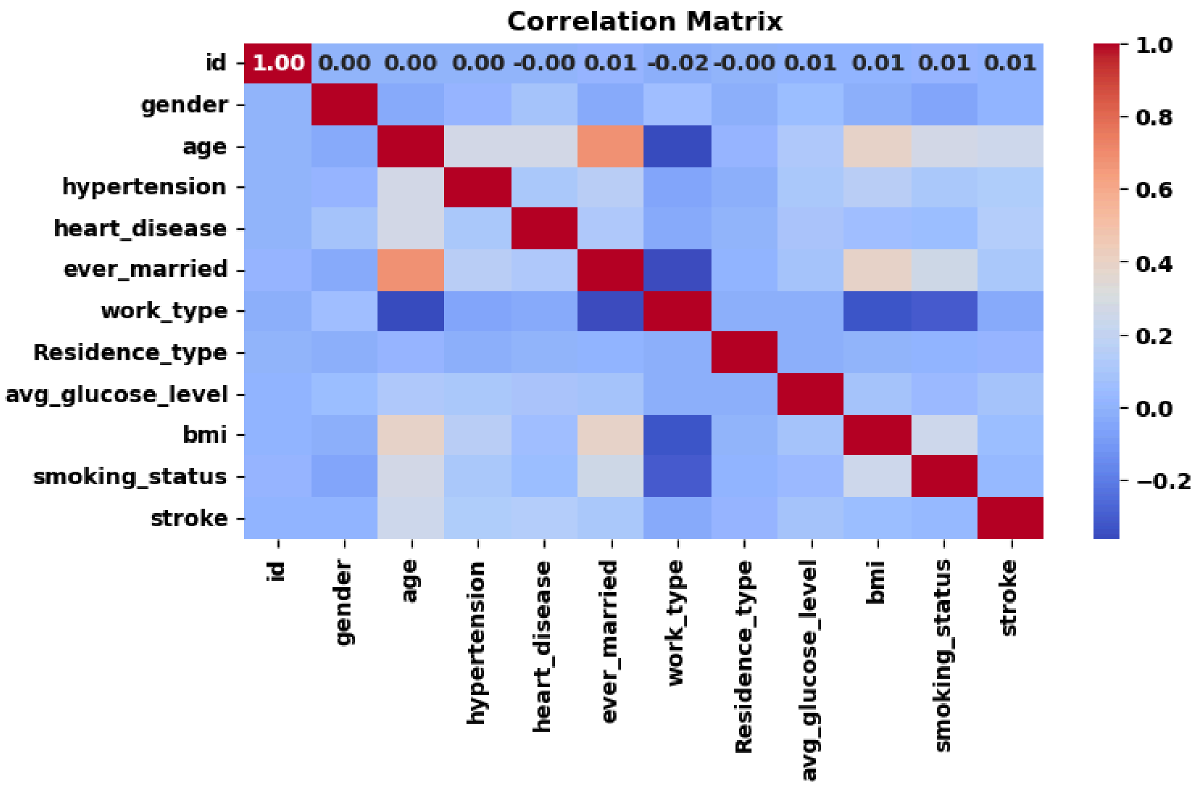
```
        id  gender  age  hypertension  heart_disease  ever_married  \
0     9046       1   67             0              1             1
1    51676       0   61             0              0             1
2    31112       1   80             0              1             1
3    60182       0   49             0              0             1
4     1665       0   79             1              0             1
...    ...     ...  ...           ...            ...           ...
5105 18234       0   80             1              0             1
5106 44873       0   81             0              0             1
5107 19723       0   35             0              0             1
5108 37544       1   51             0              0             1
5109 44679       0   44             0              0             1

      work_type  Residence_type  avg_glucose_level       bmi  smoking_status  \
0             2               1          10.681292  6.049793               1
1             3               0          10.681292  5.300943               2
2             2               0          10.291744  5.700877               2
3             2               1          10.681292  5.865151               3
4             3               0          10.681292  4.898979               2
...         ...             ...                ...       ...             ...
5105          2               1           9.151503  5.300943               2
5106          3               1          10.681292  6.324555               2
5107          3               0           9.109885  5.531727               2
5108          2               0          10.681292  5.059644               1
5109          0               1           9.234717  5.118594               0

      stroke
0          1
1          1
2          1
3          1
4          1
...      ...
5105       0
5106       0
5107       0
5108       0
5109       0

[5110 rows x 12 columns]
```

In [38]:

```
#calculate corralation coefficient
# Calculate the correlation matrix
correlation_matrix = dt.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                        id    gender       age  hypertension  heart_disease  \
id                1.000000  0.002511  0.003509      0.003550      -0.001296
gender            0.002511  1.000000 -0.028138      0.020994       0.085447
age               0.003509 -0.028138  1.000000      0.276397       0.263795
hypertension      0.003550  0.020994  0.276397      1.000000       0.108306
heart_disease    -0.001296  0.085447  0.263795      0.108306       1.000000
ever_married      0.013690 -0.031005  0.679122      0.164243       0.114644
work_type        -0.015757  0.056422 -0.361641     -0.051761      -0.028023
Residence_type   -0.001403 -0.006738  0.014208     -0.007913       0.003092
avg_glucose_level 0.008446  0.048880  0.113514      0.100975       0.091237
bmi               0.009494 -0.015869  0.379468      0.158471       0.055152
smoking_status    0.014074 -0.062581  0.265198      0.111038       0.048460
stroke            0.006388  0.008929  0.245244      0.127904       0.134914

                   ever_married  work_type  Residence_type  avg_glucose_level  \
id                     0.013690  -0.015757       -0.001403           0.008446
gender                -0.031005   0.056422       -0.006738           0.048880
age                    0.679122  -0.361641        0.014208           0.113514
hypertension           0.164243  -0.051761       -0.007913           0.100975
heart_disease          0.114644  -0.028023        0.003092           0.091237
ever_married           1.000000  -0.352722        0.006261           0.079507
work_type             -0.352722   1.000000       -0.007316          -0.014416
Residence_type         0.006261  -0.007316        1.000000          -0.018170
avg_glucose_level      0.079507  -0.014416       -0.018170           1.000000
bmi                    0.377890  -0.334163        0.005288           0.087180
smoking_status         0.259647  -0.305927        0.008237           0.033476
stroke                 0.108340  -0.032316        0.015458           0.071601

                       bmi  smoking_status    stroke
id                0.009494        0.014074  0.006388
gender           -0.015869       -0.062581  0.008929
age               0.379468        0.265198  0.245244
hypertension      0.158471        0.111038  0.127904
heart_disease     0.055152        0.048460  0.134914
ever_married      0.377890        0.259647  0.108340
work_type        -0.334163       -0.305927 -0.032316
Residence_type    0.005288        0.008237  0.015458
avg_glucose_level 0.087180        0.033476  0.071601
bmi               1.000000        0.246568  0.048593
smoking_status    0.246568        1.000000  0.028123
stroke            0.048593        0.028123  1.000000
```

In [39]:
```
# Plot the correlation matrix as a heatmap
plt.figure(figsize=(8, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", annot_kws={
plt.title('Correlation Matrix')
plt.show()
```



In [38]:
```
df = pd.read_excel("C:\\Users\\USER\\Desktop\\AI.ICA\\Health\\health care dataset.x
```

In [39]:
```
print(df)
```

```
         id  gender   age  hypertension  heart_disease ever_married  \
0      9046    Male  67.0             0              1          Yes
1     51676  Female  61.0             0              0          Yes
2     31112    Male  80.0             0              1          Yes
3     60182  Female  49.0             0              0          Yes
4      1665  Female  79.0             1              0          Yes
...     ...     ...   ...           ...            ...          ...
5105  18234  Female  80.0             1              0          Yes
5106  44873  Female  81.0             0              0          Yes
5107  19723  Female  35.0             0              0          Yes
5108  37544    Male  51.0             0              0          Yes
5109  44679  Female  44.0             0              0          Yes

          work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0           Private          Urban             228.69  36.6  formerly smoked
1     Self-employed          Rural             202.21   NaN     never smoked
2           Private          Rural             105.92  32.5     never smoked
3           Private          Urban             171.23  34.4           smokes
4     Self-employed          Rural             174.12  24.0     never smoked
...             ...            ...                ...   ...              ...
5105        Private          Urban              83.75   NaN     never smoked
5106  Self-employed          Urban             125.20  40.0     never smoked
5107  Self-employed          Rural              82.99  30.6     never smoked
5108        Private          Rural             166.29  25.6  formerly smoked
5109       Govt_job          Urban              85.28  26.2          Unknown

      stroke
0          1
1          1
2          1
3          1
4          1
...      ...
5105       0
5106       0
5107       0
5108       0
5109       0

[5110 rows x 12 columns]
```

In [40]:

```
#check datatype
print(df.dtypes)
```

```
id                      int64
gender                 object
age                   float64
hypertension            int64
heart_disease           int64
ever_married           object
work_type              object
Residence_type         object
avg_glucose_level     float64
bmi                   float64
smoking_status         object
stroke                  int64
dtype: object
```

In [41]:

```
#age can not be float
#round it up
import pandas as pd



# Round the values in the 'age' column to the nearest integer
df['age'] = df['age'].round()

# Convert 'age' column to integer type
df['age'] = df['age'].astype(int)

# Display the DataFrame with rounded 'age' values
print(df)
```

```
         id  gender  age  hypertension  heart_disease ever_married  \
0      9046    Male   67             0              1          Yes
1     51676  Female   61             0              0          Yes
2     31112    Male   80             0              1          Yes
3     60182  Female   49             0              0          Yes
4      1665  Female   79             1              0          Yes
...     ...     ...  ...           ...            ...          ...
5105  18234  Female   80             1              0          Yes
5106  44873  Female   81             0              0          Yes
5107  19723  Female   35             0              0          Yes
5108  37544    Male   51             0              0          Yes
5109  44679  Female   44             0              0          Yes

          work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0           Private          Urban             228.69  36.6  formerly smoked
1     Self-employed          Rural             202.21   NaN     never smoked
2           Private          Rural             105.92  32.5     never smoked
3           Private          Urban             171.23  34.4           smokes
4     Self-employed          Rural             174.12  24.0     never smoked
...             ...            ...                ...   ...              ...
5105        Private          Urban              83.75   NaN     never smoked
5106  Self-employed          Urban             125.20  40.0     never smoked
5107  Self-employed          Rural              82.99  30.6     never smoked
5108        Private          Rural             166.29  25.6  formerly smoked
5109       Govt_job          Urban              85.28  26.2          Unknown

      stroke
0          1
1          1
2          1
3          1
4          1
...      ...
5105       0
5106       0
5107       0
5108       0
5109       0

[5110 rows x 12 columns]
```

In [42]:

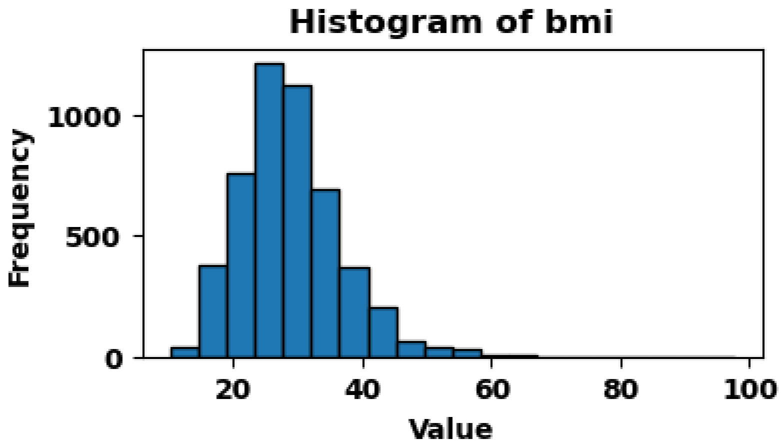
```
# confirm if age is in int.
print(df.dtypes)
```

```
id                    int64
gender               object
age                   int32
hypertension          int64
heart_disease         int64
ever_married         object
work_type            object
Residence_type       object
avg_glucose_level   float64
bmi                 float64
smoking_status       object
stroke                int64
dtype: object
```

