Basics

April 1, 2025

[2]:
```python
## Title: Heart Disease Prediction - Exploratory Data Analysis

# Description:
# This project aims to explore a dataset containing patient medical records
# to identify patterns and risk factors associated with heart disease.
# The goal is to analyze the data and prepare it for predictive modeling.

# Step 1: Import Necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 2: Load the Dataset
df = pd.read_csv("heart_disease_prediction.csv")

# Step 3: Display the First Five Rows of the Dataframe
print(df.head())

# Step 4: Print Dataset Shape (Number of Features and Observations)
print("Number of observations:", df.shape[0])
print("Number of features:", df.shape[1])
```

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
Number of observations: 918
Number of features: 12
```

[3]:

```python
# Step 5: Identify Data Types
print(df.dtypes)

# Step 6: Display Descriptive Statistics
print(df.describe())

# Observations:
# - The average age of patients can be found in the 'Age' column.
# - Checking for any extreme values in numerical features.
# - Identifying potential data inconsistencies.
# - Checking for missing values.
print("Missing values in each column:")
print(df.isnull().sum())
```

```
Age                 int64
Sex                object
ChestPainType      object
RestingBP           int64
Cholesterol         int64
FastingBS           int64
RestingECG         object
MaxHR               int64
ExerciseAngina     object
Oldpeak           float64
ST_Slope           object
HeartDisease        int64
dtype: object
              Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
count  918.000000  918.000000   918.000000  918.000000  918.000000
mean    53.510893  132.396514   198.799564    0.233115  136.809368
std      9.432617   18.514154   109.384145    0.423046   25.460334
min     28.000000    0.000000     0.000000    0.000000   60.000000
25%     47.000000  120.000000   173.250000    0.000000  120.000000
50%     54.000000  130.000000   223.000000    0.000000  138.000000
75%     60.000000  140.000000   267.000000    0.000000  156.000000
max     77.000000  200.000000   603.000000    1.000000  202.000000

          Oldpeak  HeartDisease
count  918.000000    918.000000
mean     0.887364      0.553377
std      1.066570      0.497414
min     -2.600000      0.000000
25%      0.000000      0.000000
50%      0.600000      1.000000
75%      1.500000      1.000000
max      6.200000      1.000000
Missing values in each column:
```

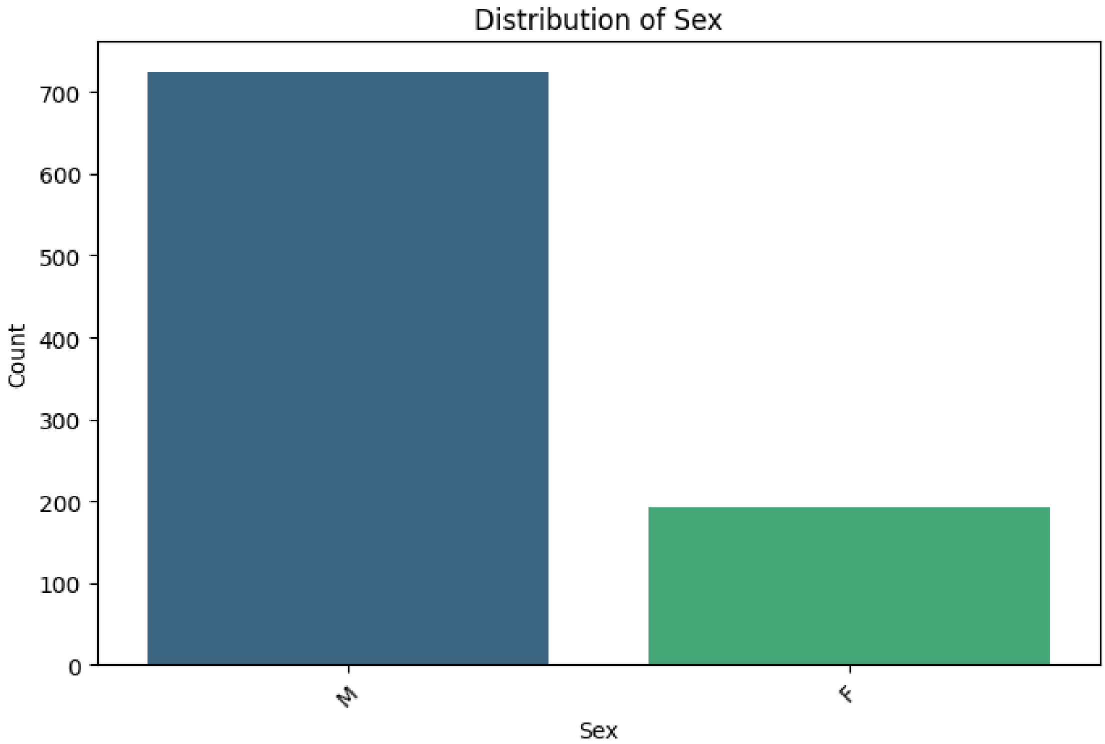
```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

[4]:
```
# Step 7: Check for Missing Values
print("Missing values in each column:")
print(df.isnull().sum())

# Step 8: Visualizing Categorical Features
categorical_columns = ["Sex", "ChestPainType", "FastingBS", "RestingECG",␣
↪"ExerciseAngina", "ST_Slope", "HeartDisease"]

for col in categorical_columns:
    plt.figure(figsize=(8, 5))
    sns.countplot(x=df[col], palette="viridis")
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.xticks(rotation=45)
    plt.show()

# Observations:
# - Number of male vs. female patients.
# - Distribution of chest pain types.
# - Frequency of different resting ECG results.

# Step 9: Visualizing Categorical Features Grouped by HeartDisease
for col in categorical_columns:
    plt.figure(figsize=(8, 5))
    sns.countplot(x=df[col], hue=df["HeartDisease"], palette="coolwarm")
    plt.title(f"Distribution of {col} by Heart Disease")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.legend(title="Heart Disease", labels=["No", "Yes"])
    plt.xticks(rotation=45)
    plt.show()
```

