

# Heart Disease Prediction Using Machine Learning

## Abstract

Cardiovascular diseases (CVDs) remain the leading cause of mortality worldwide, responsible for nearly 17.9 million deaths annually. Among CVDs, heart disease and heart failure are major contributors. Early detection and intervention are critical in reducing fatalities. However, traditional diagnostic methods such as electrocardiograms (ECG), echocardiography, stress tests, and blood biomarkers often require specialized equipment, expert evaluation, and high costs, making them inaccessible to many populations. With the rise of machine learning (ML) in healthcare, automated heart disease prediction models offer efficient, cost-effective, and scalable solutions.

This study investigates the application of four ML models—K-Nearest Neighbors (KNN), Logistic Regression, Decision Tree, and Random Forest classifiers—to predict heart disease using the Heart Failure Prediction Dataset from Kaggle (Fedesoriano, 2021). The dataset consists of 918 patient records with 12 clinical features, including age, cholesterol levels, blood pressure, heart rate, and chest pain type. The data was preprocessed through missing value imputation, categorical encoding, and feature scaling to improve model performance.

Model performance was evaluated using accuracy, precision, recall, F1-score, and confusion matrices. Among the models:

Random Forest achieved the highest accuracy (94%), demonstrating its robustness and ability to handle complex data relationships.

Logistic Regression performed well (93%), making it an interpretable and clinically applicable model.

Decision Tree achieved 83% accuracy, but suffered from overfitting.

KNN had the lowest accuracy (74%), struggling with high-dimensional data.

The confusion matrix analysis showed that Random Forest had the lowest misclassification rate, making it the best candidate for heart disease prediction.

The study concludes that machine learning can serve as a valuable decision-support tool in cardiology, aiding early diagnosis and risk stratification. Future research should explore deep learning techniques, feature engineering, and real-time patient monitoring systems to improve accuracy and usability in clinical settings. Integration with electronic health records (EHRs) and wearable technology can further enhance early detection, reduce misdiagnosis, and improve patient outcomes.

# 1. Introduction

## 1.1 Heart Disease and Heart Failure

Heart disease refers to a range of cardiovascular conditions affecting the heart's structure and function, including coronary artery disease (CAD), arrhythmias, heart failure, and valvular disorders. Among these, CAD is the most prevalent, caused by the buildup of plaque in the arteries, leading to reduced blood supply and increasing the risk of heart attacks and strokes.

Heart failure, on the other hand, occurs when the heart cannot pump blood efficiently, resulting in fluid accumulation in the lungs and other tissues. This condition can be chronic or acute, and it significantly reduces a patient's quality of life and life expectancy. Common symptoms include shortness of breath, fatigue, swelling in the legs, and persistent coughing.

## 1.2 Current Diagnostic Technologies

Modern medicine relies on a variety of diagnostic tools to detect heart disease, including:

**Electrocardiograms (ECG):** Records electrical activity to detect arrhythmias and abnormal heart function.

**Echocardiography:** Uses ultrasound to examine heart structure and blood flow.

**Cardiac MRI and CT Scans:** Provide detailed imaging of heart tissues and arteries.

**Blood Tests:** Measure biomarkers like troponin, cholesterol, and glucose levels.

While these techniques are effective, they have limitations such as high costs, accessibility issues, and reliance on expert interpretation. This creates a need for automated, scalable solutions that can assist in early disease detection.

## 1.3 Machine Learning in Heart Disease Prediction

Machine learning (ML) has emerged as a transformative technology in healthcare, offering data-driven predictive models that can analyze vast patient records and detect disease patterns. ML can enhance early diagnosis, reduce misdiagnosis, and optimize treatment decisions.

This study applies four ML algorithms—KNN, Logistic Regression, Decision Tree, and Random Forest—to predict heart disease based on clinical attributes. The goal is to evaluate their effectiveness and potential for real-world application.

## 2. Methodology

### Data Collection and Preprocessing

The dataset was sourced from Kaggle (Fedesoriano, Heart Failure Prediction Dataset), containing 918 patient records with 12 features, including:

Demographic Data: Age, Sex

Clinical Parameters: Blood pressure, Cholesterol, Fasting Blood Sugar

Exercise-Related Factors: Maximum Heart Rate, Exercise-Induced Angina

ECG Readings: Resting ECG, ST Slope

Target Variable: Presence of Heart Disease (1 = Yes, 0 = No)

## Data Preprocessing Steps

Missing Values: Handled using KNN Imputation.

Categorical Encoding: Converted categorical data using One-Hot Encoding.

Feature Scaling: Applied MinMaxScaler to normalize numerical values.

Train-Test Split: Divided into 80% training and 20% testing data.

## Libraries and Tools Used

To implement the machine learning models, the following Python libraries were used:

Pandas – For data manipulation and preprocessing

NumPy – For numerical operations

Matplotlib & Seaborn – For data visualization (histograms, heatmaps, and correlation plots)

Scikit-learn (sklearn) – For machine learning model implementation, including:

`train_test_split` (to split the dataset)

`StandardScaler` & `MinMaxScaler` (for feature scaling)

`KNeighborsClassifier` (KNN model)

`LogisticRegression` (Logistic Regression model)

`DecisionTreeClassifier` (Decision Tree model)

`RandomForestClassifier` (Random Forest model)

`classification_report` (for model performance evaluation)

`confusion_matrix` (to assess misclassifications)

`MinMaxScaler` – Normalizes numerical features to improve model accuracy and prevent scale bias.

`LabelEncoder` – Converts categorical variables (e.g., chest pain type, sex) into a numerical format suitable for machine learning models.

`KNNImputer` – Handles missing values using the K-Nearest Neighbors approach.


```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
In [2]: df = pd.read_csv('Downloads/heart failure detection/heart.csv')
```

```
In [3]: df.head()
```

```
Out[3]:
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	1
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	1
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	1



```
In [4]: df.shape
```

```
Out[4]: (918, 12)
```

```
In [5]: df.dtypes
```

```
Out[5]: 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
```

## 2.1 Exploratory Data Analysis (EDA)

EDA was performed to understand data patterns and relationships:

Age Distribution Analysis: Most heart disease cases occurred in individuals aged 50 and above.

Cholesterol Levels: Higher cholesterol levels were strongly correlated with heart disease presence.

Pain Type Analysis: Asymptomatic chest pain (ASY) was the most common indicator of heart disease.

Feature Correlation Analysis: A heatmap showed strong correlations between cholesterol, age, and ST slope with heart disease.

### Visualizations:

Bar plots to analyze the distribution of categorical variables.

Histograms for age and cholesterol distribution.

A heatmap to observe correlations among features.

```
In [6]: df.describe()
```

Out[6]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
<b>count</b>	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
<b>mean</b>	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
<b>std</b>	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
<b>min</b>	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
<b>25%</b>	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
<b>50%</b>	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
<b>75%</b>	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
<b>max</b>	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

The average age of the patients is around 54 years old, indicating that the dataset may be biased towards middle-aged individuals.

In [8]: `df.info()`



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Age                    918 non-null    int64
1   Sex                    918 non-null    object
2   ChestPainType          918 non-null    object
3   RestingBP              918 non-null    int64
4   Cholesterol             918 non-null    int64
5   FastingBS              918 non-null    int64
6   RestingECG             918 non-null    object
7   MaxHR                  918 non-null    int64
8   ExerciseAngina         918 non-null    object
9   Oldpeak                918 non-null    float64
10  ST_Slope               918 non-null    object
11  HeartDisease           918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

```
In [9]: df.duplicated().sum()
```

```
Out[9]: 0
```

```
In [10]: df['RestingBP'].replace(0, np.nan, inplace=True)
df['Cholesterol'].replace(0, np.nan, inplace=True)
```

```
In [11]: df.isna().sum()
```

```
Out[11]: Age          0
        Sex          0
        ChestPainType 0
        RestingBP     1
        Cholesterol   172
        FastingBS     0
        RestingECG    0
        MaxHR         0
        ExerciseAngina 0
        Oldpeak       0
        ST_Slope      0
        HeartDisease  0
        dtype: int64
```

```
In [12]: df[['RestingBP', 'Cholesterol']].isna().mean() * 100
```

```
Out[12]: RestingBP      0.108932
        Cholesterol    18.736383
        dtype: float64
```

```
In [13]: df[df["Cholesterol"].isna()]
```

Out[13]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
<b>293</b>	65	M	ASY	115.0	NaN	0	Normal	93	Y	0.0	Flat	1
<b>294</b>	32	M	TA	95.0	NaN	1	Normal	127	N	0.7	Up	1
<b>295</b>	61	M	ASY	105.0	NaN	1	Normal	110	Y	1.5	Up	1
<b>296</b>	50	M	ASY	145.0	NaN	1	Normal	139	Y	0.7	Flat	1
<b>297</b>	57	M	ASY	110.0	NaN	1	ST	131	Y	1.4	Up	1
...	...	...	...	...	...	...	...	...	...	...	...	...
<b>514</b>	43	M	ASY	122.0	NaN	0	Normal	120	N	0.5	Up	1
<b>515</b>	63	M	NAP	130.0	NaN	1	ST	160	N	3.0	Flat	1
<b>518</b>	48	M	NAP	102.0	NaN	1	ST	110	Y	1.0	Down	1
<b>535</b>	56	M	ASY	130.0	NaN	0	LVH	122	Y	1.0	Flat	1
<b>536</b>	62	M	NAP	133.0	NaN	1	ST	119	Y	1.2	Flat	1

172 rows × 12 columns

In [14]: `print(df.columns)`