In [43]:
```
#check for missing values
import pandas as pd


# Check for missing values using isna() or isnull()
missing_values = df.isna().sum()

# Display the count of missing values for each column
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

In [44]:
```
#handle missing values in bmi column
#bim is continuous
#check its distribution using histogram.
import matplotlib.pyplot as plt


plt.figure(figsize=(4, 2))
plt.hist(df['bmi'], bins=20, edgecolor='k')  # Adjust the number of bins as needed
plt.title('Histogram of bmi')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



In [45]:
```
#bmi is not normally distributed. so replace missing values with median
#view bim statistical summary
summary_stats = df['bmi'].describe()
print(summary_stats)
```

```
count    4909.000000
mean       28.893237
std         7.854067
min        10.300000
25%        23.500000
50%        28.100000
75%        33.100000
max        97.600000
Name: bmi, dtype: float64
```

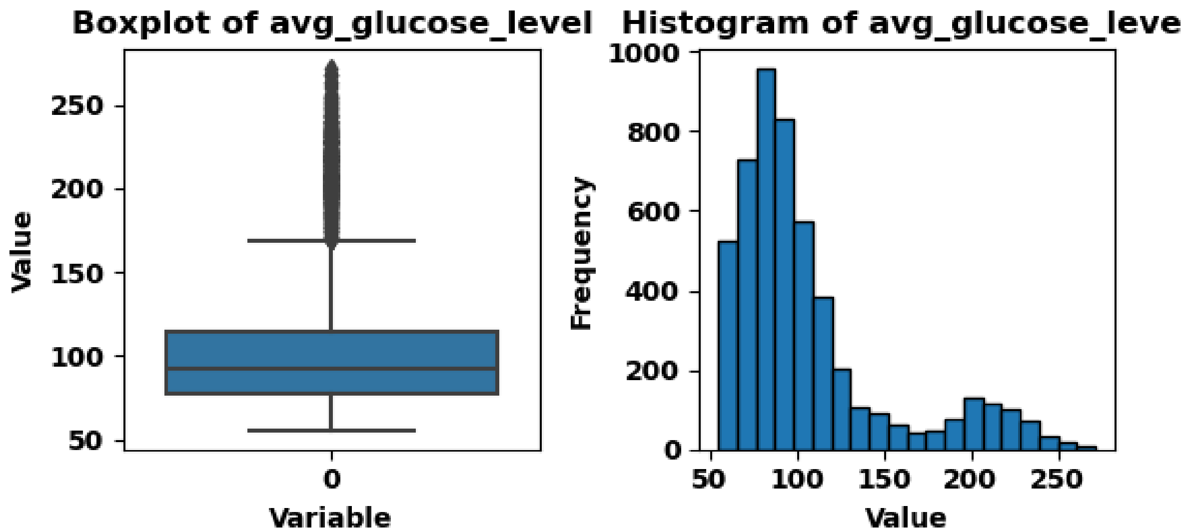
In [46]:
```
#replace missing values
median_bmi = df['bmi'].median()

# Replace missing values with the median
df['bmi'].fillna(median_bmi, inplace=True)
```

In [47]:
```
#check if the missing values have been reolaced
missing_values = df.isna().sum()

# Display the count of missing values for each column
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

In [48]:

```
#examine the numerical variable for outliers using box plot and histogram
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=df['avg_glucose_level'], ax=axs[0])
axs[0].set_title('Boxplot of avg_glucose_level')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(df['avg_glucose_level'], bins=20, edgecolor='k')  # Adjust the number o
axs[1].set_title('Histogram of avg_glucose_level')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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

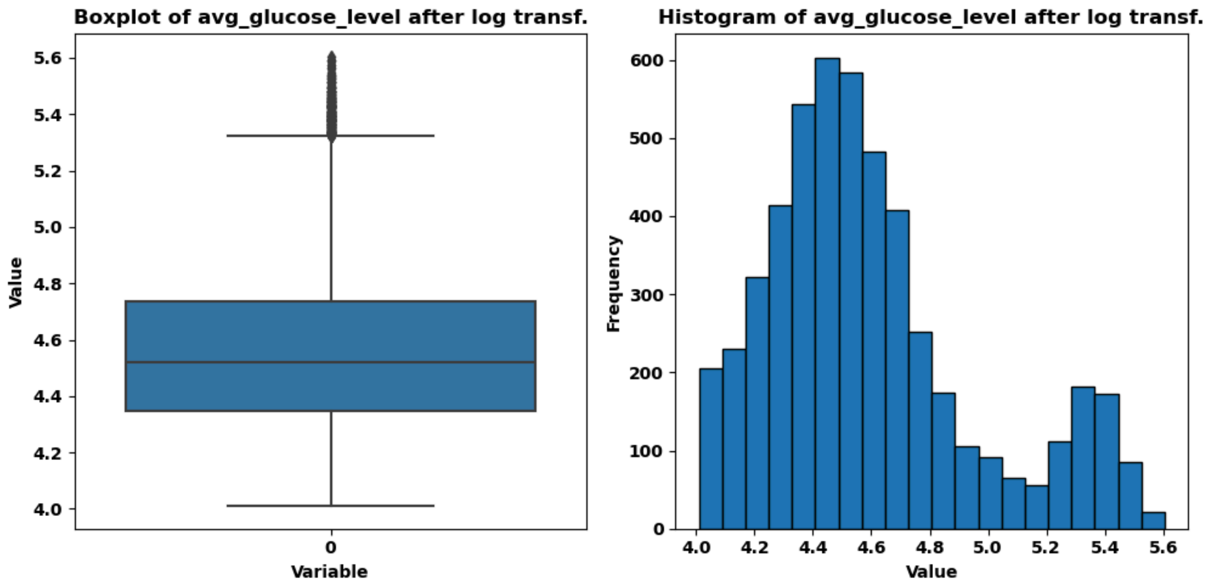


In [49]:
```
#handle outliers in avg_glucose_level column
#transform it using log transformation method
import numpy as np

# Replace zero values with a small value (e.g., 1) to avoid errors in logarithm cal
df['avg_glucose_level'] = df['avg_glucose_level'].replace(0, 1)

# Apply log transformation to 'avg_glucose_level' column
df['avg_glucose_level'] = np.log(df['avg_glucose_level'])

# Display the transformed column
print(df['avg_glucose_level'])
```

```
0       5.432367
1       5.309307
2       4.662684
3       5.143008
4       5.159745
          ...
5105    4.427836
5106    4.829912
5107    4.418720
5108    5.113733
5109    4.445940
Name: avg_glucose_level, Length: 5110, dtype: float64
```

In [50]:
```
#examine the transformation using charts
# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(10, 5))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=df['avg_glucose_level'], ax=axs[0])
axs[0].set_title('Boxplot of avg_glucose_level after log transf.')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(df['avg_glucose_level'], bins=20, edgecolor='k')  # Adjust the number o
axs[1].set_title('Histogram of avg_glucose_level after log transf.')
```

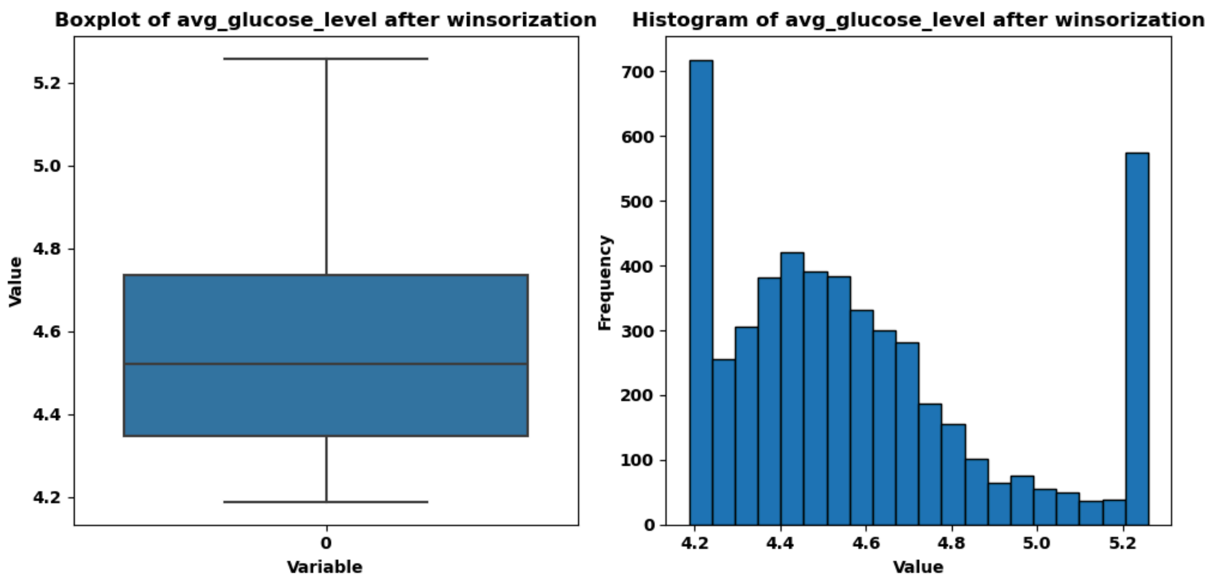
```
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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



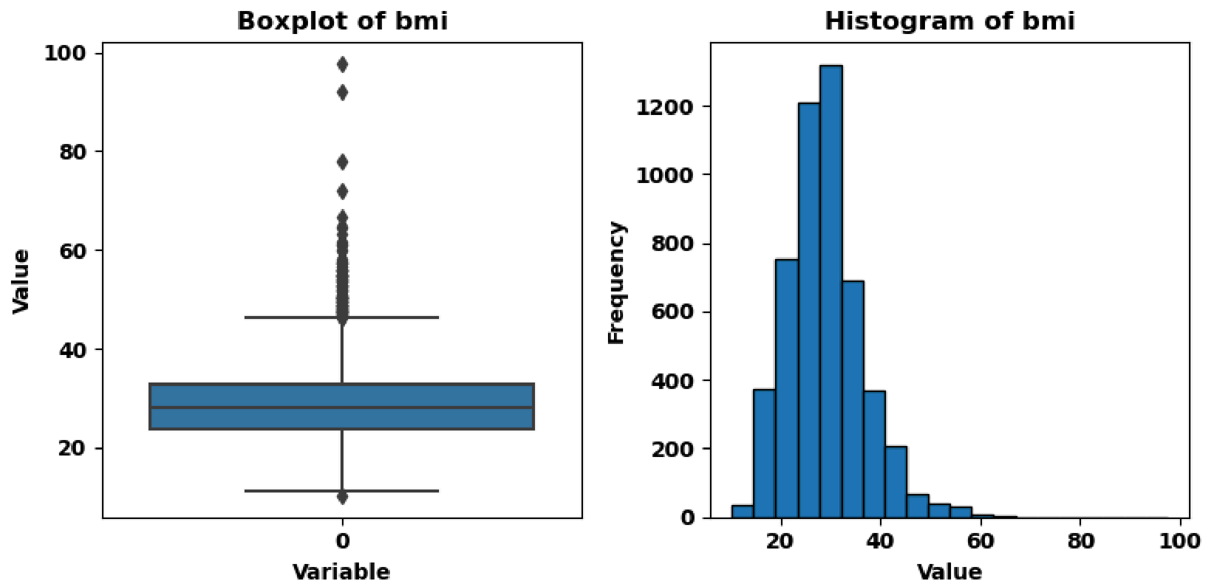
In [51]:
```
#handle the remainig outliers using winsorization methed
import numpy as np