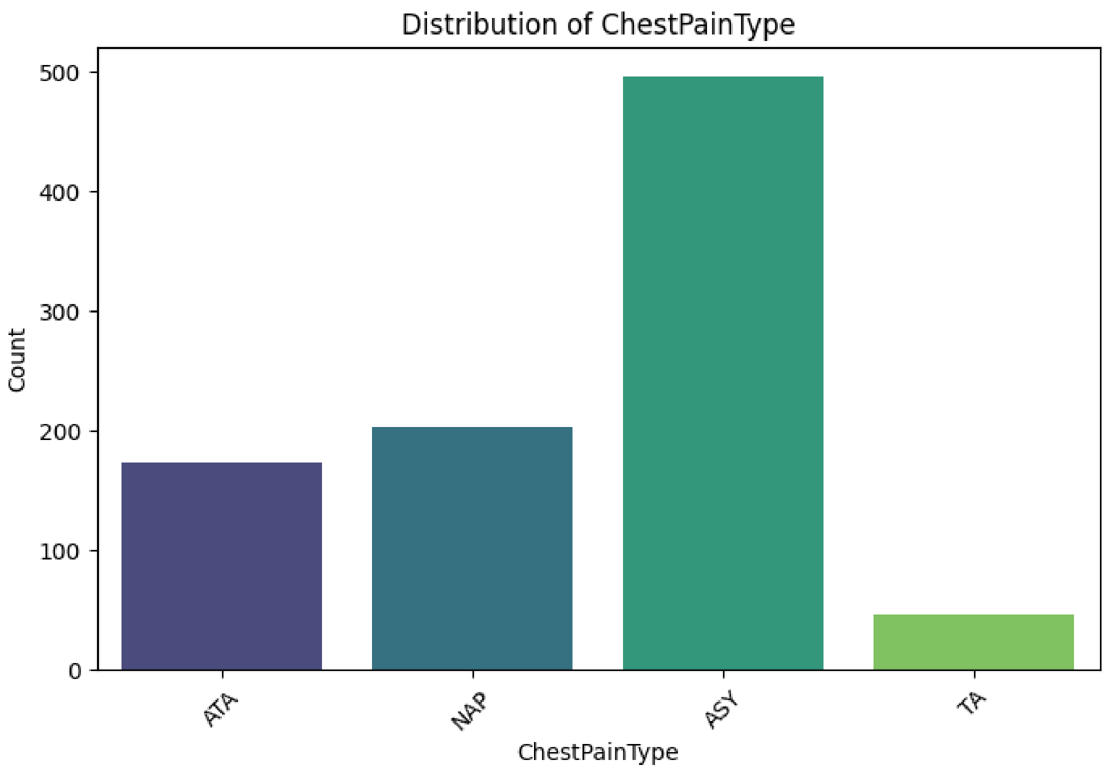
```
# Observations:
# - Which category of ChestPainType has a higher count for patients with heart␣
↪disease?
# - How FastingBS relates to heart disease prevalence?
```

```
Missing values in each column:
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

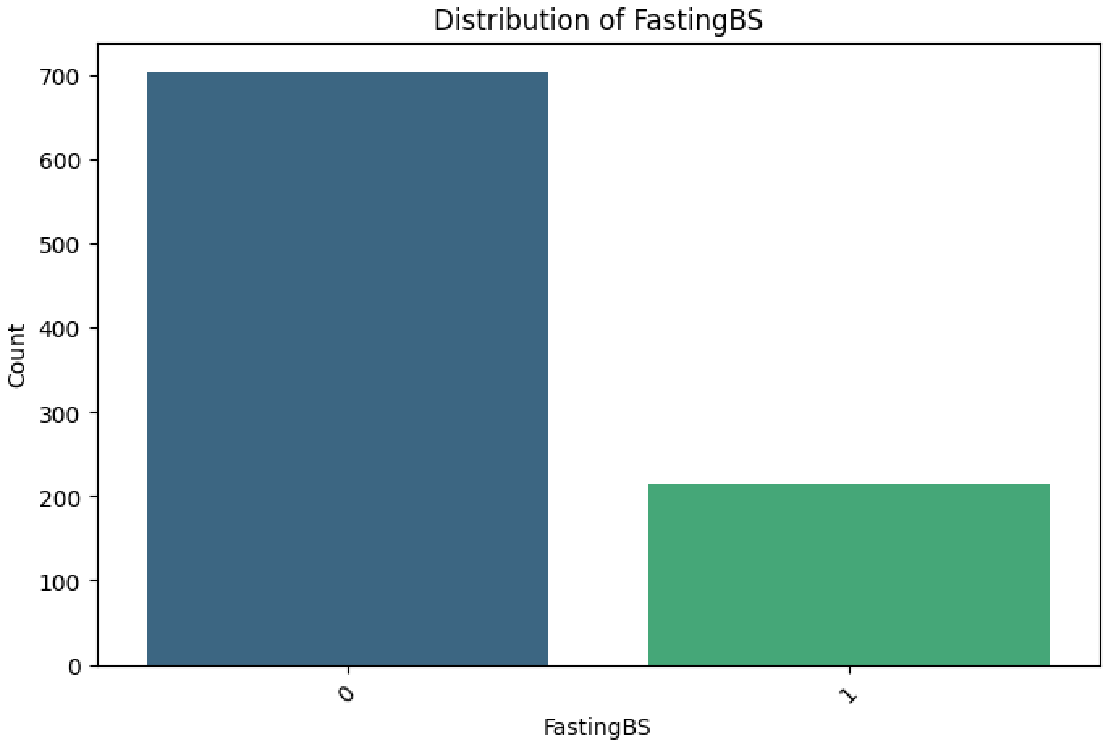
```
/tmp/ipykernel_29/3334388913.py:10: FutureWarning:

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

  sns.countplot(x=df[col], palette="viridis")
```



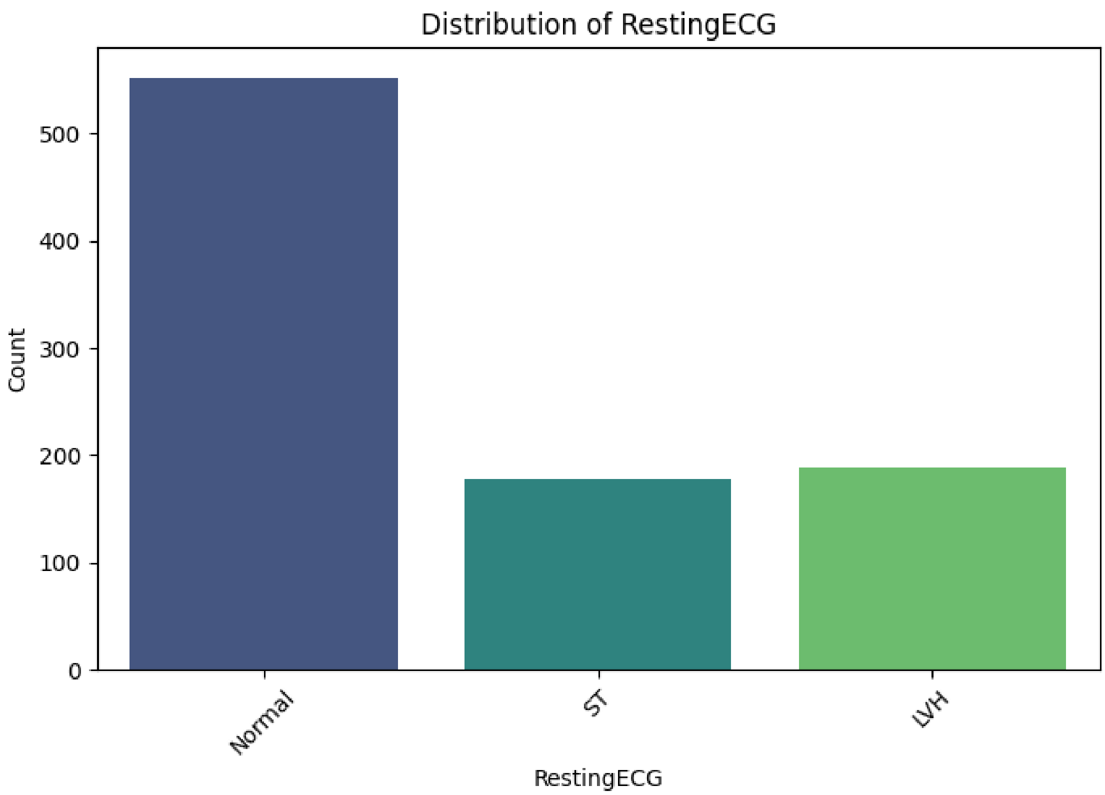