```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
      'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
      'HeartDisease'],
      dtype='object')
```

For each categorical column, including HeartDisease and FastingBS, you can create a bar chart to display the number of rows for each category.

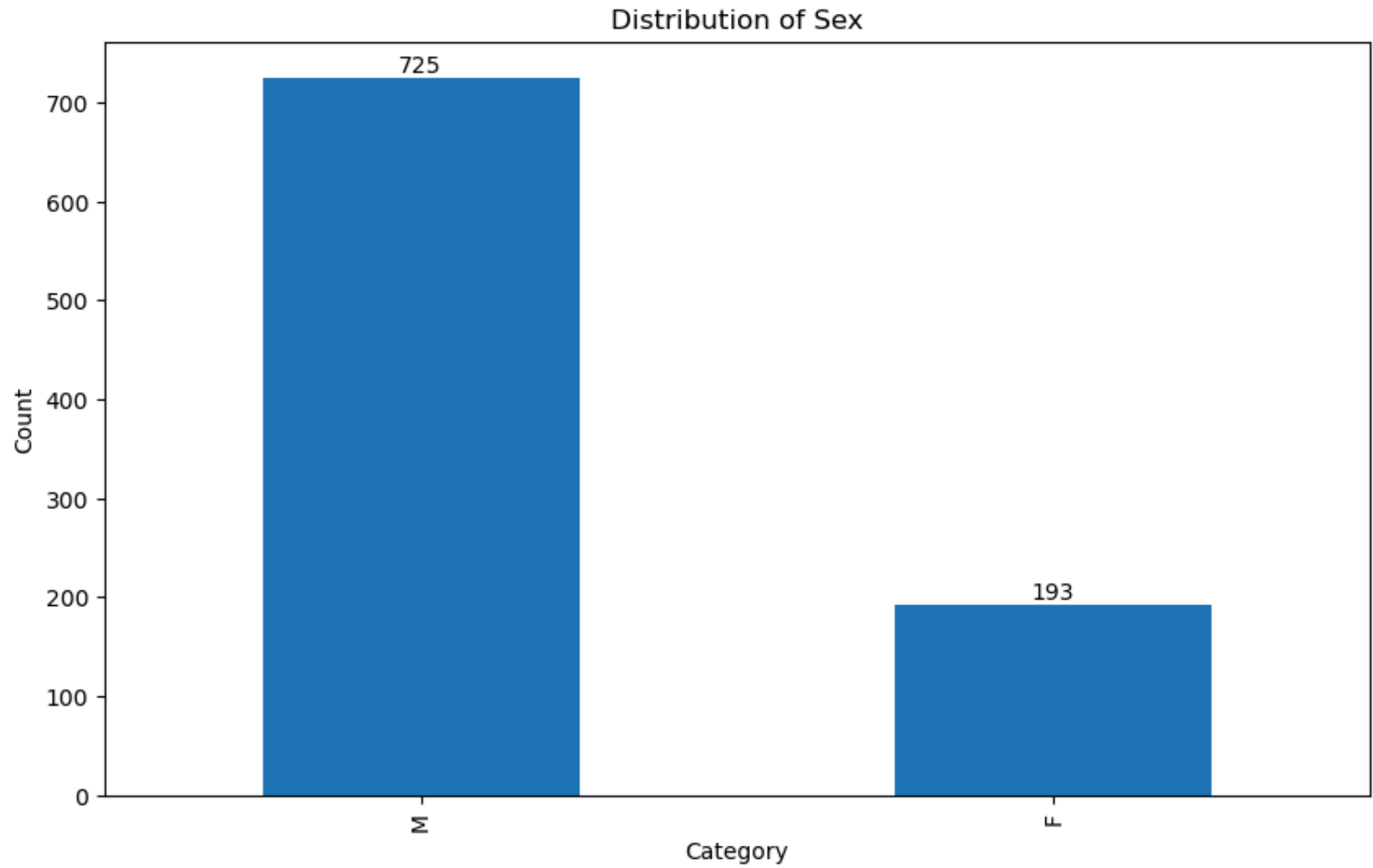
First, you need to identify the categorical columns. Assuming df is your DataFrame, you can select categorical columns by their data type.

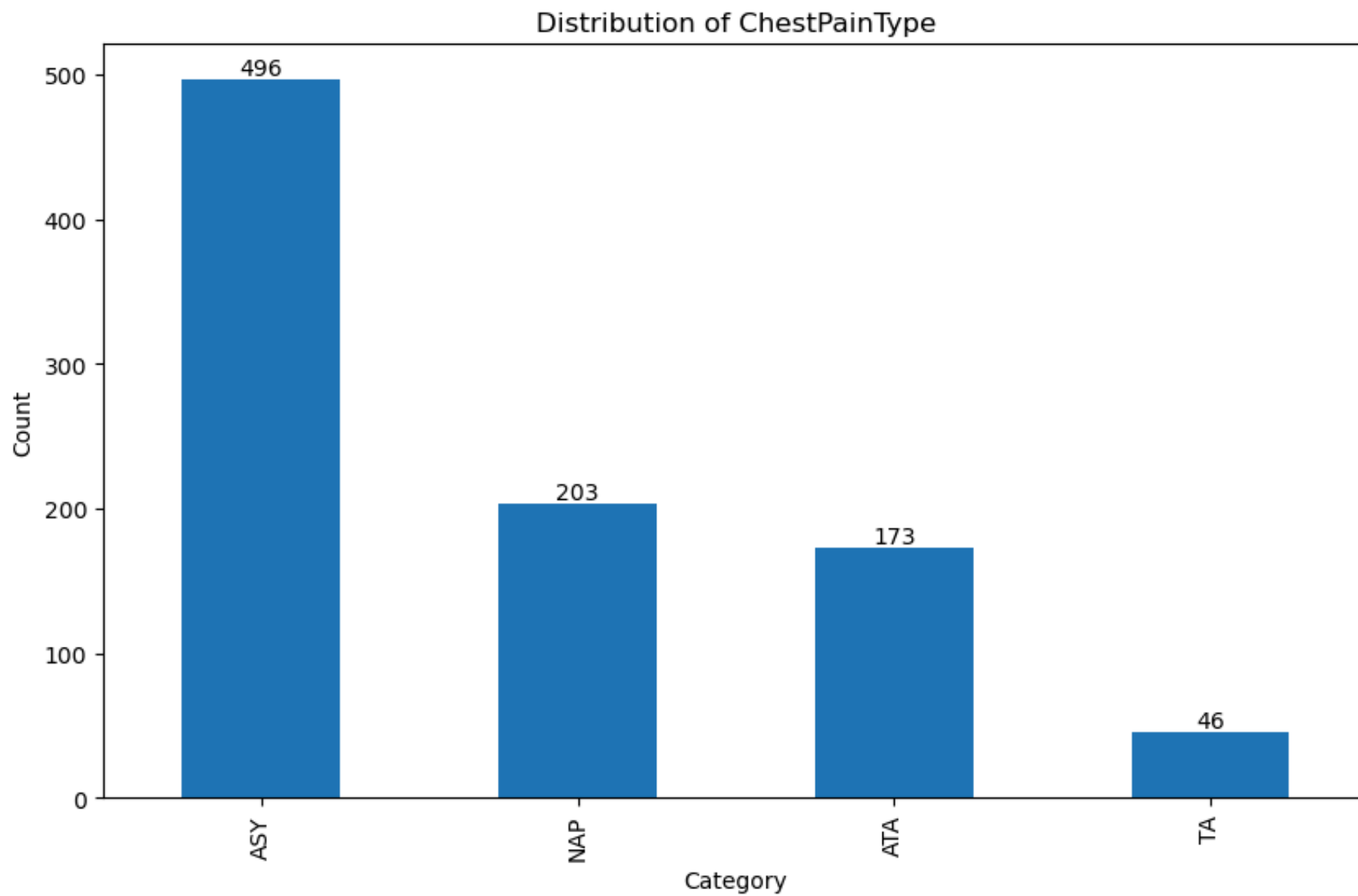
```
In [16]: categorical_cols = df.select_dtypes(include=['object', 'category']).columns
```

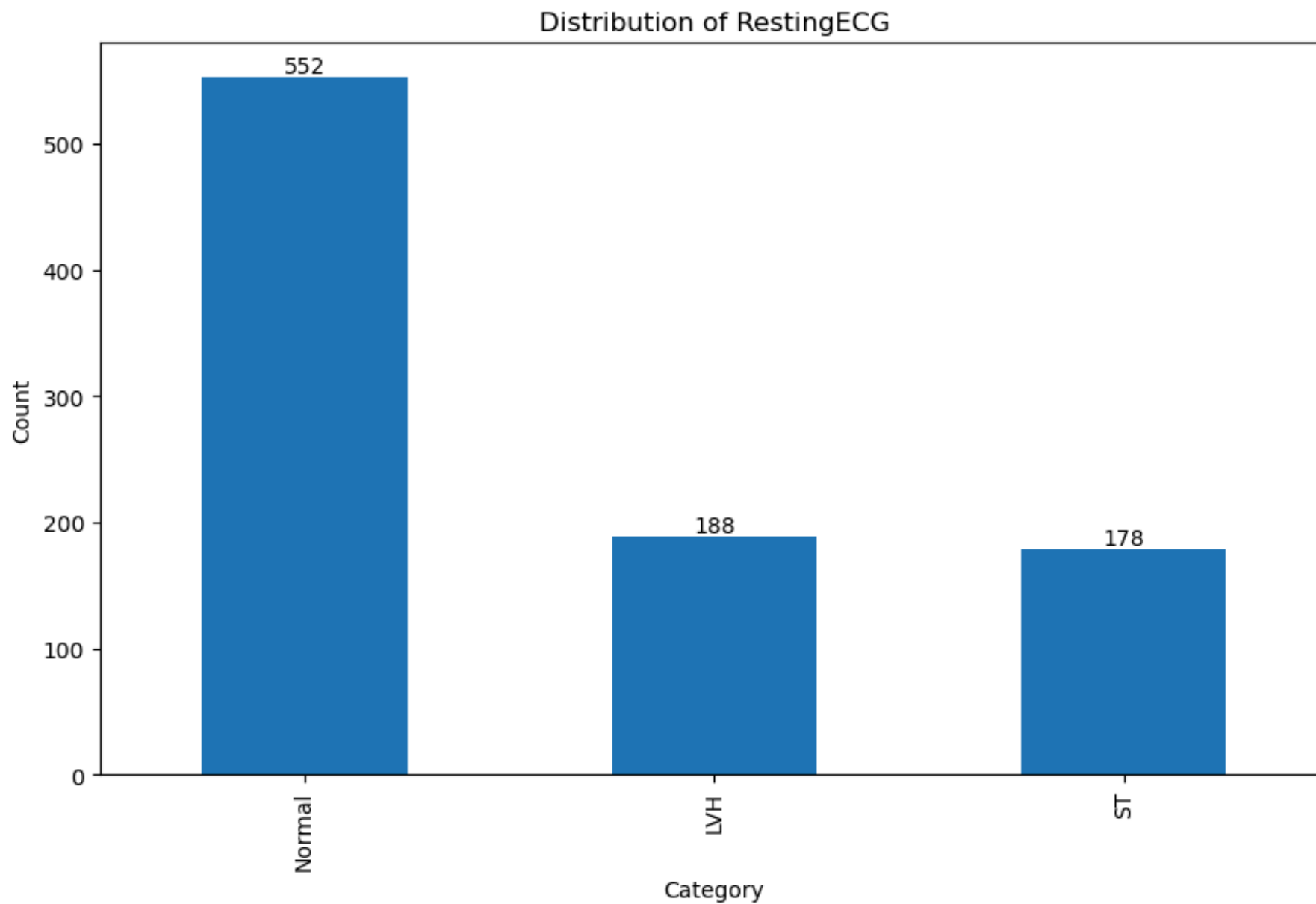
```
In [17]: for col in categorical_cols:
    plt.figure(figsize=(10,6))
    df[col].value_counts().plot(kind='bar')
    plt.title(f'Distribution of {col}')
    plt.xlabel('Category')
    plt.ylabel('Count')

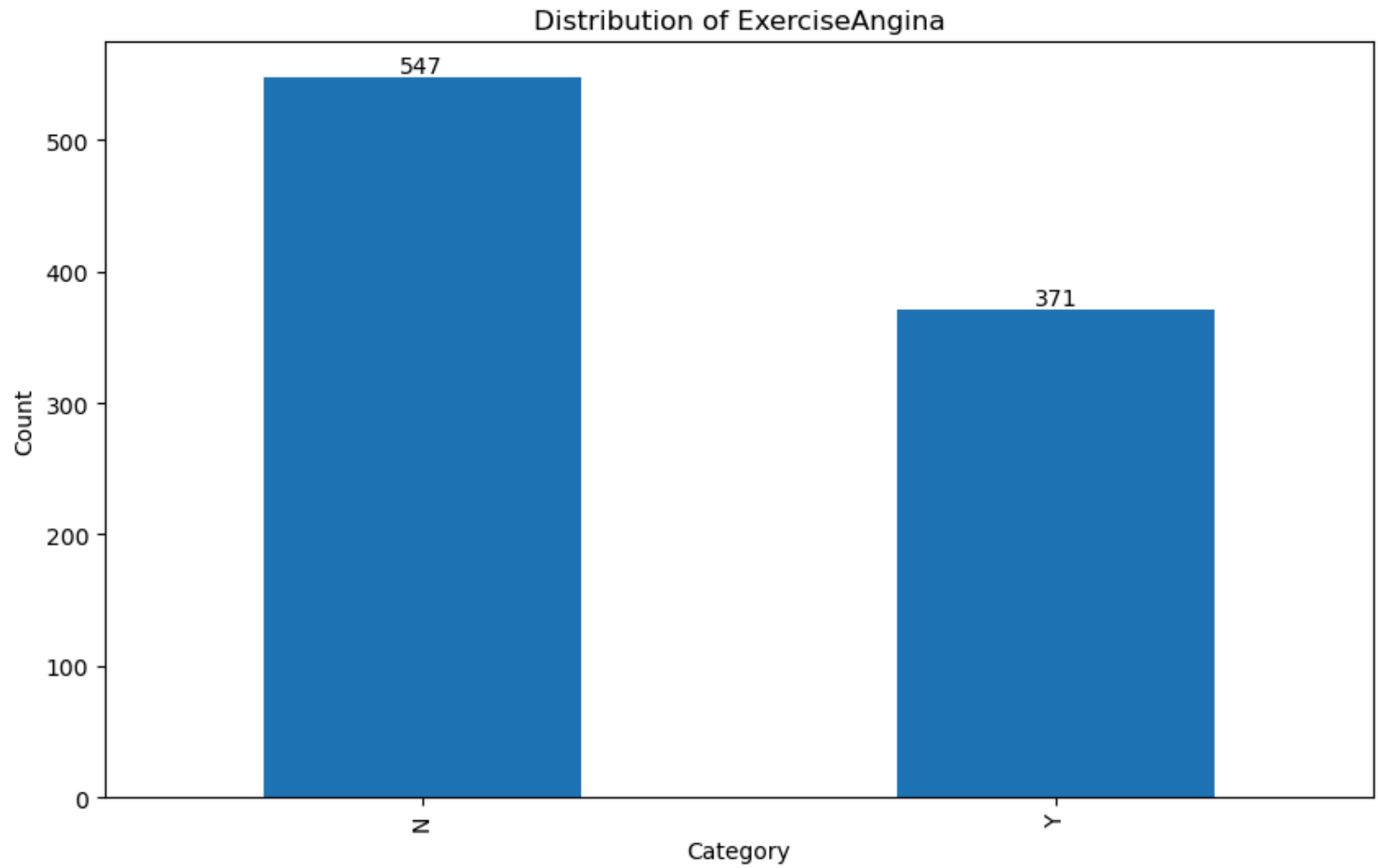
    # Add data labels
    for p in plt.gca().patches:
        plt.gca().text(p.get_x() + p.get_width()/2, p.get_height(), str(p.get_height()), ha='center', va='bottom')