# Assuming dt is your DataFrame and 'Price' is the variable of interest

# Adjust the percentile thresholds
lower_threshold = np.percentile(df['avg_glucose_level'],10)  # Lowering to 10th per
upper_threshold = np.percentile(df['avg_glucose_level'], 90)  # Raising to 90th per

# Replace outliers with threshold values
df['avg_glucose_level'] = np.where(df['avg_glucose_level'] < lower_threshold, lower
df['avg_glucose_level'] = np.where(df['avg_glucose_level'] > upper_threshold, upper
```

In [52]:
```
# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(10, 5))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=df['avg_glucose_level'], ax=axs[0])
axs[0].set_title('Boxplot of avg_glucose_level after winsorization')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(df['avg_glucose_level'], bins=20, edgecolor='k')  # Adjust the number o
axs[1].set_title('Histogram of avg_glucose_level after winsorization')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')
```

```
# Adjust layout to prevent overlap
plt.tight_layout()

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

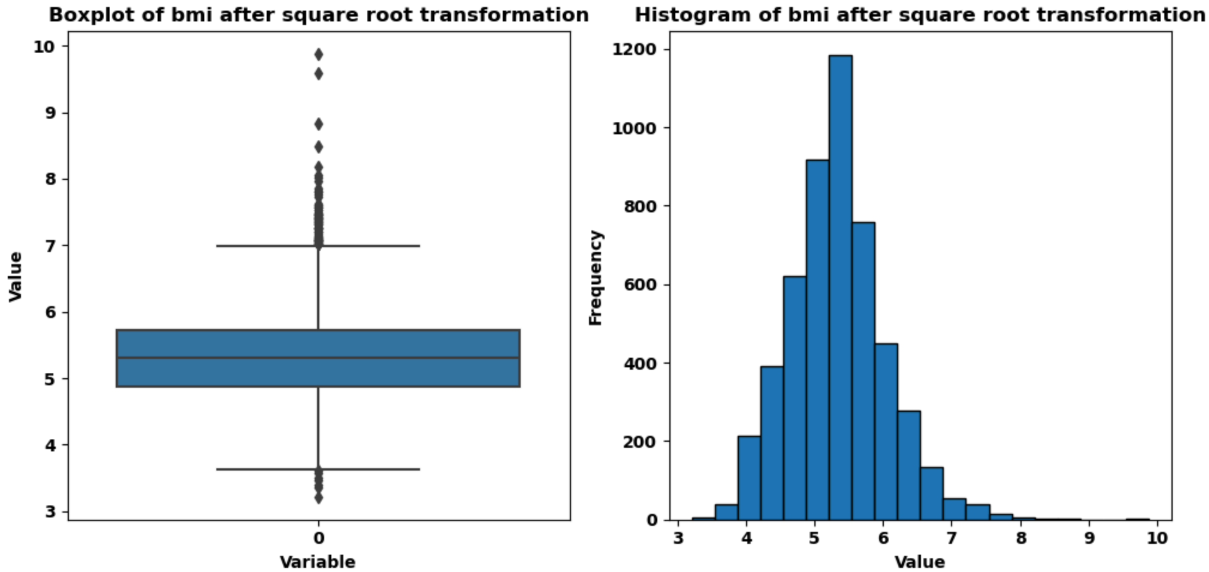


In [53]:
```
#handle outliers in bim column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=df['bmi'], ax=axs[0])
axs[0].set_title('Boxplot of bmi')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(df['bmi'], bins=20, edgecolor='k')  # Adjust the number of bins as need
axs[1].set_title('Histogram of bmi')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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



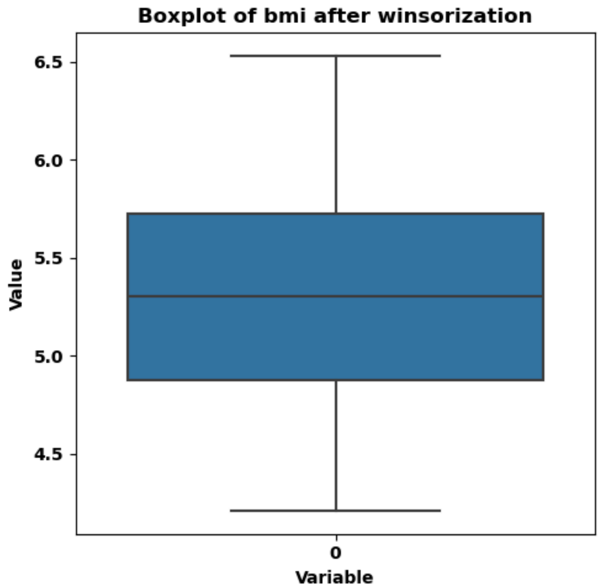
In [54]:
```
#handle outliers in bmi column
#first, transform it using square root method
import numpy as np


df['bmi'] = np.sqrt(df['bmi'])
```

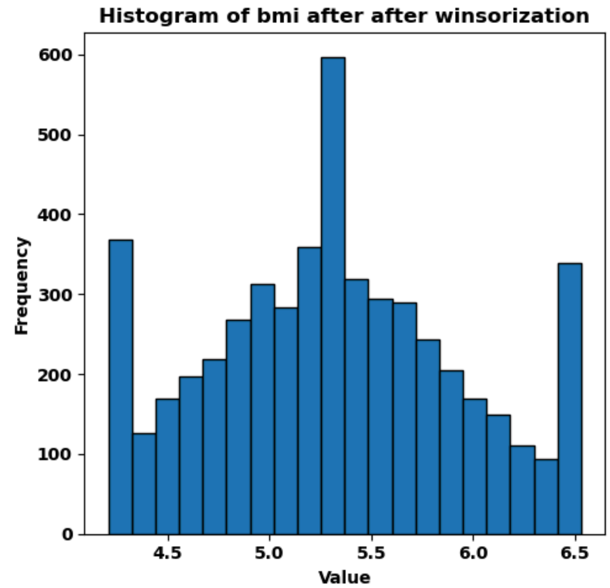
In [55]:
```
#use charts to examine the transformation
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(10, 5))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=df['bmi'], ax=axs[0])
axs[0].set_title('Boxplot of bmi after square root transformation')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(df['bmi'], bins=20, edgecolor='k')  # Adjust the number of bins as need
axs[1].set_title('Histogram of bmi after square root transformation')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

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



In [56]:
```
#handle the outliers using winsorization method
import numpy as np


# Adjust the percentile thresholds
lower_threshold = np.percentile(df['bmi'],5)  # Lowering to 0.5th percentile
upper_threshold = np.percentile(df['bmi'],95)  # Raising to 99.5th percentile

# Replace outliers with threshold values
df['bmi'] = np.where(df['bmi'] < lower_threshold, lower_threshold, df['bmi'])
df['bmi'] = np.where(df['bmi'] > upper_threshold, upper_threshold, df['bmi'])
```

In [57]:
```
#use chart to view the effectvof the winsorzing
#use charts to examine the transformation
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(10, 5))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=df['bmi'], ax=axs[0])
axs[0].set_title('Boxplot of bmi after winsorization')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(df['bmi'], bins=20, edgecolor='k')  # Adjust the number of bins as need
axs[1].set_title('Histogram of bmi after after winsorization')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()
```





In [58]:
```
# Feature engineering analysis
print(df)
```

```
         id  gender  age  hypertension  heart_disease ever_married  \
0      9046    Male   67             0              1          Yes
1     51676  Female   61             0              0          Yes
2     31112    Male   80             0              1          Yes
3     60182  Female   49             0              0          Yes
4      1665  Female   79             1              0          Yes
...     ...     ...  ...           ...            ...          ...
5105  18234  Female   80             1              0          Yes
5106  44873  Female   81             0              0          Yes
5107  19723  Female   35             0              0          Yes
5108  37544    Male   51             0              0          Yes
5109  44679  Female   44             0              0          Yes

          work_type Residence_type  avg_glucose_level       bmi  \
0           Private          Urban           5.258438  6.049793
1     Self-employed          Rural           5.258438  5.300943
2           Private          Rural           4.662684  5.700877
3           Private          Urban           5.143008  5.865151
4     Self-employed          Rural           5.159745  4.898979
...             ...            ...                ...       ...
5105        Private          Urban           4.427836  5.300943
5106  Self-employed          Urban           4.829912  6.324555
5107  Self-employed          Rural           4.418720  5.531727
5108        Private          Rural           5.113733  5.059644
5109       Govt_job          Urban           4.445940  5.118594

       smoking_status  stroke
0     formerly smoked       1
1        never smoked       1
2        never smoked       1
3              smokes       1
4        never smoked       1
...               ...     ...
5105     never smoked       0
5106     never smoked       0
5107     never smoked       0
5108  formerly smoked       0
5109          Unknown       0

[5110 rows x 12 columns]
```

In [59]:

```
#avg_glucose_level and bmi columns are continuous variables
# normalize them using min-max method
# Find minimum and maximum values of 'avg_glucose_level' column
min_val = df['avg_glucose_level'].min()
max_val = df['avg_glucose_level'].max()

# Normalize 'avg_glucose_level' column using Min-Max scaling
df['avg_glucose_level'] = (df['avg_glucose_level'] - min_val) / (max_val - min_val)

# Drop any rows with NaN values in the original 'avg_glucose_level' column
df.dropna(subset=['avg_glucose_level'], inplace=True)

# Display the DataFrame with normalized 'avg_glucose_level'
print(df)
```

```
         id  gender  age  hypertension  heart_disease ever_married  \
0      9046    Male   67             0              1          Yes
1     51676  Female   61             0              0          Yes
2     31112    Male   80             0              1          Yes
3     60182  Female   49             0              0          Yes
4      1665  Female   79             1              0          Yes
...     ...     ...  ...           ...            ...          ...
5105  18234  Female   80             1              0          Yes
5106  44873  Female   81             0              0          Yes
5107  19723  Female   35             0              0          Yes
5108  37544    Male   51             0              0          Yes
5109  44679  Female   44             0              0          Yes

          work_type Residence_type  avg_glucose_level       bmi  \
0           Private          Urban           1.000000  6.049793
1     Self-employed          Rural           1.000000  5.300943
2           Private          Rural           0.444252  5.700877
3           Private          Urban           0.892321  5.865151
4     Self-employed          Rural           0.907934  4.898979
...             ...            ...                ...       ...
5105        Private          Urban           0.225174  5.300943
5106  Self-employed          Urban           0.600251  6.324555
5107  Self-employed          Rural           0.216670  5.531727
5108        Private          Rural           0.865013  5.059644
5109       Govt_job          Urban           0.242062  5.118594

        smoking_status  stroke
0     formerly smoked       1
1        never smoked       1
2        never smoked       1
3              smokes       1
4        never smoked       1
...               ...     ...
5105     never smoked       0
5106     never smoked       0
5107     never smoked       0
5108  formerly smoked       0
5109          Unknown       0

[5110 rows x 12 columns]
```