```
/tmp/ipykernel_29/3334388913.py:10: FutureWarning:

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

  sns.countplot(x=df[col], palette="viridis")
```



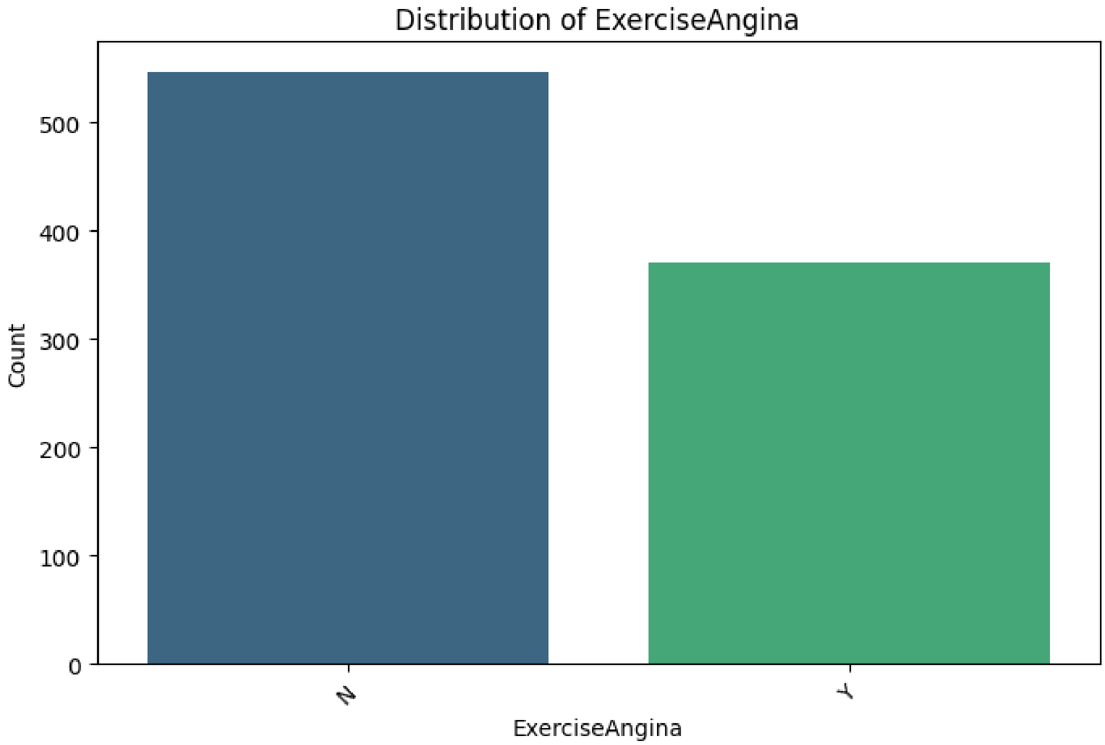

```
/tmp/ipykernel_29/3334388913.py:10: FutureWarning:

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

  sns.countplot(x=df[col], palette="viridis")
```



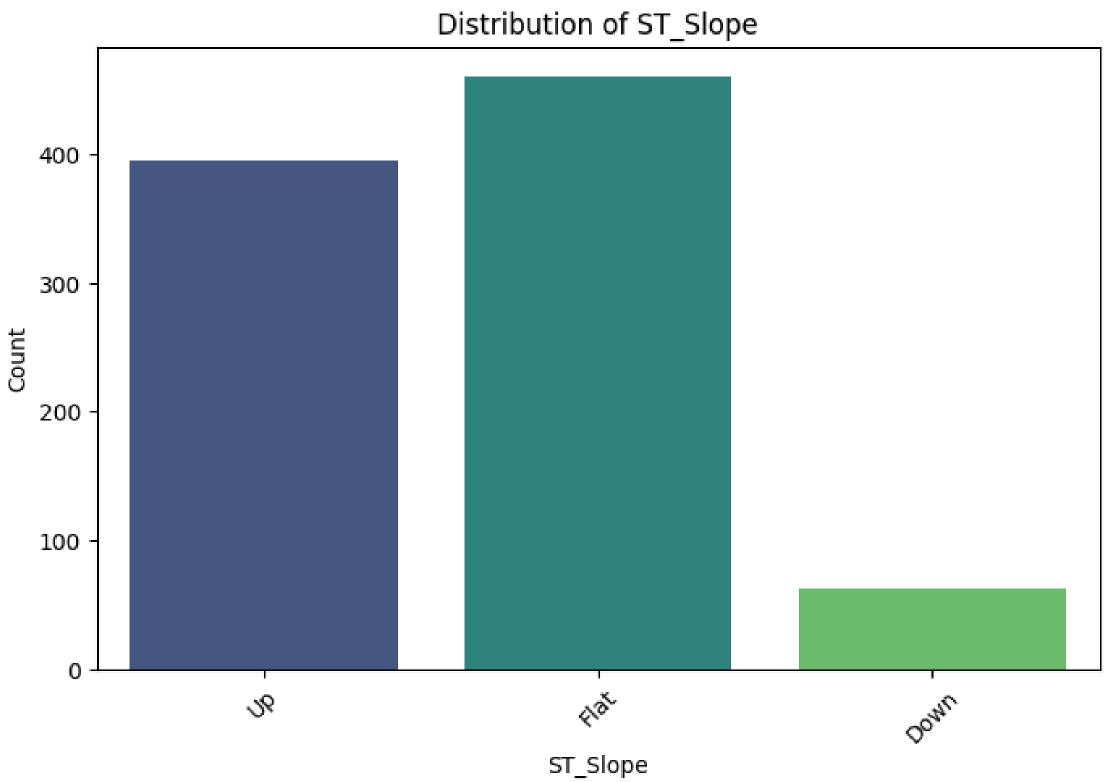