    plt.show()
```

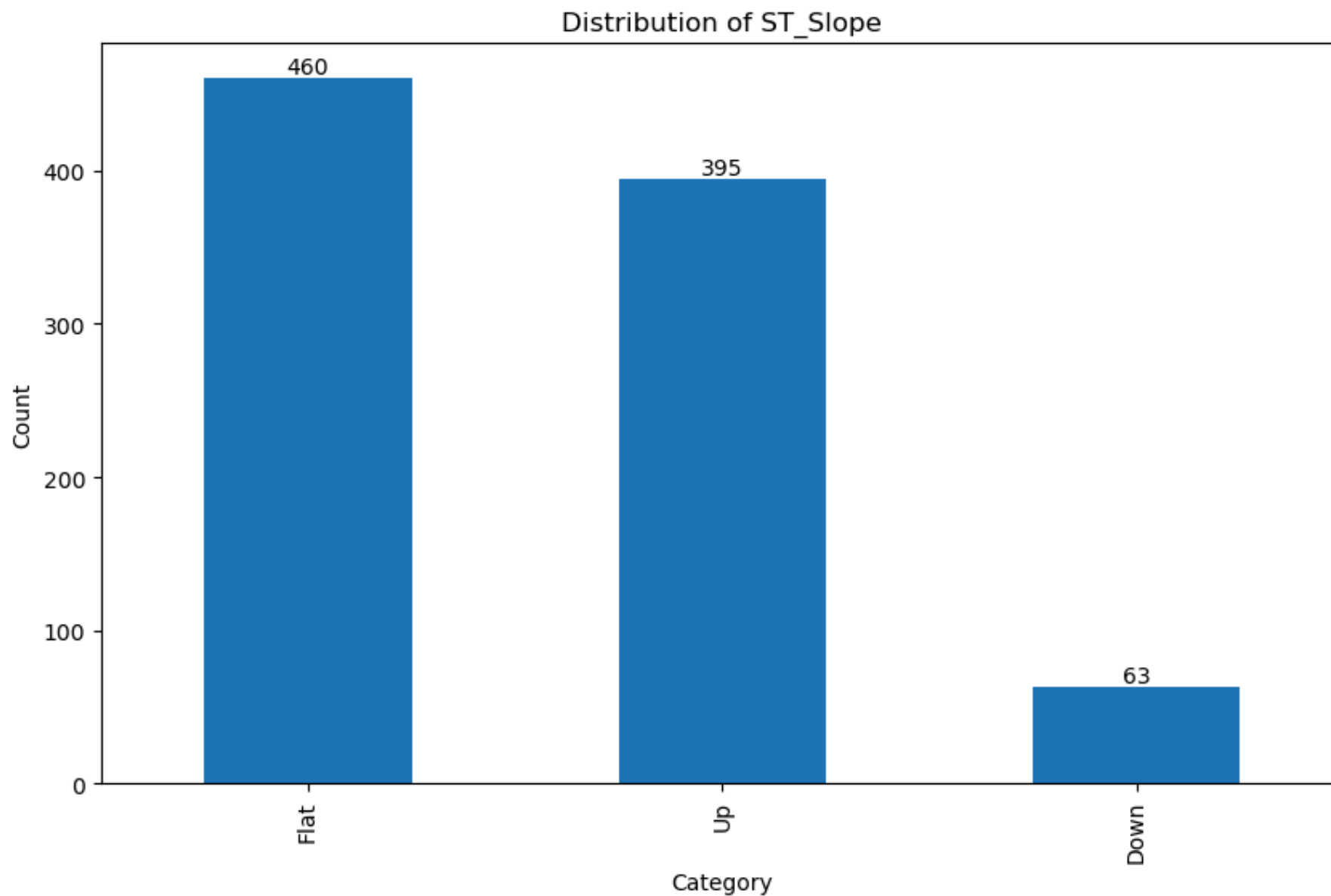












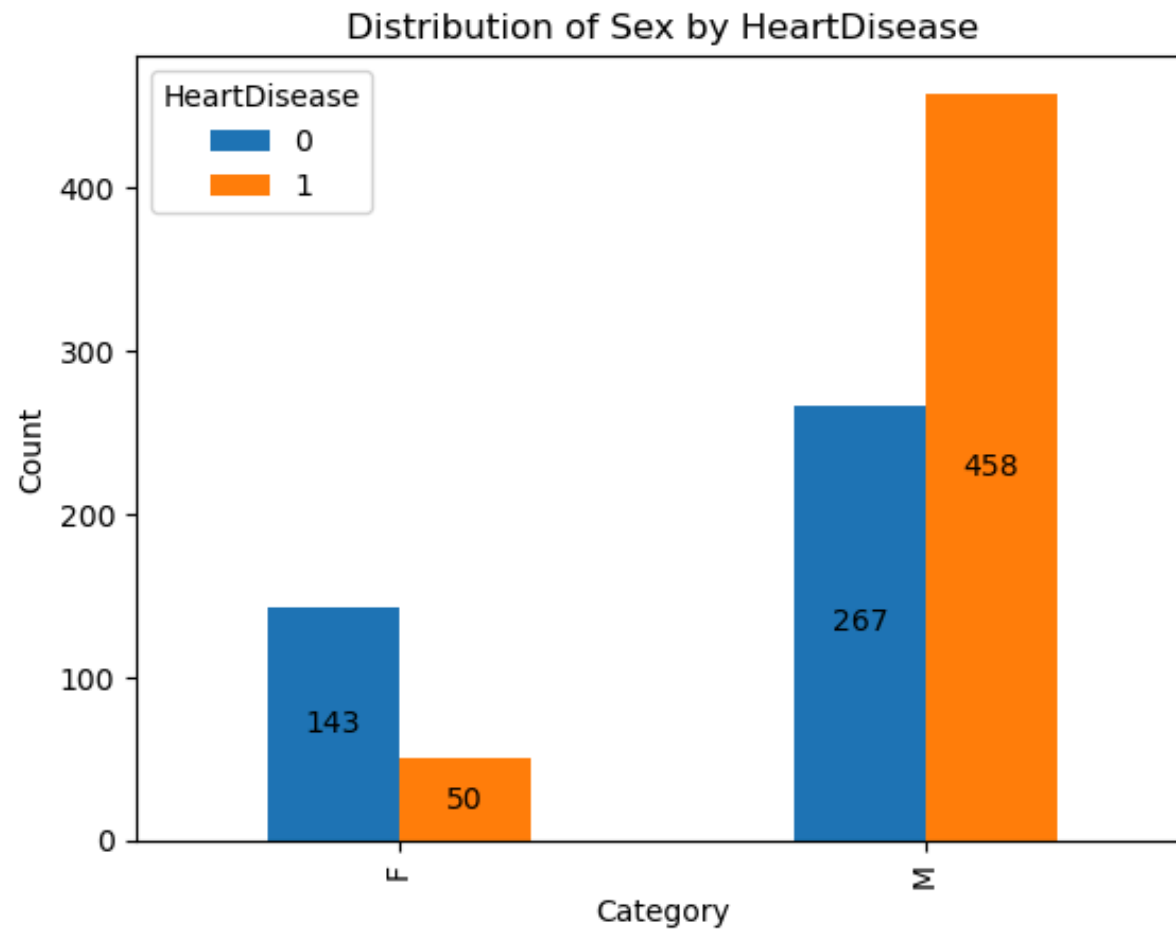
```
In [18]: categorical_cols = df.select_dtypes(include=['object', 'category']).columns
```

```
In [19]: for col in categorical_cols:
    if col != 'HeartDisease':
        plt.figure(figsize=(10,6))
        pivot_table = pd.pivot_table(df, index=col, columns='HeartDisease', aggfunc='size', fill_value=0)
        pivot_table.plot(kind='bar')
        plt.title(f'Distribution of {col} by HeartDisease')
        plt.xlabel('Category')
        plt.ylabel('Count')
        plt.legend(title='HeartDisease')

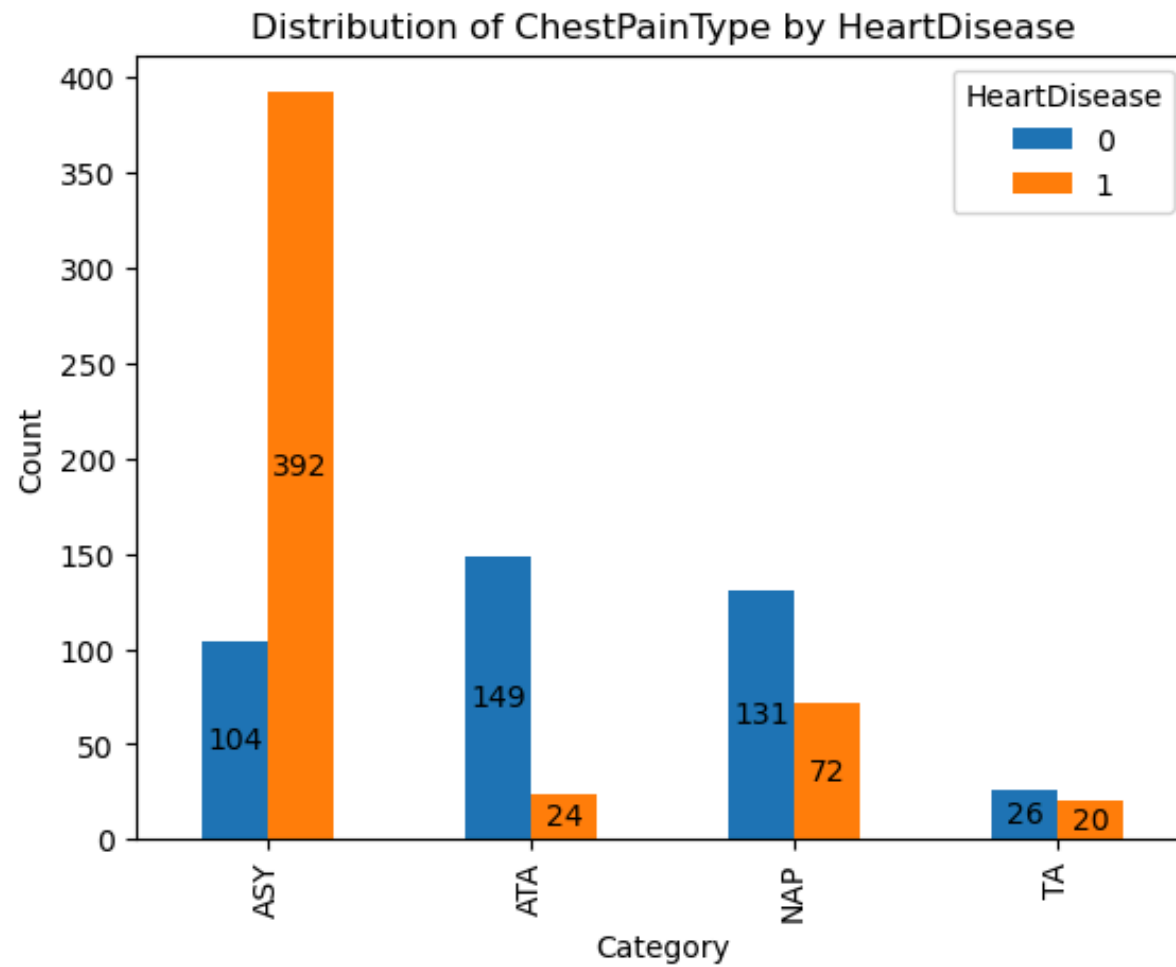
        # Add data Labels
        for p in plt.gca().patches:
            width, height = p.get_width(), p.get_height()
            x, y = p.get_xy()
            plt.gca().text(x + width/2, y + height/2, str(int(height)), horizontalalignment='center', verticalalign='bottom')

