Capstone Assignment 20.1: Initial Report & Exploratory Data Analysis (EDA)

Research Question:

Can we predict heart disease in people by looking at their basic personal and health information?

Expected Data Source:

The data will come from a dataset "https://www.kaggle.com/datasets/mirzahasnine/heart-disease-dataset?select=heart_disease.csv" called heart_disease.csv, which includes basic personal and health information, provided as a CSV file.

Techniques:

Simple data analysis to understand the data

Creating useful features from the data

Using machine learning methods (like Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting)

Checking how well the model works using methods like cross-validation, confusion matrix, and ROC curve.

Expected Results:

I aim to create a reliable tool that can predict who might develop heart disease based on their basic personal and health details. This tool will show us which factors are most important for predicting heart disease, helping doctors and patients take action early.

Why This Question is Important:

Heart disease is a major cause of death around the world. If we can predict heart disease early, we can help people make lifestyle changes or get medical treatment sooner. Without answering this question, people might not know they are at risk, missing the chance to prevent serious health issues. A simple predictive tool can help doctors and patients make better decisions, saving lives and reducing healthcare costs by allowing earlier treatment and better personal care plans.

1. Import Libraries & Load Dataset

```
# 1.1 Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# For consistent plot styles
sns.set(style="whitegrid")
```

```
# 1.2  Load the dataset
df = pd.read_csv('heart_disease.csv')
```

2. Data Overview & Initial Checks

```
# 2.1 Preview the dataset
df.head()
```

	Gender	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI
0	Male	39	postgraduate	0	0.0	0.0	no	0	0	195.0	106.0	70.0	26.97
1	Female	46	primaryschool	0	0.0	0.0	no	0	0	250.0	121.0	81.0	28.73
2	Male	48	uneducated	1	20.0	0.0	no	0	0	245.0	127.5	80.0	25.34
3	Female	61	graduate	1	30.0	0.0	no	1	0	225.0	150.0	95.0	28.58
4	Female	46	graduate	1	23.0	0.0	no	0	0	285.0	130.0	84.0	23.10

Nächste Schritte: Code mit df generieren | Empfohlene Diagramme ansehen | New interactive sheet

```
# 2.2 Check data types and non-null values
print("\nDataset Info:")
df.info()
```

```
Dataset Info:
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4238 entries, 0 to 4237
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Gender           4238 non-null   object 
 1   age              4238 non-null   int64  
 2   education        4133 non-null   object 
 3   currentSmoker    4238 non-null   int64  
 4   cigsPerDay       4209 non-null   float64
 5   BPMeds           4185 non-null   float64
 6   prevalentStroke  4238 non-null   object 
 7   prevalentHyp     4238 non-null   int64  
 8   diabetes         4238 non-null   int64  
 9   totChol          4188 non-null   float64
 10  sysBP            4238 non-null   float64
 11  diaBP            4238 non-null   float64
 12  BMI              4219 non-null   float64
 13  heartRate        4237 non-null   float64
 14  glucose          3850 non-null   float64
 15  Heart_ stroke    4238 non-null   object 
dtypes: float64(8), int64(4), object(4)
memory usage: 529.9+ KB
```

```
# 2.3 Check the shape of the dataset
print("Dataset Shape:", df.shape)
```

```
Dataset Shape: (4238, 16)
```

```
# 2.4 Check for missing values in each column
print("\nMissing Values:")
print(df.isnull().sum())
```

```
Missing Values:
Gender               0
age                  0
education          105
currentSmoker        0
cigsPerDay          29
BPMeds              53
prevalentStroke      0
prevalentHyp         0
diabetes             0
totChol             50
sysBP                0
diaBP                0
BMI                 19
heartRate            1
glucose            388
Heart_ stroke        0
dtype: int64
```

```
# 2.5 View basic statistics for numeric columns
print("\nDescriptive Statistics:")
print(df.describe())
```

```
Descriptive Statistics:
               age  currentSmoker   cigsPerDay       BPMeds  prevalentHyp  \
count  4238.000000    4238.000000  4209.000000  4185.000000   4238.000000   
mean     49.584946       0.494101     9.003089     0.029630      0.310524   
std       8.572160       0.500024    11.920094     0.169584      0.462763   
min      32.000000       0.000000     0.000000     0.000000      0.000000   
25%      42.000000       0.000000     0.000000     0.000000      0.000000   
50%      49.000000       0.000000     0.000000     0.000000      0.000000   
75%      56.000000       1.000000    20.000000     0.000000      1.000000   
max      70.000000       1.000000    70.000000     1.000000      1.000000   

          diabetes      totChol        sysBP        diaBP          BMI  \
count  4238.000000  4188.000000  4238.000000  4238.000000  4219.000000   
mean      0.025720   236.721585   132.352407    82.893464    25.802008   
std       0.158316    44.590334    22.038097    11.910850     4.080111   
min       0.000000   107.000000    83.500000    48.000000    15.540000   
25%       0.000000   206.000000   117.000000    75.000000    23.070000   
50%       0.000000   234.000000   128.000000    82.000000    25.400000   
75%       0.000000   263.000000   144.000000    89.875000    28.040000   
max       1.000000   696.000000   295.000000   142.500000    56.800000   

         heartRate      glucose  
count  4237.000000  3850.000000  
mean     75.878924    81.966753  
std      12.026596    23.959998  
min      44.000000    40.000000  
25%      68.000000    71.000000  
50%      75.000000    78.000000  
75%      83.000000    87.000000  
```

```
max     143.000000   394.000000
```

```
# 2.6 Check for duplicate rows
print("\nNumber of duplicate rows:", df.duplicated().sum())
```

```
Number of duplicate rows: 0
```

```
# 2.7 Check unique values for categorical variables
print("\nUnique values in categorical features:")
categorical_cols = df.select_dtypes(include='object').columns
for col in categorical_cols:
    print(f"{col}: {df[col].nunique()} unique values")
```

```
Unique values in categorical features:
Gender: 2 unique values
education: 4 unique values
prevalentStroke: 2 unique values
Heart_ stroke: 2 unique values
```

3. Data Cleaning

```
# 3.1 Make a copy of the dataset
df_clean = df.copy()
```

```
# 3.2 Clean column names (remove spaces, fix inconsistent naming)
df_clean.columns = df_clean.columns.str.strip().str.replace(' ', '').str.replace('-', '')
```

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4238 entries, 0 to 4237
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           4238 non-null   object
 1   age              4238 non-null   int64
 2   education        4133 non-null   object
 3   currentSmoker    4238 non-null   int64
 4   cigsPerDay       4209 non-null   float64
 5   BPMeds           4185 non-null   float64
 6   prevalentStroke  4238 non-null   object
 7   prevalentHyp     4238 non-null   int64
 8   diabetes         4238 non-null   int64
 9   totChol          4188 non-null   float64
 10  sysBP            4238 non-null   float64
 11  diaBP            4238 non-null   float64
 12  BMI              4219 non-null   float64
 13  heartRate        4237 non-null   float64
 14  glucose          3850 non-null   float64
 15  Heart_stroke     4238 non-null   object
dtypes: float64(8), int64(4), object(4)
memory usage: 529.9+ KB
```

```
# 3.3 Rename target column for clarity
df_clean.rename(columns={'Heart_stroke': 'HeartStroke'}, inplace=True)
```

```
# 3.4 Handling missing values
# Impute numerical columns with median
num_cols = df_clean.select_dtypes(include=['float64', 'int64']).columns
for col in num_cols:
    if df_clean[col].isnull().sum() > 0:
        df_clean[col] = df_clean[col].fillna(df_clean[col].median())