```
/tmp/ipykernel_29/3334388913.py:10: FutureWarning:

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

  sns.countplot(x=df[col], palette="viridis")
```



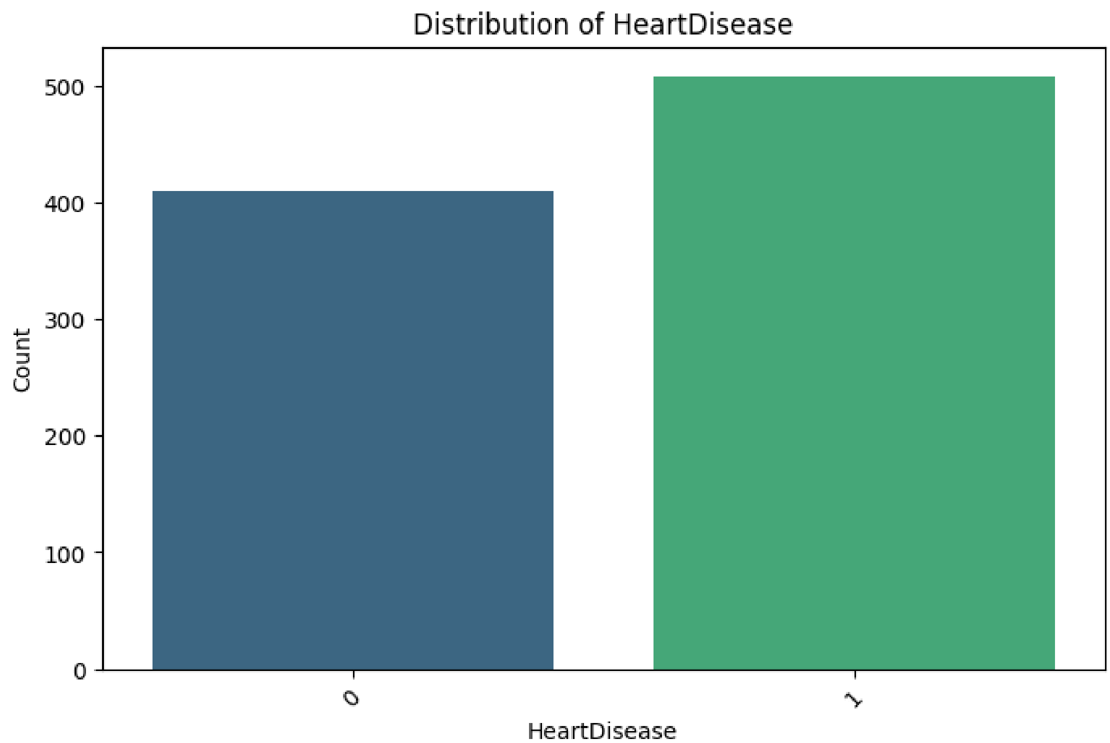

```
/tmp/ipykernel_29/3334388913.py:10: FutureWarning:

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

  sns.countplot(x=df[col], palette="viridis")
```

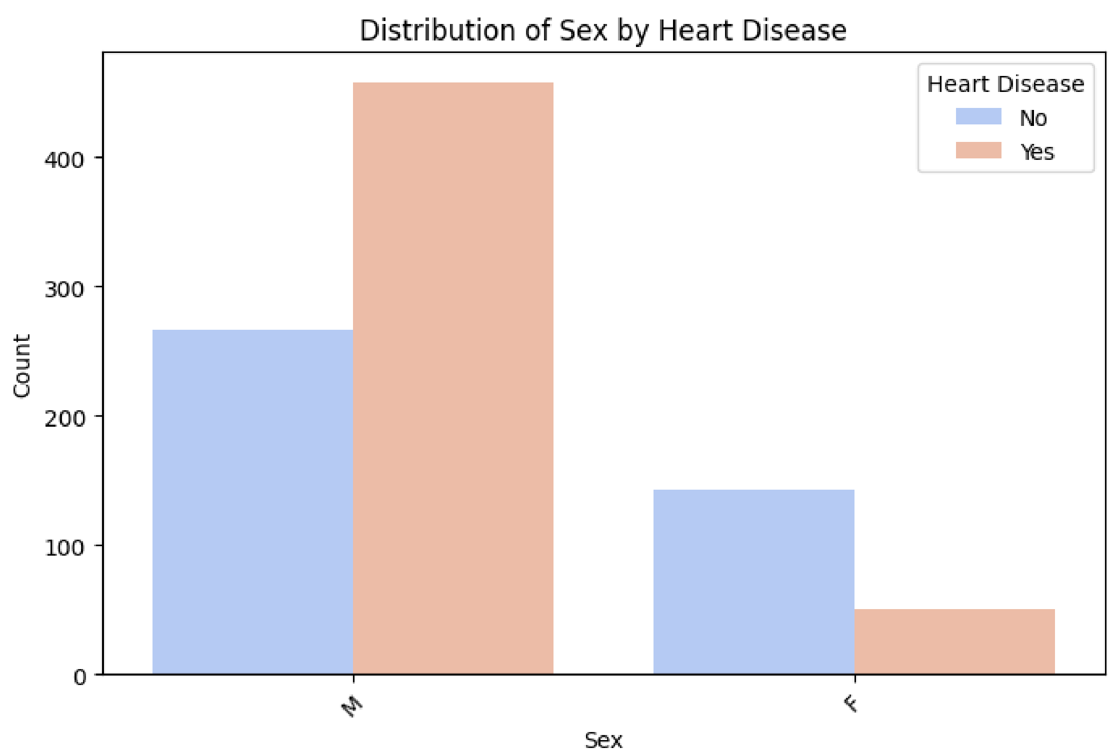


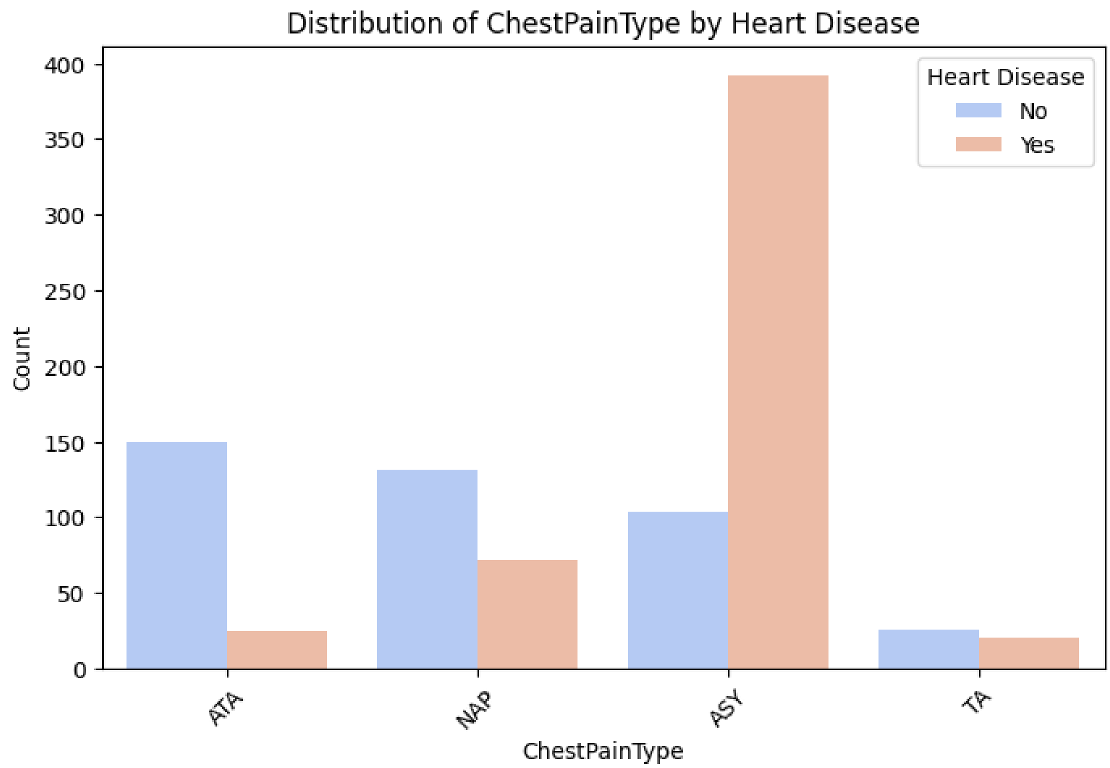