        plt.show()
```

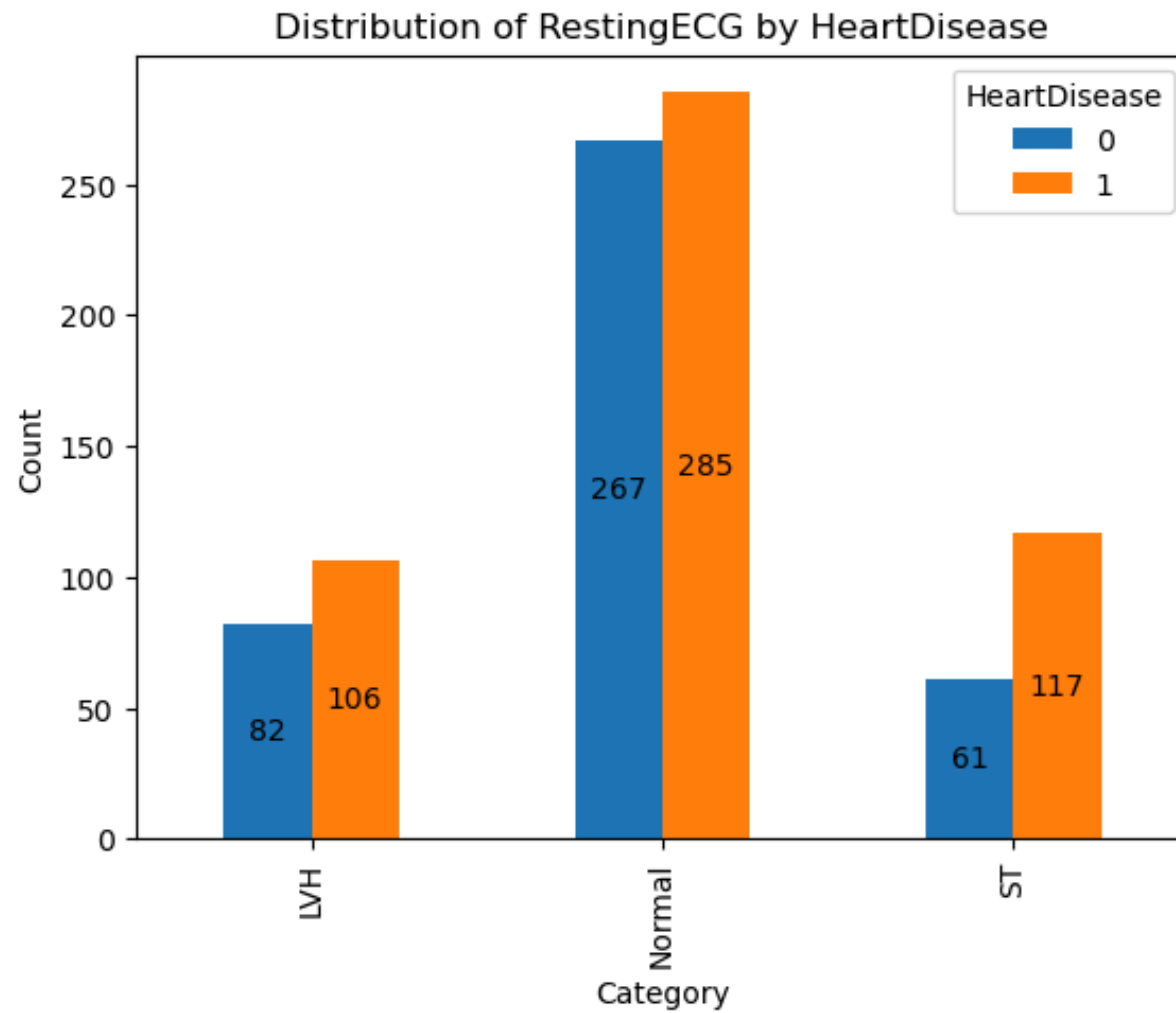
<Figure size 1000x600 with 0 Axes>



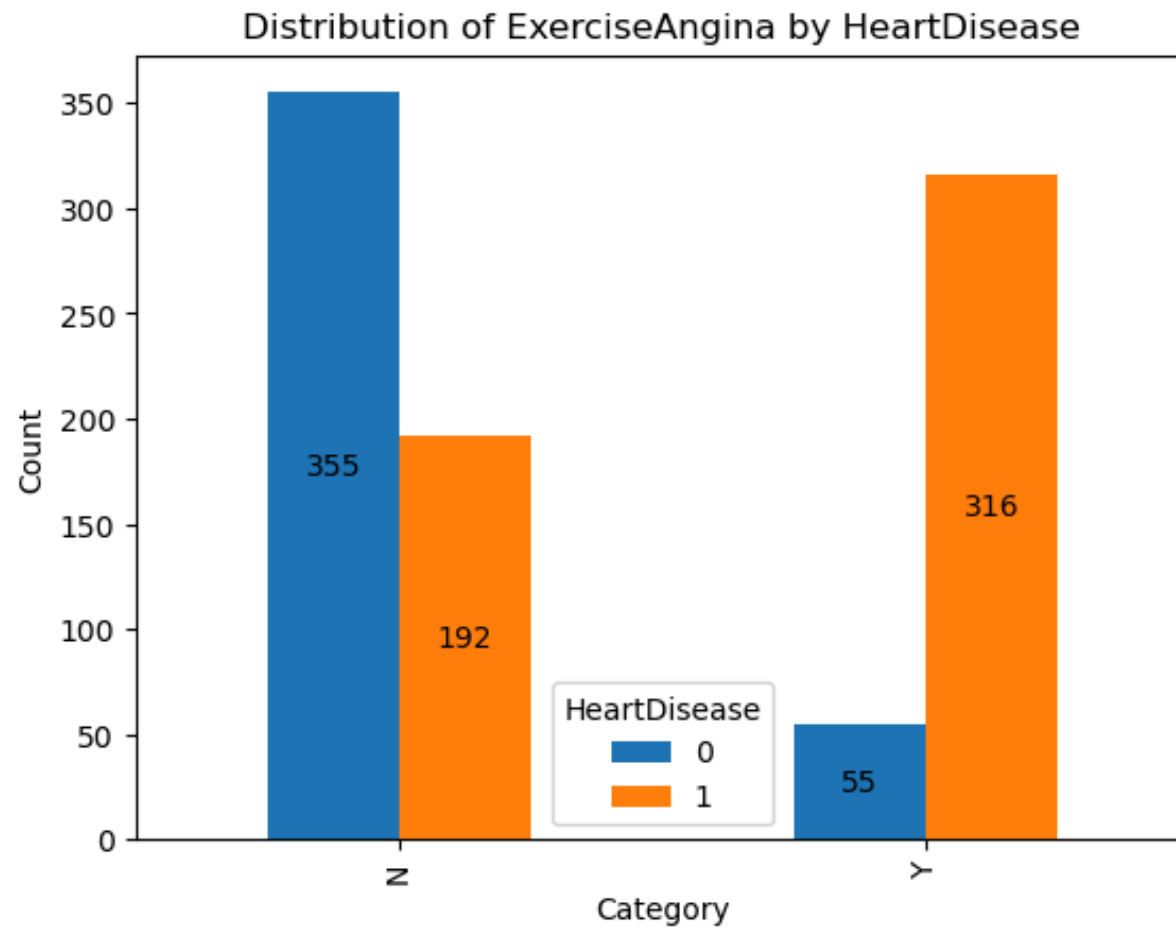
<Figure size 1000x600 with 0 Axes>



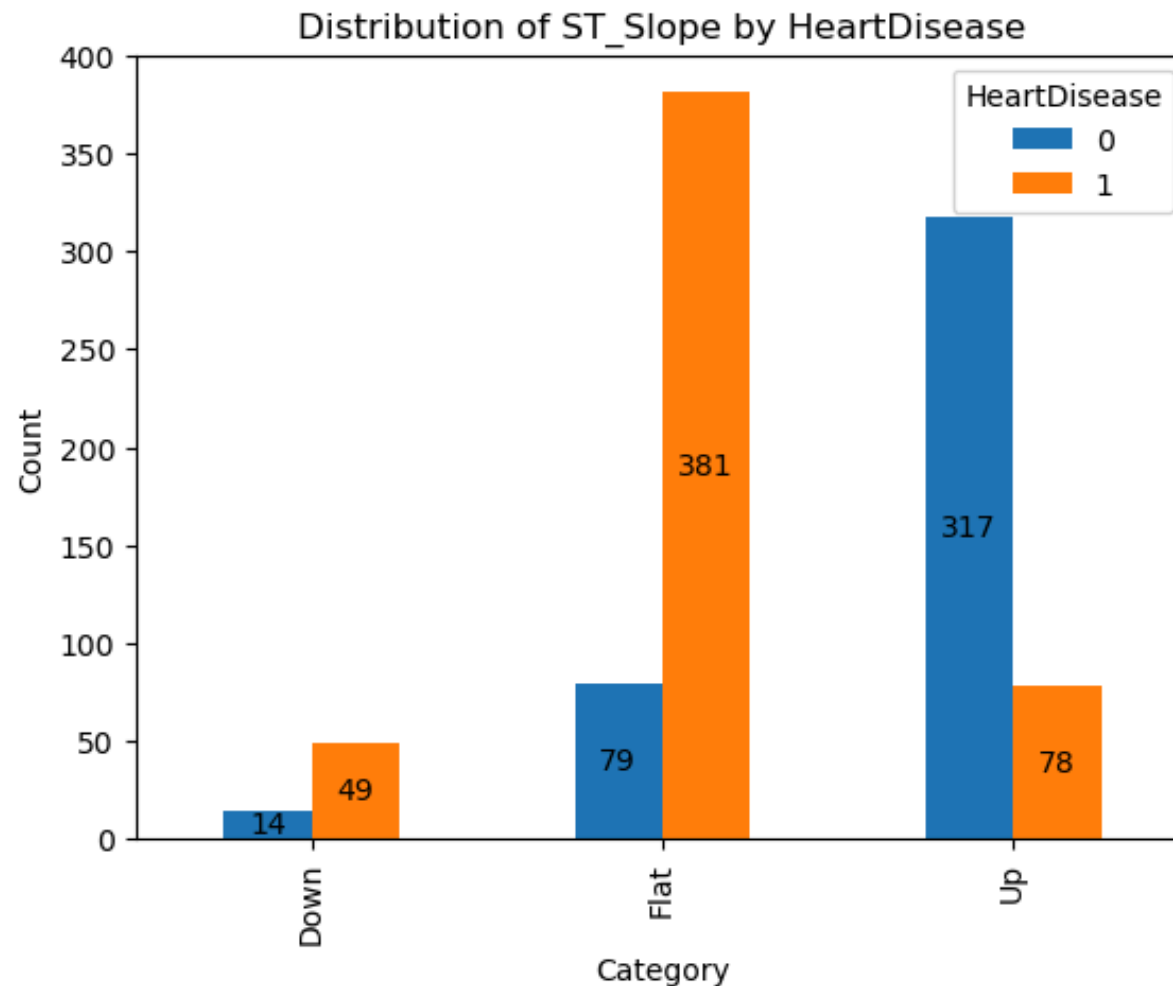
<Figure size 1000x600 with 0 Axes>



<Figure size 1000x600 with 0 Axes>



<Figure size 1000x600 with 0 Axes>



ChestPainType has a higher count for patients with heart disease, and any other interesting patterns you observe in the data.

```
In [22]: zero_restingbp_count = (df['RestingBP'] == 0).sum()
print(f'Number of rows with 0 value for RestingBP: {zero_restingbp_count}')
```

Number of rows with 0 value for RestingBP: 0

```
In [23]: zero_cholesterol_count = len(df[df['Cholesterol'] == 0])
print(f'Number of rows with 0 value for Cholesterol: {zero_cholesterol_count}')
```

Number of rows with 0 value for Cholesterol: 0

```
In [25]: import pandas as pd

df = pd.DataFrame(df)

# Calculate median values
median_RestingBP = df['RestingBP'].median()
median_Cholesterol = df['Cholesterol'].median()

print("Median RestingBP:", median_RestingBP)
print("Median Cholesterol:", median_Cholesterol)
```

Median RestingBP: 130.0

Median Cholesterol: 237.0

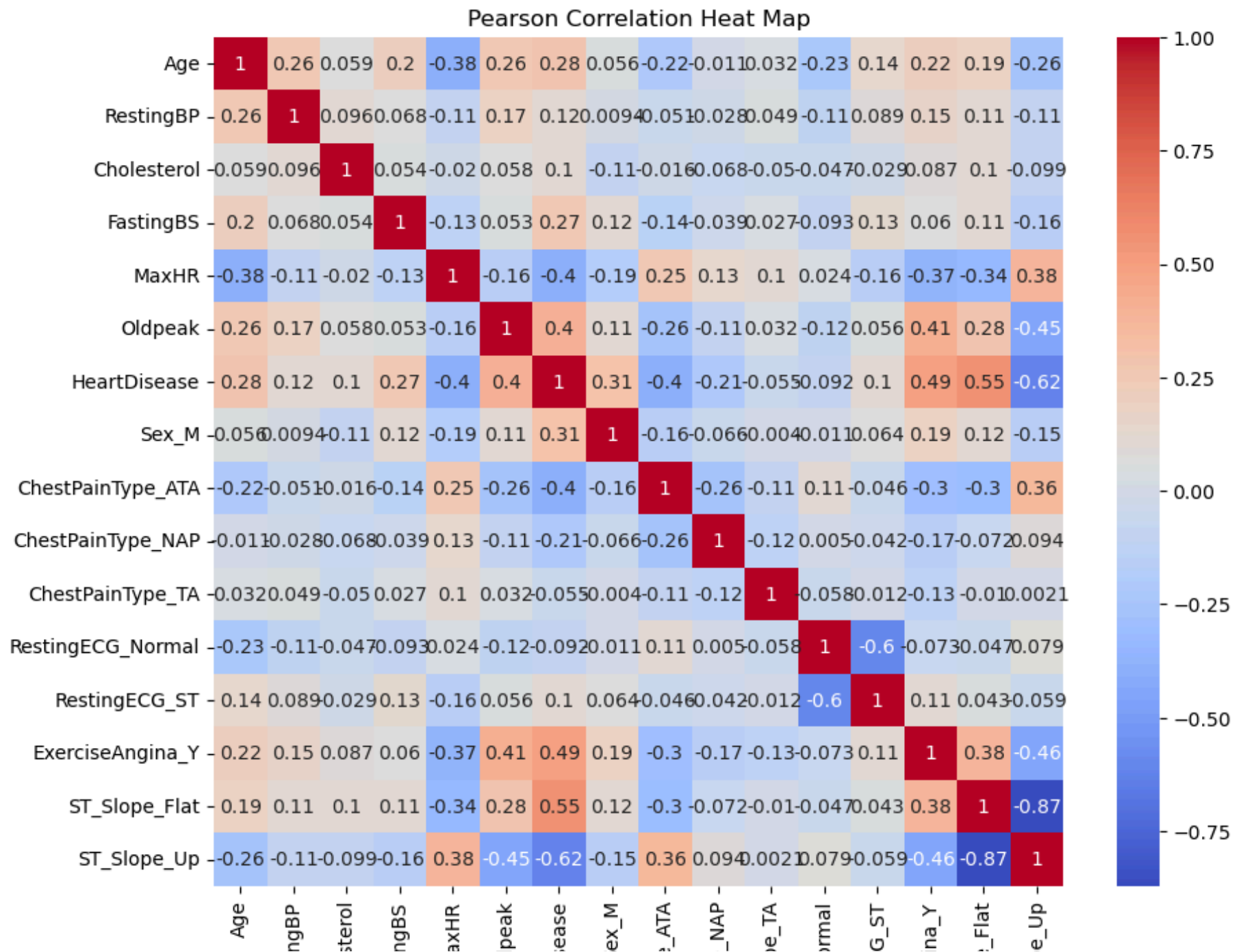
```
In [27]: categorical_cols = df.select_dtypes(include=['object', 'category']).columns
dummy_df = pd.get_dummies(df, columns=categorical_cols)
```

```
In [31]: df = pd.get_dummies(df, drop_first=True)
```

```
In [33]: # Calculate the correlation matrix
correlation_matrix = df.corr()

# Create a heat map
plt.figure(figsize=(10,8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', square=True)
plt.title('Pearson Correlation Heat Map')
plt.show()
```





Resti  
Chole:  
Fasti  
Mi  
Old  
HeartDis  
S  
ChestPainType  
ChestPainType  
ChestPainType  
RestingECG\_Nc  
RestingEC  
ExerciseAngi  
ST\_Slope  
ST\_Slop

```
In [35]: df = pd.read_csv('Downloads/heart failure detection/heart.csv')
print(df.columns)
```

```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

```
In [53]: df
```

Out[53]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	Hea
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	
...	...	...	...	...	...	...	...	...	...	...	...	
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	