# Impute categorical columns with mode
cat_cols = df_clean.select_dtypes(include='object').columns
for col in cat_cols:
    if df_clean[col].isnull().sum() > 0:
        df_clean[col] = df_clean[col].fillna(df_clean[col].mode()[0])
```

```
# 3.5 Convert categorical columns to lowercase to avoid for example 'No' vs 'no' issues
for col in cat_cols:
    df_clean[col] = df_clean[col].str.lower().str.strip()
```

```
# 3.6 Preview cleaned data
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4238 entries, 0 to 4237
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Gender           4238 non-null   object 
 1   age              4238 non-null   int64  
 2   education        4238 non-null   object 
 3   currentSmoker    4238 non-null   int64  
 4   cigsPerDay       4238 non-null   float64
 5   BPMeds           4238 non-null   float64
 6   prevalentStroke  4238 non-null   object 
 7   prevalentHyp     4238 non-null   int64  
 8   diabetes         4238 non-null   int64  
 9   totChol          4238 non-null   float64
 10  sysBP            4238 non-null   float64
 11  diaBP            4238 non-null   float64
 12  BMI              4238 non-null   float64
 13  heartRate        4238 non-null   float64
 14  glucose          4238 non-null   float64
 15  HeartStroke      4238 non-null   object 
dtypes: float64(8), int64(4), object(4)
memory usage: 529.9+ KB
```

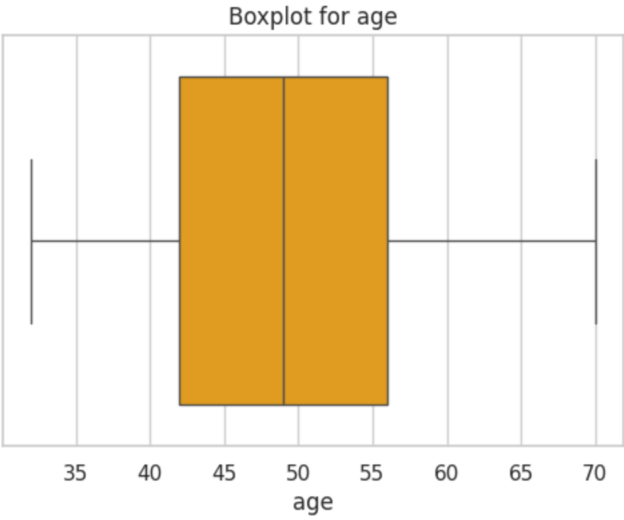
```
# 3.7 Check for missing values in each column
df_clean.isnull().sum()
```

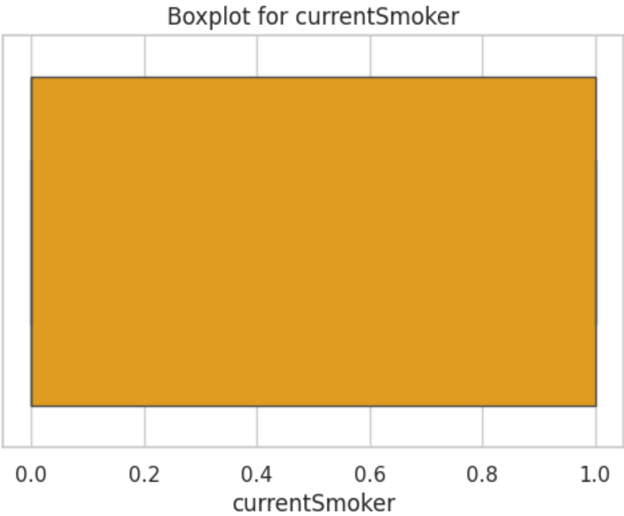
	0
Gender	0
age	0
education	0
currentSmoker	0
cigsPerDay	0
BPMeds	0
prevalentStroke	0
prevalentHyp	0
diabetes	0
totChol	0
sysBP	0
diaBP	0
BMI	0
heartRate	0
glucose	0
HeartStroke	0

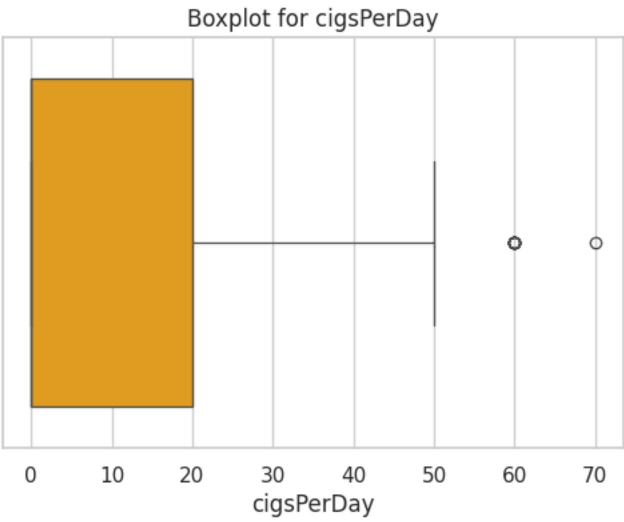
dtype: int64

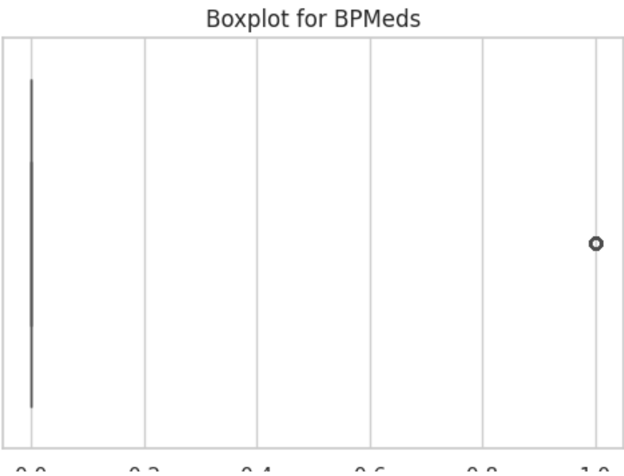
Outlier Analysis