```
/tmp/ipykernel_29/3334388913.py:10: FutureWarning:

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

  sns.countplot(x=df[col], palette="viridis")
```



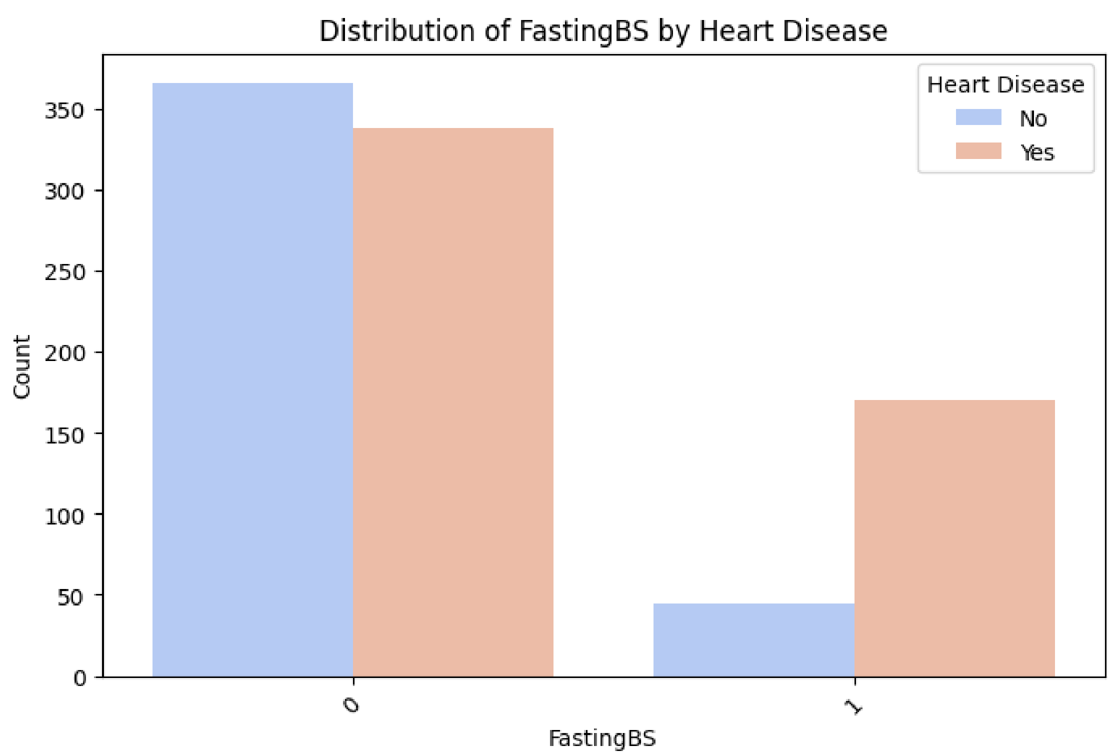

```
/tmp/ipykernel_29/3334388913.py:10: FutureWarning:

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

  sns.countplot(x=df[col], palette="viridis")
```


Distribution of HeartDisease
Count
500
400
300
200
100
0
0
1
HeartDisease

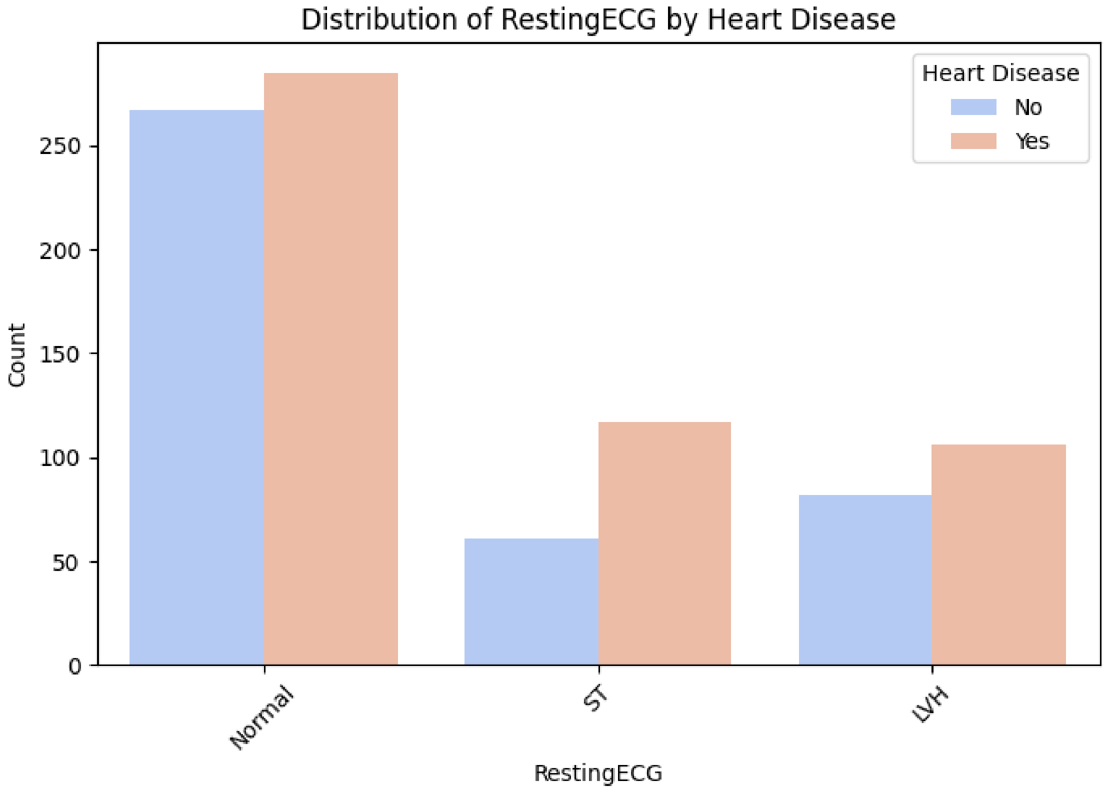

Distribution of Sex by Heart Disease
Heart Disease
No
Yes
Count
400
300
200
100
0
M
F
Sex

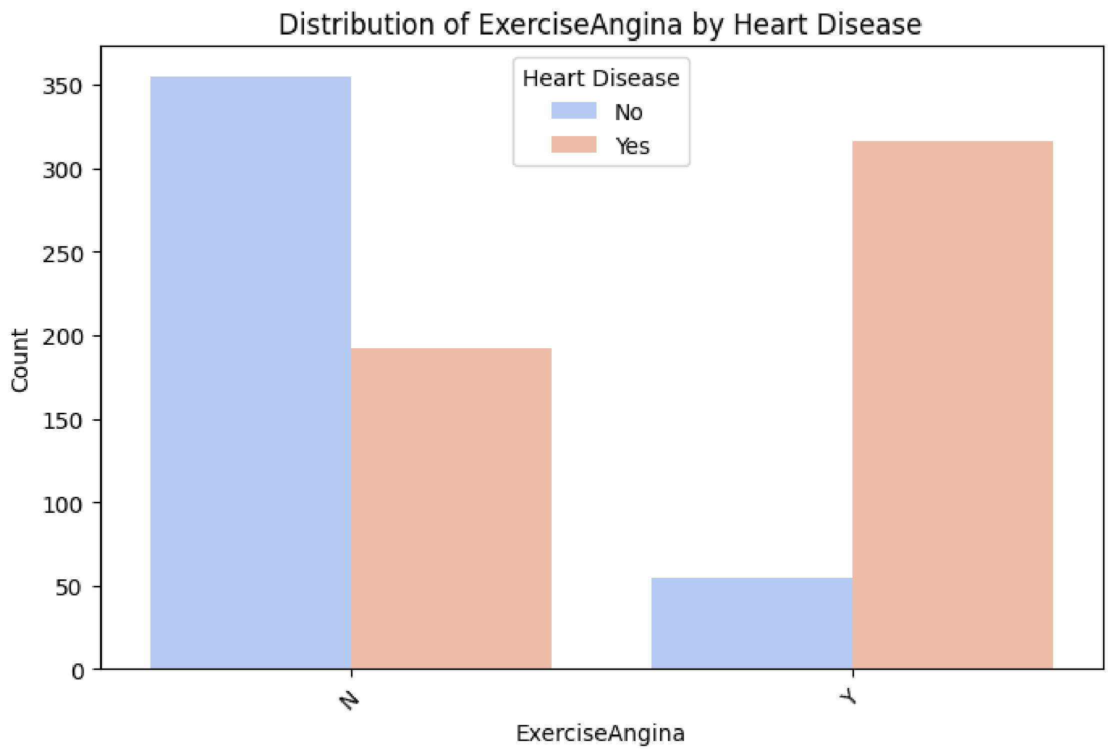


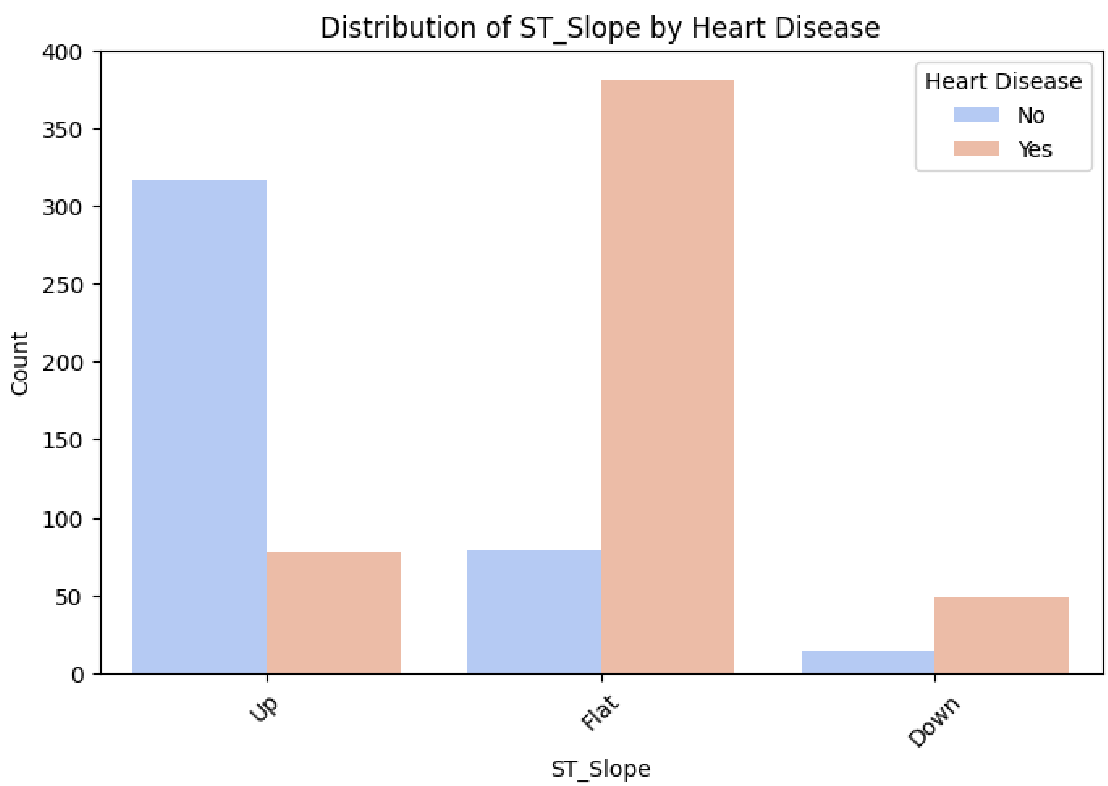
Distribution of ChestPainType by Heart Disease
Heart Disease
No
Yes
Count
400
350
300
250
200
150
100
50
0
ATA
NAP
ASY
TA
ChestPainType