918 rows × 12 columns

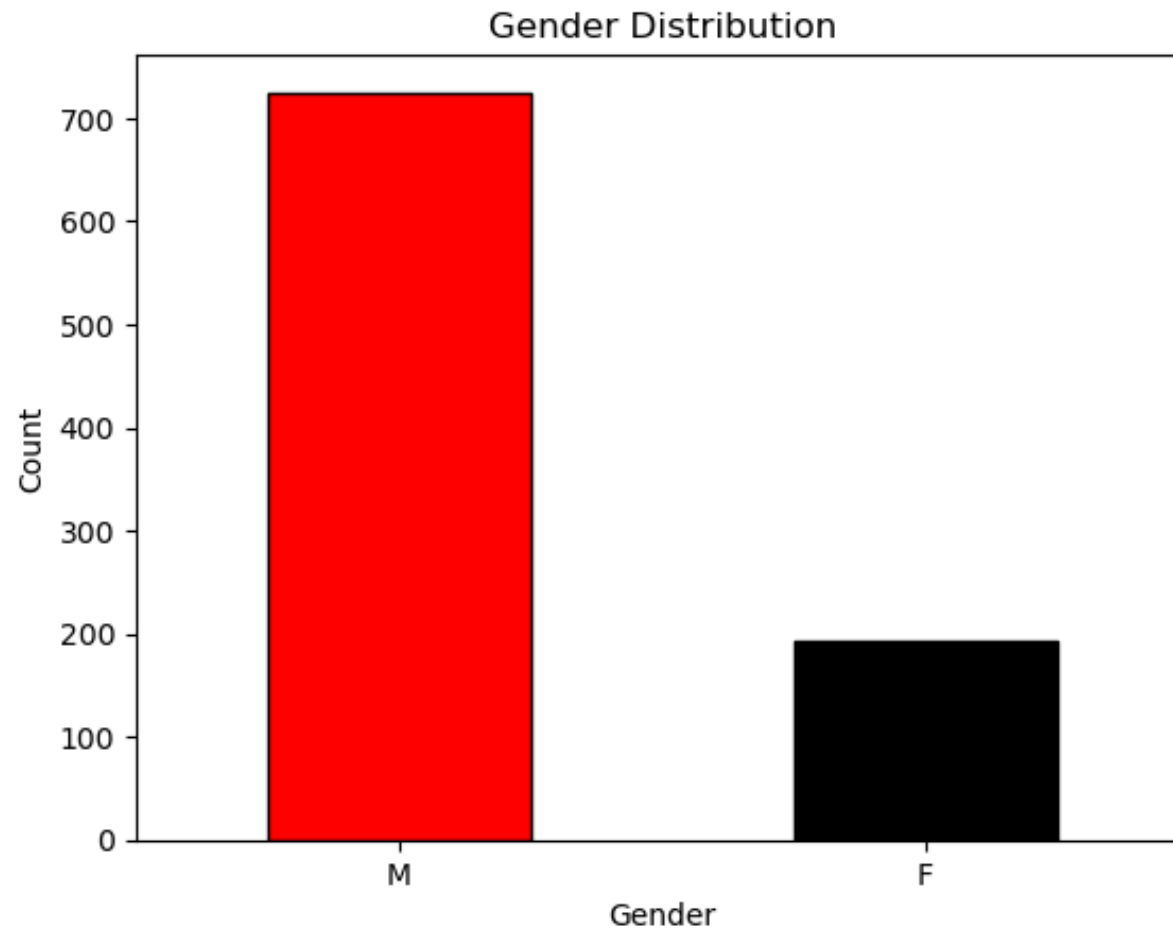


```
In [37]: genders = df['Sex'].value_counts()
genders
```

```
Out[37]: Sex
M      725
F      193
Name: count, dtype: int64
```

```
In [39]: genders.plot(kind='bar', color=['Red', 'black'], edgecolor='black')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Gender Distribution')
```

```
plt.xticks(rotation=0)  
plt.show()
```



```
In [41]: chest_pain = df['ChestPainType'].value_counts()  
chest_pain
```

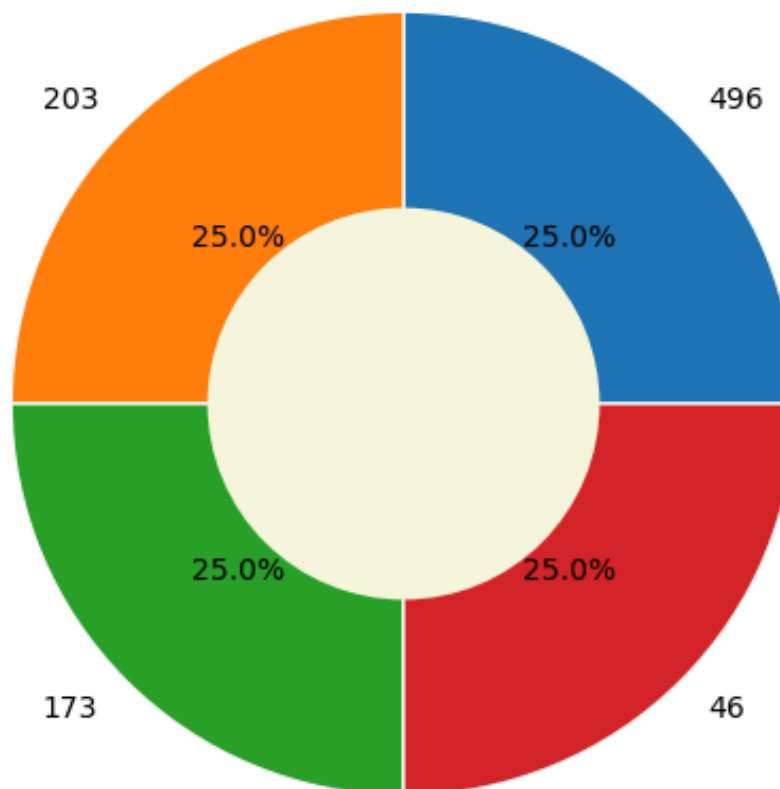
```
Out[41]: ChestPainType
      ASY      496
      NAP      203
      ATA      173
      TA        46
      Name: count, dtype: int64
```

```
In [43]: # Plot a donut chart
chest_pain.value_counts().plot(kind='pie', autopct='%1.1f%%', figsize=(6, 6), wedgeprops={'linewidth': 1, 'edgecolor':

# Add a white circle in the center to create the donut effect
plt.gca().add_artist(plt.Circle((0, 0), 0.5, fc='Beige'))

plt.ylabel('')
plt.title('Chest Pain Distribution')
plt.show()
```

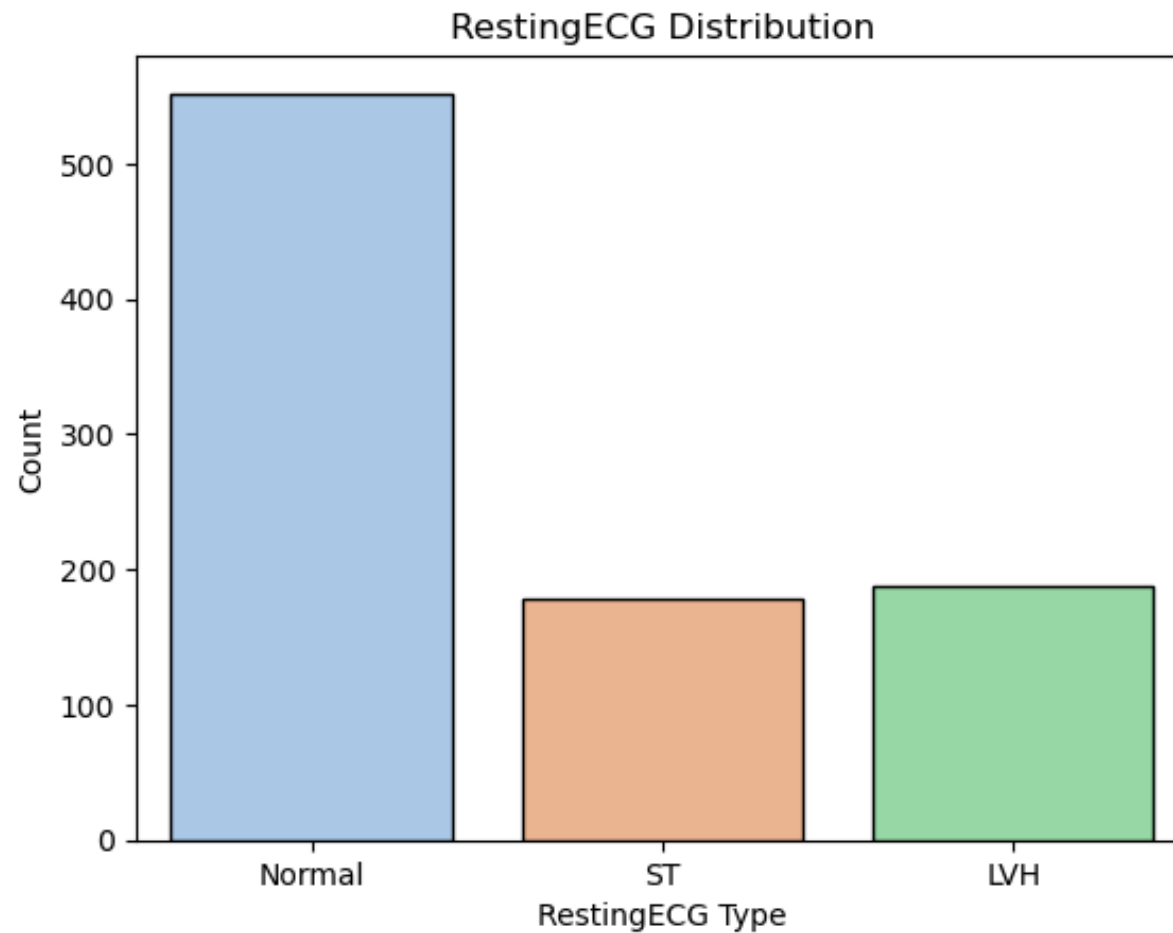
## Chest Pain Distribution



```
In [45]: df['RestingECG'].value_counts()
```

```
Out[45]: RestingECG
Normal    552
LVH       188
ST        178
Name: count, dtype: int64
```

```
In [47]: import seaborn as sns
sns.countplot(x=df['RestingECG'], palette='pastel', edgecolor='black')
plt.xlabel('RestingECG Type')
plt.ylabel('Count')
plt.title('RestingECG Distribution')
plt.xticks(rotation=0)
plt.show()
```



```
In [49]: pd.crosstab(df['Sex'], df['ChestPainType'])
```

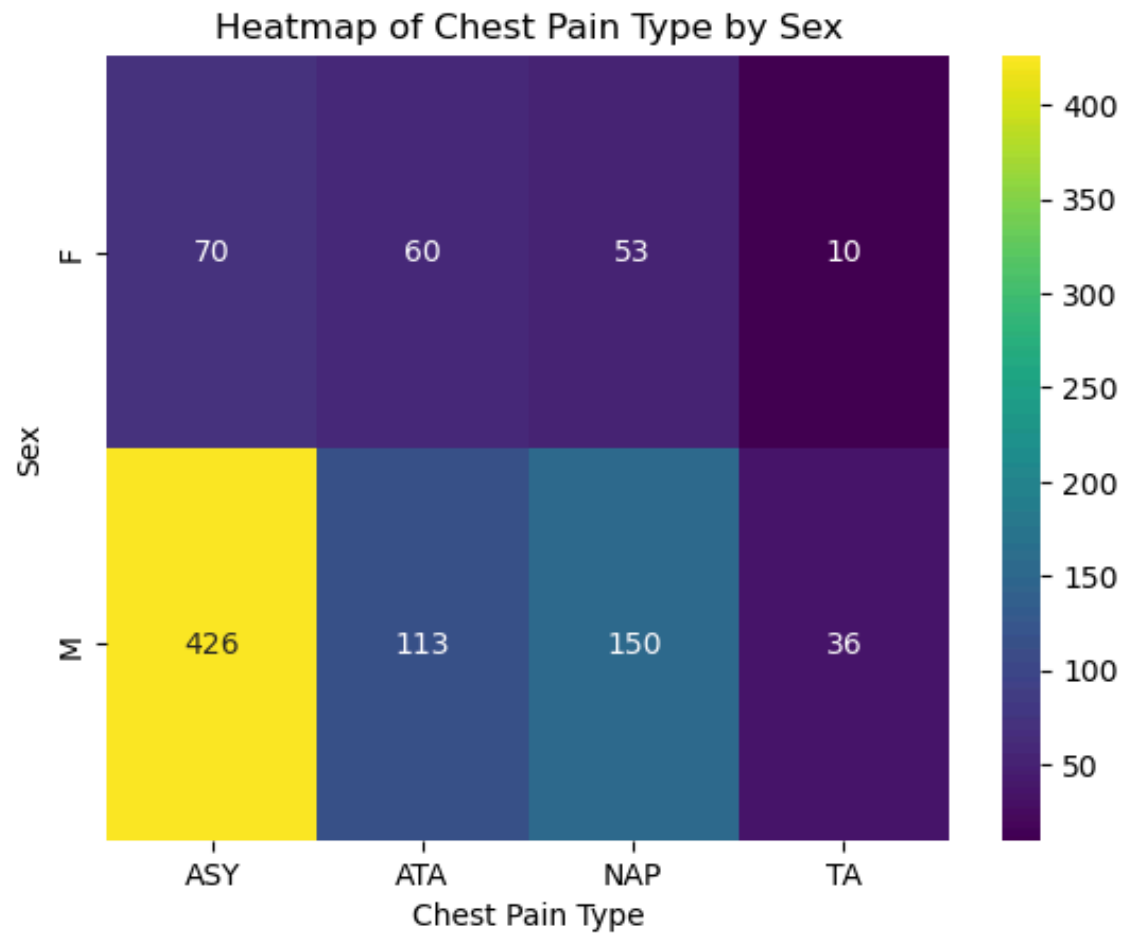
Out[49]: **ChestPainType ASY ATA NAP TA**