```
# 3.8 Outlier Analysis

numeric_cols = df_clean.select_dtypes(include=['float64', 'int64']).columns

# Visualize boxplots for each numeric column
for col in numeric_cols:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=df_clean[col], color='orange')
    plt.title(f'Boxplot for {col}')
    plt.show()
```

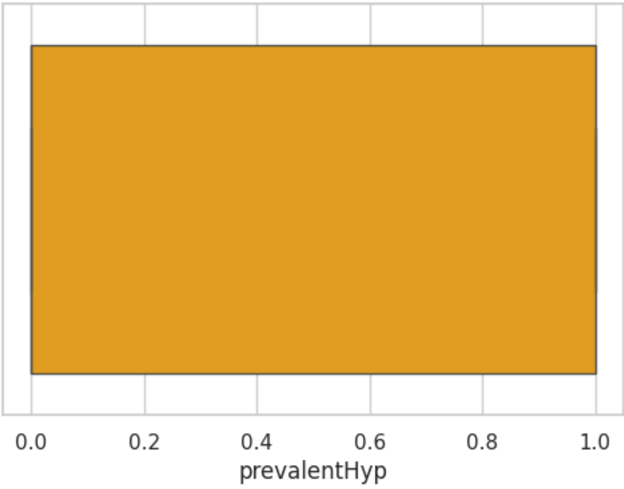

Boxplot for age
35
40
45
50
55
60
65
70
age


Boxplot for currentSmoker
0.0
0.2
0.4
0.6
0.8
1.0
currentSmoker

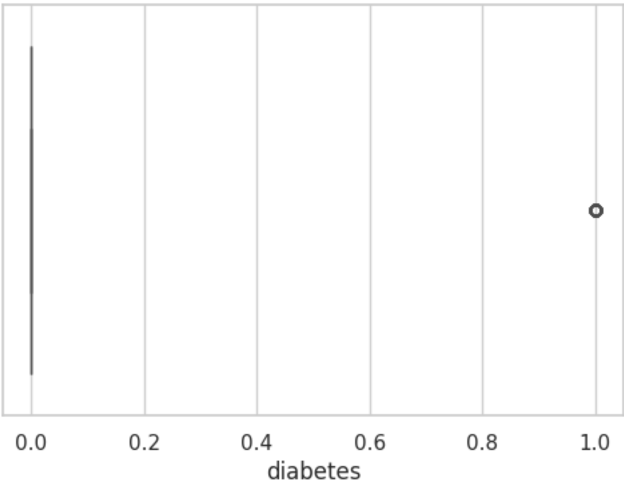

Boxplot for cigsPerDay
0
10
20
30
40
50
60
70
cigsPerDay


Boxplot for BPMeds

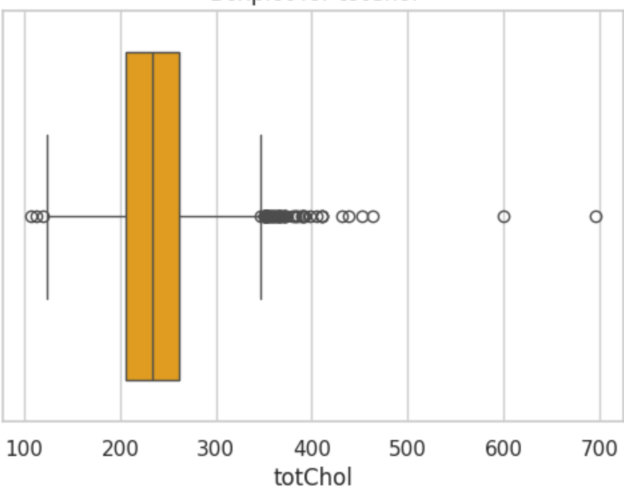
Boxplot for prevalentHyp



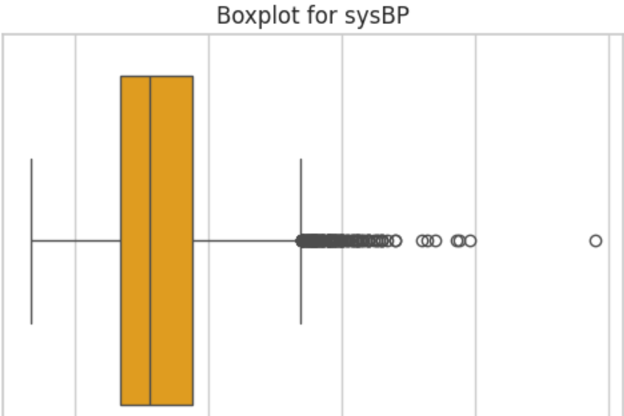
Boxplot for diabetes

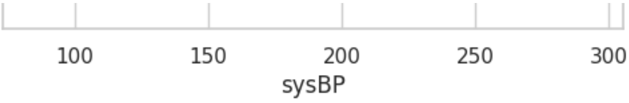


Boxplot for totChol

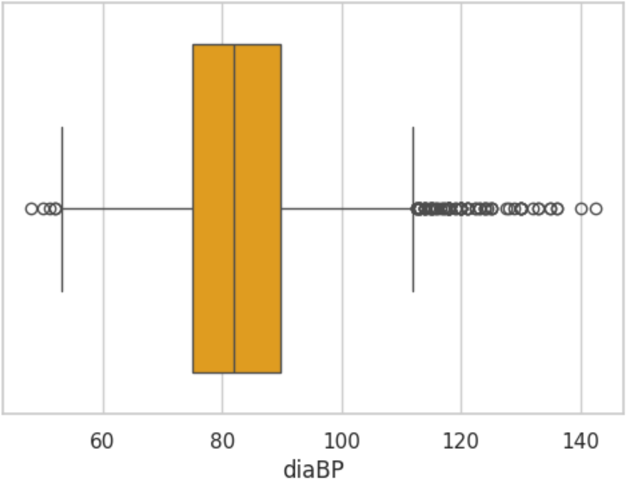


Boxplot for sysBP

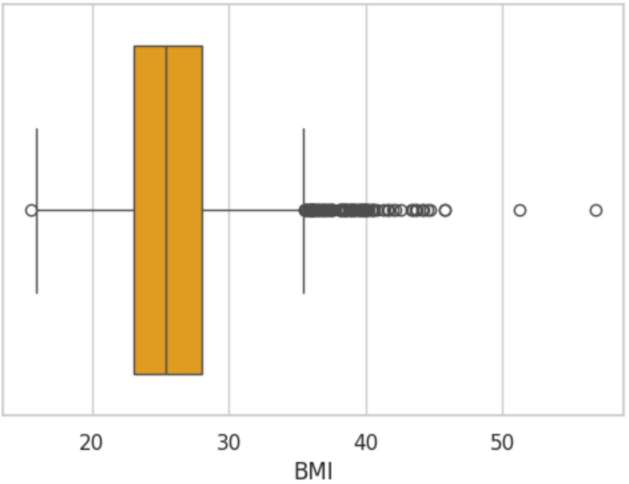




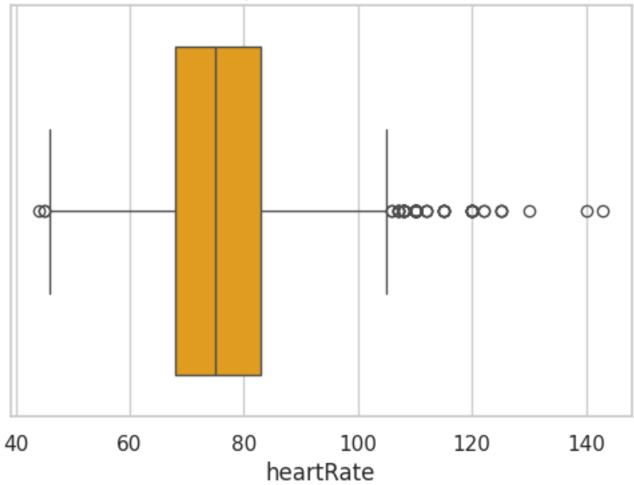
Boxplot for diaBP



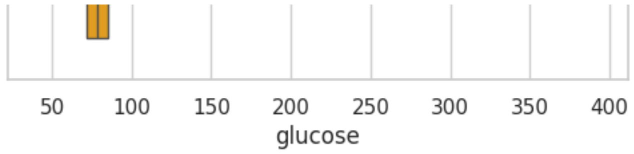
Boxplot for BMI



Boxplot for heartRate



Boxplot for glucose


50
100
150
200
250
300
350
400
glucose