In [60]:
```
# now normalize bmi column
# Find minimum and maximum values of 'avg_glucose_level' column
min_val = df['bmi'].min()
max_val = df['bmi'].max()

# Normalize 'avg_glucose_level' column using Min-Max scaling
df['bmi'] = (df['bmi'] - min_val) / (max_val - min_val)

# Drop any rows with NaN values in the original 'avg_glucose_level' column
df.dropna(subset=['bmi'], inplace=True)

# Display the DataFrame with normalized 'avg_glucose_level'
print(df)
```

```
        id  gender  age  hypertension  heart_disease ever_married  \
0      9046    Male   67             0              1          Yes   
1     51676  Female   61             0              0          Yes   
2     31112    Male   80             0              1          Yes   
3     60182  Female   49             0              0          Yes   
4      1665  Female   79             1              0          Yes   
...     ...     ...  ...           ...            ...          ...   
5105  18234  Female   80             1              0          Yes   
5106  44873  Female   81             0              0          Yes   
5107  19723  Female   35             0              0          Yes   
5108  37544    Male   51             0              0          Yes   
5109  44679  Female   44             0              0          Yes   

          work_type Residence_type  avg_glucose_level       bmi  \
0           Private          Urban           1.000000  0.792901   
1     Self-employed          Rural           1.000000  0.470669   
2           Private          Rural           0.444252  0.642762   
3           Private          Urban           0.892321  0.713449   
4     Self-employed          Rural           0.907934  0.297702   
...             ...            ...                ...       ...   
5105        Private          Urban           0.225174  0.470669   
5106  Self-employed          Urban           0.600251  0.911132   
5107  Self-employed          Rural           0.216670  0.569976   
5108        Private          Rural           0.865013  0.366837   
5109       Govt_job          Urban           0.242062  0.392203   

       smoking_status  stroke  
0     formerly smoked       1  
1        never smoked       1  
2        never smoked       1  
3              smokes       1  
4        never smoked       1  
...               ...     ...  
5105     never smoked       0  
5106     never smoked       0  
5107     never smoked       0  
5108  formerly smoked       0  
5109          Unknown       0  

[5110 rows x 12 columns]
```

In [61]:

```python
# Encoding
import pandas as pd



# One-hot encode the 'gender' column with specified categories
df = pd.get_dummies(df, columns=['gender'], drop_first=False, prefix='gender')

# Replace True and False with 1 and 0
df.replace({True: 1, False: 0}, inplace=True)

# Display the DataFrame with one-hot encoded 'gender' column
print(df)
```

```
         id  age  hypertension  heart_disease ever_married      work_type  \
0      9046   67             0              1          Yes        Private   
1     51676   61             0              0          Yes  Self-employed   
2     31112   80             0              1          Yes        Private   
3     60182   49             0              0          Yes        Private   
4      1665   79             1              0          Yes  Self-employed   
...     ...  ...           ...            ...          ...            ...   
5105  18234   80             1              0          Yes        Private   
5106  44873   81             0              0          Yes  Self-employed   
5107  19723   35             0              0          Yes  Self-employed   
5108  37544   51             0              0          Yes        Private   
5109  44679   44             0              0          Yes       Govt_job   

     Residence_type  avg_glucose_level       bmi   smoking_status  stroke  \
0             Urban           1.000000  0.792901  formerly smoked       1   
1             Rural           1.000000  0.470669     never smoked       1   
2             Rural           0.444252  0.642762     never smoked       1   
3             Urban           0.892321  0.713449           smokes       1   
4             Rural           0.907934  0.297702     never smoked       1   
...             ...                ...       ...              ...     ...   
5105          Urban           0.225174  0.470669     never smoked       0   
5106          Urban           0.600251  0.911132     never smoked       0   
5107          Rural           0.216670  0.569976     never smoked       0   
5108          Rural           0.865013  0.366837  formerly smoked       0   
5109          Urban           0.242062  0.392203          Unknown       0   

      gender_Female  gender_Male  gender_Other  
0                 0            1             0  
1                 1            0             0  
2                 0            1             0  
3                 1            0             0  
4                 1            0             0  
...             ...          ...           ...  
5105              1            0             0  
5106              1            0             0  
5107              1            0             0  
5108              0            1             0  
5109              1            0             0  

[5110 rows x 14 columns]
```

In [62]:

```python
# Convert 'Yes' and 'No' entries in ever_married column to 1s and 0s
df['ever_married'] = df['ever_married'].map({'Yes': 1, 'No': 0})

# Display the DataFrame
print(df)
```

```
        id  age  hypertension  heart_disease  ever_married      work_type  \
0     9046   67             0              1             1        Private   
1    51676   61             0              0             1  Self-employed   
2    31112   80             0              1             1        Private   
3    60182   49             0              0             1        Private   
4     1665   79             1              0             1  Self-employed   
...    ...  ...           ...            ...           ...            ...   
5105 18234   80             1              0             1        Private   
5106 44873   81             0              0             1  Self-employed   
5107 19723   35             0              0             1  Self-employed   
5108 37544   51             0              0             1        Private   
5109 44679   44             0              0             1       Govt_job   

     Residence_type  avg_glucose_level       bmi   smoking_status  stroke  \
0             Urban           1.000000  0.792901  formerly smoked       1   
1             Rural           1.000000  0.470669     never smoked       1   
2             Rural           0.444252  0.642762     never smoked       1   
3             Urban           0.892321  0.713449           smokes       1   
4             Rural           0.907934  0.297702     never smoked       1   
...             ...                ...       ...              ...     ...   
5105          Urban           0.225174  0.470669     never smoked       0   
5106          Urban           0.600251  0.911132     never smoked       0   
5107          Rural           0.216670  0.569976     never smoked       0   
5108          Rural           0.865013  0.366837  formerly smoked       0   
5109          Urban           0.242062  0.392203          Unknown       0   

      gender_Female  gender_Male  gender_Other  
0                 0            1             0  
1                 1            0             0  
2                 0            1             0  
3                 1            0             0  
4                 1            0             0  
...             ...          ...           ...  
5105              1            0             0  
5106              1            0             0  
5107              1            0             0  
5108              0            1             0  
5109              1            0             0  

[5110 rows x 14 columns]
```

In [63]:
```python
# Get the distinct entries in the 'gender' column
distinct_work_type = df['work_type'].unique()

# Display the distinct entries
print(distinct_work_type)
```

```
['Private' 'Self-employed' 'Govt_job' 'children' 'Never_worked']
```

In [64]:
```python
#hot encode them
import pandas as pd



# One-hot encode the 'work_type' column with specified categories
df = pd.get_dummies(df, columns=['work_type'], drop_first=False, prefix='work_type'
```

```
# Replace True and False with 1 and 0
df.replace({True: 1, False: 0}, inplace=True)

# Display the DataFrame with one-hot encoded 'gender' column
print(df)
```

```
         id  age  hypertension  heart_disease  ever_married Residence_type  \
0      9046   67             0              1             1          Urban
1     51676   61             0              0             1          Rural
2     31112   80             0              1             1          Rural
3     60182   49             0              0             1          Urban
4      1665   79             1              0             1          Rural
...     ...  ...           ...            ...           ...            ...
5105  18234   80             1              0             1          Urban
5106  44873   81             0              0             1          Urban
5107  19723   35             0              0             1          Rural
5108  37544   51             0              0             1          Rural
5109  44679   44             0              0             1          Urban

      avg_glucose_level       bmi   smoking_status  stroke  gender_Female  \
0              1.000000  0.792901  formerly smoked       1              0
1              1.000000  0.470669     never smoked       1              1
2              0.444252  0.642762     never smoked       1              0
3              0.892321  0.713449           smokes       1              1
4              0.907934  0.297702     never smoked       1              1
...                 ...       ...              ...     ...            ...
5105           0.225174  0.470669     never smoked       0              1
5106           0.600251  0.911132     never smoked       0              1
5107           0.216670  0.569976     never smoked       0              1
5108           0.865013  0.366837  formerly smoked       0              0
5109           0.242062  0.392203          Unknown       0              1

      gender_Male  gender_Other  work_type_Govt_job  work_type_Never_worked  \
0               1             0                   0                       0
1               0             0                   0                       0
2               1             0                   0                       0
3               0             0                   0                       0
4               0             0                   0                       0
...           ...           ...                 ...                     ...
5105            0             0                   0                       0
5106            0             0                   0                       0
5107            0             0                   0                       0
5108            1             0                   0                       0
5109            0             0                   1                       0

      work_type_Private  work_type_Self-employed  work_type_children
0                     1                        0                   0
1                     0                        1                   0
2                     1                        0                   0
3                     1                        0                   0
4                     0                        1                   0
...                 ...                      ...                 ...
5105                  1                        0                   0
5106                  0                        1                   0
5107                  0                        1                   0
5108                  1                        0                   0
5109                  0                        0                   0

[5110 rows x 18 columns]
```

In [65]:
```
# Get the distinct entries in the 'Residence_type' column
distinct_Residence_type = df['Residence_type'].unique()
```

```
# Display the distinct entries
print(distinct_Residence_type)
```

['Urban' 'Rural']

In [66]:

```
#hot econde them
#hot encode them
import pandas as pd



# One-hot encode the 'gender' column with specified categories
df = pd.get_dummies(df, columns=['Residence_type'], drop_first=False, prefix='Resid

# Replace True and False with 1 and 0
df.replace({True: 1, False: 0}, inplace=True)

# Display the DataFrame with one-hot encoded 'gender' column
print(df)
```

```
         id  age  hypertension  heart_disease  ever_married  \
0      9046   67             0              1             1   
1     51676   61             0              0             1   
2     31112   80             0              1             1   
3     60182   49             0              0             1   
4      1665   79             1              0             1   
...     ...  ...           ...            ...           ...   
5105  18234   80             1              0             1   
5106  44873   81             0              0             1   
5107  19723   35             0              0             1   
5108  37544   51             0              0             1   
5109  44679   44             0              0             1   

      avg_glucose_level       bmi   smoking_status  stroke  gender_Female  \
0              1.000000  0.792901  formerly smoked       1              0   
1              1.000000  0.470669     never smoked       1              1   
2              0.444252  0.642762     never smoked       1              0   
3              0.892321  0.713449           smokes       1              1   
4              0.907934  0.297702     never smoked       1              1   
...                 ...       ...              ...     ...            ...   
5105           0.225174  0.470669     never smoked       0              1   
5106           0.600251  0.911132     never smoked       0              1   
5107           0.216670  0.569976     never smoked       0              1   
5108           0.865013  0.366837  formerly smoked       0              0   
5109           0.242062  0.392203          Unknown       0              1   

      gender_Male  gender_Other  work_type_Govt_job  work_type_Never_worked  \
0               1             0                   0                       0   
1               0             0                   0                       0   
2               1             0                   0                       0   
3               0             0                   0                       0   
4               0             0                   0                       0   
...           ...           ...                 ...                     ...   
5105            0             0                   0                       0   
5106            0             0                   0                       0   
5107            0             0                   0                       0   
5108            1             0                   0                       0   
5109            0             0                   1                       0   

      work_type_Private  work_type_Self-employed  work_type_children  \
0                     1                        0                   0   
1                     0                        1                   0   
2                     1                        0                   0   
3                     1                        0                   0   
4                     0                        1                   0   
...                 ...                      ...                 ...   
5105                  1                        0                   0   
5106                  0                        1                   0   
5107                  0                        1                   0   
5108                  1                        0                   0   
5109                  0                        0                   0   

      Residence_type_Rural  Residence_type_Urban  
0                        0                     1  
1                        1                     0  
2                        1                     0  
```

```
3                     0                    1
4                     1                    0
...                 ...                  ...
5105                  0                    1
5106                  0                    1
5107                  1                    0
5108                  1                    0
5109                  0                    1

[5110 rows x 19 columns]
```

In [67]:
```
# Get the distinct entries in the 'smoking_status' column
distinct_smoking_status = df['smoking_status'].unique()

# Display the distinct entries
print(distinct_smoking_status)
```

```
['formerly smoked' 'never smoked' 'smokes' 'Unknown']
```