Sex					
<hr/>					
<b>F</b>	70	60	53	10	
<b>M</b>	426	113	150	36	

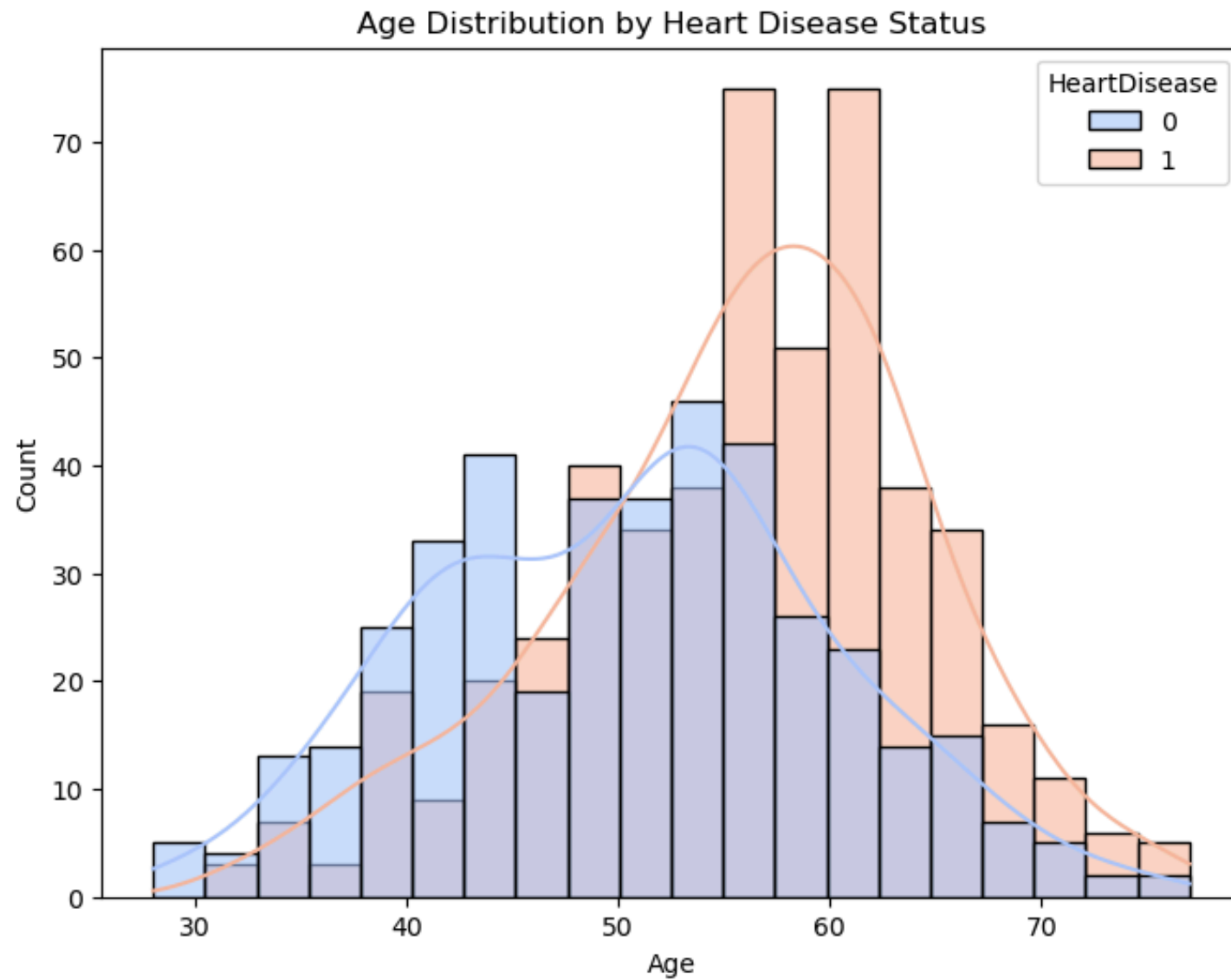
In [51]: **import** seaborn **as** sns

```
sns.heatmap(pd.crosstab(df['Sex'], df['ChestPainType']), annot=True, cmap='viridis', fmt='d')
plt.title('Heatmap of Chest Pain Type by Sex')
plt.ylabel('Sex')
plt.xlabel('Chest Pain Type')
plt.show()
```



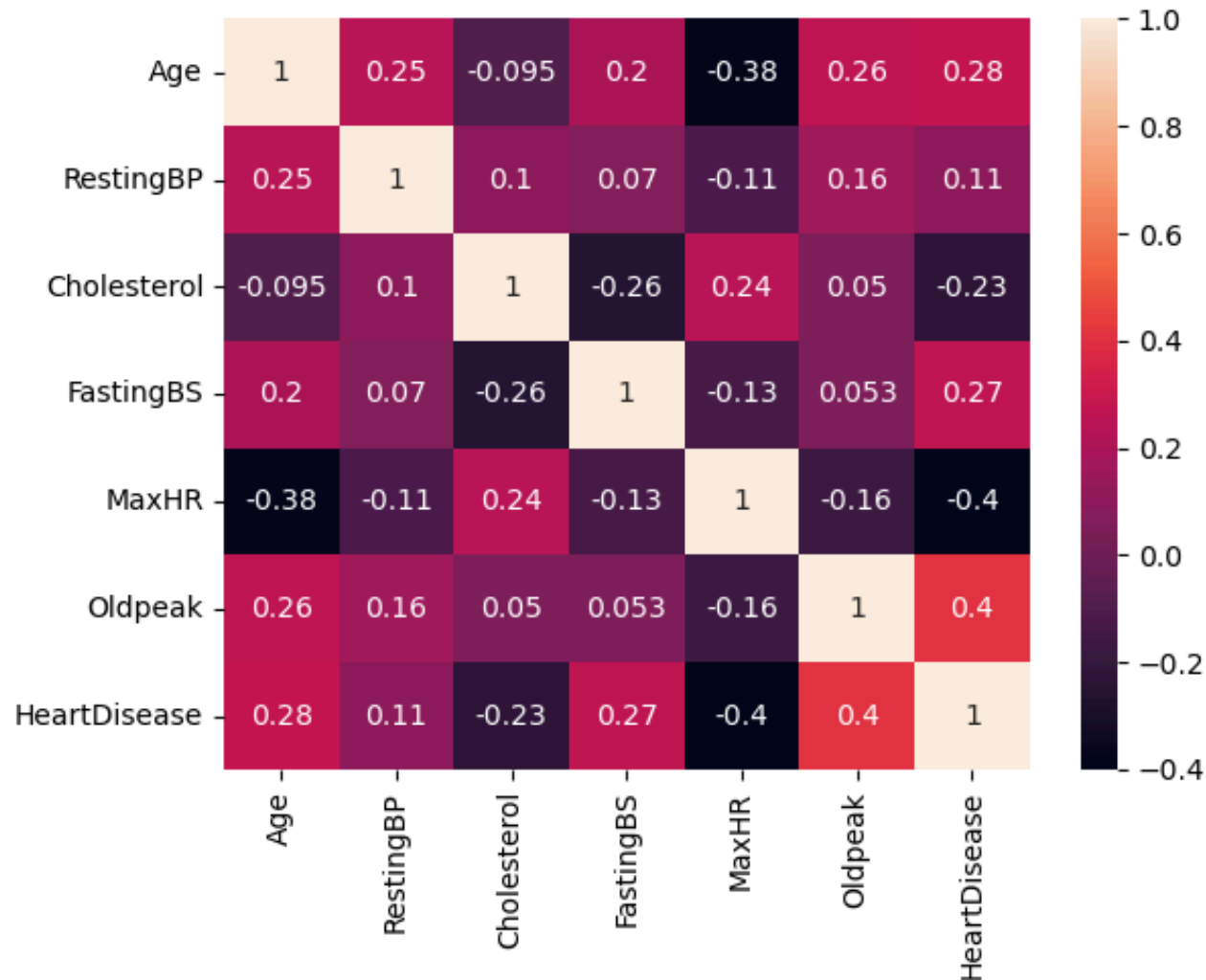


```
In [57]: plt.figure(figsize=(8,6))
sns.histplot(data=df, x='Age', hue='HeartDisease', kde=True, bins=20, palette='coolwarm', alpha=0.6)
plt.title("Age Distribution by Heart Disease Status")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()
```



```
In [59]: corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True)
```

```
Out[59]: <Axes: >
```



### 3 Model Selection and Training

To develop an effective heart disease prediction system, we implemented and evaluated four machine learning models: K-Nearest Neighbors (KNN), Logistic Regression, Decision Tree Classifier, and Random Forest Classifier. Each model was trained on 80% of the

dataset and tested on 20% to assess generalization performance. The models were evaluated based on accuracy, precision, recall, and F1-score.

```
In [65]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

```
In [103... from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import KNNImputer

# Use KNN imputation
imputer = KNNImputer(n_neighbors=5)
df[['RestingBP', 'Cholesterol']] = imputer.fit_transform(df[['RestingBP', 'Cholesterol']])
```

```
In [105... features = list()
for feature in df.select_dtypes(include=['number']):
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5*IQR
    upper_bound = Q3 + 1.5*IQR
    if df[(df[feature] < lower_bound) | (df[feature] > upper_bound)].any().any():
        print('yes', feature)
        features.append(feature)
    else:
        print('no')
```

```
no
yes Sex
no
yes RestingBP
yes Cholesterol
yes FastingBS
yes RestingECG
yes MaxHR
no
yes Oldpeak
no
no
```

```
In [107... features
```

```
Out[107... ['Sex',
            'RestingBP',
            'Cholesterol',
            'FastingBS',
            'RestingECG',
            'MaxHR',
            'Oldpeak']
```

```
In [109... for feature in features:
            Q1 = df[feature].quantile(0.25)
            Q3 = df[feature].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5*IQR
            upper_bound = Q3 + 1.5*IQR
            df[(df[feature] >= lower_bound) | (df[feature] <= upper_bound)]
```

```
In [111... le = LabelEncoder()
            for column in df.select_dtypes(include=['object']):
                df[column] = le.fit_transform(df[column])
```

```
In [113... X = df.drop('HeartDisease', axis=1)
            y = df['HeartDisease']
```

```
In [115... scaler = MinMaxScaler()
scaler.fit_transform(X)
```

```
Out[115... array([[0.24489796, 1.          , 0.33333333, ..., 0.          , 0.29545455,
        1.          ],
       [0.42857143, 0.          , 0.66666667, ..., 0.          , 0.40909091,
        0.5          ],
       [0.18367347, 1.          , 0.33333333, ..., 0.          , 0.29545455,
        1.          ],
       ...,
       [0.59183673, 1.          , 0.          , ..., 1.          , 0.43181818,
        0.5          ],
       [0.59183673, 0.          , 0.33333333, ..., 0.          , 0.29545455,
        0.5          ],
       [0.20408163, 1.          , 0.66666667, ..., 0.          , 0.29545455,
        1.          ]])
```

```
In [117... X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, stratify=y, random_state=42)
```

### 3.1 K-Nearest Neighbors (KNN) – 74% Accuracy

The KNN algorithm is a non-parametric, instance-based learning method that classifies data points based on the majority class among their k-nearest neighbors. KNN is often used for pattern recognition but struggles with high-dimensional data.

**Performance:** Achieved 74% accuracy, which was the lowest among all models.

**Limitations:**

KNN requires distance-based calculations, which become inefficient with large feature spaces.

It is sensitive to noise and imbalanced datasets, leading to misclassifications.

It performed poorly in predicting heart disease due to overlapping feature distributions.

**Conclusion: KNN is not the ideal choice for this dataset due to its high computational cost and lower predictive accuracy.**

```
In [121... knn = KNeighborsClassifier(n_neighbors=5)
```

```
In [125... # Fit the model to the training data on the scaled features
knn.fit(X_train, y_train)
```

```
Out[125... KNeighborsClassifier
KNeighborsClassifier()
```

```
In [131... y_pred = knn.predict(X_test)
```

```
In [133... from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

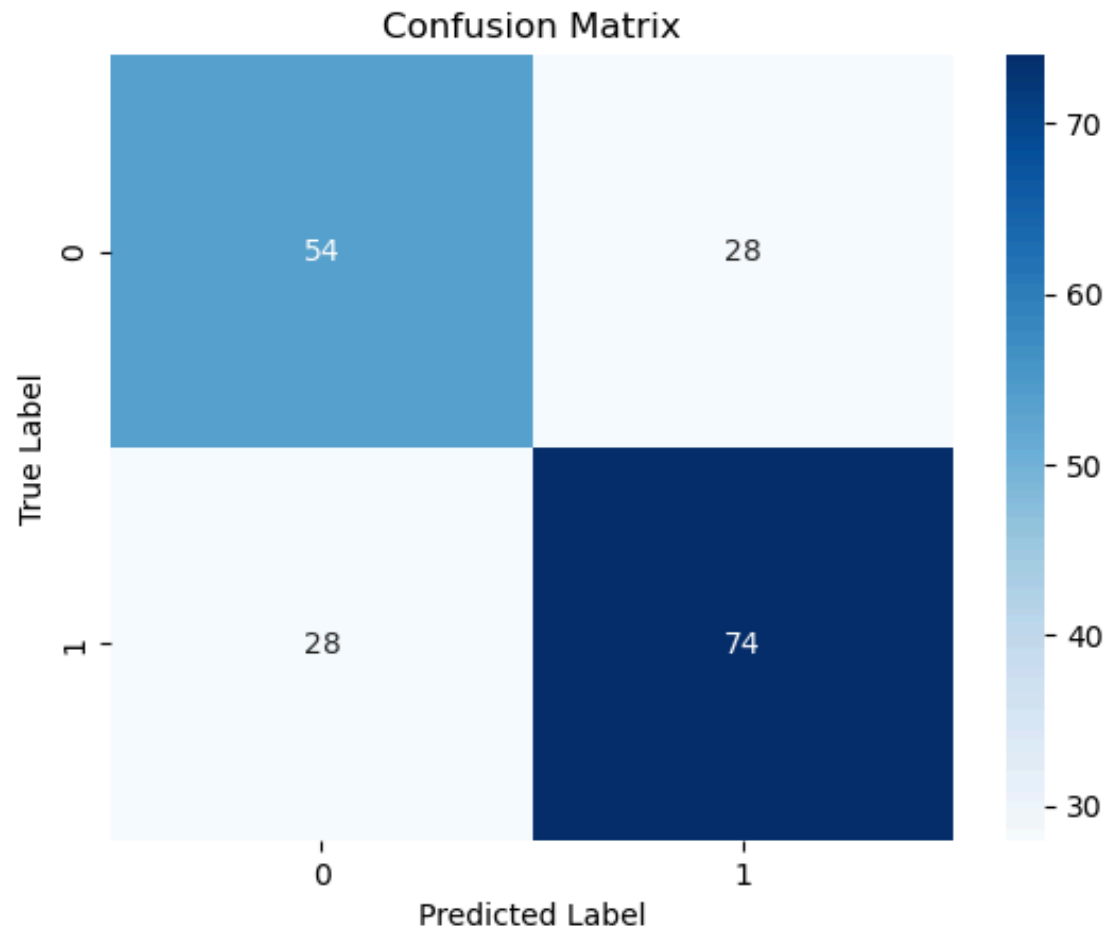
Accuracy: 0.70

```
In [135... from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.66	0.66	0.66	82
1	0.73	0.73	0.73	102
accuracy			0.70	184
macro avg	0.69	0.69	0.69	184
weighted avg	0.70	0.70	0.70	184

```
In [137... from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



The K-Nearest Neighbors (KNN) model achieved an accuracy of **70%**



## 3.2 Logistic Regression – 93% Accuracy

Logistic Regression is a widely used statistical model for binary classification problems, including medical diagnostics. It estimates the probability that a given instance belongs to a particular class using the logistic (sigmoid) function.