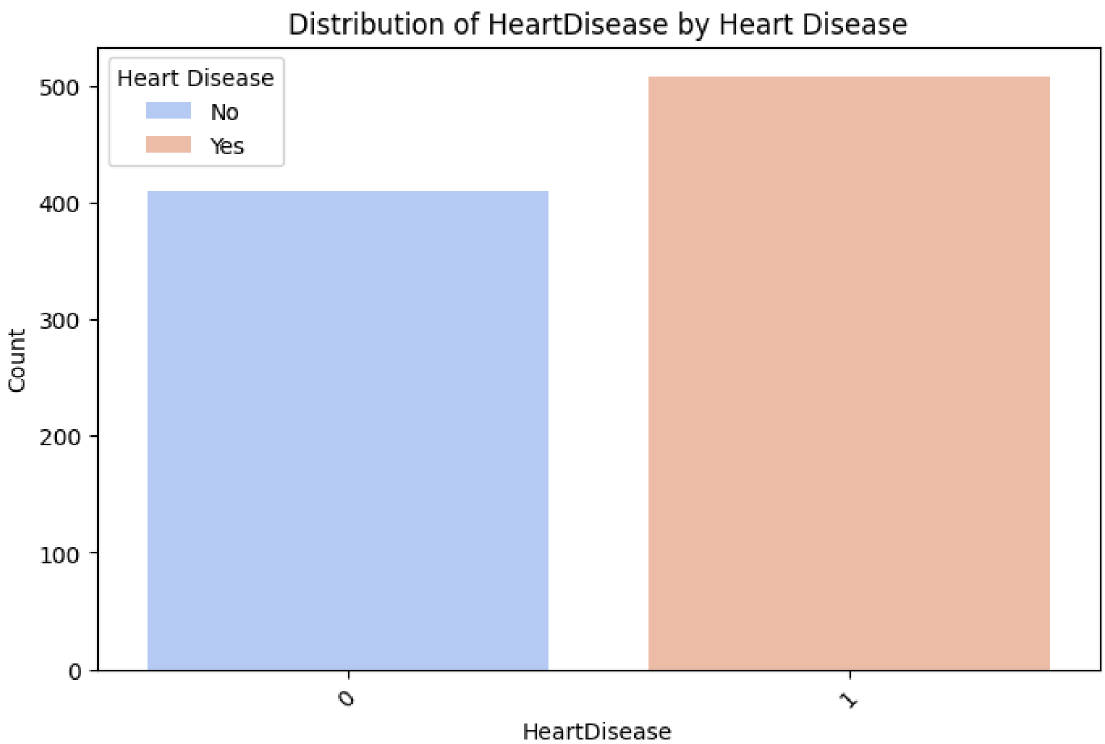


Distribution of FastingBS by Heart Disease
Heart Disease
No
Yes
Count
350
300
250
200
150
100
50
0
0
1
FastingBS


Distribution of RestingECG by Heart Disease
Heart Disease
No
Yes
Count
0
50
100
150
200
250
Normal
ST
LVH
RestingECG


Distribution of ExerciseAngina by Heart Disease
Heart Disease
No
Yes
Count
0
50
100
150
200
250
300
350
N
Y
ExerciseAngina


Distribution of ST_Slope by Heart Disease
Heart Disease
No
Yes
Count
0
50
100
150
200
250
300
350
400
Up
Flat
Down
ST_Slope



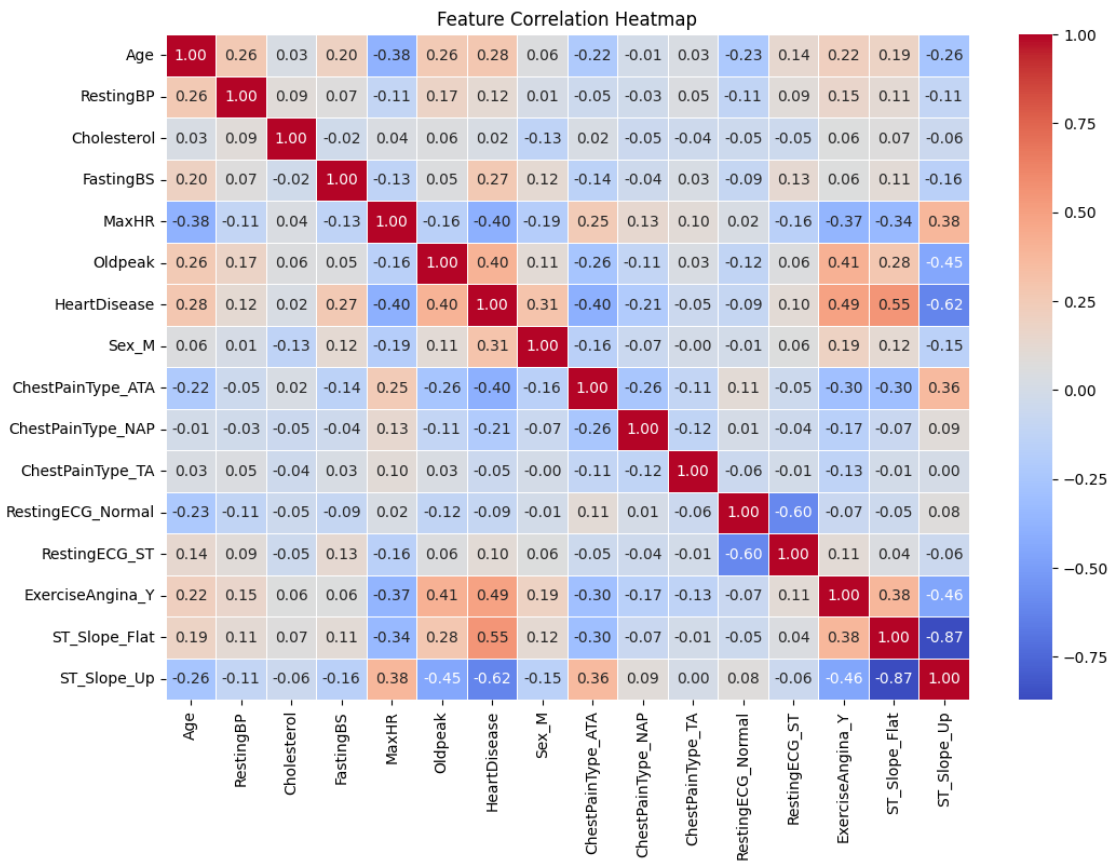
[5]:

```
# Step 10: Handling Incorrect Zero Values
zero_bp_count = (df["RestingBP"] == 0).sum()
zero_cholesterol_count = (df["Cholesterol"] == 0).sum()
print(f"Number of rows with RestingBP = 0: {zero_bp_count}")
print(f"Number of rows with Cholesterol = 0: {zero_cholesterol_count}")

# Impute zero values with median based on HeartDisease category
def impute_median(df, column):
    for heart_disease_value in df["HeartDisease"].unique():
        median_value = df[df["HeartDisease"] == heart_disease_value][column].
 ↪median()
        df.loc[(df[column] == 0) & (df["HeartDisease"] == heart_disease_value),␣
 ↪column] = median_value
    return df

df = impute_median(df, "RestingBP")
df = impute_median(df, "Cholesterol")

print("Updated dataset after handling zero values:")
print(df.describe())
```

```
Number of rows with RestingBP = 0: 1
Number of rows with Cholesterol = 0: 172
Updated dataset after handling zero values:
              Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
count  918.000000  918.000000   918.000000  918.000000  918.000000
mean    53.510893  132.540305   239.675381    0.233115  136.809368
std      9.432617   17.989941    54.328249    0.423046   25.460334
min     28.000000   80.000000    85.000000    0.000000   60.000000
25%     47.000000  120.000000   214.000000    0.000000  120.000000
50%     54.000000  130.000000   225.000000    0.000000  138.000000
75%     60.000000  140.000000   267.000000    0.000000  156.000000
max     77.000000  200.000000   603.000000    1.000000  202.000000

          Oldpeak  HeartDisease
count  918.000000    918.000000
mean     0.887364      0.553377
std      1.066570      0.497414
min     -2.600000      0.000000
25%      0.000000      0.000000
50%      0.600000      1.000000
75%      1.500000      1.000000
max      6.200000      1.000000
```

[6]:

```
# Step 11: Convert Categorical Features into Dummy Variables
df = pd.get_dummies(df, columns=["Sex", "ChestPainType", "RestingECG",
 ↪"ExerciseAngina", "ST_Slope"], drop_first=True)

# Step 12: Create Pearson Correlation Heatmap
plt.figure(figsize=(12, 8))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()

# Observations:
# - Identify features that have a strong correlation with HeartDisease.
# - Determine any multicollinearity issues between features.
```



[7]:
```python
# Step 13: Feature Selection Based on Correlation
target = "HeartDisease"
selected_features = ["Oldpeak", "MaxHR", "ChestPainType_ATA", "Sex_M",␣
↪"ExerciseAngina_Y", "ST_Slope_Flat", "ST_Slope_Up"]

# Step 14: Split the dataset into training and validation sets
X = df[selected_features]
y = df[target]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Step 15: Train and Evaluate k-NN Classifier for Each Feature
k = 5  # Number of neighbors
for feature in selected_features:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train[[feature]], y_train)
    y_pred = knn.predict(X_val[[feature]])
    accuracy = accuracy_score(y_val, y_pred)
    print(f"Accuracy using {feature}: {accuracy:.2f}")
```

```
# Observations:
# - Identify which feature contributes most to predicting heart disease.
# - Experiment with different numbers of neighbors for improved accuracy.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [7], in <cell line: 0>()
      6 X = df[selected_features]
      7 y = df[target]
----> 8 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
     10 # Step 15: Train and Evaluate k-NN Classifier for Each Feature
     11 k = 5  # Number of neighbors

NameError: name 'train_test_split' is not defined
```

[8]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Step 13: Feature Selection Based on Correlation
target = "HeartDisease"
selected_features = ["Oldpeak", "MaxHR", "ChestPainType_ATA", "Sex_M",␣
 ↪"ExerciseAngina_Y", "ST_Slope_Flat", "ST_Slope_Up"]

# Step 14: Split the dataset into training and validation sets
X = df[selected_features]
y = df[target]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Step 15: Train and Evaluate k-NN Classifier for Each Feature
k = 5  # Number of neighbors
for feature in selected_features:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train[[feature]], y_train)
    y_pred = knn.predict(X_val[[feature]])
    accuracy = accuracy_score(y_val, y_pred)
    print(f"Accuracy using {feature}: {accuracy:.2f}")
```

```
Accuracy using Oldpeak: 0.70
Accuracy using MaxHR: 0.61
Accuracy using ChestPainType_ATA: 0.42
Accuracy using Sex_M: 0.42
Accuracy using ExerciseAngina_Y: 0.66
Accuracy using ST_Slope_Flat: 0.75
```

Accuracy using ST_Slope_Up: 0.80

[9]:
```python
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Step 1: Create a MinMaxScaler object
scaler = MinMaxScaler()

# Step 2: Fit and transform the training set features
X_train_scaled = scaler.fit_transform(X_train)

# Step 3: Transform the validation set features
X_val_scaled = scaler.transform(X_val)

# Step 4: Create and train a k-NN classifier using the scaled features
k = 5  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_scaled, y_train)

# Step 5: Evaluate the model on the validation set
y_pred_scaled = knn.predict(X_val_scaled)

# Step 6: Calculate and print the accuracy
accuracy_scaled = accuracy_score(y_val, y_pred_scaled)
print(f"Accuracy using all scaled features: {accuracy_scaled:.2f}")
```

Accuracy using all scaled features: 0.81

[10]:
```python
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score

# Step 1: Split the dataset into training and test sets
X = df[selected_features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Step 2: Scale the training set using MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 3: Define the parameter grid to search over
param_grid = {
```

```python
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

# Step 4: Instantiate a k-NN model
knn = KNeighborsClassifier()

# Step 5: Create the GridSearchCV instance
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid,␣
 ↪scoring='accuracy', cv=5)

# Step 6: Fit the GridSearchCV instance on the scaled training data
grid_search.fit(X_train_scaled, y_train)

# Step 7: Print out the best score and best parameters
print(f"Best accuracy: {grid_search.best_score_:.2f}")
print(f"Best parameters: {grid_search.best_params_}")

# Step 8: Evaluate the model on the test set using the best parameters
best_knn = grid_search.best_estimator_
y_pred_test = best_knn.predict(X_test_scaled)

# Step 9: Print the accuracy on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
print(f"Test accuracy: {test_accuracy:.2f}")
```

```
Best accuracy: 0.84
Best parameters: {'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'uniform'}
Test accuracy: 0.81
```

[11]:
```python
# Step 1: Scale the test set using the same scaler (do not fit, just transform)
X_test_scaled = scaler.transform(X_test)

# Step 2: Predict using the best model from GridSearchCV
best_knn = grid_search.best_estimator_
y_pred_test = best_knn.predict(X_test_scaled)

# Step 3: Calculate and print the accuracy on the test set
test_accuracy = accuracy_score(y_test, y_pred_test)
print(f"Test accuracy: {test_accuracy:.2f}")
```

```
Test accuracy: 0.81
```

[ ]:
```python

```