In [68]:
```
#hot encode them
#hot econde them
#hot encode them
import pandas as pd



# One-hot encode the 'gender' column with specified categories
df = pd.get_dummies(df, columns=['smoking_status'], drop_first=False, prefix='smoki

# Replace True and False with 1 and 0
df.replace({True: 1, False: 0}, inplace=True)

# Display the DataFrame with one-hot encoded 'gender' column
print(df)
```

```
         id  age  hypertension  heart_disease  ever_married  \
0      9046   67             0              1             1
1     51676   61             0              0             1
2     31112   80             0              1             1
3     60182   49             0              0             1
4      1665   79             1              0             1
...     ...  ...           ...            ...           ...
5105  18234   80             1              0             1
5106  44873   81             0              0             1
5107  19723   35             0              0             1
5108  37544   51             0              0             1
5109  44679   44             0              0             1

      avg_glucose_level       bmi  stroke  gender_Female  gender_Male  ...  \
0              1.000000  0.792901       1              0            1  ...
1              1.000000  0.470669       1              1            0  ...
2              0.444252  0.642762       1              0            1  ...
3              0.892321  0.713449       1              1            0  ...
4              0.907934  0.297702       1              1            0  ...
...                 ...       ...     ...            ...          ...  ...
5105           0.225174  0.470669       0              1            0  ...
5106           0.600251  0.911132       0              1            0  ...
5107           0.216670  0.569976       0              1            0  ...
5108           0.865013  0.366837       0              0            1  ...
5109           0.242062  0.392203       0              1            0  ...

      work_type_Never_worked  work_type_Private  work_type_Self-employed  \
0                          0                  1                        0
1                          0                  0                        1
2                          0                  1                        0
3                          0                  1                        0
4                          0                  0                        1
...                      ...                ...                      ...
5105                       0                  1                        0
5106                       0                  0                        1
5107                       0                  0                        1
5108                       0                  1                        0
5109                       0                  0                        0

      work_type_children  Residence_type_Rural  Residence_type_Urban  \
0                      0                     0                     1
1                      0                     1                     0
2                      0                     1                     0
3                      0                     0                     1
4                      0                     1                     0
...                  ...                   ...                   ...
5105                   0                     0                     1
5106                   0                     0                     1
5107                   0                     1                     0
5108                   0                     1                     0
5109                   0                     0                     1

      smoking_status_Unknown  smoking_status_formerly smoked  \
0                          0                               1
1                          0                               0
2                          0                               0
```

```
3                          0                               0
4                          0                               0
...                      ...                             ...
5105                       0                               0
5106                       0                               0
5107                       0                               0
5108                       0                               1
5109                       1                               0

      smoking_status_never smoked  smoking_status_smokes
0                               0                      0
1                               1                      0
2                               1                      0
3                               0                      1
4                               1                      0
...                           ...                    ...
5105                            1                      0
5106                            1                      0
5107                            1                      0
5108                            0                      0
5109                            0                      0

[5110 rows x 22 columns]
```

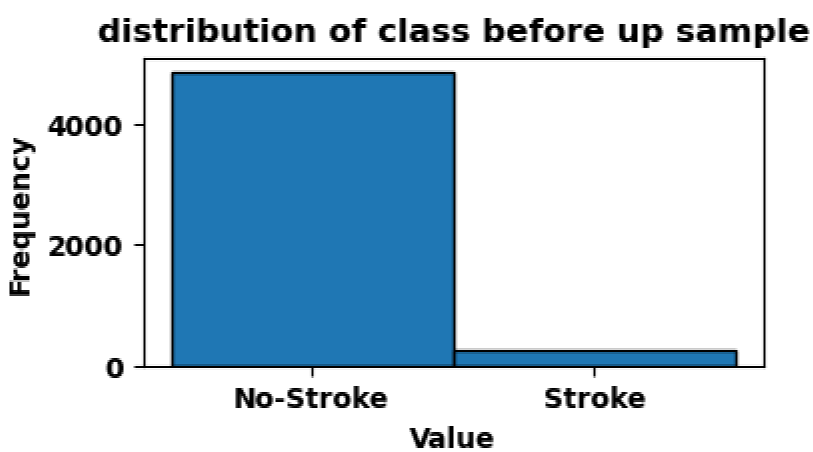
In [69]:

```python
# drop id column for it does not contribute to the analysis
import pandas as pd


# Drop the 'id' column
df = df.drop('id', axis=1)

print(df)
```

```
      age  hypertension  heart_disease  ever_married  avg_glucose_level  \
0      67             0              1             1           1.000000
1      61             0              0             1           1.000000
2      80             0              1             1           0.444252
3      49             0              0             1           0.892321
4      79             1              0             1           0.907934
...   ...           ...            ...           ...                ...
5105   80             1              0             1           0.225174
5106   81             0              0             1           0.600251
5107   35             0              0             1           0.216670
5108   51             0              0             1           0.865013
5109   44             0              0             1           0.242062

           bmi  stroke  gender_Female  gender_Male  gender_Other  ...  \
0     0.792901       1              0            1             0  ...
1     0.470669       1              1            0             0  ...
2     0.642762       1              0            1             0  ...
3     0.713449       1              1            0             0  ...
4     0.297702       1              1            0             0  ...
...        ...     ...            ...          ...           ...  ...
5105  0.470669       0              1            0             0  ...
5106  0.911132       0              1            0             0  ...
5107  0.569976       0              1            0             0  ...
5108  0.366837       0              0            1             0  ...
5109  0.392203       0              1            0             0  ...

      work_type_Never_worked  work_type_Private  work_type_Self-employed  \
0                          0                  1                        0
1                          0                  0                        1
2                          0                  1                        0
3                          0                  1                        0
4                          0                  0                        1
...                      ...                ...                      ...
5105                       0                  1                        0
5106                       0                  0                        1
5107                       0                  0                        1
5108                       0                  1                        0
5109                       0                  0                        0

      work_type_children  Residence_type_Rural  Residence_type_Urban  \
0                      0                     0                     1
1                      0                     1                     0
2                      0                     1                     0
3                      0                     0                     1
4                      0                     1                     0
...                  ...                   ...                   ...
5105                   0                     0                     1
5106                   0                     0                     1
5107                   0                     1                     0
5108                   0                     1                     0
5109                   0                     0                     1

      smoking_status_Unknown  smoking_status_formerly smoked  \
0                          0                               1
1                          0                               0
2                          0                               0
```

```
3                          0                               0
4                          0                               0
...                      ...                             ...
5105                       0                               0
5106                       0                               0
5107                       0                               0
5108                       0                               1
5109                       1                               0

      smoking_status_never smoked  smoking_status_smokes
0                               0                      0
1                               1                      0
2                               1                      0
3                               0                      1
4                               1                      0
...                           ...                    ...
5105                            1                      0
5106                            1                      0
5107                            1                      0
5108                            0                      0
5109                            0                      0

[5110 rows x 21 columns]
```

In [70]:
```
#upsample the target variable column
#check its distribution using histogram.
import matplotlib.pyplot as plt


plt.figure(figsize=(4, 2))
plt.hist(df['stroke'], bins=2, edgecolor='k')  # Adjust the number of bins as neede
plt.title('distribution of class before up sample')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.xticks([0.25, 0.75], ['No-Stroke', 'Stroke'])
plt.show()
```



In [71]:
```
import pandas as pd


# View distinct entries of the 'stroke' column and their count
```

```
stroke_counts = df['stroke'].value_counts()

# Display the result
print(stroke_counts)
```

```
stroke
0    4861
1     249
Name: count, dtype: int64
```

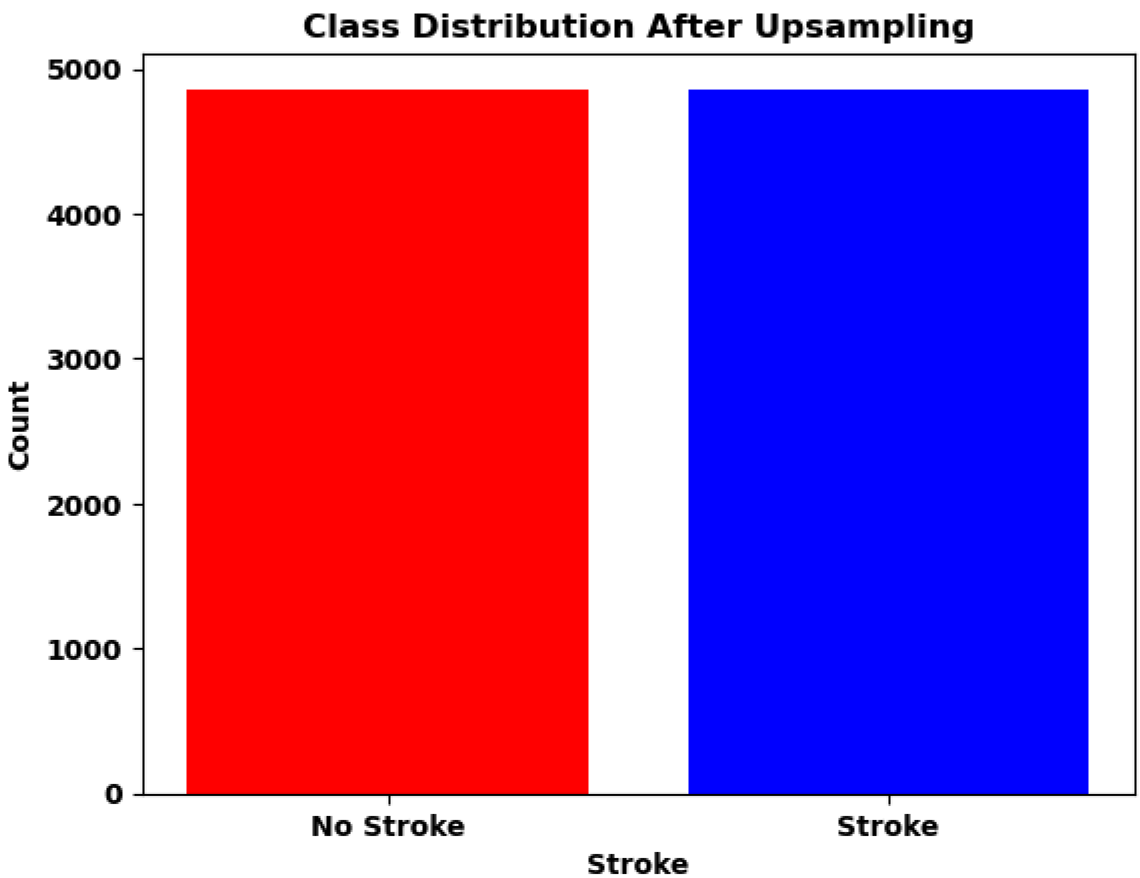
In [72]:
```
# up sample the target variable
from imblearn.over_sampling import SMOTE

# Separate features and target variable
X = df.drop(columns=['stroke'])
y = df['stroke']

# Instantiate SMOTE
smote = SMOTE()

# Upsample the minority class
X_resampled, y_resampled = smote.fit_resample(X, y)
```

In [73]:
```
import pandas as pd
from collections import Counter

# Convert the resampled data to a DataFrame
df_resampled = pd.DataFrame(X_resampled, columns=X.columns)
df_resampled['stroke'] = y_resampled

# Check class distribution before and after upsampling
print("Class distribution before upsampling:", Counter(y))
print("Class distribution after upsampling:", Counter(y_resampled))

# Alternatively, you can directly print the value counts of the 'stroke' column
print("Class distribution after upsampling:")
print(df_resampled['stroke'].value_counts())
```

```
Class distribution before upsampling: Counter({0: 4861, 1: 249})
Class distribution after upsampling: Counter({1: 4861, 0: 4861})
Class distribution after upsampling:
stroke
1    4861
0    4861
Name: count, dtype: int64
```