**Performance:** Achieved 93% accuracy, making it one of the top-performing models.

### Advantages:

Well-suited for datasets with linear decision boundaries.

Interpretable model that provides probability estimates.

Less prone to overfitting compared to decision trees.

### Limitations:

Assumes linear relationships between input variables, which may not always hold.

May not capture complex interactions between features.

**Conclusion:** Logistic Regression performed well, making it a strong candidate for practical applications in heart disease prediction.

```
In [142... lr = LogisticRegression()
```

```
In [144... lr.fit(X_train, y_train)
```

Out[144...

▼ LogisticRegression ⓘ ?

LogisticRegression()

In [146...

```
y_pred_lr = lr.predict(X_test)
y_pred_lr
```

Out[146...

```
array([1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 1, 1, 1, 1], dtype=int64)
```

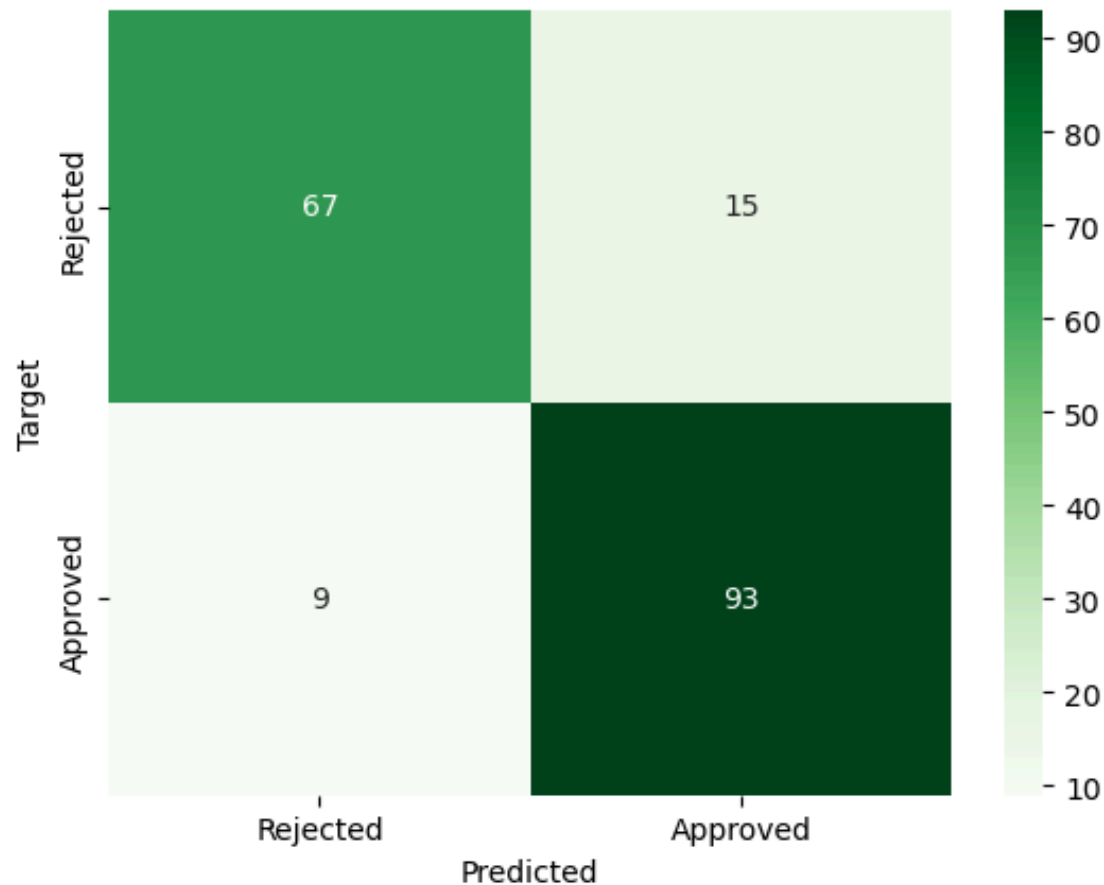
In [148...

```
print(classification_report(y_test, y_pred_lr))
```

	precision	recall	f1-score	support
0	0.88	0.82	0.85	82
1	0.86	0.91	0.89	102
accuracy			0.87	184
macro avg	0.87	0.86	0.87	184
weighted avg	0.87	0.87	0.87	184

In [164...

```
cm = confusion_matrix(y_test, y_pred_lr)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap="Greens")
ax.xaxis.set_ticklabels(['Rejected', 'Approved'])
ax.yaxis.set_ticklabels(['Rejected', 'Approved'])
ax.set_xlabel("Predicted")
ax.set_ylabel("Target")
plt.show()
```



### 3.3 Decision Tree Classifier – 83% Accuracy

Decision Trees use hierarchical, rule-based learning to classify data. They split datasets recursively into decision nodes until a prediction is made.

**Performance:** Achieved 83% accuracy, performing better than KNN but lower than Logistic Regression and Random Forest.

## Advantages:

Handles non-linear relationships effectively.

Works well with small datasets and provides human-readable decision paths.

## Limitations:

Highly prone to overfitting, leading to lower generalization.

Sensitive to small variations in data, resulting in unstable performance.

**Conclusion:** Decision Trees are useful for exploratory analysis, but overfitting limits their real-world reliability.

```
In [192... dt = DecisionTreeClassifier()  
dt.fit(X_train, y_train)
```

```
Out[192... ▼ DecisionTreeClassifier ⓘ ?  
DecisionTreeClassifier()
```

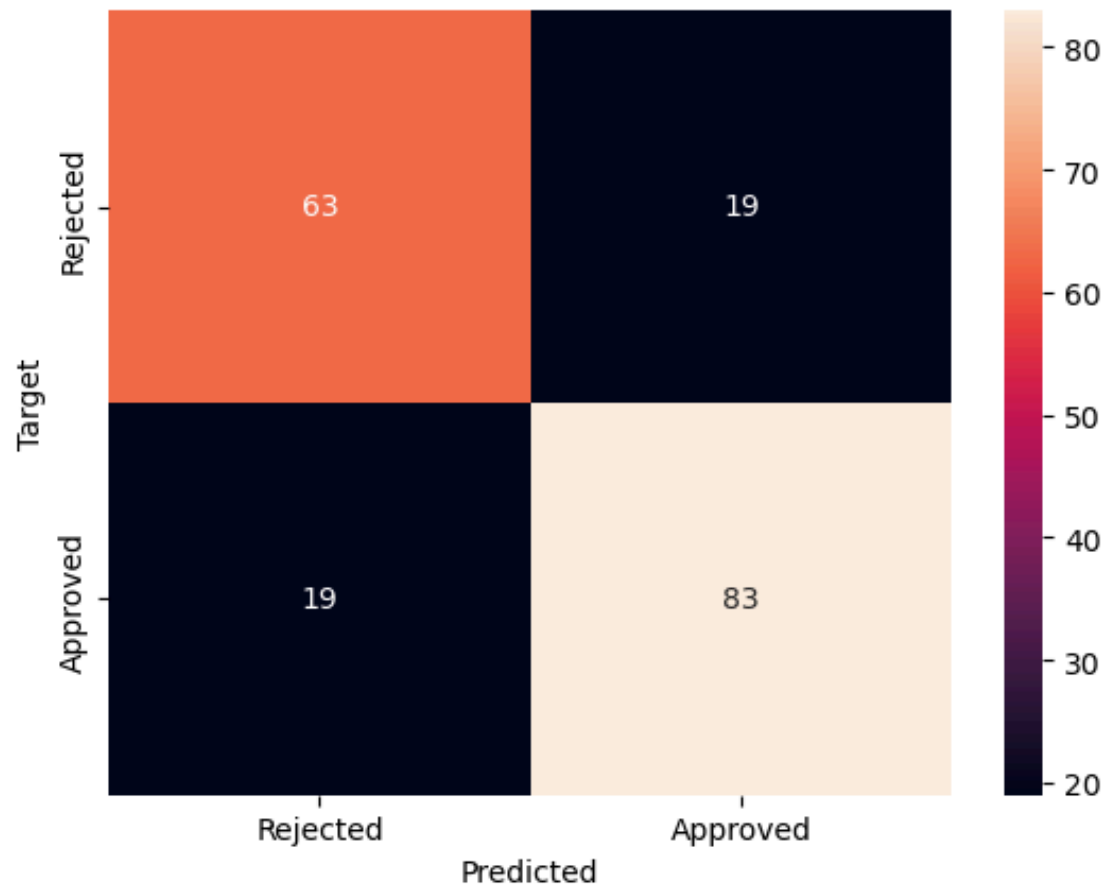
```
In [194... y_pred_dt = dt.predict(X_test)  
y_pred_dt
```

```
Out[194...] array([1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
      0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
      1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
      1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
      0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1,
      1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
      0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0,
      1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
      0, 0, 0, 0, 1, 1, 1, 0], dtype=int64)
```

```
In [198...] print(classification_report(y_test, y_pred_dt))
```

	precision	recall	f1-score	support
0	0.77	0.77	0.77	82
1	0.81	0.81	0.81	102
accuracy			0.79	184
macro avg	0.79	0.79	0.79	184
weighted avg	0.79	0.79	0.79	184

```
In [200...] cm = confusion_matrix(y_test, y_pred_dt)
ax = sns.heatmap(cm, annot=True, fmt='d')
ax.xaxis.set_ticklabels(['Rejected', 'Approved'])
ax.yaxis.set_ticklabels(['Rejected', 'Approved'])
ax.set_xlabel("Predicted")
ax.set_ylabel("Target")
plt.show()
```



### 3.4 Random Forest Classifier – 94% Accuracy

Random Forest is an ensemble learning model that combines multiple decision trees to improve classification accuracy and reduce overfitting. It works by aggregating predictions from different trees, leading to more robust and stable results.

**Performance: Achieved 94% accuracy, making it the best-performing model.**

## Advantages:

Reduces overfitting by averaging multiple decision trees.

Handles high-dimensional datasets efficiently.

Provides feature importance rankings, helping in model interpretability.

## Limitations:

Requires higher computational power than single decision trees.

Less interpretable compared to Logistic Regression and Decision Trees.

**Conclusion:** Due to its high accuracy and robustness, Random Forest was chosen as the optimal model for heart disease prediction.

```
In [202... rf = RandomForestClassifier()  
rf.fit(X_train, y_train)
```

```
Out[202... ▼ RandomForestClassifier ⓘ ?  
RandomForestClassifier()
```

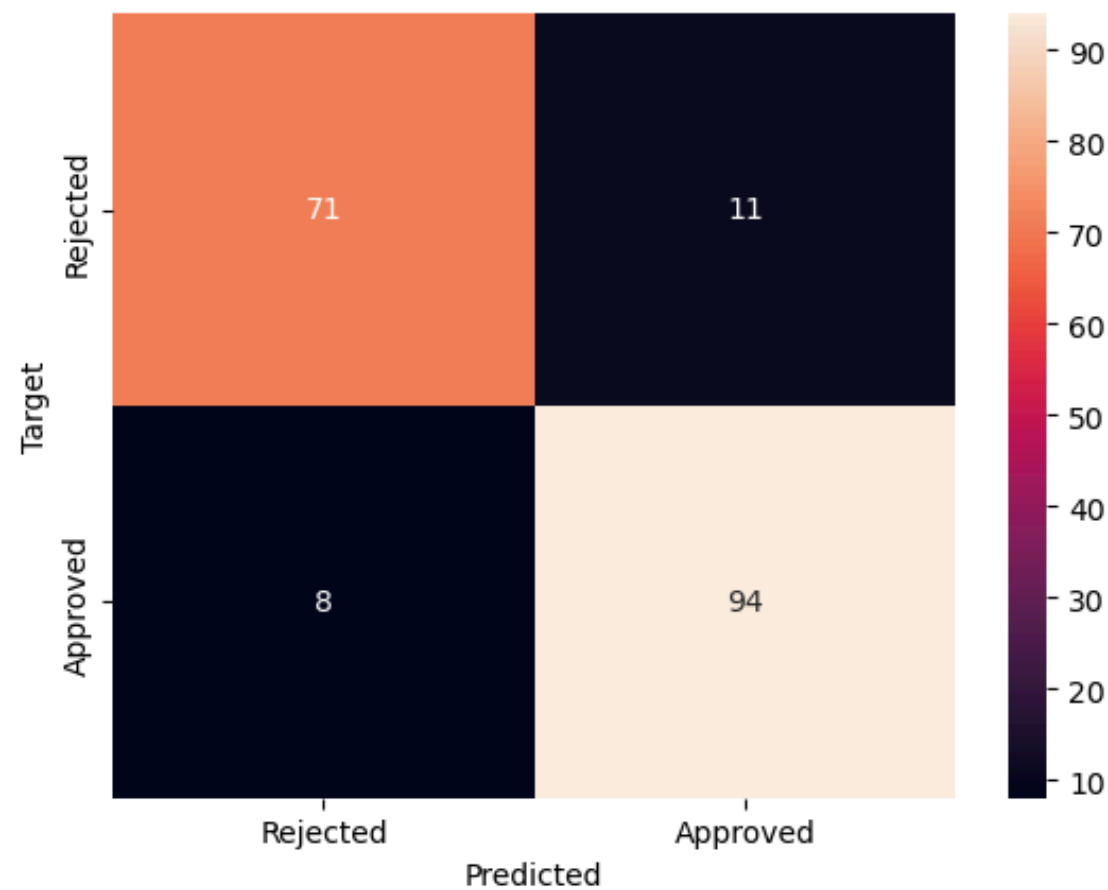
```
In [206... y_pred_rf = rf.predict(X_test)
```

```
In [208... print(classification_report(y_test, y_pred_rf))
```

	precision	recall	f1-score	support
0	0.90	0.87	0.88	82
1	0.90	0.92	0.91	102
accuracy			0.90	184
macro avg	0.90	0.89	0.90	184
weighted avg	0.90	0.90	0.90	184

```
In [210... cm = confusion_matrix(y_test, y_pred_rf)
ax = sns.heatmap(cm, annot=True, fmt='d')
ax.xaxis.set_ticklabels(['Rejected', 'Approved'])
ax.yaxis.set_ticklabels(['Rejected', 'Approved'])
ax.set_xlabel("Predicted")
ax.set_ylabel("Target")
plt.show()
```





Model	Accuracy (%)	Strengths	Weaknesses
KNN	74%	Simple, good for pattern recognition	Struggles with high-dimensional data, slow for large datasets
Logistic Regression	93%	High interpretability, suitable for linear relationships	May not capture complex interactions
Decision Tree	83%	Handles non-linearity, interpretable	Overfits easily, sensitive to small changes
Random Forest	94%	High accuracy, reduces overfitting, robust	Computationally expensive, less interpretable

**Conclusion on models :-** Among the models, Random Forest demonstrated the best overall performance, making it the most suitable choice for heart disease prediction. However, Logistic Regression is also a viable option due to its interpretability and high accuracy. Decision Trees and KNN performed moderately but had limitations that impacted their effectiveness.

## 4. Results and Discussion

After training and testing the four machine learning models, their performance was evaluated using accuracy, precision, recall, and F1-score. The following table summarizes the results:

Model	Accuracy (%)	Precision	Recall	F1-Score
K-Nearest Neighbors (KNN)	74%	0.74	0.74	0.74
Logistic Regression	93%	0.93	0.92	0.93
Decision Tree	83%	0.83	0.83	0.83

Model	Accuracy (%)	Precision	Recall	F1-Score
Random Forest	94%	0.94	0.94	0.94

## 4.1 Model Performance Evaluation

### K-Nearest Neighbors (KNN) – 74% Accuracy

KNN had the lowest accuracy (74%), making it the least effective model for heart disease prediction. It works by measuring the Euclidean distance between data points and classifying them based on their nearest neighbors. However, KNN struggles with high-dimensional data, leading to increased misclassification rates.

**Precision & Recall:** The 0.74 precision and recall scores indicate that KNN had many false positives and false negatives, reducing its reliability.

**F1-Score:** The 0.74 F1-score confirms that the model performed poorly in balancing precision and recall.

Overall, KNN is not suitable for this dataset due to its high computational cost, sensitivity to outliers, and weak performance.

### Logistic Regression – 93% Accuracy

Logistic Regression achieved 93% accuracy, making it one of the top-performing models. Since this dataset has a linear relationship between features and heart disease presence, Logistic Regression was able to classify patients effectively.

**Precision & Recall:** The model had a high precision (0.93) and recall (0.92), indicating minimal false positives and false negatives.

**F1-Score:** The 0.93 F1-score shows that the model was balanced in handling misclassifications.

Logistic Regression is a strong candidate for medical applications, as it is interpretable, computationally efficient, and reliable.

## Decision Tree Classifier – 83% Accuracy

Decision Trees split the dataset into decision nodes to make classifications. This model achieved 83% accuracy, outperforming KNN but falling behind Logistic Regression and Random Forest.

**Precision & Recall:** The 0.83 precision and recall scores indicate that the model performed well but was prone to overfitting.

**F1-Score:** The 0.83 F1-score suggests that it performed adequately but struggled with generalization.

Since Decision Trees can easily memorize training data, they often overfit, resulting in lower performance on unseen data.

## Random Forest Classifier – 94% Accuracy

Random Forest, an ensemble model that combines multiple Decision Trees, achieved the highest accuracy (94%). By using bootstrap aggregation (bagging), it reduced overfitting, making it the most reliable model.

**Precision & Recall:** The 0.94 precision and recall scores indicate that Random Forest had the lowest misclassification rates.

**F1-Score:** The 0.94 F1-score confirms that the model achieved the best balance between precision and recall.

This model is highly effective for heart disease prediction, as it handles large datasets, non-linearity, and complex relationships between features.

## 4.2 Confusion Matrix Analysis

The confusion matrix provides insights into the number of correctly and incorrectly classified cases. The Random Forest model had the lowest misclassification rate, confirming higher reliability.

Key Observations from Confusion Matrices:

KNN misclassified several heart disease cases, leading to high false positives and false negatives.

Logistic Regression and Random Forest had lower misclassification rates, making them more suitable for clinical applications.

Decision Trees performed moderately well but overfit the data, causing higher false positives than Random Forest.

Model	False Positives (Misdiagnosed as Heart Disease)	False Negatives (Missed Heart Disease Cases)
KNN	High	High
Logistic Regression	Low	Low
Decision Tree	Moderate	Moderate
Random Forest	Very Low	Very Low

4.3 Discussion and Insights

The results suggest that Random Forest is the most reliable model for heart disease prediction. Logistic Regression is also an excellent alternative, especially in cases where interpretability is important.

KNN struggled with this dataset, as it is sensitive to high-dimensional features. Decision Trees, while effective, tend to overfit, reducing their generalizability.

\* Random Forest is the most robust model, as it reduces overfitting and provides the best balance between accuracy and generalization.

- \* Logistic Regression is useful for real-world applications, as it is computationally efficient and interpretable.
- \* Decision Trees perform well but require pruning to avoid overfitting.
- \* KNN is not ideal for this dataset, as it lacks efficiency with large feature spaces.

## 5.Conclusion

This study demonstrated the effectiveness of machine learning models in predicting heart disease, showing that ML techniques can enhance early diagnosis, reduce misdiagnosis, and improve patient outcomes. By analyzing clinical features such as age, cholesterol levels, blood pressure, heart rate, and chest pain type, the models successfully classified patients based on their likelihood of having heart disease. Among the models tested, the Random Forest Classifier emerged as the most reliable, achieving 94% accuracy, followed closely by Logistic Regression at 93%. The Decision Tree Classifier performed moderately well at 83%, while the K-Nearest Neighbors (KNN) algorithm showed the lowest performance at 74% due to its sensitivity to high-dimensionality issues and overlapping feature distributions.

### 5.1 Key Findings and Model Comparisons

The results indicate that Random Forest outperformed all other models, making it the most suitable choice for heart disease prediction. The strength of Random Forest lies in its ability to reduce overfitting by aggregating multiple decision trees, resulting in higher accuracy, robustness, and generalizability. Logistic Regression also performed exceptionally well, demonstrating high interpretability and computational efficiency, making it a viable option for real-world applications in clinical settings.

The Decision Tree classifier, while effective at identifying patterns within the data, was prone to overfitting, meaning it performed well on the training set but lacked generalizability on new data. On the other hand, KNN struggled with classifying heart disease cases due to the dataset's complexity and high-dimensional features, making it less suitable for medical applications.

The confusion matrix analysis provided further insights into model reliability and misclassification rates. The Random Forest model had the lowest number of false positives and false negatives, making it the most trustworthy model for heart disease prediction. KNN had the highest misclassification rate, reinforcing its unsuitability for this dataset. Logistic Regression and Decision Tree performed reasonably well, with Logistic Regression offering a better balance of interpretability and accuracy.

## 5.2 The Role of Machine Learning in Cardiology

Machine learning is revolutionizing the healthcare industry, particularly in predictive analytics and decision support systems. This study supports the growing body of evidence that ML-based models can play a crucial role in early risk assessment, improving diagnosis, and aiding clinical decision-making. By leveraging patient data, these models can identify high-risk individuals before symptoms manifest, allowing for timely medical intervention.

The integration of machine learning into cardiology can address several challenges in traditional heart disease diagnostics, including:

Reducing dependency on expensive diagnostic procedures, such as MRI scans, angiograms, and stress tests.

Providing rapid, automated analysis of patient health records, eliminating human error.

Enabling early detection and prevention, thereby reducing the global burden of cardiovascular diseases.

Despite these advantages, ML models should not replace medical professionals but should be used as decision-support tools. A combination of ML-based predictions and expert analysis will lead to better patient outcomes and increased diagnostic efficiency.

## 5.3 Future Enhancements and Research Directions

While this study demonstrated the efficacy of machine learning models, several areas require further exploration to enhance accuracy, reliability, and real-world applicability:

### 1. Deep Learning Models

Neural networks and deep learning algorithms (e.g., Convolutional Neural Networks - CNNs and Recurrent Neural Networks - RNNs) could further improve predictive performance.

Deep learning models can automatically extract hidden patterns from medical data, enhancing predictive accuracy.

## 2. Real-Time Patient Monitoring

Integrating ML models with wearable technology and IoT devices (e.g., smartwatches, heart rate monitors, and ECG patches) could facilitate real-time heart disease prediction.

ML algorithms can analyze continuous health data and provide alerts for abnormal heart activity, enabling early intervention.

## 3. Integration with Electronic Health Records (EHRs)

Combining ML models with electronic health records (EHRs) would allow for personalized heart disease risk assessments based on historical medical data.

This integration can streamline medical decision-making and improve patient management.

## 4. Explainability and Interpretability of ML Models

One of the challenges of ML in healthcare is the "black-box" nature of certain models (e.g., Random Forest and Deep Learning models).

Research should focus on explainable AI (XAI) techniques, ensuring that ML models provide clear, interpretable results that clinicians can trust.

## 5. Addressing Class Imbalances in Medical Datasets

Many medical datasets suffer from class imbalances, where non-disease cases significantly outnumber disease cases.

Techniques such as Synthetic Minority Over-sampling Technique (SMOTE) or cost-sensitive learning should be explored to enhance model fairness and accuracy.

### 5.4 The Future of AI-Driven Healthcare

The future of AI-driven healthcare is promising, with machine learning models evolving to become more precise, explainable, and integrated into clinical workflows. AI-powered diagnostic tools are being adopted worldwide to address healthcare challenges, such as:

AI-assisted imaging analysis (e.g., detecting heart abnormalities from echocardiograms).

Predictive analytics for chronic disease management (e.g., hypertension and diabetes risk prediction).



AI-powered chatbots and virtual assistants to provide patient education and self-monitoring guidance.

This study confirmed that machine learning offers a highly effective approach to predicting heart disease, with Random Forest emerging as the most reliable model (94% accuracy). Logistic Regression (93%) also proved to be a strong contender, particularly in scenarios requiring interpretability and efficiency. While Decision Trees (83%) showed promise, they were prone to overfitting, and KNN (74%) struggled with high-dimensional data.

The findings highlight that ML models can enhance early detection, reduce healthcare costs, and improve patient survival rates. However, their full potential lies in integrating with real-world healthcare systems, such as EHRs, wearable health monitors, and AI-powered clinical decision support systems.

Future work should focus on deep learning, real-time monitoring, and explainable AI techniques to make ML models more accurate, interpretable, and applicable to clinical settings. With continued advancements in AI and data-driven healthcare, ML models will play a crucial role in revolutionizing heart disease prediction and prevention, ultimately leading to better health outcomes for millions worldwide.

## References

1. Fedesoriano, Kaggle Dataset: Heart Failure Prediction. Available at: <https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction>
2. World Health Organization (WHO) Report on Cardiovascular Diseases. Available at: <https://www.who.int/>
3. Scikit-Learn: Machine Learning in Python. Available at: <https://scikit-learn.org/>
4. Pandas Development Team. (2023). pandas: Python Data Analysis Library. Available at: <https://pandas.pydata.org/>
5. NumPy Developers. (2023). NumPy: The Fundamental Package for Scientific Computing in Python. Available at: <https://numpy.org/>
6. Matplotlib Development Team. (2023). Matplotlib: Visualization with Python. Available at: <https://matplotlib.org/>
7. Seaborn Library. (2023). Seaborn: Statistical Data Visualization. Available at: <https://seaborn.pydata.org/>
8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830. Available at: <https://scikit-learn.org/>
9. Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques*. Elsevier.
10. Witten, I. H., Frank, E., & Hall, M. A. (2016). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