```
# Define a function to cap outliers using IQR method
def cap_outliers(df, columns):
    df_capped = df.copy()  # Create a copy so we don't modify original
    for col in columns:
        Q1 = df_capped[col].quantile(0.25)
        Q3 = df_capped[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Cap lower and upper values
        df_capped[col] = np.where(df_capped[col] < lower_bound, lower_bound, df_capped[col])
        df_capped[col] = np.where(df_capped[col] > upper_bound, upper_bound, df_capped[col])

    return df_capped

# Apply the capping function to all numeric columns
df_clean_capped = cap_outliers(df_clean, numeric_cols)

# Check new basic statistics
df_clean_capped.describe()
```

	age	currentSmoker	cigsPerDay	BPMeds	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	h
count	4238.000000	4238.000000	4238.000000	4238.0	4238.000000	4238.0	4238.000000	4238.000000	4238.000000	4238.000000	423
mean	49.584946	0.494101	8.910807	0.0	0.310524	0.0	236.191600	131.913285	82.733291	25.715456	7
std	8.572160	0.500024	11.781028	0.0	0.462763	0.0	42.297346	20.647252	11.389783	3.788664	[illegible]
min	32.000000	0.000000	0.000000	0.0	0.000000	0.0	122.000000	83.500000	52.687500	15.643750	4
25%	42.000000	0.000000	0.000000	0.0	0.000000	0.0	206.000000	117.000000	75.000000	23.080000	6
50%	49.000000	0.000000	0.000000	0.0	0.000000	0.0	234.000000	128.000000	82.000000	25.400000	7
75%	56.000000	1.000000	20.000000	0.0	1.000000	0.0	262.000000	144.000000	89.875000	28.037500	8
max	70.000000	1.000000	50.000000	0.0	1.000000	0.0	346.000000	184.500000	112.187500	35.473750	10

```
# Define a function to detect outliers
def detect_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    return outliers

# Find outliers in
# Loop through each numeric column
for col in numeric_cols:
    outliers = detect_outliers(df_clean, col)  # Use your function
    print(f'Number of outliers in {col}: {len(outliers)}')
```

```
Number of outliers in age: 0
Number of outliers in currentSmoker: 0
Number of outliers in cigsPerDay: 12
Number of outliers in BPMeds: 124
Number of outliers in prevalentHyp: 0
Number of outliers in diabetes: 109
Number of outliers in totChol: 57
Number of outliers in sysBP: 126
Number of outliers in diaBP: 81
Number of outliers in BMI: 97
Number of outliers in heartRate: 76
Number of outliers in glucose: 262
```

```
# Save df_clean to a CSV file
df_clean.to_csv('df_clean.csv', index=False)
```

```
def remove_outliers(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        # Keep only rows within bounds
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df
```

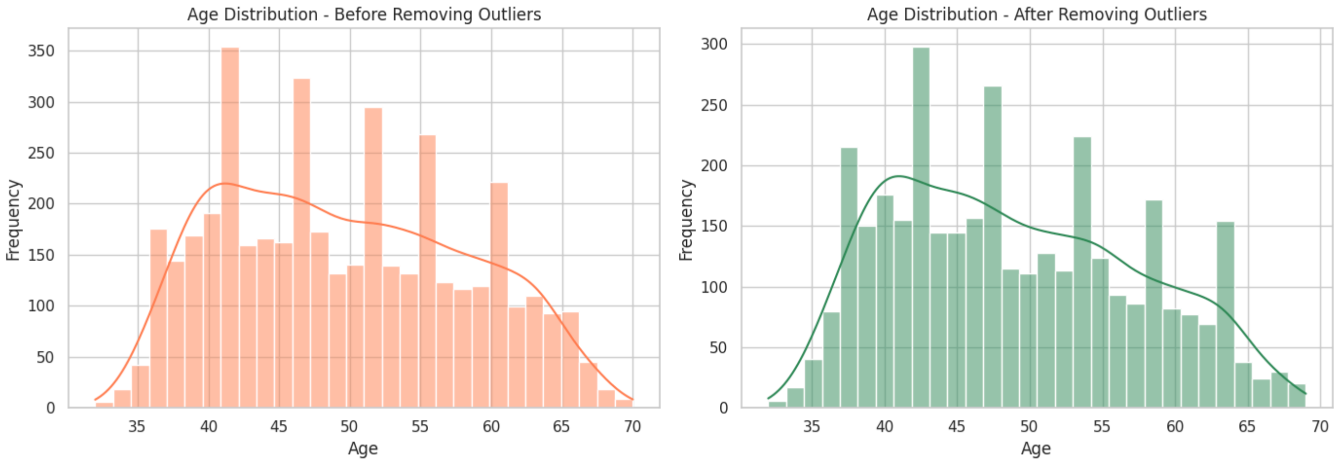
```
# Apply to df_clean
numeric_cols = df_clean.select_dtypes(include=['float64', 'int64']).columns
df_clean_no_outliers = remove_outliers(df_clean, numeric_cols)

# Check new shape
print("Original shape:", df_clean.shape)
print("New shape after removing outliers:", df_clean_no_outliers.shape)
```

```
Original shape: (4238, 17)
New shape after removing outliers: (3510, 17)
```

```
# Save df_clean_no_outliers to a CSV file
df_clean.to_csv('df_clean_no_outliers.csv', index=False)
```

4. Visual Exploratory Data Analysis (EDA)

```
# 4.1 Distribution Plots
# Histogram of Age
# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Plot 1: Age distribution in df_clean (before removing outliers)
sns.histplot(data=df_clean, x='age', kde=True, bins=30, color='coral', ax=axes[0])
axes[0].set_title('Age Distribution - Before Removing Outliers')
axes[0].set_xlabel('Age')
axes[0].set_ylabel('Frequency')

# Plot 2: Age distribution in df_clean_no_outliers (after removing outliers)
sns.histplot(data=df_clean_no_outliers, x='age', kde=True, bins=30, color='seagreen', ax=axes[1])
axes[1].set_title('Age Distribution - After Removing Outliers')
axes[1].set_xlabel('Age')
axes[1].set_ylabel('Frequency')

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



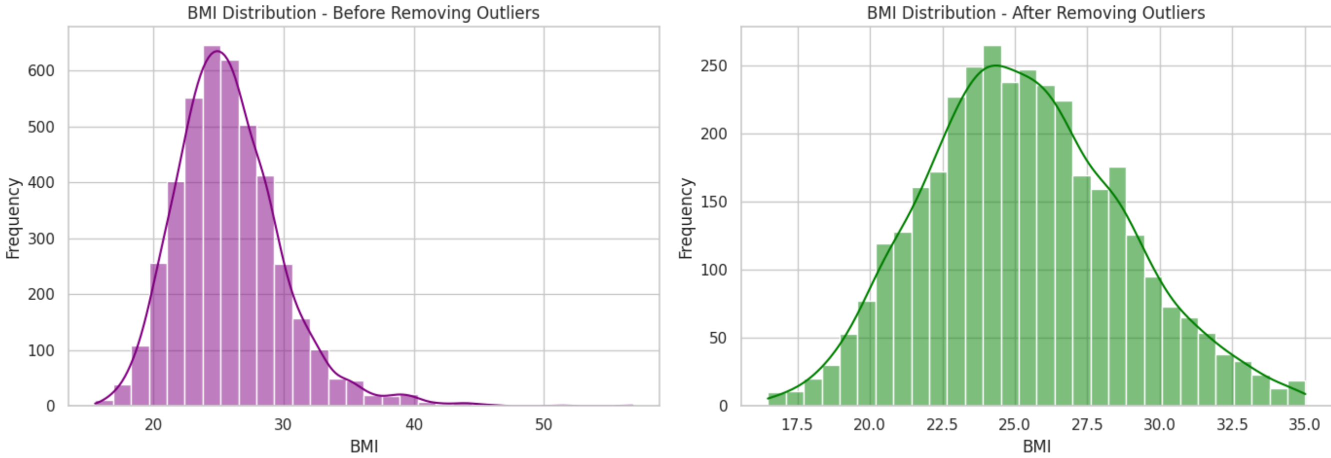
@finding:

The side-by-side comparison of Age distribution shows that removing outliers had minimal impact on the overall age distribution, confirming that the dataset is stable in this feature

The age distribution is approximately normal, with most individuals falling between 40 and 60 years old. This suggests the dataset mainly includes middle-aged adults, a group at higher risk for heart disease and stroke.

```
# 4.2 Distribution Plots
# Histogram of BMI
# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Plot 1: BMI distribution in df_clean (before removing outliers)
sns.histplot(data=df_clean, x='BMI', kde=True, bins=30, color='purple', ax=axes[0])
axes[0].set_title('BMI Distribution - Before Removing Outliers')
axes[0].set_xlabel('BMI')
axes[0].set_ylabel('Frequency')
```

```
# Plot 2: BMI distribution in df_clean_no_outliers (after removing outliers)
sns.histplot(data=df_clean_no_outliers, x='BMI', kde=True, bins=30, color='green', ax=axes[1])
axes[1].set_title('BMI Distribution - After Removing Outliers')
axes[1].set_xlabel('BMI')
axes[1].set_ylabel('Frequency')

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

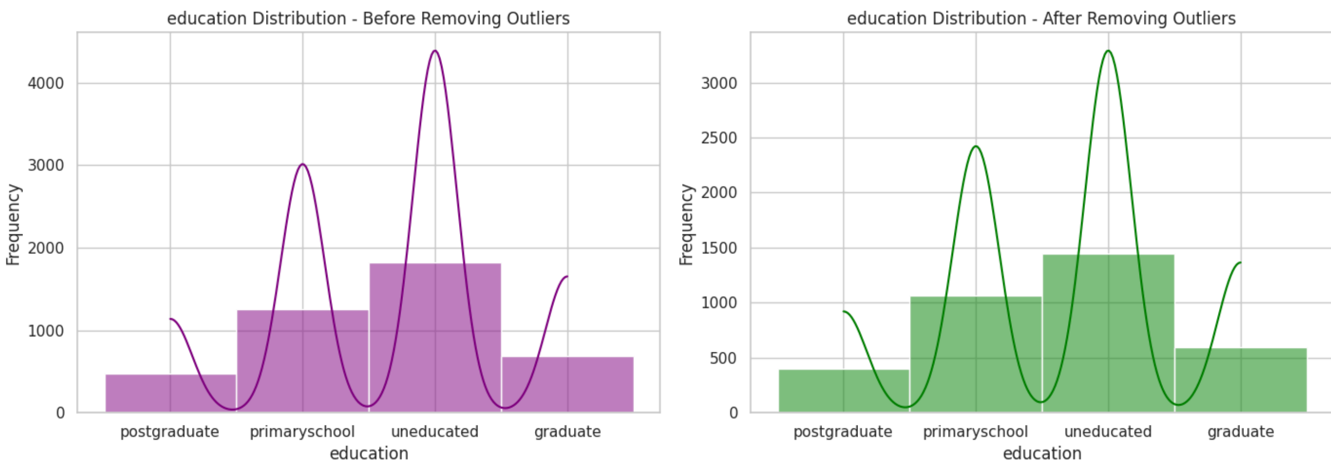


@finding:

The BMI distribution is right-skewed, meaning more people are slightly overweight or obese. A few individuals have high BMI values above 40, which are considered extreme and may be outliers.

The side-by-side comparison shows that removing outliers made the BMI distribution tighter and reduced extreme values above 40.

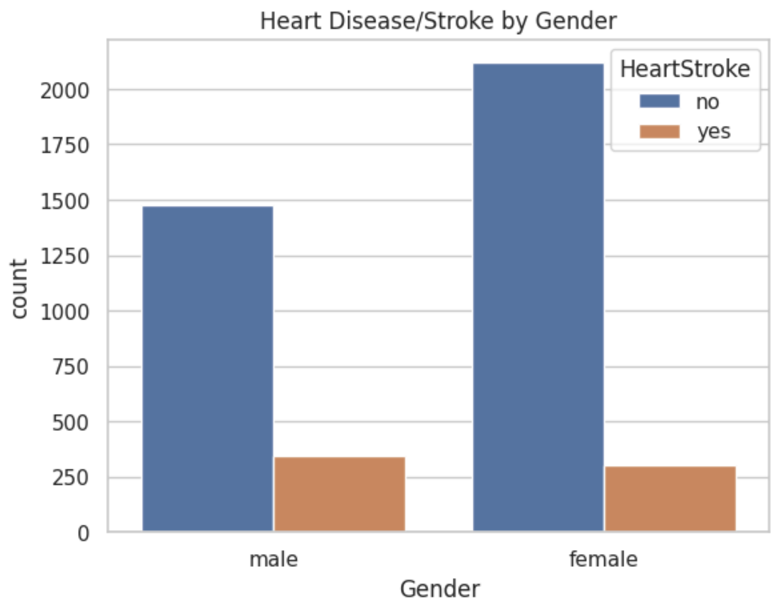
```
# 4.3 Distribution Plots
# education distribution
# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Plot 1: BMI distribution in df_clean (before removing outliers)
sns.histplot(data=df_clean, x='education', kde=True, bins=30, color='purple', ax=axes[0])
axes[0].set_title('education Distribution - Before Removing Outliers')
axes[0].set_xlabel('education')
axes[0].set_ylabel('Frequency')

# Plot 2: BMI distribution in df_clean_no_outliers (after removing outliers)
sns.histplot(data=df_clean_no_outliers, x='education', kde=True, bins=30, color='green', ax=axes[1])
axes[1].set_title('education Distribution - After Removing Outliers')
axes[1].set_xlabel('education')
axes[1].set_ylabel('Frequency')

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



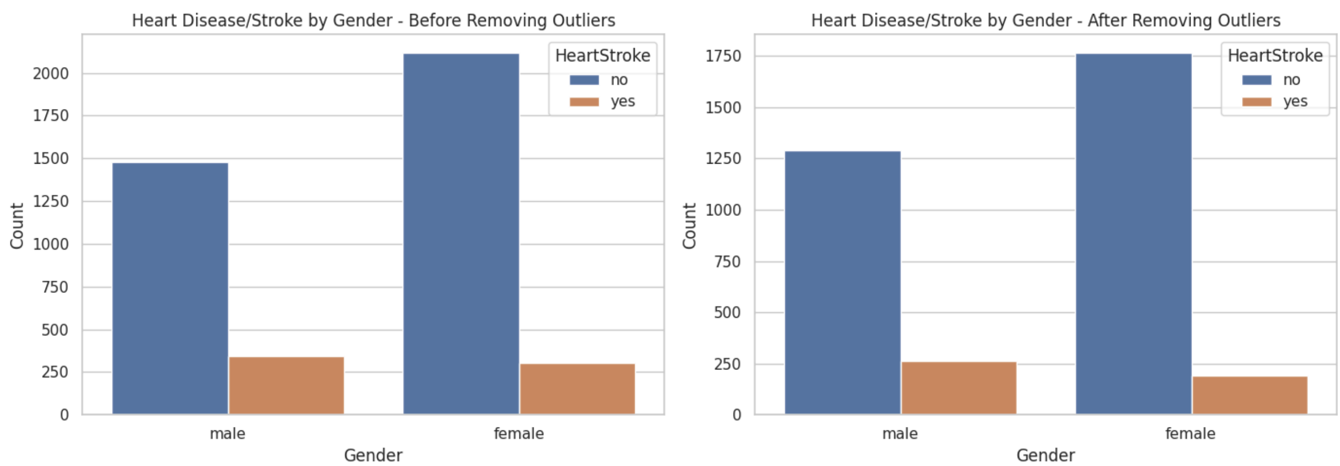
@finding:

There are more people with low education (primary or no schooling) than with higher education (graduate or postgraduate).

```
# 4.4 Bar Plots for Categorical vs Target (HeartStroke)
# Gender vs HeartStroke
sns.countplot(data=df_clean, x='Gender', hue='HeartStroke')
plt.title('Heart Disease/Stroke by Gender')
plt.show()
```



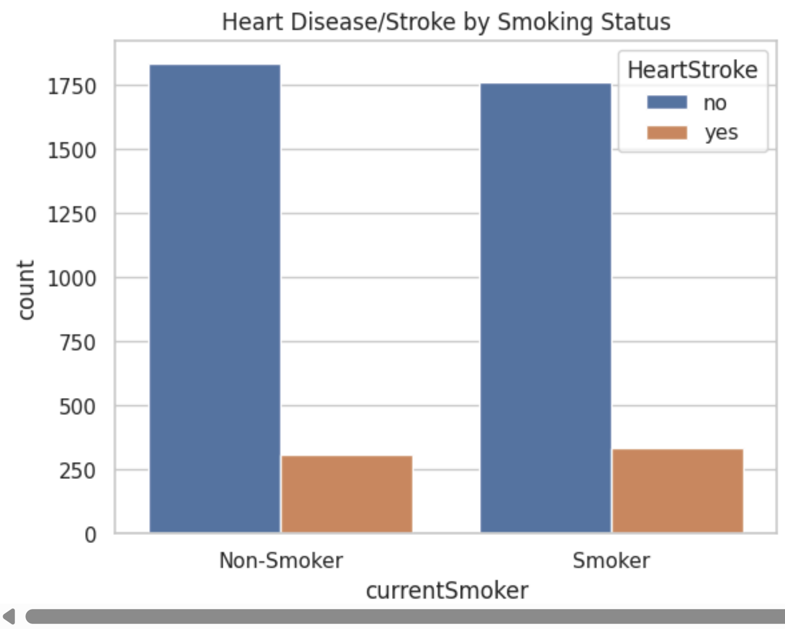

```
# Create side-by-side plots
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Plot 1: df_clean (before removing outliers)
sns.countplot(data=df_clean, x='Gender', hue='HeartStroke', ax=axes[0])
axes[0].set_title('Heart Disease/Stroke by Gender - Before Removing Outliers')
axes[0].set_xlabel('Gender')
axes[0].set_ylabel('Count')

# Plot 2: df_clean_no_outliers (after removing outliers)
sns.countplot(data=df_clean_no_outliers, x='Gender', hue='HeartStroke', ax=axes[1])
axes[1].set_title('Heart Disease/Stroke by Gender - After Removing Outliers')
axes[1].set_xlabel('Gender')
axes[1].set_ylabel('Count')

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



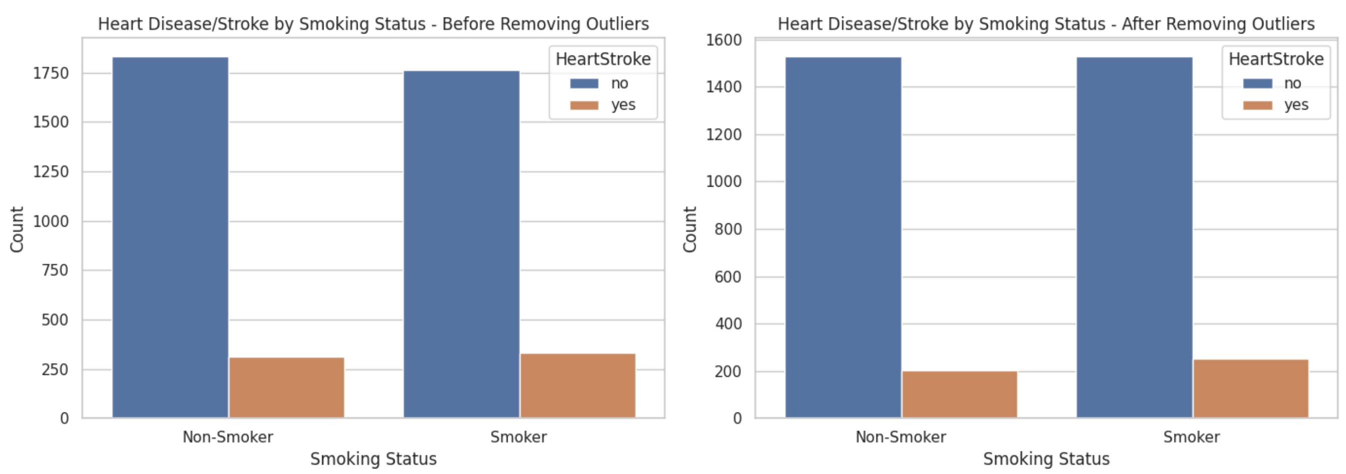
@finding:

Both males and females are represented in the dataset. However, the number of people with heart disease/stroke appears slightly higher among males. This could indicate a higher risk among male participants, though more analysis is needed.

```
# 4.5 Bar Plots for Categorical vs Target (HeartStroke)
# Smoking status vs HeartStroke
sns.countplot(data=df_clean, x='currentSmoker', hue='HeartStroke')
plt.title('Heart Disease/Stroke by Smoking Status')
plt.xticks([0, 1], ['Non-Smoker', 'Smoker'])
plt.show()
```



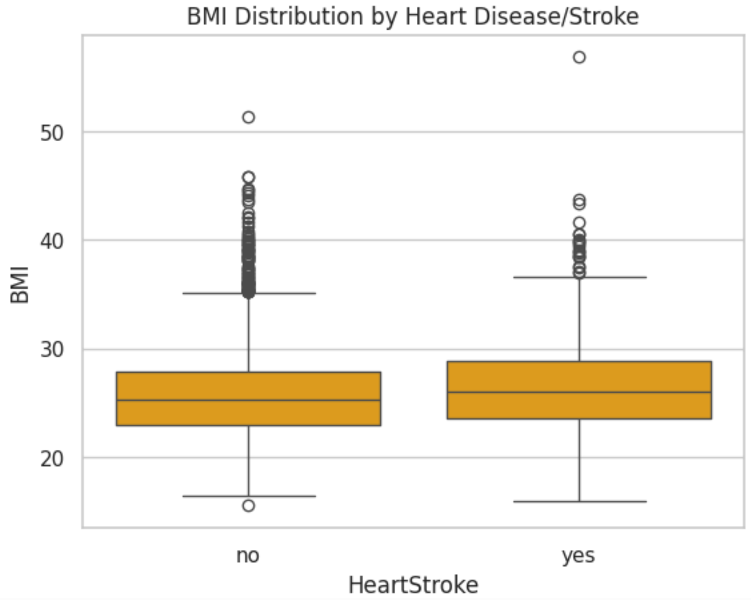

```
# Create side-by-side plots

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Plot 1: df_clean (before removing outliers)
sns.countplot(data=df_clean, x='currentSmoker', hue='HeartStroke', ax=axes[0])
axes[0].set_title('Heart Disease/Stroke by Smoking Status - Before Removing Outliers')
axes[0].set_xlabel('Smoking Status')
axes[0].set_ylabel('Count')
axes[0].set_xticks([0, 1])  # Set positions
axes[0].set_xticklabels(['Non-Smoker', 'Smoker'])  # Set labels

# Plot 2: df_clean_no_outliers (after removing outliers)
sns.countplot(data=df_clean_no_outliers, x='currentSmoker', hue='HeartStroke', ax=axes[1])
axes[1].set_title('Heart Disease/Stroke by Smoking Status - After Removing Outliers')
axes[1].set_xlabel('Smoking Status')
axes[1].set_ylabel('Count')
axes[1].set_xticks([0, 1])  # Set positions
axes[1].set_xticklabels(['Non-Smoker', 'Smoker'])  # Set labels

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



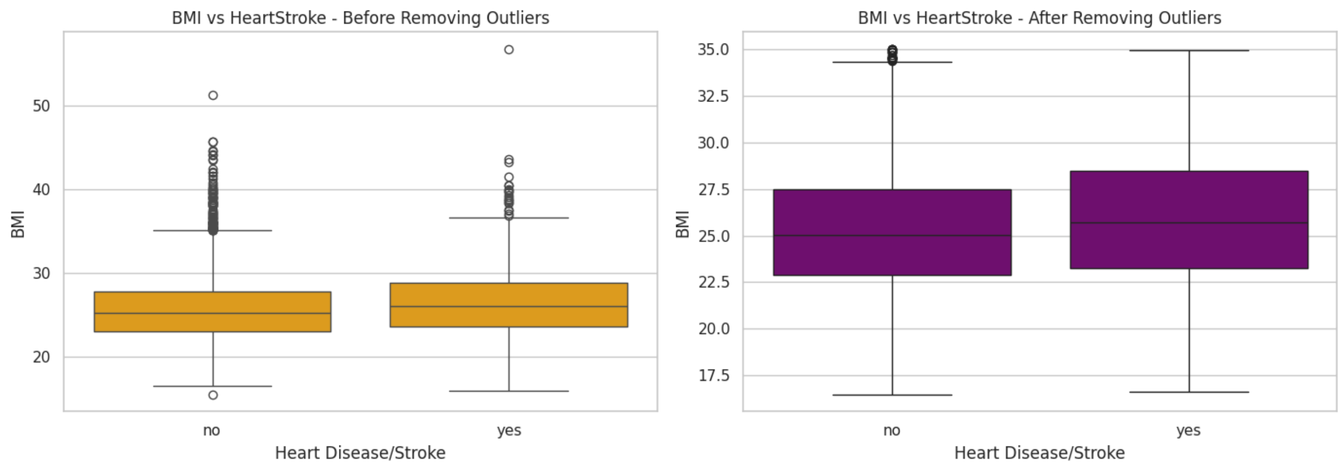
@finding:

Smokers are more likely to be in the heart disease/stroke group than non-smokers. This supports the well-known link between smoking and cardiovascular risk.

```
# 4.6 Boxplots (To Check Impact of Numeric on Target)
# BMI vs HeartStroke
sns.boxplot(data=df_clean, x='HeartStroke', y='BMI', color='orange')
plt.title('BMI Distribution by Heart Disease/Stroke')
plt.show()
```



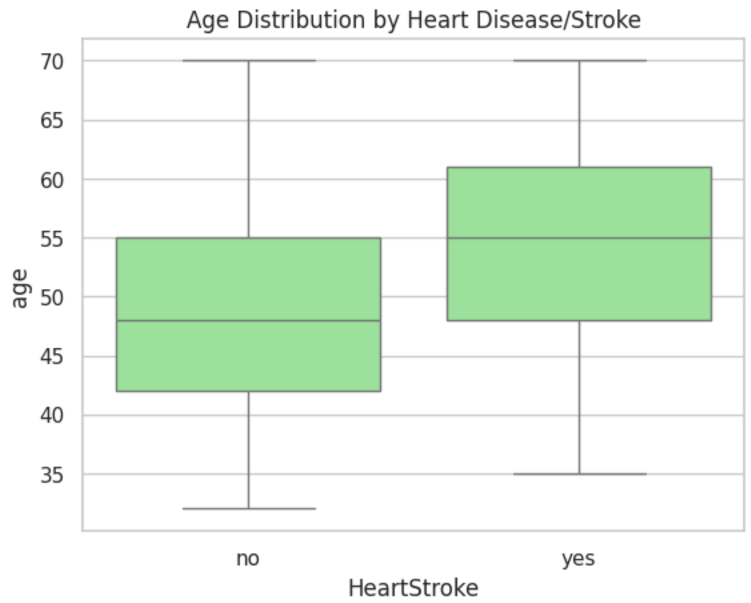

```
# Create side-by-side boxplots
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Boxplot 1: Before Removing Outliers
sns.boxplot(data=df_clean, x='HeartStroke', y='BMI', color='orange', ax=axes[0])
axes[0].set_title('BMI vs HeartStroke - Before Removing Outliers')
axes[0].set_xlabel('Heart Disease/Stroke')
axes[0].set_ylabel('BMI')

# Boxplot 2: After Removing Outliers
sns.boxplot(data=df_clean_no_outliers, x='HeartStroke', y='BMI', color='purple', ax=axes[1])
axes[1].set_title('BMI vs HeartStroke - After Removing Outliers')
axes[1].set_xlabel('Heart Disease/Stroke')
axes[1].set_ylabel('BMI')

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



@finding:

The boxplot shows that individuals with heart disease or stroke tend to have slightly higher BMI on average. However, the difference is not dramatic, suggesting BMI may be a contributing factor, but not the strongest one.

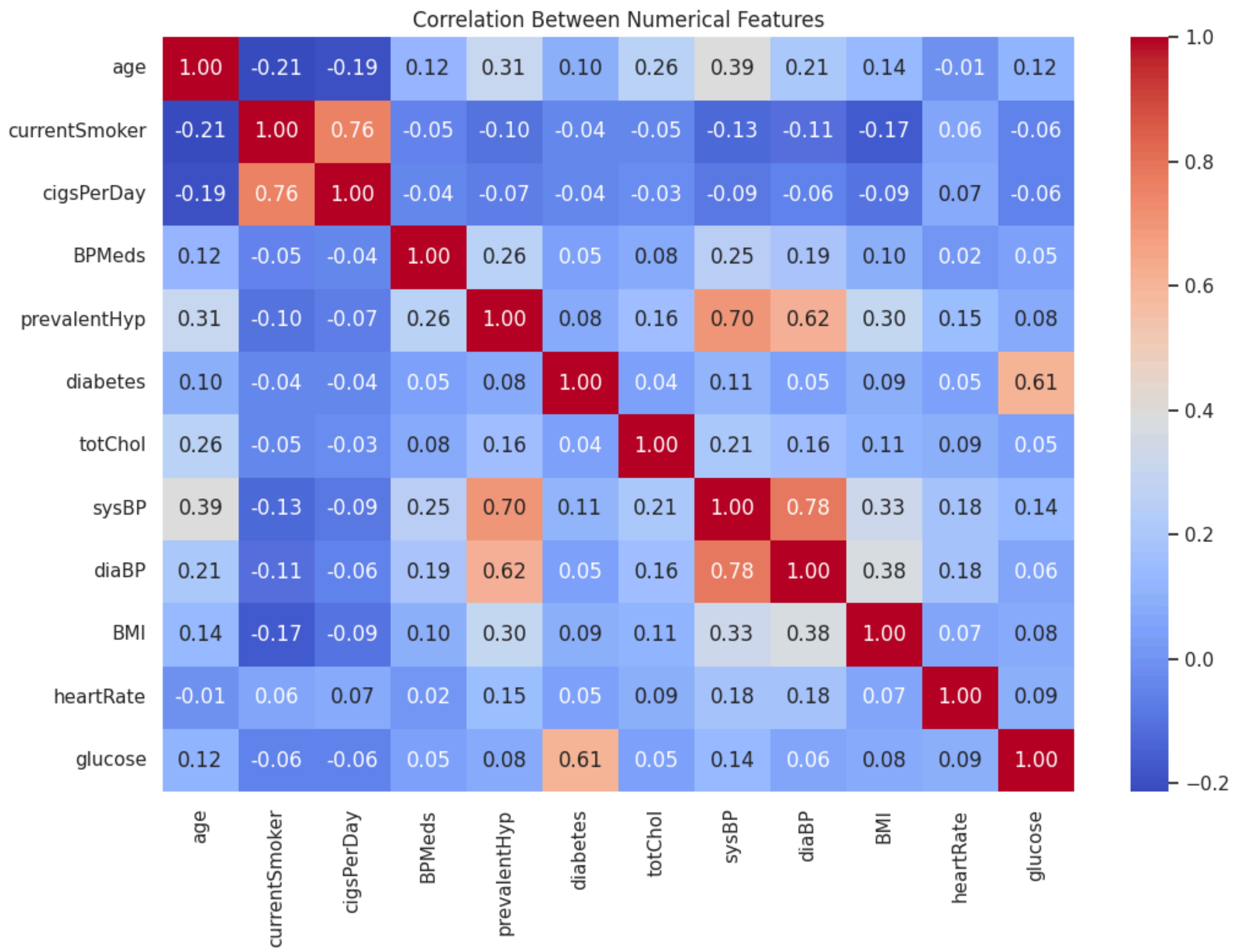
```
# 4.7 Age vs HeartStroke
sns.boxplot(data=df_clean, x='HeartStroke', y='age', color='lightgreen')
plt.title('Age Distribution by Heart Disease/Stroke')
plt.show()
```



@finding:

There is a clear trend: individuals who experienced heart disease or stroke tend to be older. This reinforces the role of age as a key risk factor in cardiovascular events.

```
# 4.8 Correlation Heatmap (Numerical Features)
plt.figure(figsize=(12, 8))
sns.heatmap(df_clean.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Between Numerical Features')
plt.show()
```



@finding:

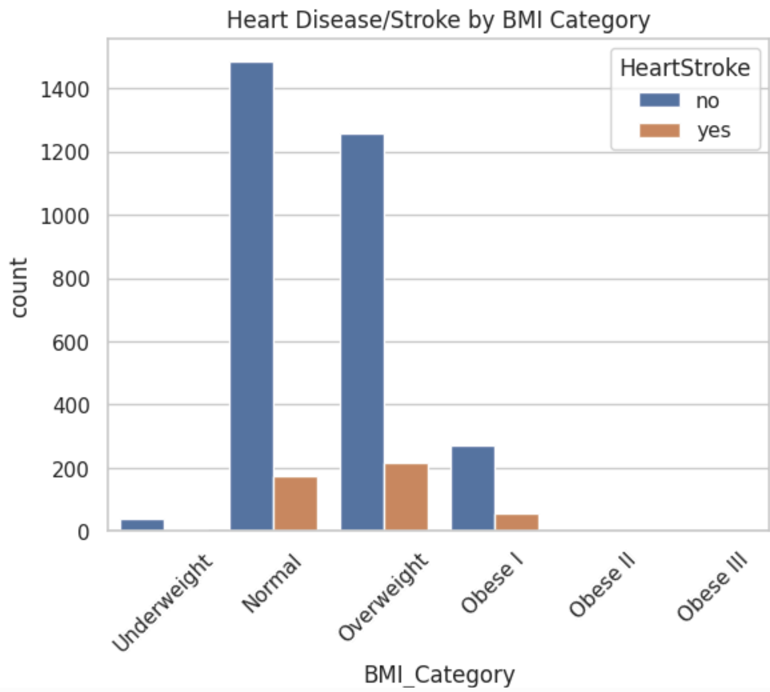
The heatmap highlights positive correlations between systolic blood pressure, BMI, and age with heart-related outcomes. It also shows that some variables like diastolic BP and heart rate are only weakly related to the target. This helps narrow down which variables may be most useful for prediction

```
# 4.9 Create BMI categories
def categorize_bmi(bmi):
    if bmi < 18.5:
        return 'Underweight'
    elif 18.5 <= bmi < 25:
        return 'Normal'
    elif 25 <= bmi < 30:
        return 'Overweight'
    elif 30 <= bmi < 35:
        return 'Obese I'
    elif 35 <= bmi < 40:
        return 'Obese II'
    else:
        return 'Obese III'

# Apply function to the BMI column
df_clean_no_outliers['BMI_Category'] = df_clean_no_outliers['BMI'].apply(categorize_bmi)

# Check the value counts
df_clean_no_outliers['BMI_Category'].value_counts()
```

	count
BMI_Category	
Normal	1660
Overweight	1478
Obese I	324
Underweight	45
Obese II	3

dtype: int64

```
# 4.9 BMI Category vs Heart Disease/Stroke
sns.countplot(data=df_clean_no_outliers, x='BMI_Category', hue='HeartStroke', order=[
    'Underweight', 'Normal', 'Overweight', 'Obese I', 'Obese II', 'Obese III'])
plt.title('Heart Disease/Stroke by BMI Category')
plt.xticks(rotation=45)
plt.show()
```



@finding:

People in the Overweight category appear to have higher risk than those at Normal weight.

Very high BMI (Obese II and III) might also increase risk, but the sample size is too small in the dataset to be confident.

Being "Normal" BMI doesn't completely protect against heart disease/stroke, other factors (like age, smoking, blood pressure) are likely at play.

5. Create a simple classification model to predict whether a person has heart disease/stroke using the available health features.

Baseline Model — Logistic Regression I will use:

Features (like age, BMI, smoking status, blood pressure, etc.)

Target: HeartStroke (convert to binary: yes -> 1, no -> 0)

```
# 5.1 Encode the Target Variable
# Convert target column to binary numeric values
df_clean_no_outliers['HeartStroke'] = df_clean_no_outliers['HeartStroke'].map({'yes': 1, 'no': 0})
```

```
# 5.2 One-Hot Encode Categorical Variables
# Drop BMI category (it's a derived feature, optional)
df_model = df_clean_no_outliers.drop(columns=['BMI_Category'])

# One-hot encoding for all categorical variables
df_model = pd.get_dummies(df_model, drop_first=True)
```

```
# Remove rows where the target is missing
df_model = df_model[df_model['HeartStroke'].notna()]

# Just to be safe: drop any remaining rows with any NaN
df_model = df_model.dropna()

# Double-Check Size Before Splitting
print("Data shape after cleaning:", df_model.shape)
```

```
Data shape after cleaning: (3510, 18)
```

```
# 5.3 Split the Data
X = df_model.drop('HeartStroke', axis=1)
y = df_model['HeartStroke']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

```
# To check again
print("Train size:", X_train.shape)
print("Test size:", X_test.shape)
print("Target distribution:", y_train.value_counts(normalize=True))
```

```
Train size: (2808, 17)
Test size: (702, 17)
Target distribution: HeartStroke
0    0.874288
1    0.125712
Name: proportion, dtype: float64
```

```
# The Data

X = df_model.drop('HeartStroke', axis=1)
y = df_model['HeartStroke']
```

```
# 5.4 Train Logistic Regression Model
from sklearn.linear_model import LogisticRegression
```