In [74]:
```
import matplotlib.pyplot as plt

# Data
classes = [1, 0]
counts = [4861, 4861]

# Define labels for the bins
labels = ['Stroke', 'No Stroke']

# Plot
plt.bar(classes, counts, color=['blue', 'red'])
```

```
# Add labels and title
plt.xlabel('Stroke')
plt.ylabel('Count')
plt.title('Class Distribution After Upsampling')

# Set x-axis tick labels
plt.xticks(classes, labels)

# Show plot
plt.show()
```



In [75]:
```
print(df_resampled)
```

```
      age  hypertension  heart_disease  ever_married  avg_glucose_level  \
0      67             0              1             1           1.000000
1      61             0              0             1           1.000000
2      80             0              1             1           0.444252
3      49             0              0             1           0.892321
4      79             1              0             1           0.907934
...   ...           ...            ...           ...                ...
9717   61             0              1             1           0.712481
9718   78             1              0             1           0.310189
9719   38             0              0             1           0.369671
9720   76             0              0             1           0.572216
9721   72             1              0             1           0.999291

           bmi  gender_Female  gender_Male  gender_Other  work_type_Govt_job  \
0     0.792901              0            1             0                   0
1     0.470669              1            0             0                   0
2     0.642762              0            1             0                   0
3     0.713449              1            0             0                   0
4     0.297702              1            0             0                   0
...        ...            ...          ...           ...                 ...
9717  0.671560              0            1             0                   0
9718  0.470669              0            1             0                   0
9719  0.505799              1            0             0                   0
9720  0.470669              1            0             0                   0
9721  0.563799              1            0             0                   0

      ...  work_type_Private  work_type_Self-employed  work_type_children  \
0     ...                  1                        0                   0
1     ...                  0                        1                   0
2     ...                  1                        0                   0
3     ...                  1                        0                   0
4     ...                  0                        1                   0
...   ...                ...                      ...                 ...
9717  ...                  1                        0                   0
9718  ...                  0                        0                   0
9719  ...                  1                        0                   0
9720  ...                  0                        1                   0
9721  ...                  1                        0                   0

      Residence_type_Rural  Residence_type_Urban  smoking_status_Unknown  \
0                        0                     1                       0
1                        1                     0                       0
2                        1                     0                       0
3                        0                     1                       0
4                        1                     0                       0
...                    ...                   ...                     ...
9717                     0                     1                       0
9718                     0                     1                       0
9719                     1                     0                       0
9720                     0                     1                       0
9721                     0                     0                       0

      smoking_status_formerly smoked  smoking_status_never smoked  \
0                                  1                            0
1                                  0                            1
2                                  0                            1
```

```
3                                  0                             0
4                                  0                             1
...                              ...                           ...
9717                               0                             0
9718                               1                             0
9719                               0                             0
9720                               0                             0
9721                               0                             1

      smoking_status_smokes  stroke
0                         0       1
1                         0       1
2                         0       1
3                         1       1
4                         0       1
...                     ...     ...
9717                      0       1
9718                      0       1
9719                      0       1
9720                      0       1
9721                      0       1

[9722 rows x 21 columns]
```

In [76]:

```
import pandas as pd

# Create a DataFrame with the upsampled data
df_resampled = pd.DataFrame(X_resampled, columns=X.columns)
df_resampled['stroke'] = y_resampled

# Specify the file path where you want to save the new CSV file
file_path = 'resampled_data.csv'

# Save the DataFrame to a new CSV file
df_resampled.to_csv(file_path, index=False)

print("Data saved to", file_path)
```

```
Data saved to resampled_data.csv
```

In [77]:

```
import os

# Get the current working directory
current_directory = os.getcwd()

# Combine the current directory with the file name to get the full path
full_path = os.path.join(current_directory, file_path)

# Print the full path
print("Full path of the saved file:", full_path)
```

```
Full path of the saved file: C:\Users\USER\resampled_data.csv
```

In [1]:

```
import pandas as pd

# Specify the full path of the CSV file
file_path = 'C:\\Users\\USER\\resampled_data.csv'
```

```
# Import the CSV file into a DataFrame
df_imported = pd.read_csv(file_path)

# Display the first few rows of the imported DataFrame
print(df_imported.head())
```

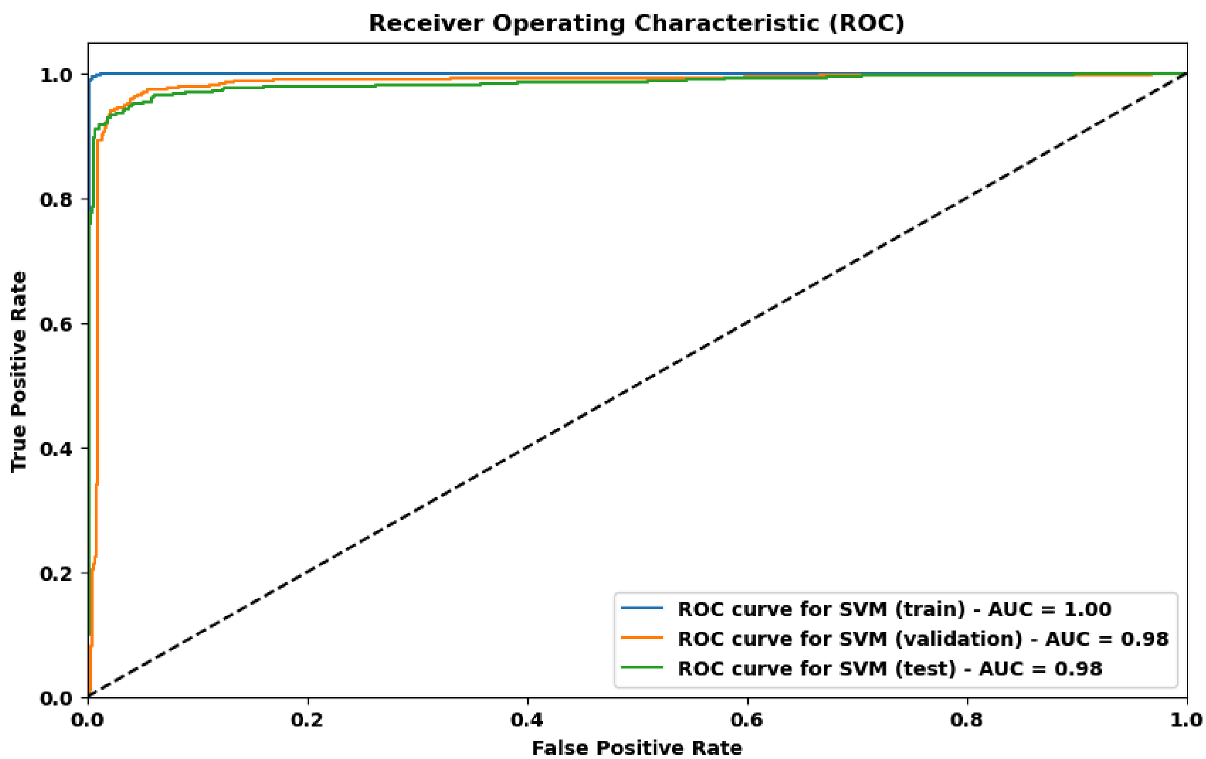
```
   age  hypertension  heart_disease  ever_married  avg_glucose_level  \
0   67             0              1             1           1.000000   
1   61             0              0             1           1.000000   
2   80             0              1             1           0.444252   
3   49             0              0             1           0.892321   
4   79             1              0             1           0.907934   

        bmi  gender_Female  gender_Male  gender_Other  work_type_Govt_job  \
0  0.792901              0            1             0                   0   
1  0.470669              1            0             0                   0   
2  0.642762              0            1             0                   0   
3  0.713449              1            0             0                   0   
4  0.297702              1            0             0                   0   

   ...  work_type_Private  work_type_Self-employed  work_type_children  \
0  ...                  1                        0                   0   
1  ...                  0                        1                   0   
2  ...                  1                        0                   0   
3  ...                  1                        0                   0   
4  ...                  0                        1                   0   

   Residence_type_Rural  Residence_type_Urban  smoking_status_Unknown  \
0                     0                     1                       0   
1                     1                     0                       0   
2                     1                     0                       0   
3                     0                     1                       0   
4                     1                     0                       0   

   smoking_status_formerly smoked  smoking_status_never smoked  \
0                               1                            0   
1                               0                            1   
2                               0                            1   
3                               0                            0   
4                               0                            1   

   smoking_status_smokes  stroke  
0                      0       1  
1                      0       1  
2                      0       1  
3                      1       1  
4                      0       1  

[5 rows x 21 columns]
```

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, reca
import matplotlib.pyplot as plt
```

```
# Load the cleaned data
df = pd.read_csv('C:\\Users\\USER\\resampled_data.csv')

# Separate features and target variable
X = df.drop(columns=['stroke'])
y = df['stroke']

# Split the dataset into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

# Define the SVM model
svm = SVC()

# Define the hyperparameters to tune
param_grid = {
    'kernel': ['rbf'],
    'gamma': [0.1, 1, 10],
    'class_weight': ['balanced', None],
    'C': [0.1, 1, 10],
    'tol': [1e-3, 1e-4, 1e-5]
}

# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5, scoring='acc
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

# Print the best parameters
print("Best parameters of SVM:")
print(best_params)

# Train the SVM model with the best parameters
best_svm = SVC(**best_params)
best_svm.fit(X_train, y_train)

# Predictions
y_train_pred = best_svm.predict(X_train)
y_val_pred = best_svm.predict(X_val)
y_test_pred = best_svm.predict(X_test)

# Performance evaluation
print("\nPerformance on training set:")
print("Confusion Matrix:\n", confusion_matrix(y_train, y_train_pred))
print("Accuracy:", accuracy_score(y_train, y_train_pred))
print("Precision:", precision_score(y_train, y_train_pred))
print("Recall:", recall_score(y_train, y_train_pred))

print("\nPerformance on validation set:")
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Precision:", precision_score(y_val, y_val_pred))
print("Recall:", recall_score(y_val, y_val_pred))
```

```
print("\nPerformance on test set:")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Precision:", precision_score(y_test, y_test_pred))
print("Recall:", recall_score(y_test, y_test_pred))

# ROC curve and AUC
y_train_score = best_svm.decision_function(X_train)
fpr_train, tpr_train, _ = roc_curve(y_train, y_train_score)
roc_auc_train = auc(fpr_train, tpr_train)

y_val_score = best_svm.decision_function(X_val)
fpr_val, tpr_val, _ = roc_curve(y_val, y_val_score)
roc_auc_val = auc(fpr_val, tpr_val)

y_test_score = best_svm.decision_function(X_test)
fpr_test, tpr_test, _ = roc_curve(y_test, y_test_score)
roc_auc_test = auc(fpr_test, tpr_test)

# Plot ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr_train, tpr_train, label='ROC curve for SVM (train) - AUC = {:.2f}'.for
plt.plot(fpr_val, tpr_val, label='ROC curve for SVM (validation) - AUC = {:.2f}'.fo
plt.plot(fpr_test, tpr_test, label='ROC curve for SVM (test) - AUC = {:.2f}'.format
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Best parameters of SVM:
{'C': 10, 'class_weight': 'balanced', 'gamma': 1, 'kernel': 'rbf', 'tol': 0.001}

Performance on training set:
Confusion Matrix:
 [[3373   31]
 [   9 3392]]
Accuracy: 0.994121969140338
Precision: 0.9909436167104879
Recall: 0.9973537194942664

Performance on validation set:
Confusion Matrix:
 [[678  30]
 [ 30 720]]
Accuracy: 0.9588477366255144
Precision: 0.96
Recall: 0.96

Performance on test set:
Confusion Matrix:
 [[722  27]
 [ 41 669]]
Accuracy: 0.9533927347498287
Precision: 0.9612068965517241
Recall: 0.9422535211267605
```



In [5]:
```python
import numpy as np

#generate indices for gender_Male and gender_Female from the test data
PROTECTED_MALE = "gender_Male"
PROTECTED_FEMALE = "gender_Female"
```

```python
male_indices = np.where(X_test[PROTECTED_MALE] == 1)[0]
female_indices = np.where(X_test[PROTECTED_FEMALE] == 1)[0]

# Split predicted values into gender_Male and gender_Female groups
pred_male = y_test_pred[male_indices]
pred_female = y_test_pred[female_indices]

# Print the number of true and predicted values for each group
print("Predicted values for Men:", pred_male)
print("Predicted values for women:", pred_female)
```

```
Predicted values for Men: [0 0 0 0 0 1 1 1 1 0 1 0 1 0 0 1 1 0 1 0 1 0 1 1 0 0 0 0 1
0 1 1 0 0 0 1 0
 1 0 1 1 1 0 0 1 1 1 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 0 1
 0 1 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 1
 1 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0
 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0
 1 0 1 0 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0
 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 0 0 1 0 0 1
 0 1 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 0
 0 1 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 1 1 1 0 0
 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0
 0 1 1 0 0 1 1 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 1 0 1 0 0
 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 0
 1 1 1 0 1 0 0 1 1 0 1 1 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 1 1 1 0 0
 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0]
Predicted values for women: [1 0 1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 1 1 1 1 1 1
1 1 0 0 1 1 1 1 0
 0 1 0 0 1 0 0 1 1 0 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1 0 0 1 0 1
 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 1
 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 1
 0 0 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0
 0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 1
 1 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 1 1 0 0 1 1 1
 1 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1
 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 1 0
 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 1 0 1 1 1 0 1 1 0 1 1 0 1 0
 0 1 0 1 0 0 0 1 1 1 0 1 0 0 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 0 0 1 1
 1 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 1 1 0 1 1 0 0 0 1 1 1 1 1
 1 0 1 0 0 1 0 1 1 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 0 0
 0 1 1 1 0 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1 0 1 0 1
 0 1 1 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0
 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0
 1 0 1 1 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 1 1
 1 1 0 1 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 0
 0 0 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0 1 0 1 0 0 0 1 1 1 0 1 0 0 1
 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0
 1 0 0 0 0 0 1 1 1 1 0 1]
```

In [97]:
```python
# Check if 'gender_Male' and 'gender_Female' columns are present in X_test
print("Columns in X_test:", X_test.columns)

# Check the first few rows of X_test to ensure the columns are present and have cor
```

```
print("First few rows of X_test:\n", X_test.head())

# Check if the indices extracted for men and women are within the range of the inde
print("Indices for Men:", male_indices)
print("Indices for Women:", female_indices)

# Check if the indices are correctly assigned to gender_Male and gender_Female
print("Check if Men indices correspond to gender_Male in X_test:", all(X_test.iloc[
print("Check if Women indices correspond to gender_Female in X_test:", all(X_test.i
```

```
Columns in X_test: Index(['age', 'hypertension', 'heart_disease', 'ever_married',
       'avg_glucose_level', 'bmi', 'gender_Female', 'gender_Male',
       'gender_Other', 'work_type_Govt_job', 'work_type_Never_worked',
       'work_type_Private', 'work_type_Self-employed', 'work_type_children',
       'Residence_type_Rural', 'Residence_type_Urban',
       'smoking_status_Unknown', 'smoking_status_formerly smoked',
       'smoking_status_never smoked', 'smoking_status_smokes'],
      dtype='object')
First few rows of X_test:
       age  hypertension  heart_disease  ever_married  avg_glucose_level  \
5173   81             0              0             1           0.163735   
4649    8             0              0             0           0.550203   
7787   76             1              0             1           1.000000   
2611   25             0              0             1           0.230727   
6795   72             1              0             1           0.275810   

           bmi  gender_Female  gender_Male  gender_Other  work_type_Govt_job  \
5173  0.314916              1            0             0                   0   
4649  0.000000              0            1             0                   0   
7787  0.441722              0            0             0                   0   
2611  0.319548              1            0             0                   0   
6795  0.354178              1            0             0                   0   

      work_type_Never_worked  work_type_Private  work_type_Self-employed  \
5173                       0                  1                        0   
4649                       0                  0                        0   
7787                       0                  0                        1   
2611                       0                  1                        0   
6795                       0                  1                        0   

      work_type_children  Residence_type_Rural  Residence_type_Urban  \
5173                   0                     1                     0   
4649                   1                     0                     1   
7787                   0                     0                     0   
2611                   0                     0                     1   
6795                   0                     1                     0   

      smoking_status_Unknown  smoking_status_formerly smoked  \
5173                       0                               0   
4649                       1                               0   
7787                       0                               1   
2611                       1                               0   
6795                       0                               0   

      smoking_status_never smoked  smoking_status_smokes  
5173                            1                      0  
4649                            0                      0  
7787                            0                      0  
2611                            0                      0  
6795                            0                      0  
Indices for Men: [    1    13    17    19    21    22    23    25    32    37    38    39    41
42
    45    47    50    51    52    53    54    57    58    64    66    67    71    72
    76    78    82    83    87    90    93    95    99   100   109   110   112   114
   117   119   123   124   125   126   127   129   132   140   141   146   147   149
   152   153   154   156   159   160   163   164   168   171   174   175   176   185
```

```
  186  190  196  198  201  202  203  208  210  213  218  219  221  223
  224  225  226  227  233  234  236  237  238  239  241  245  246  248
  250  252  253  254  255  257  259  260  261  262  265  266  267  272
  275  276  277  279  280  281  282  283  287  290  296  298  299  302
  304  306  307  310  316  317  318  320  321  332  336  337  339  340
  346  347  351  352  355  356  359  362  366  367  369  372  374  376
  377  384  385  387  390  391  393  394  397  400  406  409  412  414
  415  417  420  421  423  425  427  430  431  432  435  445  448  452
  455  457  458  461  462  467  472  473  483  486  487  489  492  493
  494  495  496  497  500  504  506  507  508  509  510  511  515  516
  518  519  523  524  530  531  533  537  541  543  545  550  555  556
  557  558  559  560  564  565  566  568  569  572  576  578  580  581
  591  593  594  595  596  599  603  604  608  610  612  613  614  616
  617  618  621  623  624  626  627  628  631  634  638  639  642  646
  648  649  651  656  658  659  661  664  665  666  668  672  677  681
  685  686  689  690  691  692  693  696  697  699  701  703  707  708
  712  714  717  725  728  730  745  746  747  752  756  757  758  759
  761  763  765  766  767  771  773  777  781  786  787  791  792  794
  795  796  798  799  805  810  811  814  817  818  820  821  824  825
  827  828  831  832  838  841  844  846  847  850  851  857  866  870
  872  874  883  886  888  890  892  893  898  902  903  904  908  911
  912  915  918  921  926  936  937  939  942  943  945  946  948  953
  957  958  959  961  963  964  973  975  979  980  981  990  992  994
  995 1000 1001 1002 1003 1004 1007 1009 1011 1012 1014 1015 1025 1026
 1027 1029 1038 1041 1046 1048 1050 1051 1052 1055 1058 1060 1061 1064
 1065 1066 1067 1073 1075 1081 1082 1083 1087 1089 1093 1105 1107 1117
 1121 1123 1125 1130 1132 1134 1138 1139 1143 1147 1148 1150 1157 1160
 1161 1168 1169 1174 1178 1179 1181 1187 1188 1189 1190 1194 1206 1208
 1222 1223 1225 1226 1228 1229 1231 1233 1236 1238 1246 1248 1251 1254
 1255 1257 1258 1262 1263 1267 1269 1270 1274 1282 1283 1284 1285 1289
 1299 1301 1302 1303 1305 1311 1318 1329 1331 1333 1335 1339 1340 1342
 1349 1350 1352 1354 1357 1360 1362 1363 1364 1366 1369 1375 1376 1383
 1385 1388 1392 1393 1396 1397 1398 1401 1402 1403 1408 1409 1410 1412
 1413 1414 1418 1419 1424 1427 1428 1433 1434 1441 1445 1446 1450 1453
 1458]
Indices for Women: [   0    3    4    5    7    8   10   11   12   14   15   16   18
20
   24   26   27   28   29   30   33   34   35   36   40   43   44   46
   48   49   55   59   60   61   62   63   65   68   69   70   73   74
   75   77   79   80   81   84   85   86   88   89   91   92   94   98
  101  103  104  105  106  107  108  111  113  115  116  118  120  121
  128  130  131  133  135  136  137  138  139  142  143  144  145  148
  150  155  157  158  161  162  165  166  167  169  170  172  173  177
  179  180  181  182  183  184  187  188  189  191  192  193  195  197
  199  200  204  205  206  209  211  212  214  215  216  217  220  222
  228  229  230  231  232  242  243  244  249  251  256  258  263  264
  269  270  271  273  274  278  284  285  286  288  289  291  292  293
  294  295  297  300  301  308  309  311  313  314  315  319  322  323
  324  325  326  327  328  329  330  331  333  334  335  338  341  342
  343  345  348  349  350  354  357  358  360  361  363  364  365  368
  370  371  373  375  378  379  380  381  383  386  388  389  392  396
  398  399  401  402  404  405  407  411  416  418  419  422  426  428
  429  433  434  436  437  439  440  446  447  449  450  451  453  454
  459  460  463  464  465  466  468  470  471  474  475  476  477  478
  479  480  481  484  485  488  490  491  498  499  501  502  503  505
  512  513  514  517  520  521  522  525  528  529  534  535  536  538
```

```
  540  542  544  546  547  548  549  551  552  553  554  561  562  567
  570  571  573  574  575  577  579  582  583  584  589  590  592  598
  600  601  602  606  607  609  611  615  620  622  625  629  632  633
  636  641  643  644  645  647  650  652  653  654  655  657  660  662
  663  667  669  670  671  673  674  675  676  678  680  682  683  684
  687  688  694  695  698  700  702  704  705  706  709  710  713  715
  716  718  719  720  721  724  726  727  732  734  735  736  737  738
  739  740  741  743  744  749  750  751  753  754  760  762  764  768
  770  772  775  778  779  780  783  784  785  788  789  793  797  800
  802  803  804  806  807  808  809  812  813  815  816  819  822  823
  826  829  830  833  834  835  836  837  839  840  842  843  848  849
  852  853  854  855  856  858  859  862  863  864  865  869  871  873
  875  876  877  878  879  880  881  882  887  889  891  894  895  896
  897  899  900  905  906  907  909  910  913  914  916  917  919  920
  922  923  924  925  927  929  930  931  932  933  934  938  940  941
  944  947  949  951  952  954  955  960  962  965  966  967  968  969
  970  971  972  974  976  977  982  983  984  985  986  987  989  991
  993  996  997  998 1010 1013 1016 1017 1018 1019 1020 1022 1023 1024
 1028 1030 1032 1033 1034 1035 1036 1037 1039 1040 1042 1043 1044 1045
 1047 1049 1053 1054 1057 1059 1062 1068 1069 1070 1071 1072 1074 1076
 1077 1078 1079 1080 1084 1085 1088 1090 1091 1092 1094 1095 1096 1097
 1098 1100 1101 1102 1103 1104 1106 1108 1109 1110 1111 1112 1113 1114
 1115 1116 1118 1119 1120 1124 1126 1127 1128 1129 1131 1133 1135 1136
 1137 1140 1142 1144 1146 1149 1151 1152 1153 1154 1155 1156 1158 1159
 1162 1163 1165 1166 1167 1170 1171 1172 1173 1175 1176 1177 1182 1183
 1186 1191 1195 1196 1197 1198 1199 1200 1201 1202 1204 1205 1207 1209
 1210 1211 1212 1213 1214 1215 1217 1218 1221 1224 1227 1230 1232 1234
 1235 1237 1239 1240 1242 1243 1244 1245 1247 1249 1250 1252 1253 1256
 1259 1260 1261 1265 1266 1268 1272 1273 1275 1276 1277 1278 1280 1286
 1288 1292 1293 1294 1295 1296 1297 1298 1304 1306 1307 1308 1309 1310
 1312 1313 1314 1315 1316 1317 1319 1320 1321 1322 1323 1324 1325 1326
 1327 1328 1330 1332 1334 1336 1337 1338 1341 1343 1344 1346 1347 1348
 1351 1353 1355 1356 1358 1359 1361 1365 1367 1368 1370 1372 1373 1374
 1377 1378 1379 1380 1381 1382 1384 1386 1387 1389 1390 1391 1394 1399
 1405 1407 1411 1415 1416 1417 1420 1421 1422 1423 1425 1426 1429 1430
 1431 1432 1435 1436 1437 1438 1439 1440 1442 1444 1447 1448 1449 1451
 1452 1454 1455 1456 1457]
Check if Men indices correspond to gender_Male in X_test: True
Check if Women indices correspond to gender_Female in X_test: True
```

In [16]:

```python
from sklearn.metrics import confusion_matrix, classification_report

# Reset the index of y_test
y_test_reset_index = y_test.reset_index(drop=True)

# Calculate confusion matrix and classification report for men
conf_matrix_Men = confusion_matrix(y_test_reset_index[male_indices], pred_male)
report_Men = classification_report(y_test_reset_index[male_indices], pred_male)

# Calculate confusion matrix and classification report for women
conf_matrix_Women = confusion_matrix(y_test_reset_index[female_indices], pred_femal
report_Women = classification_report(y_test_reset_index[female_indices], pred_femal

# Print the confusion matrix and classification report for men
print("Confusion Matrix for Men:")
print(conf_matrix_Men)
```

```
print("\nClassification Report for men:")
print(report_Men)

# Print the confusion matrix and classification report for women
print("\nConfusion Matrix for women:")
print(conf_matrix_Women)
print("\nClassification Report for women:")
print(report_Women)
```

```
Confusion Matrix for Men:
[[310  13]
 [ 10 214]]

Classification Report for men:
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       323
           1       0.94      0.96      0.95       224

    accuracy                           0.96       547
   macro avg       0.96      0.96      0.96       547
weighted avg       0.96      0.96      0.96       547


Confusion Matrix for women:
[[412  14]
 [ 31 332]]

Classification Report for women:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       426
           1       0.96      0.91      0.94       363

    accuracy                           0.94       789
   macro avg       0.94      0.94      0.94       789
weighted avg       0.94      0.94      0.94       789
```

In [17]:

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

# Reset the index of y_test
y_test_reset_index = y_test.reset_index(drop=True)

# Calculate confusion matrix for men
conf_matrix_Men = confusion_matrix(y_test_reset_index[male_indices], pred_male)

# Calculate metrics for men
accuracy_Men = accuracy_score(y_test_reset_index[male_indices], pred_male)
precision_Men = precision_score(y_test_reset_index[male_indices], pred_male)
recall_Men = recall_score(y_test_reset_index[male_indices], pred_male)
positive_rate_Men = conf_matrix_Men[1, 1] / (conf_matrix_Men[1, 1] + conf_matrix_Me

# Print confusion matrix and metrics for men
print("Confusion Matrix for Men:")
print(conf_matrix_Men)
```

```
print("\nAccuracy for Men:", accuracy_Men)
print("Precision for Men:", precision_Men)
print("Recall for Men:", recall_Men)
print("Positive Rate for Men:", positive_rate_Men)

# Calculate confusion matrix for women
conf_matrix_Women = confusion_matrix(y_test_reset_index[female_indices], pred_femal

# Calculate metrics for women
accuracy_Women = accuracy_score(y_test_reset_index[female_indices], pred_female)
precision_Women = precision_score(y_test_reset_index[female_indices], pred_female)
recall_Women = recall_score(y_test_reset_index[female_indices], pred_female)
positive_rate_Women = conf_matrix_Women[1, 1] / (conf_matrix_Women[1, 1] + conf_mat

# Print confusion matrix and metrics for women
print("\nConfusion Matrix for Women:")
print(conf_matrix_Women)
print("\nAccuracy for Women:", accuracy_Women)
print("Precision for Women:", precision_Women)
print("Recall for Women:", recall_Women)
print("Positive Rate for Women:", positive_rate_Women)
```

```
Confusion Matrix for Men:
[[310  13]
 [ 10 214]]

Accuracy for Men: 0.9579524680073126
Precision for Men: 0.9427312775330396
Recall for Men: 0.9553571428571429
Positive Rate for Men: 0.9553571428571429

Confusion Matrix for Women:
[[412  14]
 [ 31 332]]

Accuracy for Women: 0.9429657794676806
Precision for Women: 0.9595375722543352
Recall for Women: 0.9146005509641874
Positive Rate for Women: 0.9146005509641874
```

In [4]:

```
import numpy as np

# Define confusion matrix for men and women
confusion_matrix_men = np.array([[310, 13], [10, 214]])
confusion_matrix_women = np.array([[412, 14], [31, 332]])

# Define functions to calculate TP, FN, TN, FP
def calculate_metrics(confusion_matrix):
    TP = confusion_matrix[0, 0]
    FN = confusion_matrix[0, 1]
    FP = confusion_matrix[1, 0]
    TN = confusion_matrix[1, 1]
    return TP, FN, FP, TN

# Calculate metrics for men and women
TP_men, FN_men, FP_men, TN_men = calculate_metrics(confusion_matrix_men)
```

```
TP_women, FN_women, FP_women, TN_women = calculate_metrics(confusion_matrix_women)

# Calculate fairness criteria
def calculate_fairness_criteria(TP, FN, FP, TN):
    # Equal Opportunity: TPR for positive class
    equal_opportunity = TP / (TP + FN)

    # Demographic Parity: Proportion of positive predictions
    demographic_parity = (TP + FP) / (TP + FN + FP + TN)

    # Equalized Odds: TPR and FPR for positive class
    TPR = TP / (TP + FN)
    FPR = FP / (FP + TN)
    equalized_odds = TPR / FPR

    return equal_opportunity, demographic_parity, equalized_odds

# Calculate fairness criteria for men and women
equal_opportunity_men, demographic_parity_men, equalized_odds_men = calculate_fairn
equal_opportunity_women, demographic_parity_women, equalized_odds_women = calculate

# Print results
print("Fairness Criteria for Men:")
print("Equal Opportunity:", equal_opportunity_men)
print("Demographic Parity:", demographic_parity_men)
print("Equalized Odds:", equalized_odds_men)

print("\nFairness Criteria for Women:")
print("Equal Opportunity:", equal_opportunity_women)
print("Demographic Parity:", demographic_parity_women)
print("Equalized Odds:", equalized_odds_women)
```

```
Fairness Criteria for Men:
Equal Opportunity: 0.9597523219814241
Demographic Parity: 0.5850091407678245
Equalized Odds: 21.4984520123839

Fairness Criteria for Women:
Equal Opportunity: 0.9671361502347418
Demographic Parity: 0.5614702154626109
Equalized Odds: 11.324852339845524
```

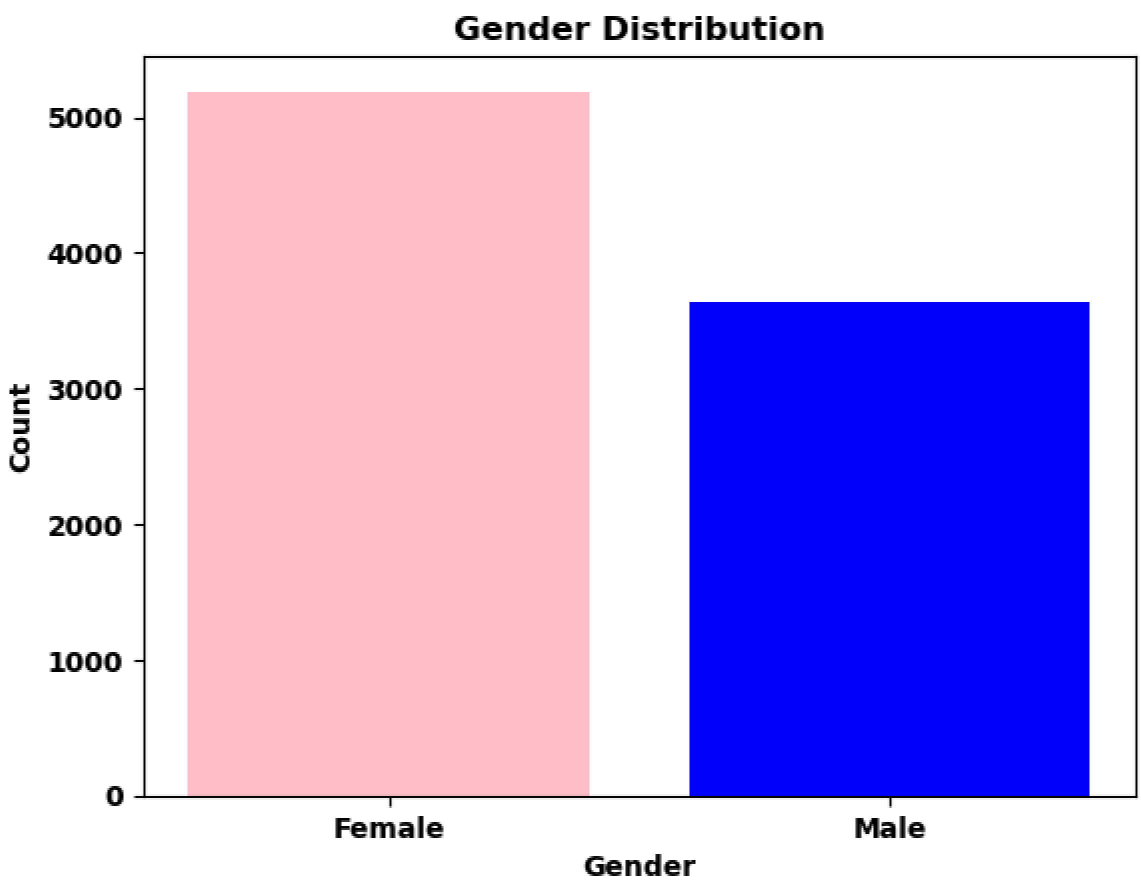
In [2]:

```
import pandas as pd

# Load the dataset
file_path = 'C:\\Users\\USER\\resampled_data.csv'
df_imported = pd.read_csv(file_path)

# Check the balance of gender_Female and gender_Male
female_count = df_imported['gender_Female'].sum()
male_count = df_imported['gender_Male'].sum()

# Check if the counts are balanced
if female_count == male_count:
    print("Gender balance: Equal number of females and males.")
else:
```

```
    print("Gender balance: Not equal number of females and males.")
    print("Female count:", female_count)
    print("Male count:", male_count)
```

```
Gender balance: Not equal number of females and males.
Female count: 5189
Male count: 3635
```

In [3]:
```
import matplotlib.pyplot as plt

# Counts of females and males
female_count = 5189
male_count = 3635

# Create a bar chart
plt.bar(['Female', 'Male'], [female_count, male_count], color=['pink', 'blue'])
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Gender Distribution')
plt.show()
```



In [ ]: