Heart Failure Prediction

Gerard Corrales

Gabriel Duffy

Shiley-Marcos School of Engineering, University of San Diego

Abstract

Heart disease remains one of the leading causes of death worldwide, making early and accurate detection crucial for improving patient outcomes. This project investigates the application of machine learning techniques to predict heart disease using a dataset consisting of 918 patient records and 12 clinical features, including variables such as age, blood pressure, cholesterol levels, and other vital health indicators. The primary goal is to assess the predictive accuracy and interpretability of various machine learning models, including CART, C5.0, Random Forest, Naïve Bayes, and Neural Networks.

Extensive data preprocessing was conducted to prepare the dataset for analysis, addressing issues such as outliers, skewed distributions, and the transformation of categorical variables through one-hot encoding. The dataset was split into training and testing sets with a 75/25 ratio, and each model's performance was rigorously evaluated using metrics such as accuracy, precision, recall, and F1-score. Notably, the C5.0 algorithm emerged as the most accurate model, achieving an accuracy rate of 83%. Although the Neural Network model demonstrated a slightly higher accuracy of 87%, it lacked the interpretability that is often crucial in clinical settings.

This project highlights the potential of machine learning in supporting early detection of heart disease, with the C5.0 model offering a compelling balance between accuracy and interpretability. The Random Forest model further provided valuable insights into the importance of specific features, such as ST_Slope and ExerciseAngina, as key predictors. Future work could focus on enhancing predictive performance by exploring ensemble methods or more complex neural network architectures.

Table of Contents

Introduction

Heart disease is one of the leading causes of mortality worldwide (Roth & Mensah, 2020). Early detection and intervention are crucial for reducing heart disease and improving patient results. This project aims to leverage machine learning techniques to predict the presence of heart disease based on a set of descriptive features collected from patients. The first objective of this project is to conduct feature analysis and exploration to identify and understand which features are most indicative of heart disease. This involves exploratory data analysis (EDA) to uncover the relationships and patterns among the variables. Second, to develop and evaluate various classification models to predict heart disease, including CART, C5.0, Random Forests, Naïve Bayes Classification, and Neural Networks. Each model will be evaluated based on criteria such as accuracy, sensitivity, specificity, and misclassification.

The data set used in this project includes a variety of features such as age, resting blood pressure, cholesterol levels, fasting blood sugar, maximum heart rate, exercise-induced angina, and other medical indicators. These features will be preprocessed and standardized to ensure model performance. The expected outcome of this project is to identify the most accurate machine learning model for predicting heart disease.

Methodology

The dataset for this project was obtained from a public repository on GitHub. The dataset contains a total of 918 records and 12 features. The dataset includes a set of features collected from patients, such as age, sex, chest pain, resting blood pressure, cholesterol levels, fasting blood sugar, resting electrocardiogram results, maximum heart rate, exercise-induced angina, numeric value measured in depression (oldpeak), the slope of the peak exercise (ST), and heart disease.

Initial observations revealed no missing values in the dataset. However, outliers were identified in the MaxHR and Oldpeak columns, particularly for the z-scores of MaxHR and Oldpeak, indicating some extreme values that required attention. The dataset also exhibited varying degrees of skewness across features, with notable skewness in the Cholesterol, FastingBS, and Oldpeak columns, among others.

To ensure the data was suitable for analysis, a series of preprocessing steps were carried out. Initially, wrong values in the dataset were handled using K-Nearest Neighbors (KNN) imputation to estimate and fill in the gaps for numerical variables. Categorical variables such as Sex, ChestPainType, RestingECG, ExerciseAngina, and ST_Slope were transformed into numerical format using one-hot encoding, creating dummy variables for each category level. The Oldpeak variable, which had negative values, was standardized using Min-Max scaling to bring all features onto a common scale. This helped in ensuring that all features contribute equally during model training. Identified outliers in the MaxHR and Oldpeak columns were treated to ensure they do not skew the results. Figure 1 and Figure 2 show histograms of age distribution for patients with and without heart disease, with Figure 2 normalizing the distribution to proportionally compare the two groups.

Figure 1

Histogram of Age with Response Overlay

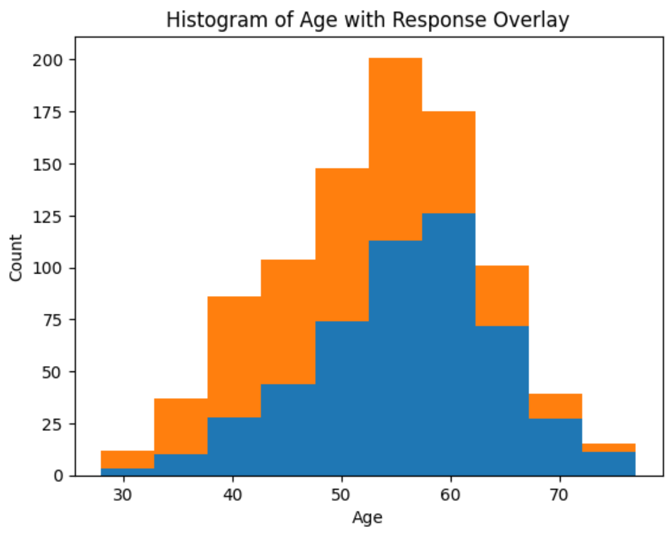
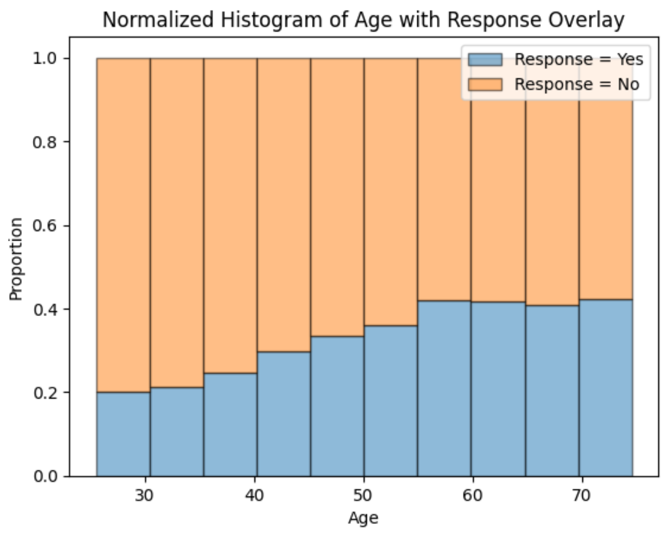


Figure 2

Normalized Histogram of Age with Response Overlay



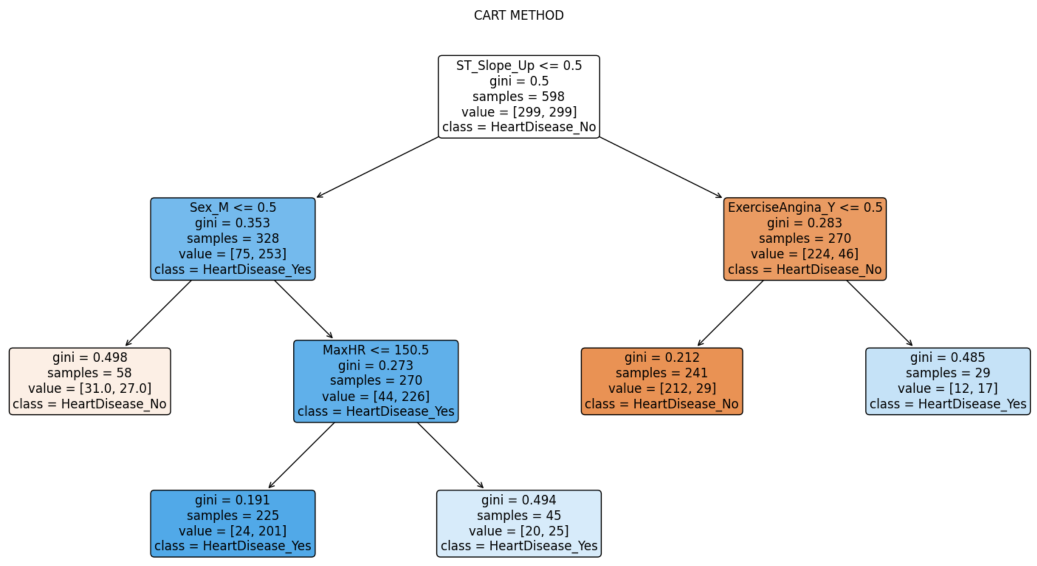
CART

The CART algorithm was employed to develop a predictive model for heart disease. CART is a decision tree algorithm that uses the Gini impurity criterion to split the data into homogenous groups. The CART model was trained on the preprocessed dataset, which included both numerical and categorical variables transformed via one-hot encoding. The dataset was split into training and testing sets, with 75% of the data used for training and 25% reserved for testing. The training set was balanced 50% yes and 50% no responses, it was used to build the decision tree, and it was utilized to evaluate the model's performance.

The model identified several key predictors of heart disease. Among the most significant features were ST_Slope, Sex, MaxHR, and ExerciseAngina (as seen in Figure 3). These variables exhibited the strongest influence on the model’s predictions, highlighting their critical role in assessing heart disease risk.

Figure 3

CART Model Decision Tree



The resulting decision tree provided a clear and interpretable set of rules for predicting heart disease. For instance, the root node split on ST_Slope, indicating that patients with a ST_Slope value of less than or equal to 0.5 were more likely to not have heart disease. Subsequent splits on features like Sex and MaxHR further refined the classification, demonstrating that male patients with a MaxHR of less than or equal to 150.5 were more likely to have heart disease. The decision tree also indicated that patients with an ExerciseAngina value of less than or equal to 0.5 were more likely to not have heart disease. The accuracy of the model was found to be 80%, indicating a strong ability to correctly classify patients as having heart disease or not. Additionally, the confusion matrix showed a good balance between sensitivity (true positive rate) and specificity (true negative rate), suggesting that the model effectively identifies both positive and negative cases. In Table 1 shows the results of the classification report.

Table 1

Results of Classification Report

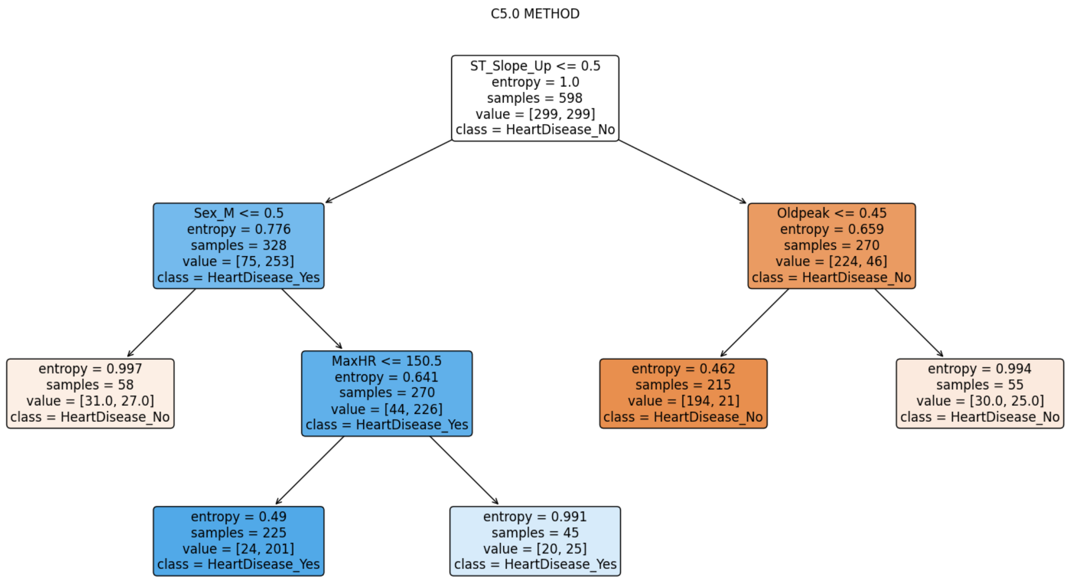
	Precision	**Recall**	**F1-Score**	**Support**
0	0.80	0.77	0.79	111
1	0.80	0.82	0.81	119
Accuracy			0.80	230

C5.0

The C5.0 model was trained on the training dataset. The training and testing splits were identical, with 75% of the data used for training and 25% reserved for testing. The training set was used to construct the decision tree, and the testing set was employed to evaluate the model's performance. The C5.0 model identified several critical predictors of heart disease. The most influential features were ST_Slope, Sex, MaxHR, and Oldpeak. The resulting decision tree provided an interpretable set of rules for heart disease prediction. As seen in Figure 4, the root node split on ST_Slope, indicating that patients with an ST_Slope value of less than or equal to 0.5 were more likely to not have heart disease. For example, male patients (Sex_M <= 0.5) with a MaxHR of less than or equal to 150.5 were more likely to have heart disease. Additionally, patients with an Oldpeak value of less than or equal to 0.45 were more likely to not have heart disease.

Figure 4

C5.0 Results



The C5.0 model demonstrated robust performance on the test dataset, with an accuracy of 83%. The confusion matrix (Table 2) revealed a good balance between sensitivity and specificity, indicating the model's efficacy in identifying both positive and negative cases of heart disease.

Table 2

C5.0 Confusion Matrix

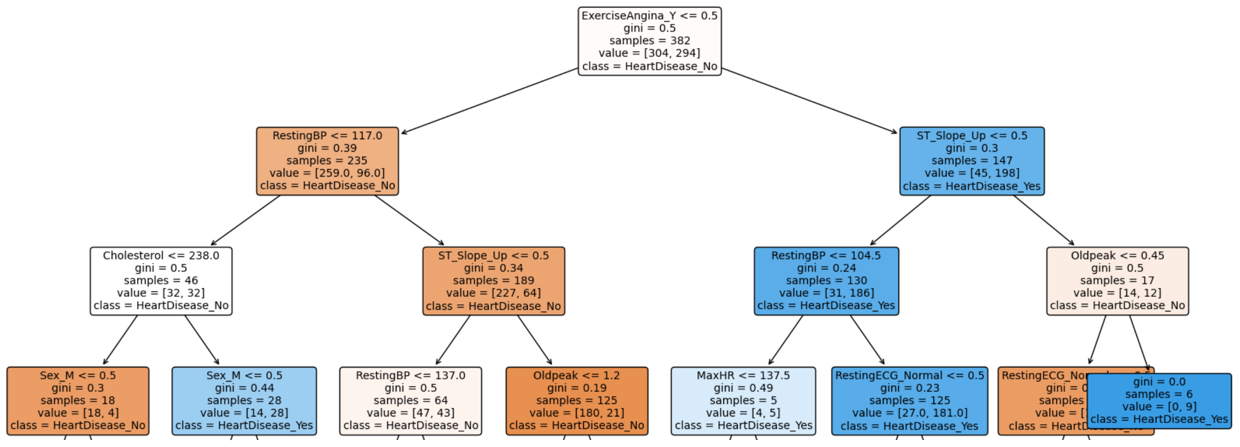
	Precision	**Recall**	**F1-Score**	**Support**
0	0.78	0.90	0.83	111
1	0.89	0.76	0.82	119
Accuracy			0.83	230

Random Forest

The Random Forest model provided valuable insights into the importance of different features in predicting heart disease. As seen in Figure 5, the most important features identified by the model included ExerciseAngina_Y, ST_Slope_Up, Cholesterol and Sex_M, MaxHR and Oldpeak, and RestingBP. ExerciseAngina indicates that patients without exercise-induced angina (ExerciseAngina_Y <= 0.5) had a higher likelihood of not having heart disease. The slope of the ST segment during peak exercise was also a crucial factor. Patients with an upward ST slope (ST_Slope_Up <= 0.5) showed a lower likelihood of heart disease. Resting blood pressure appeared multiple times in the tree, with various thresholds influencing the likelihood of heart disease. Lower resting blood pressure values were associated with a higher probability of heart disease. Cholesterol levels and gender (Sex_M) also played significant roles in the decision tree, with different thresholds impacting the classification. Maximum heart rate achieved and the ST depression induced by exercise relative to rest (Oldpeak) were other important predictors, influencing subsequent splits in the tree.

Figure 5

Random Forest Decision Tree



Naïve Bayes

When evaluating the Naive Bayes model, the confusion matrix (Table 3) yielded the following results: 88 true negatives (cases where no heart disease was correctly identified), 23 false positives (cases where heart disease was incorrectly predicted), 20 false negatives (cases where heart disease was not predicted but was present), and 99 true positives (cases where heart disease was correctly predicted). This evaluation resulted in an overall model accuracy of 81%. The feature log probabilities for each class (heart disease present or not) were visualized using a heatmap.

Table 3

Naïve Bayes Confusion Matrix

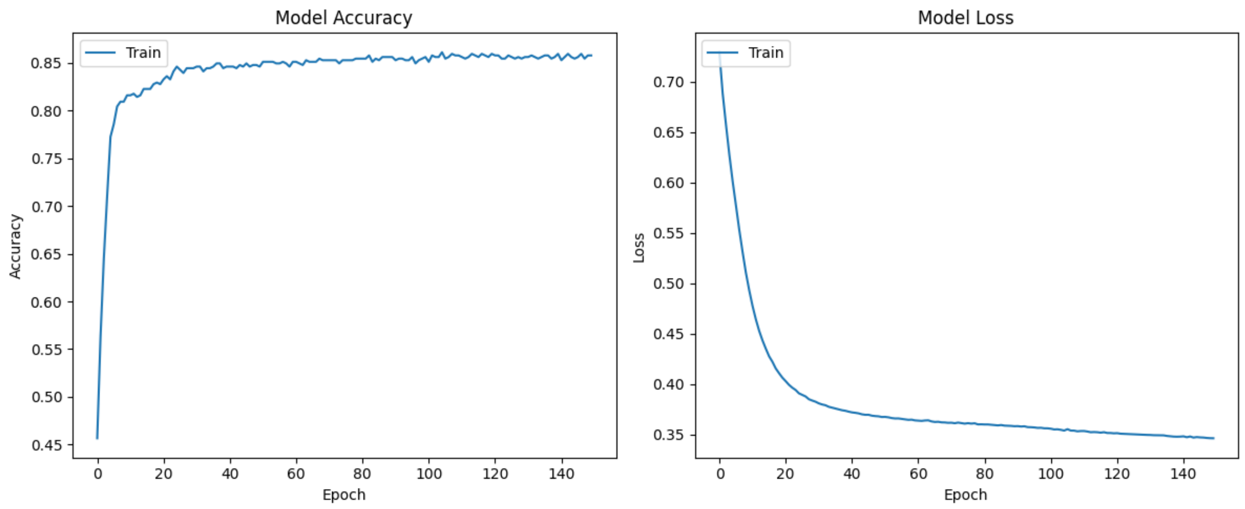
Predicted	**False**	**True**	**Total**
Actual			
0	88	23	111
1	20	99	119
Total	108	122	230

Neural Networks

The model was compiled using the Adam optimizer and binary cross-entropy loss function, suitable for binary classification tasks. It was trained over 150 epochs with a batch size determined automatically by Keras. The training dataset was used to fit the model, achieving an accuracy of approximately 87% (see Figure 6), indicating the model's ability to correctly classify the presence or absence of heart disease. After training, the model was evaluated on the test dataset. While neural networks do not provide feature importance in the same way as decision trees or other interpretable models, the architecture and weights learned during training highlight the complex interactions between features.

Figure 6

Neural Networks Accuracy



Results

The comparative analysis of various machine learning models yielded compelling insights into their effectiveness in predicting heart disease. The C5.0 algorithm demonstrated superior performance, achieving an accuracy of 83%. Its balance between precision and recall was evident in a robust F1-score, underscoring its ability to accurately classify both positive and negative cases. The confusion matrix revealed strong sensitivity and specificity, highlighting the model's reliability in a clinical context where minimizing misclassification is critical.

While the Neural Network model achieved the highest accuracy at 87%, its application was constrained by a lack of interpretability, which poses challenges in healthcare settings requiring transparent decision-making. Despite its predictive strength, the model's opacity in feature contribution limits its practical utility for clinicians. The Random Forest model, with an accuracy of 81%, provided valuable feature importance rankings, identifying variables such as ST_Slope and ExerciseAngina as key predictors of heart disease. This model's capacity to elucidate feature significance is particularly advantageous for understanding the underlying risk factors.

The CART and Naïve Bayes models, achieving accuracies of 80% and 81% respectively, offered strong performance through their simplicity and ease of interpretation. These models provided clear decision rules and probabilistic estimates, making them suitable for scenarios requiring quick, transparent analysis. Overall, the results indicate that while C5.0 offers a well-rounded balance of accuracy and interpretability, Random Forest is particularly beneficial for its insights into feature importance, making both models highly effective for heart disease prediction.

Conclusion

The comprehensive evaluation of machine learning models for heart disease prediction has provided valuable insights into their strengths and practical applications. The C5.0 algorithm, with an accuracy of 83%, proved to be highly effective in balancing precision and recall, making it a robust tool for clinical environments where interpretability is crucial. This model's clear decision rules enable healthcare practitioners to understand and trust the predictions, facilitating informed decision-making. The practical implication of this is that C5.0 can be effectively utilized in diagnostic tools to aid clinicians in identifying patients at risk of heart disease, thus enhancing early intervention and treatment strategies.

Conversely, the Neural Network model achieved the highest accuracy of 87%, demonstrating its superior predictive power. However, its complexity poses challenges for clinical adoption due to the "black box" nature, where the decision-making process is not transparent. Despite its high performance, deploying Neural Networks in clinical settings would necessitate additional efforts to ensure that predictions are interpretable and actionable. This suggests a need for complementary methods or tools that can bridge the gap between high accuracy and practical usability, potentially through explainable AI techniques or hybrid models that combine the strengths of neural networks with more transparent methods.

The Random Forest model, with an accuracy of 81%, provided significant insights into feature importance, such as ST_Slope and ExerciseAngina, which are critical for understanding heart disease risk factors. This feature importance allows for more targeted risk assessments and personalized healthcare strategies, improving the precision of patient evaluations. The CART and Naïve Bayes models, while offering simpler decision rules and probabilistic predictions, can be valuable for scenarios requiring quick, transparent assessments. Integrating these models into a cohesive strategy, perhaps through ensemble techniques, could enhance overall predictive performance while maintaining clarity and usability. By leveraging the strengths of these various approaches, healthcare providers can develop more effective tools for early heart disease detection, ultimately leading to improved patient outcomes.

References

Larose, C., & Larose, D. (2019). *Data Science Using Python and R*. Wiley.

Tan, P.-N., Steinbach, M., Karpatne, A., & Kumar, V. (2020). *Introduction to data mining*

(Second Edition). Pearson.

Roth, G. A., Mensah. (2020). Global Burden of Cardiovascular Diseases and Risk Factors, 1990-2019: Update From the GBD 2019 Study. *Journal of the American College of Cardiology*, *76*(25), 2982–3021. https://doi.org/10.1016/j.jacc.2020.11.010

Jayachandru001. (2021). *GitHub - jayachandru001/Heart-Failure-Prediction-: This project involves training of Machine Learning models to predict the Heart Failure for Heart Disease event. In this KNN gives a high Accuracy of 89%.* GitHub. https://github.com/jayachandru001/Heart-Failure-Prediction-/tree/main

Appendix

August 9, 2024

0.1 Data Preparation

All The Libraries Used

```
[ ]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     from scipy import stats
     import seaborn as sns
     from sklearn.model_selection import train_test_split
     import random
     import statsmodels.tools.tools as stattools
     from sklearn.tree import DecisionTreeClassifier, export_graphviz, plot_tree
     import graphviz
     from sklearn.impute import KNNImputer
     from sklearn.metrics import classification_report, confusion_matrix,␣
      ↪accuracy_score
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.naive_bayes import MultinomialNB
     from sklearn.preprocessing import MinMaxScaler
     import tensorflow as tf
     from sklearn import tree
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Input
     from sklearn.preprocessing import LabelEncoder
     import networkx as nx
```

Importing data set

```
[ ]: heart = pd.read_csv("heart.csv")
```

Visualizing the header

```
[ ]: heart.head()
```

```
[ ]:    Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
     0   55   M           NAP          0            0          0     Normal    155
     1   53   M           ASY         80            0          0     Normal    141
     2   32   M            TA         95            0          1     Normal    127
     3   51   M           ASY         95            0          1     Normal    126
     4   57   M           ASY         95            0          1     Normal    182
```

```
  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      1.5     Flat             1
1              Y      2.0     Down             0
2              N      0.7       Up             1
3              N      2.2     Flat             1
4              N      0.7     Down             1
```

Exploring data types

[ ]:
```
heart.dtypes
```

[ ]:
```
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
```

[ ]:
```
heart.size
```

[ ]:
```
11016
```

Changing columns object to category type

[ ]:
```
for column in ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina',␣
 ↪'ST_Slope']:
    heart[column] = heart[column].astype('category')

heart.dtypes
```

[ ]:
```
Age                  int64
Sex               category
ChestPainType     category
RestingBP            int64
Cholesterol          int64
FastingBS            int64
RestingECG        category
MaxHR                int64
ExerciseAngina    category
Oldpeak            float64
ST_Slope          category
```

```
HeartDisease         int64
dtype: object
```

Checking for missing values

```
[ ]: heart.isna().sum()
```

```
[ ]: 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
```

Checking for duplicates

```
[ ]: duplicates = heart.duplicated().sum()

     print("Duplicates found: ", duplicates)
```

```
Duplicates found:  0
```

Unique values

```
[ ]: heart.nunique()
```

```
[ ]: Age                50
     Sex                 2
     ChestPainType       4
     RestingBP          67
     Cholesterol       222
     FastingBS           2
     RestingECG          3
     MaxHR             119
     ExerciseAngina      2
     Oldpeak            53
     ST_Slope            3
     HeartDisease        2
     dtype: int64
```

Summary of The Central Tendency, Dispersion, and Shape

```
[ ]: heart.describe().T
[ ]:                count        mean         std   min     25%    50%    75%    max
     Age            918.0   53.510893    9.432617  28.0   47.00   54.0   60.0   77.0
     RestingBP      918.0  132.396514   18.514154   0.0  120.00  130.0  140.0  200.0
     Cholesterol    918.0  198.799564  109.384145   0.0  173.25  223.0  267.0  603.0
     FastingBS      918.0    0.233115    0.423046   0.0    0.00    0.0    0.0    1.0
     MaxHR          918.0  136.809368   25.460334  60.0  120.00  138.0  156.0  202.0
     Oldpeak        918.0    0.887364    1.066570  -2.6    0.00    0.6    1.5    6.2
     HeartDisease   918.0    0.553377    0.497414   0.0    0.00    1.0    1.0    1.0
```

Creating an index in our data set

```
[ ]: print("Number of rows: ", heart.shape[0])
     print("Number of columns: ", heart.shape[1])

     # Creating new variable Index
     heart['Index'] = pd.Series(range(0, 918))

     heart.head()
```

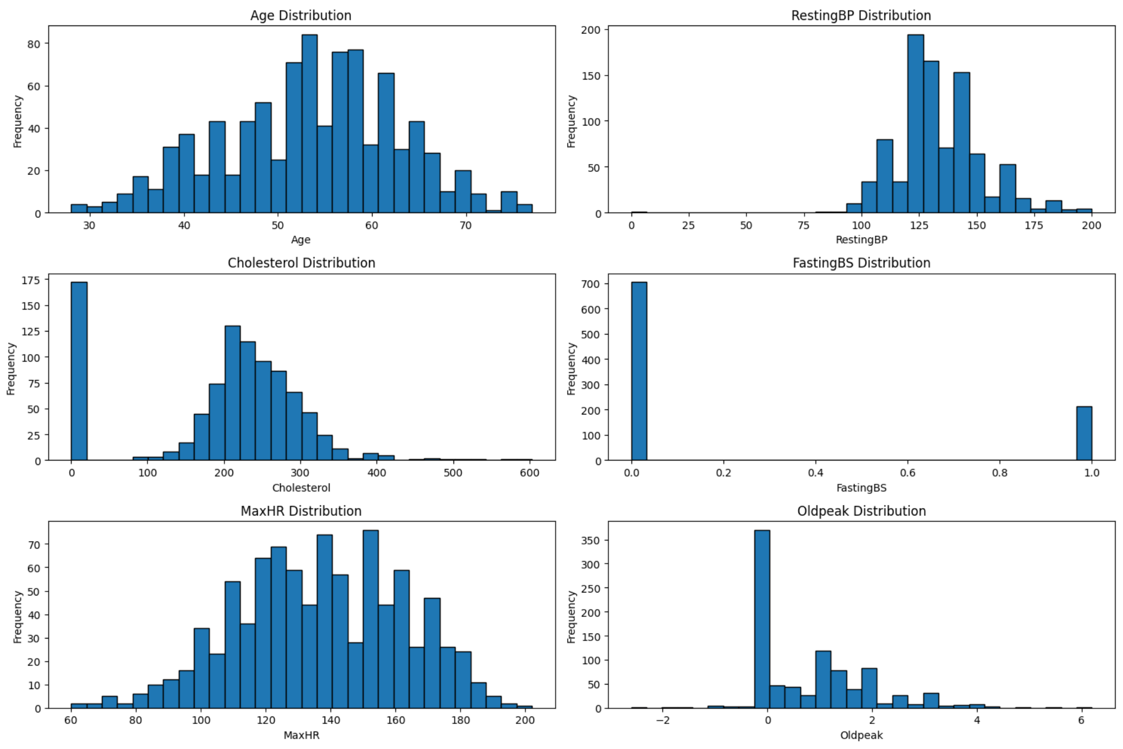
```
Number of rows:  918
Number of columns:  12
```

```
[ ]:    Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
     0   55   M           NAP          0            0          0     Normal    155
     1   53   M           ASY         80            0          0     Normal    141
     2   32   M            TA         95            0          1     Normal    127
     3   51   M           ASY         95            0          1     Normal    126
     4   57   M           ASY         95            0          1     Normal    182

       ExerciseAngina  Oldpeak ST_Slope  HeartDisease  Index
     0              N      1.5     Flat             1      0
     1              Y      2.0     Down             0      1
     2              N      0.7       Up             1      2
     3              N      2.2     Flat             1      3
     4              N      0.7     Down             1      4
```

First Visualization to Detect Anomalies in Numerical Data

```
[ ]: numerical_columns = ['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR',␣
       ↪'Oldpeak']

     fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 10))
     axes = axes.flatten()

     for i, column in enumerate(numerical_columns):
         axes[i].hist(heart[column], bins=30, edgecolor='black')
         axes[i].set_title(f'{column} Distribution')
         axes[i].set_xlabel(column)
         axes[i].set_ylabel('Frequency')
```

```
plt.tight_layout()

plt.show()
```



Changing Misleading Field Values Replacing 0 values in “Cholesterol” and “RestingBP” attributes with “nan” because it’s highly unlikely to have 0 cholesterol or 0 Blood pressure.

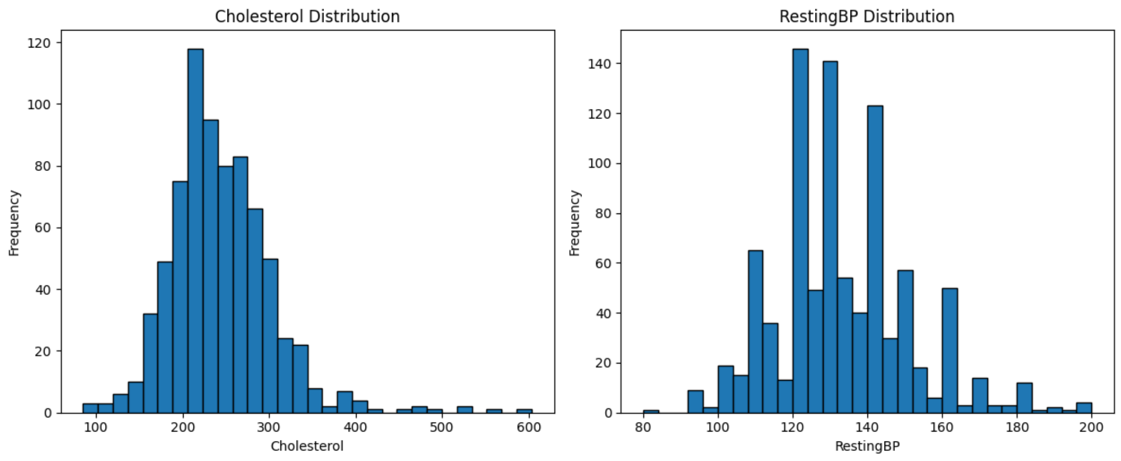
[ ]:
```
heart['Cholesterol'] = heart['Cholesterol'].replace({0: np.nan})
heart['RestingBP'] = heart['RestingBP'].replace({0: np.nan})
```

Showing results after changing misleading values in those columns

[ ]:
```
columns_to_show = ['Cholesterol', 'RestingBP']

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))

for i, column in enumerate(columns_to_show):
    axes[i].hist(heart[column], bins=30, edgecolor='black')
    axes[i].set_title(f'{column} Distribution')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Frequency')
```

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

```
[ ]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Standardizing Numeric Fields to Detect Outliers

```
[ ]: numerical_columns = ['Age', 'RestingBP', 'Cholesterol', 'FastingBS',
                          'MaxHR', 'Oldpeak']
     heart_ZScore = pd.DataFrame()

     for column in numerical_columns:
         # Calculating Z-score (standardizing)
         heart_ZScore[f'{column}_Z'] = stats.zscore(heart[column])



     # Identifying Outliers
     heart_age_outliers = heart_ZScore.query('Age_Z > 3 | Age_Z < -3')[['Age_Z']]
     heart_resting_outliers = heart_ZScore.query('RestingBP_Z > 3 | RestingBP_Z <␣
      ↪-3')[[ 'RestingBP_Z']]
     heart_cholesterol_outliers = heart_ZScore.query('Cholesterol_Z > 3 |␣
      ↪Cholesterol_Z < -3')[['Cholesterol_Z']]
     heart_fasting_outliers = heart_ZScore.query('FastingBS_Z > 3 | FastingBS_Z <␣
      ↪-3')[[ 'FastingBS_Z']]
     heart_maxhr_outliers = heart_ZScore.query('MaxHR_Z > 3 | MaxHR_Z <␣
      ↪-3')[['MaxHR_Z']]
     heart_oldpeak_outliers = heart_ZScore.query('Oldpeak_Z > 3 | Oldpeak_Z < -3')[[␣
      ↪'Oldpeak_Z']]

     print("Age")
     print(heart_age_outliers)
```

```python
print("\nRestingBP_Z")
print(heart_resting_outliers)
print("\nCholesterol_Z")
print(heart_cholesterol_outliers)
print("\nFastingBS_Z")
print(heart_fasting_outliers)
print("\nMaxHR_Z")
print(heart_maxhr_outliers)
print("\nOldpeak_Z")
print(heart_oldpeak_outliers)
```

```
Age
Empty DataFrame
Columns: [Age_Z]
Index: []

RestingBP_Z
Empty DataFrame
Columns: [RestingBP_Z]
Index: []

Cholesterol_Z
Empty DataFrame
Columns: [Cholesterol_Z]
Index: []

FastingBS_Z
Empty DataFrame
Columns: [FastingBS_Z]
Index: []

MaxHR_Z
      MaxHR_Z
126 -3.018469

Oldpeak_Z
     Oldpeak_Z
9    -3.271482
208   4.983762
421   4.420905
512   3.858047
707   3.107570
809   3.107570
855   3.295190
```

Columns “MaxHR_Z” and “Oldpeak_Z” contain many outliers.

Changing target variable values to ‘Yes’ and ‘No’

```
[ ]: Disease_dict = {1: 'Yes', 0: 'No'}

     heart['HeartDisease_categorical'] = heart['HeartDisease'].replace(Disease_dict)

     heart['HeartDisease_categorical'] = heart['HeartDisease_categorical'].
      ↪astype('category')

     heart['HeartDisease_categorical']
```

```
[ ]: 0      Yes
     1       No
     2      Yes
     3      Yes
     4      Yes
           …
     913    Yes
     914    Yes
     915    Yes
     916     No
     917    Yes
     Name: HeartDisease_categorical, Length: 918, dtype: category
     Categories (2, object): ['No', 'Yes']
```

0.2 Exploratory Data Analysis (EDA)

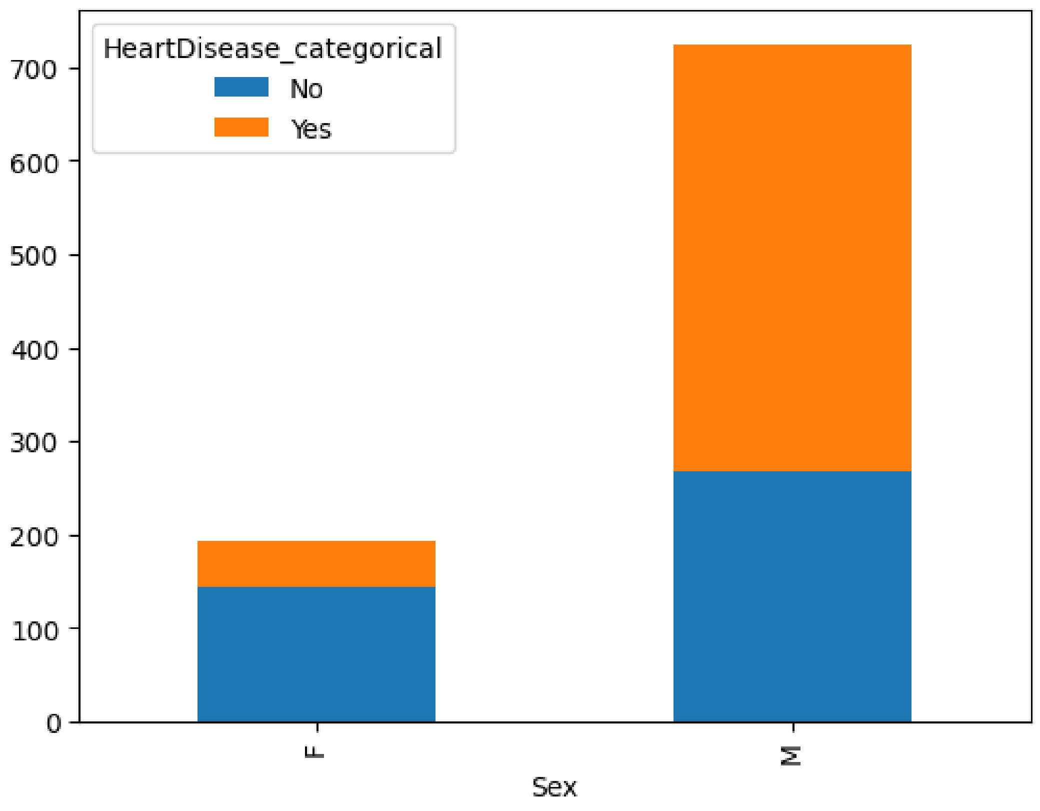
Exploring Categorical Features Using Bar Graph with Response Overlay

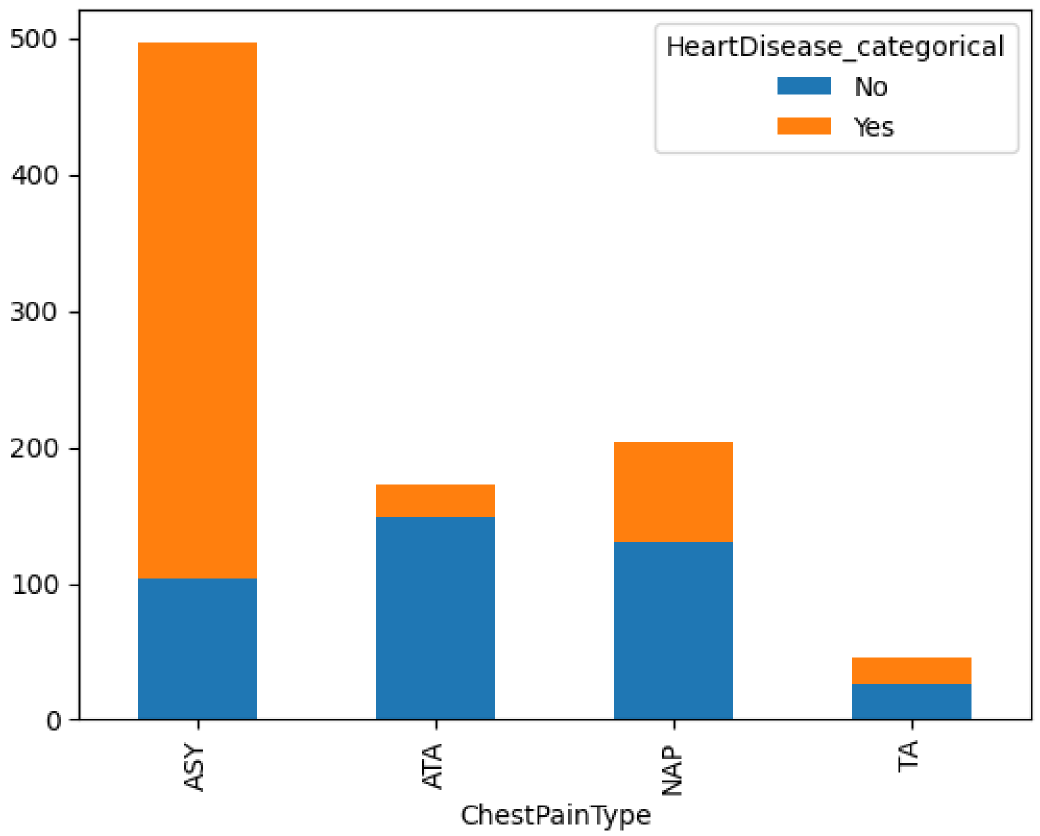
```
[ ]: # Creating a contingency table
     crosstab_01 = pd.crosstab(heart['Sex'],  heart['HeartDisease_categorical'])
     crosstab_02 = pd.crosstab(heart['ChestPainType'],␣
      ↪heart['HeartDisease_categorical'])
     crosstab_03 = pd.crosstab(heart['RestingECG'],␣
      ↪heart['HeartDisease_categorical'])
     crosstab_04 = pd.crosstab(heart['ExerciseAngina'],␣
      ↪heart['HeartDisease_categorical'])
     crosstab_05 = pd.crosstab(heart['ST_Slope'], heart['HeartDisease_categorical'])

     # Calculating Column Proportions
     proportions_01 = round(crosstab_01.div(crosstab_01.sum(0), axis=1) * 100, 1)
     proportions_02 = round(crosstab_02.div(crosstab_02.sum(0), axis=1) * 100, 1)
     proportions_03 = round(crosstab_03.div(crosstab_03.sum(0), axis=1) * 100, 1)
     proportions_04 = round(crosstab_04.div(crosstab_04.sum(0), axis=1) * 100, 1)
     proportions_05 = round(crosstab_05.div(crosstab_05.sum(0), axis=1) * 100, 1)


     print(proportions_01)
     print("\n", proportions_02)
     print("\n", proportions_03)
```

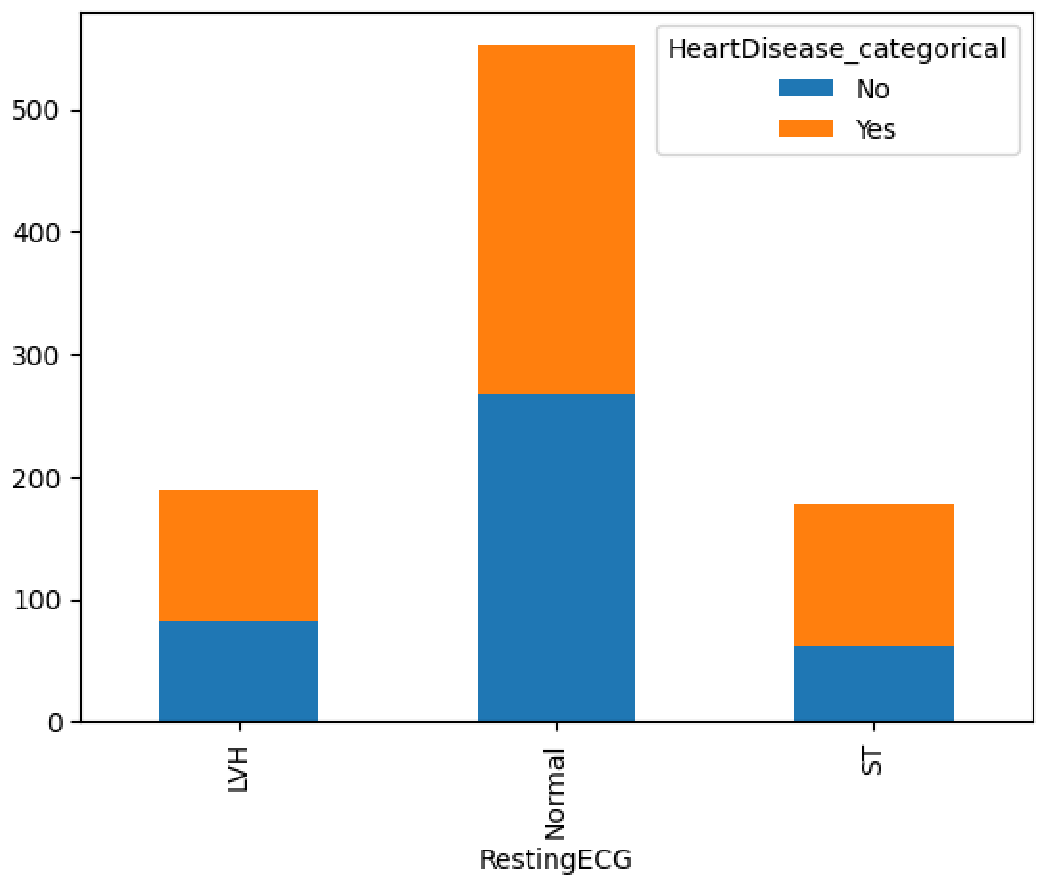
```python
print("\n", proportions_04)
print("\n", proportions_05)

# Bar graph
crosstab_01.plot(kind='bar', stacked= True)
crosstab_02.plot(kind='bar', stacked= True)
crosstab_03.plot(kind='bar', stacked= True)
crosstab_04.plot(kind='bar', stacked= True)
crosstab_05.plot(kind='bar', stacked= True)
```

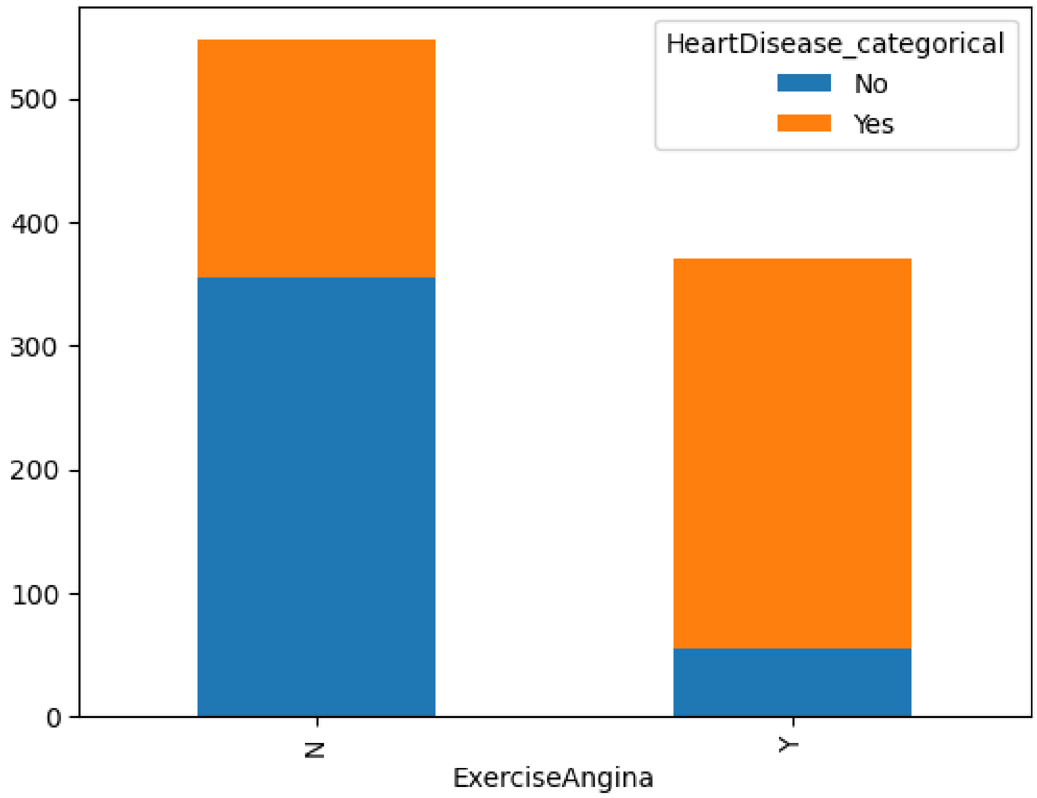
```
HeartDisease_categorical    No   Yes
Sex
F                         34.9   9.8
M                         65.1  90.2

 HeartDisease_categorical    No   Yes
ChestPainType
ASY                       25.4  77.2
ATA                       36.3   4.7
NAP                       32.0  14.2
TA                         6.3   3.9

 HeartDisease_categorical    No   Yes
RestingECG
LVH                       20.0  20.9
Normal                    65.1  56.1
ST                        14.9  23.0

 HeartDisease_categorical    No   Yes
ExerciseAngina
N                         86.6  37.8
Y                         13.4  62.2

 HeartDisease_categorical    No   Yes
ST_Slope
Down                       3.4   9.6
Flat                      19.3  75.0
Up                        77.3  15.4
```

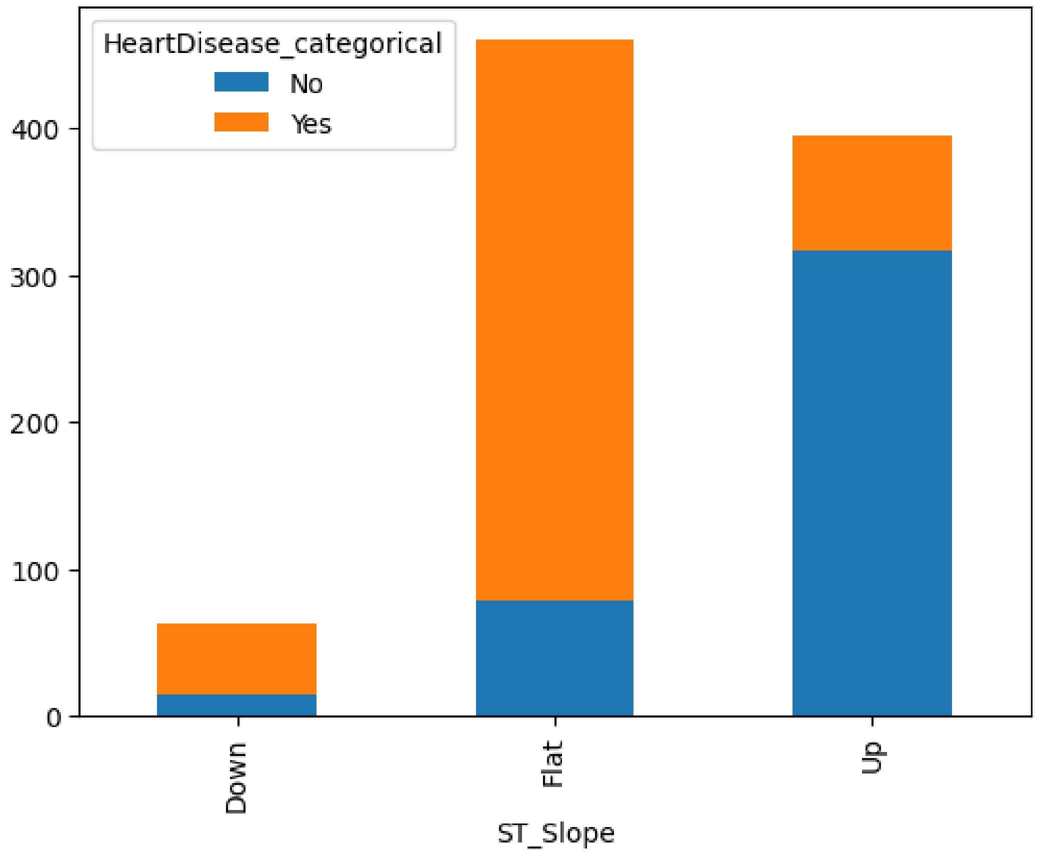
```
[ ]: <Axes: xlabel='ST_Slope'>
```


HeartDisease_categorical
No
Yes
0
100
200
300
400
500
600
700
F
M
Sex


HeartDisease_categorical
No
Yes
0
100
200
300
400
500
ASY
ATA
NAP
TA
ChestPainType


HeartDisease_categorical
No
Yes
0
100
200
300
400
500
LVH
Normal
ST
RestingECG

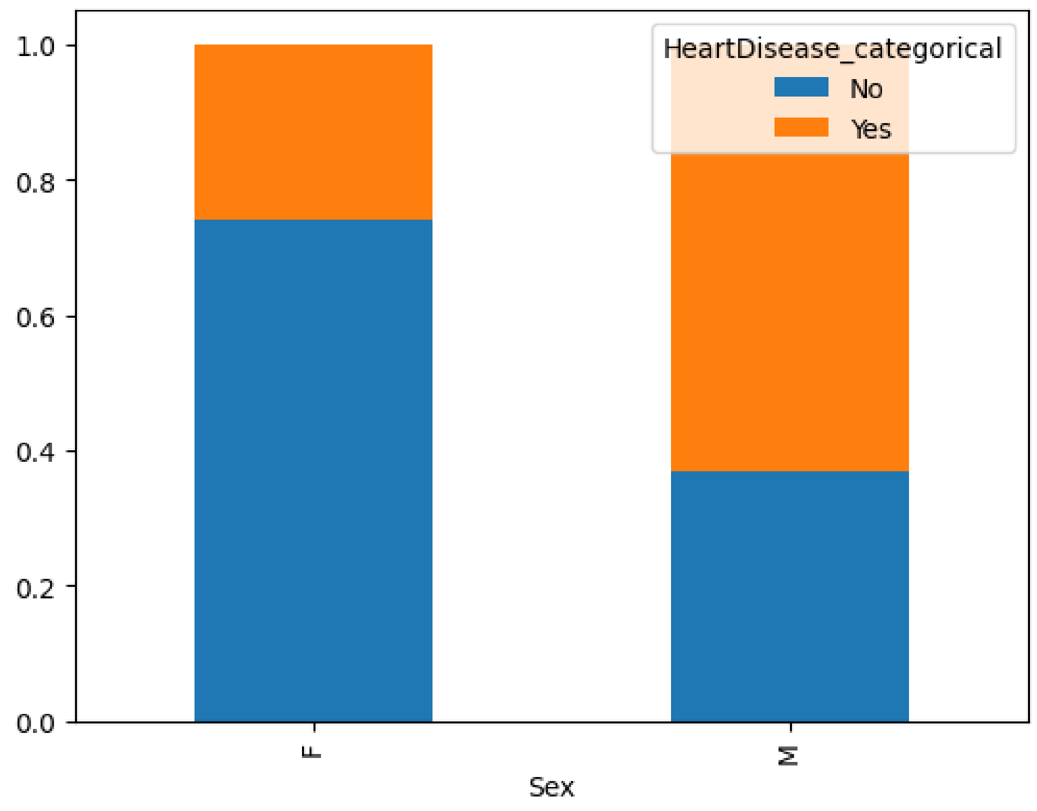

HeartDisease_categorical
No
Yes
0
100
200
300
400
500
N
Y
ExerciseAngina

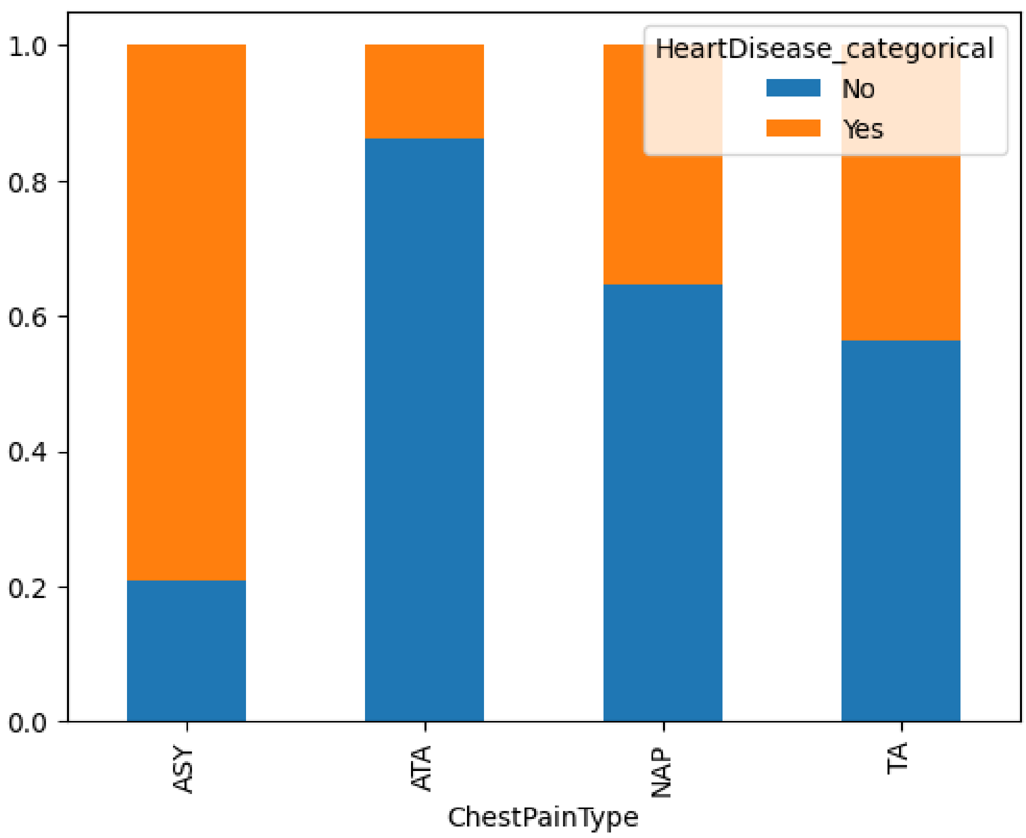


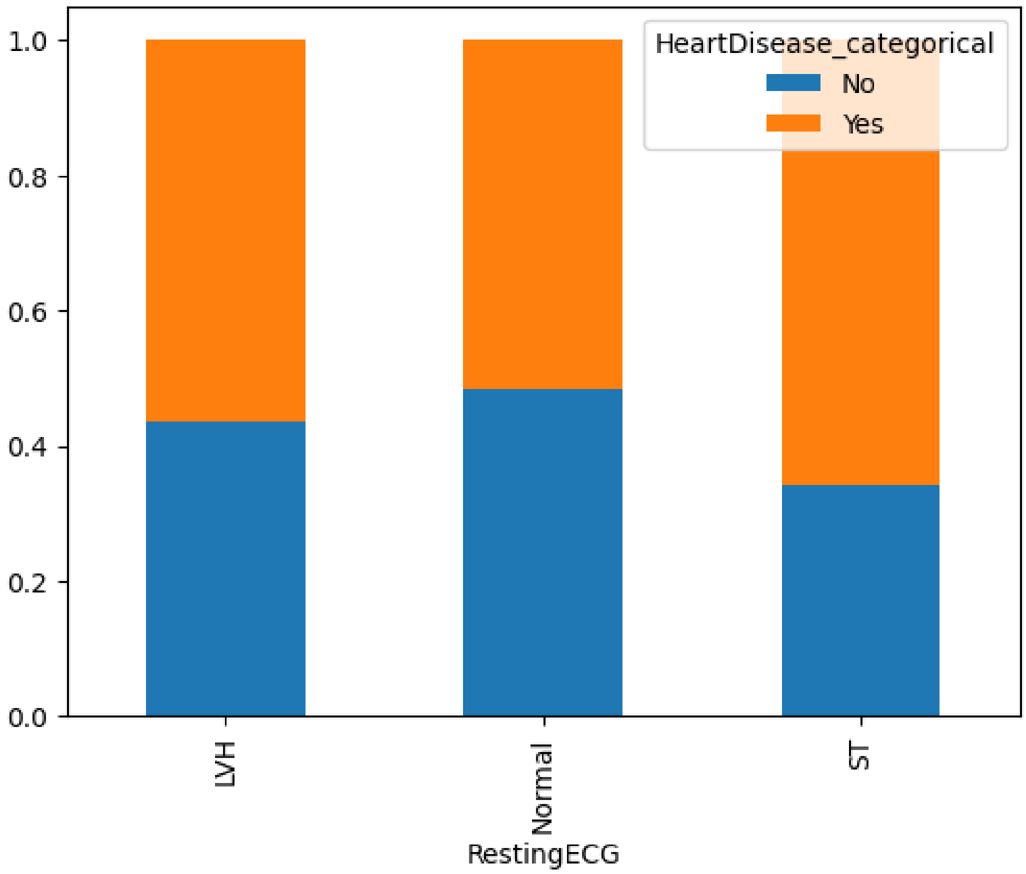
Creating a Normalized Bar Graph

```
[ ]: crosstab_norm_01 = crosstab_01.div(crosstab_01.sum(1), axis= 0)
     crosstab_norm_02 = crosstab_02.div(crosstab_02.sum(1), axis= 0)
     crosstab_norm_03 = crosstab_03.div(crosstab_03.sum(1), axis= 0)
     crosstab_norm_04 = crosstab_04.div(crosstab_04.sum(1), axis= 0)
     crosstab_norm_05 = crosstab_05.div(crosstab_05.sum(1), axis= 0)

     crosstab_norm_01.plot(kind='bar', stacked= True)
     crosstab_norm_02.plot(kind='bar', stacked= True)
     crosstab_norm_03.plot(kind='bar', stacked= True)
     crosstab_norm_04.plot(kind='bar', stacked= True)
     crosstab_norm_05.plot(kind='bar', stacked= True)
```

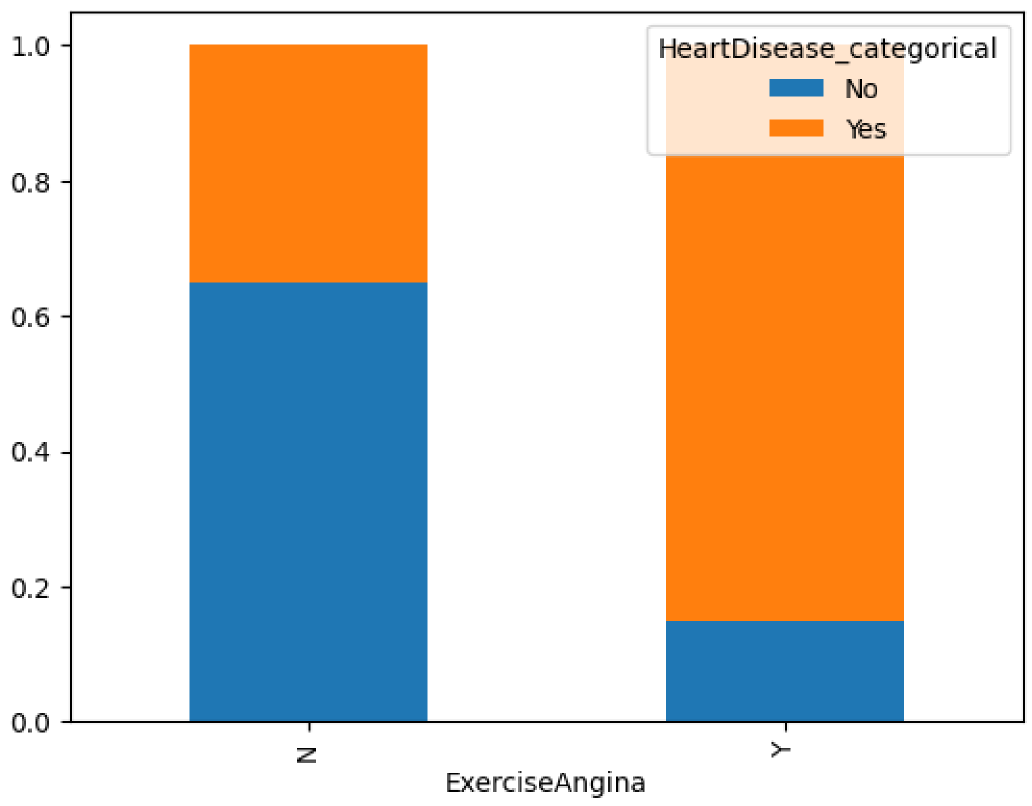
```
[ ]: <Axes: xlabel='ST_Slope'>
```

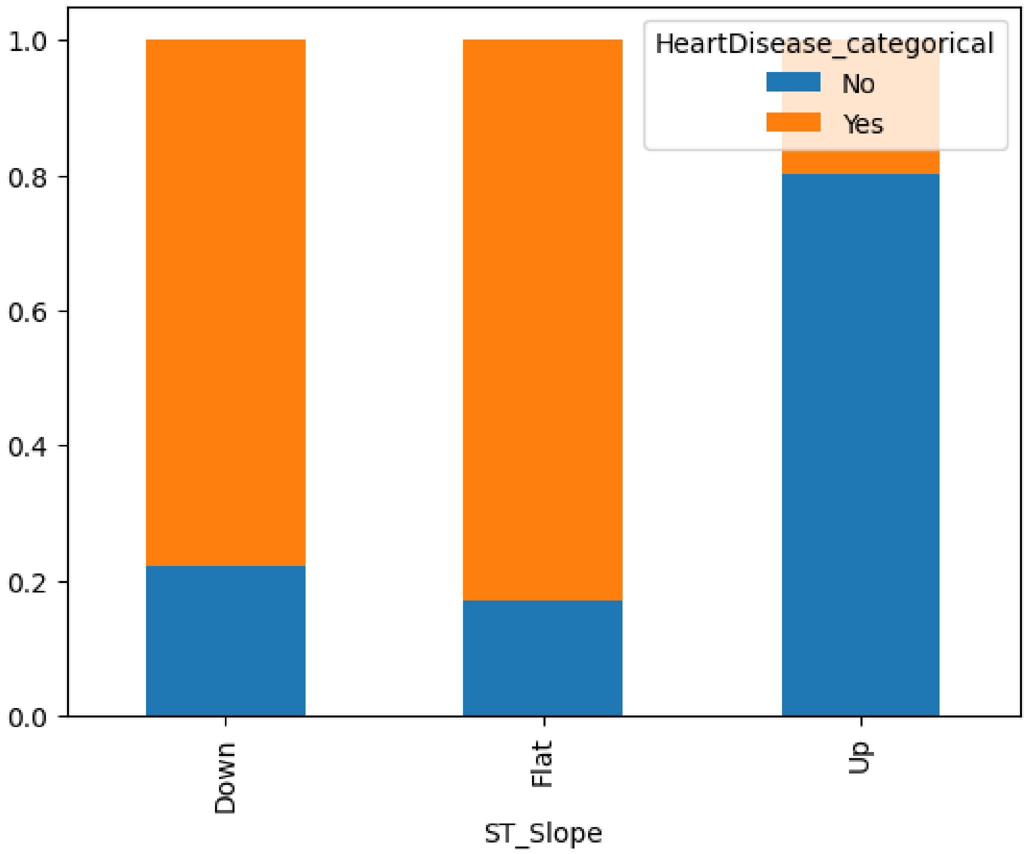

HeartDisease_categorical
No
Yes
1.0
0.8
0.6
0.4
0.2
0.0
F
M
Sex


HeartDisease_categorical
No
Yes
1.0
0.8
0.6
0.4
0.2
0.0
ASY
ATA
NAP
TA
ChestPainType



HeartDisease_categorical
No
Yes
1.0
0.8
0.6
0.4
0.2
0.0
LVH
Normal
ST
RestingECG

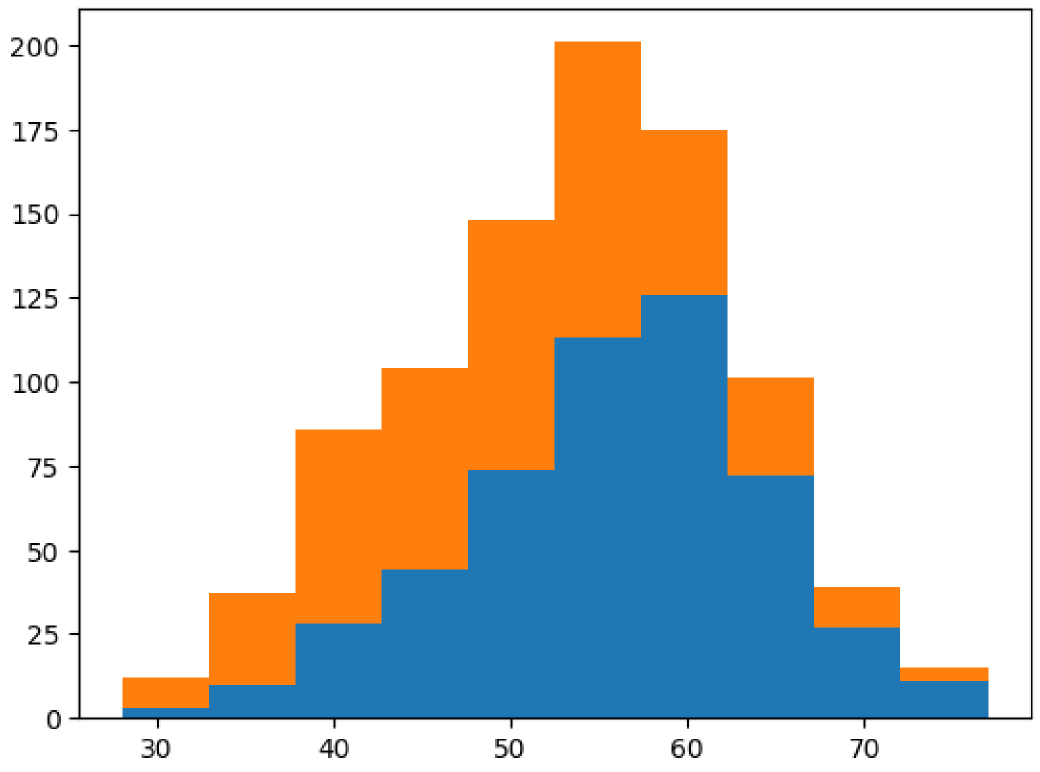

HeartDisease_categorical
No
Yes
1.0
0.8
0.6
0.4
0.2
0.0
N
Y
ExerciseAngina

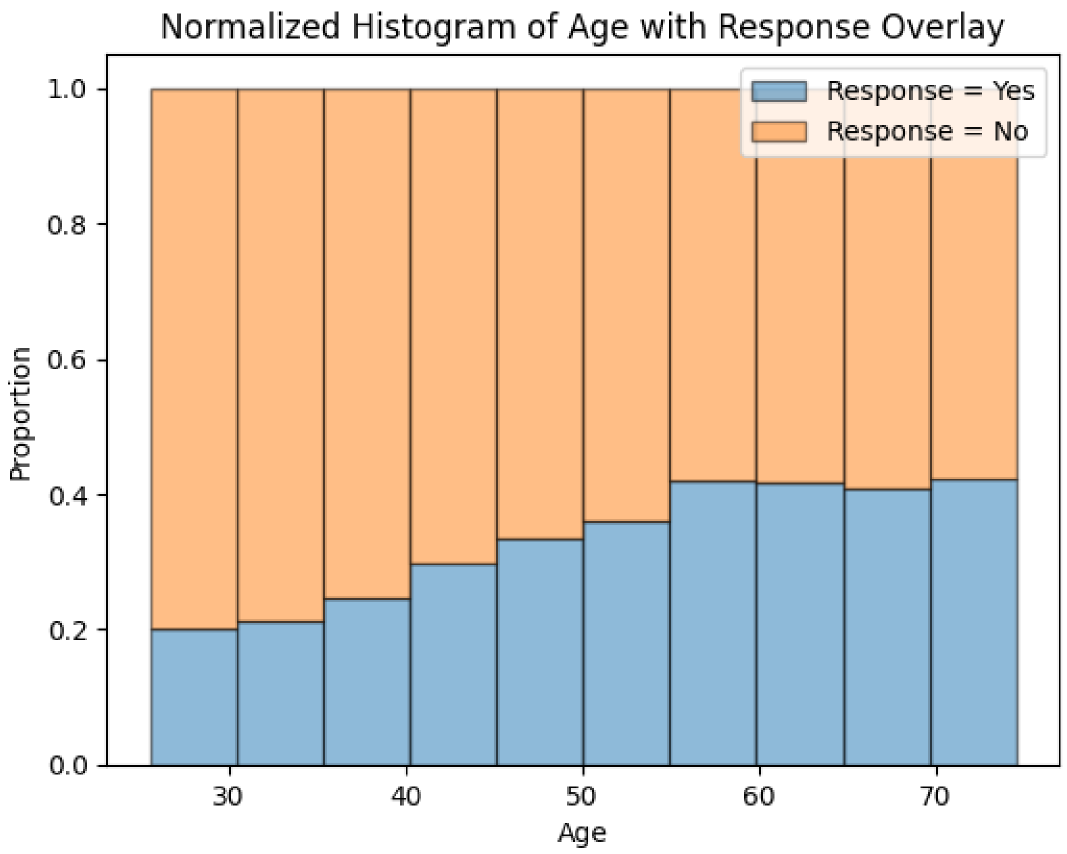


Histogram with Response Overlay for Numerical Data

```
[ ]: #numerical_columns = ['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR',␣
      ↪'Oldpeak']

     heart_age_y = heart[heart.HeartDisease_categorical == "Yes"]['Age']
     heart_age_n = heart[heart.HeartDisease_categorical == "No"]['Age']

     (n, bins, patches) = plt.hist([heart_age_y, heart_age_n], bins=10, stacked=True)

     n_table = np.column_stack((n[0], n[1]))

     n_norm = n_table / n_table.sum(axis=1)[:, None]

     ourbins = np.column_stack((bins[:10], bins[1:11]))

     fig, ax = plt.subplots()

     p1 = plt.bar(x=ourbins[:, 0], height=n_norm[:, 0], width=ourbins[:, 1] -␣
      ↪ourbins[:, 0], alpha=0.5, edgecolor='black', label='Heart Disease: Yes')
```

```
p2 = plt.bar(x=ourbins[:, 0], height=n_norm[:, 1], width=ourbins[:, 1] -␣
 ↪ourbins[:, 0], bottom=n_norm[:, 0], alpha=0.5, edgecolor='black',␣
 ↪label='Heart Disease: No')


plt.legend(['Response = Yes', 'Response = No'])
plt.title('Normalized Histogram of Age with Response Overlay')
plt.xlabel('Age')
plt.ylabel('Proportion')
plt.show()
```

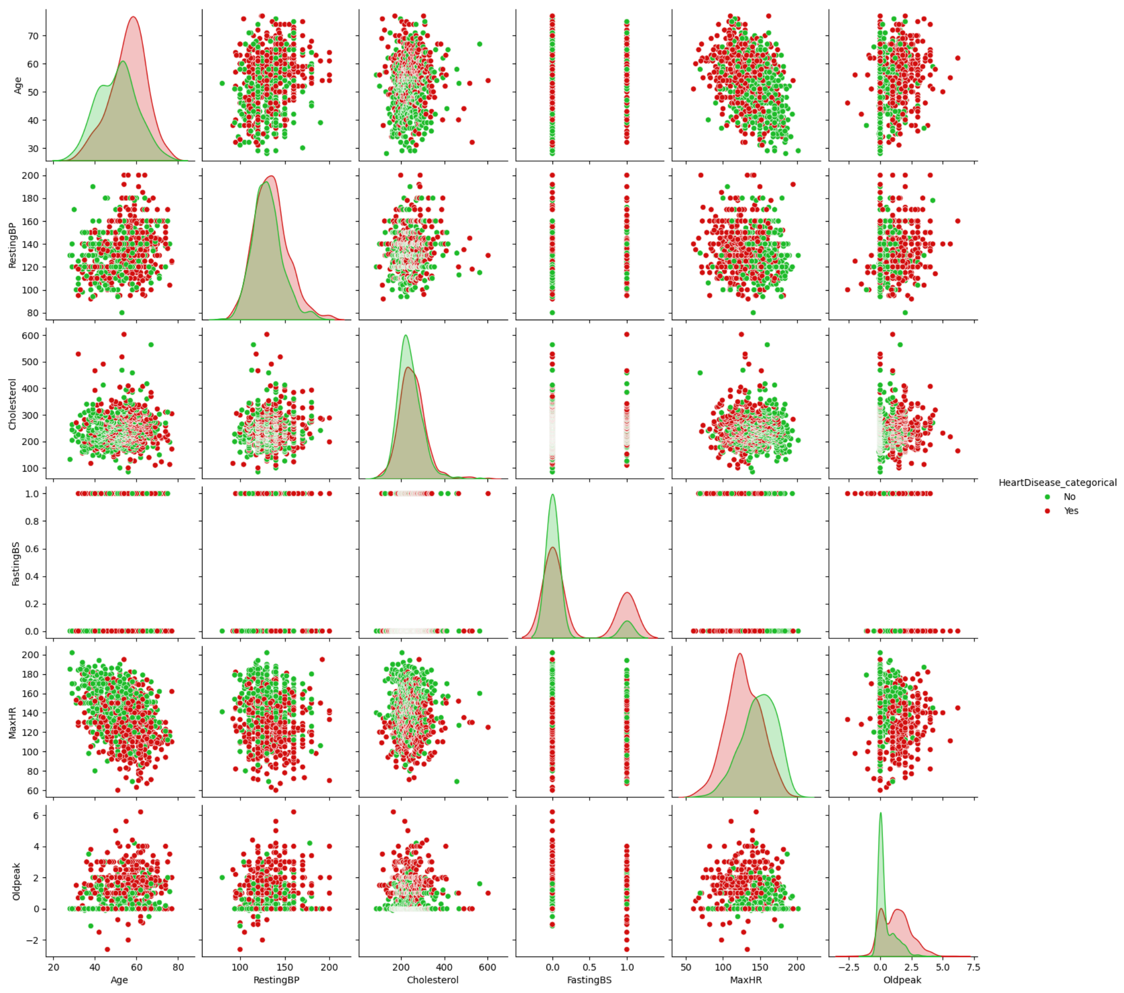




This normalized bar graph shows how the risk of having a heart disease increase with age but at the same time exist a higher number of people without the disease.

```
[ ]: palette = {'No': '#1DBB29', 'Yes': '#D20E0E'}
     sns.pairplot(heart,
                  vars=['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR',␣
       ↪'Oldpeak'],
                  diag_kind="kde",
                  hue='HeartDisease_categorical',
                  palette=palette)
```

```
[ ]: <seaborn.axisgrid.PairGrid at 0x29d2a9b5c70>
```



0.3 Preparing to Model The Data

One Hot Encoding, Data Imputation, and Partitioning the Data

```
[ ]: # Transforming categorical data into numerical data ( One Hot Encoding)
     heart_encoded = pd.get_dummies(heart, columns=['Sex', 'ChestPainType',
                                                            'RestingECG',
                                                           ␣
       ↪'ExerciseAngina', 'ST_Slope',
                                                             ␣
       ↪'HeartDisease_categorical'],
                                              drop_first=True)
     # KNN Imputation
     KNN_imputer = KNNImputer(n_neighbors=5)

     heart_encoded[['Cholesterol']] = KNN_imputer.
       ↪fit_transform(heart_encoded[['Cholesterol']])
```

```
heart_encoded[['RestingBP']] = KNN_imputer.
 ↪fit_transform(heart_encoded[['RestingBP']])

# Partitioning the Data
heart_train, heart_test = train_test_split(heart_encoded, test_size= 0.25,␣
 ↪random_state= 7)

print("Original Data Set:", heart_encoded.shape)
print("Training Data Set:", heart_train.shape, round(heart_train.shape[0] /␣
 ↪heart_encoded.shape[0] * 100, 2), "%")
print("Test Data Set:", heart_test.shape, round(heart_test.shape[0] /␣
 ↪heart_encoded.shape[0] * 100, 2), "%")
```

```
Original Data Set: (918, 18)
Training Data Set: (688, 18) 74.95 %
Test Data Set: (230, 18) 25.05 %
```

Correlations

```
[ ]: heart_encoded.corr()
```

```
[ ]:                                  Age  RestingBP  Cholesterol  FastingBS  \
     Age                         1.000000   0.263081     0.053373   0.198039
     RestingBP                   0.263081   1.000000     0.083076   0.067811
     Cholesterol                 0.053373   0.083076     1.000000   0.043008
     FastingBS                   0.198039   0.067811     0.043008   1.000000
     MaxHR                      -0.382045  -0.109662    -0.017239  -0.131438
     Oldpeak                     0.258612   0.174220     0.053029   0.052698
     HeartDisease                0.282039   0.117938     0.094071   0.267291
     Index                      -0.028882   0.145586     0.672500  -0.198319
     Sex_M                       0.055750   0.009425    -0.101706   0.120076
     ChestPainType_ATA          -0.218165  -0.051367    -0.015288  -0.140514
     ChestPainType_NAP          -0.011335  -0.027483    -0.062229  -0.039249
     ChestPainType_TA            0.032042   0.049463    -0.047322   0.026885
     RestingECG_Normal          -0.230566  -0.113718    -0.042407  -0.093028
     RestingECG_ST               0.136798   0.089145    -0.024530   0.127110
     ExerciseAngina_Y            0.215793   0.153008     0.077549   0.060451
     ST_Slope_Flat               0.185568   0.110111     0.093627   0.107006
     ST_Slope_Up                -0.258067  -0.105926    -0.089995  -0.161730
     HeartDisease_categorical_Yes  0.282039   0.117938     0.094071   0.267291

                                MaxHR    Oldpeak  HeartDisease     Index  \
     Age                        -0.382045   0.258612      0.282039 -0.028882
     RestingBP                  -0.109662   0.174220      0.117938  0.145586
     Cholesterol                -0.017239   0.053029      0.094071  0.672500
     FastingBS                  -0.131438   0.052698      0.267291 -0.198319
     MaxHR                       1.000000  -0.160691     -0.400421  0.175307
     Oldpeak                    -0.160691   1.000000      0.403951  0.072980
```

```
HeartDisease                 -0.400421  0.403951      1.000000 -0.139166
Index                         0.175307  0.072980     -0.139166  1.000000
Sex_M                        -0.189186  0.105734      0.305445 -0.181174
ChestPainType_ATA             0.253735 -0.262124     -0.401924  0.123510
ChestPainType_NAP             0.134580 -0.106212     -0.212964 -0.039486
ChestPainType_TA              0.100025  0.032231     -0.054790  0.005106
RestingECG_Normal             0.023801 -0.116719     -0.091580 -0.065735
RestingECG_ST                -0.157879  0.055958      0.102527 -0.105835
ExerciseAngina_Y             -0.370425  0.408752      0.494282  0.024372
ST_Slope_Flat                -0.342581  0.283295      0.554134 -0.004604
ST_Slope_Up                   0.383397 -0.450577     -0.622164  0.043002
HeartDisease_categorical_Yes -0.400421  0.403951      1.000000 -0.139166

                                 Sex_M  ChestPainType_ATA  ChestPainType_NAP  \
Age                           0.055750          -0.218165          -0.011335
RestingBP                     0.009425          -0.051367          -0.027483
Cholesterol                  -0.101706          -0.015288          -0.062229
FastingBS                     0.120076          -0.140514          -0.039249
MaxHR                        -0.189186           0.253735           0.134580
Oldpeak                       0.105734          -0.262124          -0.106212
HeartDisease                  0.305445          -0.401924          -0.212964
Index                        -0.181174           0.123510          -0.039486
Sex_M                         1.000000          -0.161522          -0.066486
ChestPainType_ATA            -0.161522           1.000000          -0.256767
ChestPainType_NAP            -0.066486          -0.256767           1.000000
ChestPainType_TA             -0.004031          -0.110679          -0.122381
RestingECG_Normal            -0.010634           0.107941           0.005010
RestingECG_ST                 0.063715          -0.046111          -0.042236
ExerciseAngina_Y              0.190664          -0.300365          -0.166030
ST_Slope_Flat                 0.116077          -0.304667          -0.072031
ST_Slope_Up                  -0.150942           0.357588           0.093583
HeartDisease_categorical_Yes  0.305445          -0.401924          -0.212964

                              ChestPainType_TA  RestingECG_Normal  \
Age                                   0.032042          -0.230566
RestingBP                             0.049463          -0.113718
Cholesterol                          -0.047322          -0.042407
FastingBS                             0.026885          -0.093028
MaxHR                                 0.100025           0.023801
Oldpeak                               0.032231          -0.116719
HeartDisease                         -0.054790          -0.091580
Index                                 0.005106          -0.065735
Sex_M                                -0.004031          -0.010634
ChestPainType_ATA                    -0.110679           0.107941
ChestPainType_NAP                    -0.122381           0.005010
ChestPainType_TA                      1.000000          -0.057719
RestingECG_Normal                    -0.057719           1.000000
```

```
RestingECG_ST                      -0.011611          -0.602314
ExerciseAngina_Y                   -0.128105          -0.072924
ST_Slope_Flat                      -0.010486          -0.047172
ST_Slope_Up                         0.002087           0.078563
HeartDisease_categorical_Yes       -0.054790          -0.091580

                              RestingECG_ST  ExerciseAngina_Y  ST_Slope_Flat  \
Age                                0.136798          0.215793       0.185568
RestingBP                          0.089145          0.153008       0.110111
Cholesterol                       -0.024530          0.077549       0.093627
FastingBS                          0.127110          0.060451       0.107006
MaxHR                             -0.157879         -0.370425      -0.342581
Oldpeak                            0.055958          0.408752       0.283295
HeartDisease                       0.102527          0.494282       0.554134
Index                             -0.105835          0.024372      -0.004604
Sex_M                              0.063715          0.190664       0.116077
ChestPainType_ATA                 -0.046111         -0.300365      -0.304667
ChestPainType_NAP                 -0.042236         -0.166030      -0.072031
ChestPainType_TA                  -0.011611         -0.128105      -0.010486
RestingECG_Normal                 -0.602314         -0.072924      -0.047172
RestingECG_ST                      1.000000          0.107036       0.043017
ExerciseAngina_Y                   0.107036          1.000000       0.382237
ST_Slope_Flat                      0.043017          0.382237       1.000000
ST_Slope_Up                       -0.058936         -0.455676      -0.870951
HeartDisease_categorical_Yes       0.102527          0.494282       0.554134

                              ST_Slope_Up  HeartDisease_categorical_Yes
Age                             -0.258067                      0.282039
RestingBP                       -0.105926                      0.117938
Cholesterol                     -0.089995                      0.094071
FastingBS                       -0.161730                      0.267291
MaxHR                            0.383397                     -0.400421
Oldpeak                         -0.450577                      0.403951
HeartDisease                    -0.622164                      1.000000
Index                            0.043002                     -0.139166
Sex_M                           -0.150942                      0.305445
ChestPainType_ATA                0.357588                     -0.401924
ChestPainType_NAP                0.093583                     -0.212964
ChestPainType_TA                 0.002087                     -0.054790
RestingECG_Normal                0.078563                     -0.091580
RestingECG_ST                   -0.058936                      0.102527
ExerciseAngina_Y                -0.455676                      0.494282
ST_Slope_Flat                   -0.870951                      0.554134
ST_Slope_Up                      1.000000                     -0.622164
HeartDisease_categorical_Yes    -0.622164                      1.000000
```

Skewness of The Distribution

```
[ ]: heart_encoded.skew()
```

```
[ ]: Age                            -0.195933
     RestingBP                       0.607525
     Cholesterol                     1.373396
     FastingBS                       1.264484
     MaxHR                          -0.144359
     Oldpeak                         1.022872
     HeartDisease                   -0.215086
     Index                           0.000000
     Sex_M                          -1.424540
     ChestPainType_ATA               1.595899
     ChestPainType_NAP               1.346107
     ChestPainType_TA                4.130983
     RestingECG_Normal              -0.414489
     RestingECG_ST                   1.551033
     ExerciseAngina_Y                0.391329
     ST_Slope_Flat                  -0.004364
     ST_Slope_Up                     0.282079
     HeartDisease_categorical_Yes   -0.215086
     dtype: float64
```

Validating The Data Partition

```
[ ]: from statsmodels.stats.proportion import proportions_ztest

     # Get counts for HeartDisease_categorical in training and test sets
     count_train = np.sum(heart_train['HeartDisease'] == 1)
     count_test = np.sum(heart_test['HeartDisease'] == 1)

     n_train = len(heart_train)
     n_test = len(heart_test)

     # Perform two-sample Z-test for proportion
     count = np.array([count_train, count_test])
     nobs = np.array([n_train, n_test])

     z_stat, p_val = proportions_ztest(count, nobs)

     print(f'Two-sample Z-test for HeartDisease: z-statistic = {z_stat}, p-value =␣
      ↪{p_val}')
```

```
Two-sample Z-test for HeartDisease: z-statistic = 1.268056802569942, p-value =
0.20477766636519124
```

Based on the Z-test result, there is no significant difference between the training and test sets for the HeartDisease_categorical variable. This indicates that the partitioning of the dataset into training and test sets did not introduce a systematic difference in the proportions of the “Yes” and “No” responses, which is a good indication that the partitioning is valid.

Balancing The Training Data Set

```
[ ]: total_train = heart_train['HeartDisease_categorical_Yes'].value_counts()

     Yes_total = total_train.iloc[1] / heart_train.shape[0] * 100
     No_total = total_train.iloc[0] / heart_train.shape[0] * 100

     print("Total number of 'Yes' in the Training:", round(Yes_total, 2), "%")
     print("Total number of 'No' in the Training:", round(No_total, 2), "%")

     total_train
```

```
Total number of 'Yes' in the Training: 43.46 %
Total number of 'No' in the Training: 56.54 %
```

```
[ ]: HeartDisease_categorical_Yes
     True     389
     False    299
     Name: count, dtype: int64
```

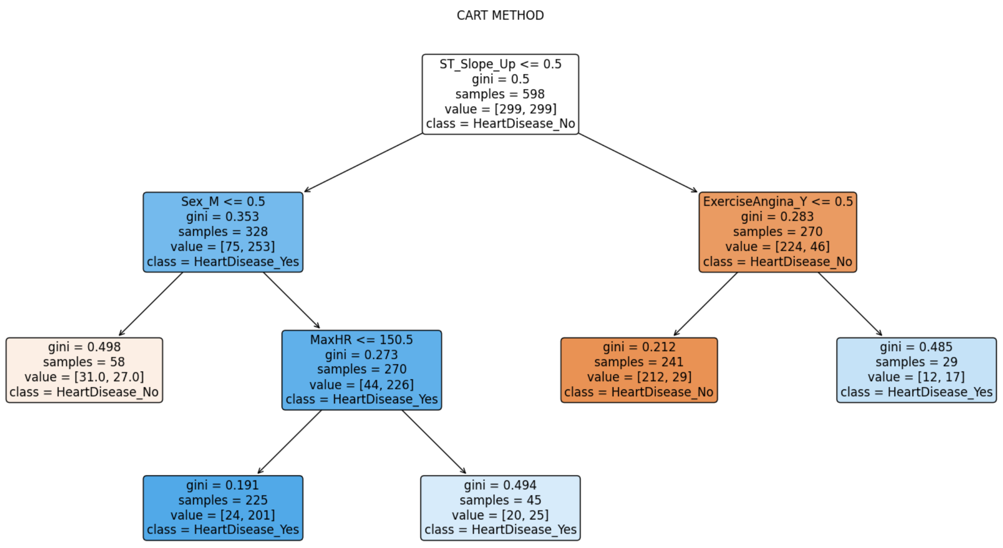
```
[ ]: # Increasing the percentage of "Yes" and "No" responses to 50%
     total_train = heart_train['HeartDisease_categorical_Yes'].value_counts()
     current_yes_count = total_train[True]
     current_no_count = total_train[False]

     # Determine the target count for each class to achieve 50-50 balance
     target_count = min(current_yes_count, current_no_count)

     # Resample to achieve balance
     if current_yes_count > target_count:
         # Downsample Yes
         to_downsample_yes = heart_train[heart_train['HeartDisease_categorical_Yes']␣
      ↪== True]
         downsampled_yes = to_downsample_yes.sample(n=target_count, random_state=7)
         balanced_train = pd.concat([downsampled_yes,␣
      ↪heart_train[heart_train['HeartDisease_categorical_Yes'] == False]])
     elif current_no_count > target_count:
         # Downsample No
         to_downsample_no = heart_train[heart_train['HeartDisease_categorical_Yes']␣
      ↪== False]
         downsampled_no = to_downsample_no.sample(n=target_count, random_state=7)
         balanced_train = pd.concat([downsampled_no,␣
      ↪heart_train[heart_train['HeartDisease_categorical_Yes'] == True]])

     # Shuffle the balanced dataset
     balanced_train = balanced_train.sample(frac=1, random_state=7).
      ↪reset_index(drop=True)
```

```
# Verify the balance
balanced_counts = balanced_train['HeartDisease_categorical_Yes'].value_counts()
print("Balanced distribution in the training dataset:")
print(balanced_counts)
```

```
Balanced distribution in the training dataset:
HeartDisease_categorical_Yes
True     299
False    299
Name: count, dtype: int64
```

Modeling Phase, Decision Trees CART Method

```
[ ]: # Separate data into features and target variable
X = balanced_train.drop(['HeartDisease', 'HeartDisease_categorical_Yes',␣
↪'Index'], axis= 1)
y = balanced_train[['HeartDisease_categorical_Yes']]

# Defining feature names
X_names = list(balanced_train.columns)
X_names.remove('Index')
X_names.remove('HeartDisease')
X_names.remove('HeartDisease_categorical_Yes')

y_names = ["HeartDisease_No", 'HeartDisease_Yes']

# Running the CART algorithm / training
cart01 = DecisionTreeClassifier(criterion= 'gini', max_leaf_nodes= 5).fit(X, y)

# Visualize the decision tree
plt.figure(figsize=(20,10))
plot_tree(cart01, feature_names=X_names, class_names=y_names, filled=True,␣
↪rounded=True, fontsize=12)
plt.title("CART METHOD")
plt.show()
```



Making predictions

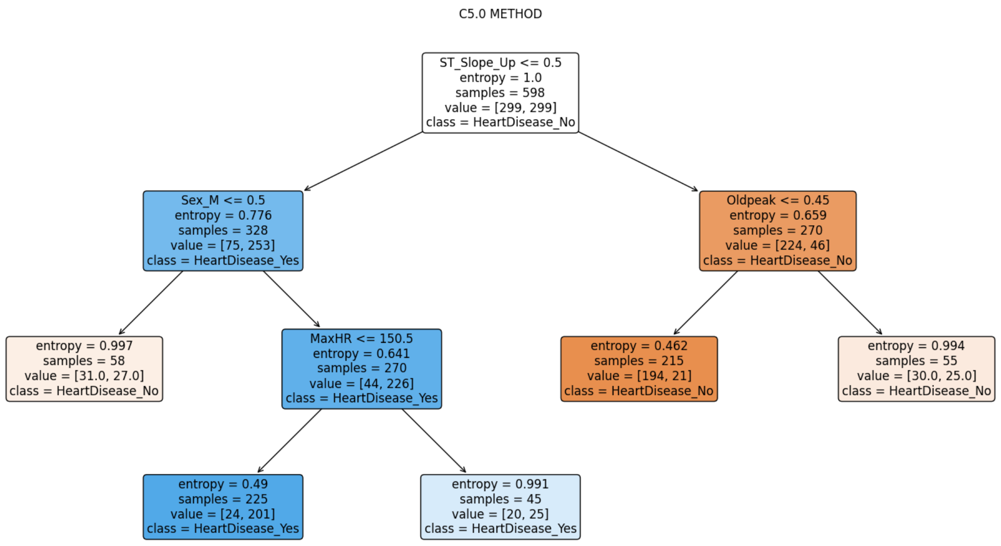
```
[ ]: feature_columns = [
         'Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak',
         'Sex_M', 'ChestPainType_ATA', 'ChestPainType_NAP', 'ChestPainType_TA',
         'RestingECG_Normal', 'RestingECG_ST', 'ExerciseAngina_Y', 'ST_Slope_Flat',␣
       ↪'ST_Slope_Up'
     ]

     # Selecting feature columns
     X_test = heart_test[feature_columns]

     # Making predictions
     predHeartDiseaseCART = cart01.predict(X_test)

     # Selecting response variable
     y_test = heart_test['HeartDisease']

     # Results
     print("Confusion Matrix:")
     print(confusion_matrix(y_test, predHeartDiseaseCART))
     print("\nClassification Report:")
     print(classification_report(y_test, predHeartDiseaseCART, target_names=["0",␣
       ↪"1"]))
     print("\nAccuracy Score:")
     print(accuracy_score(y_test, predHeartDiseaseCART))
```

```
Confusion Matrix:
[[86 25]
```

```
 [21 98]]

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.77      0.79       111
           1       0.80      0.82      0.81       119

    accuracy                           0.80       230
   macro avg       0.80      0.80      0.80       230
weighted avg       0.80      0.80      0.80       230


Accuracy Score:
0.8
```

C5.0 Method

```
[ ]: # Training algorithm
     c50_01 = DecisionTreeClassifier(criterion= 'entropy', max_leaf_nodes= 5).
      ↪fit(X,y)

     # Visualizing c5.0
     plt.figure(figsize=(20,10))
     plot_tree(c50_01, feature_names=X_names, class_names=y_names, filled= True,␣
      ↪rounded= True, fontsize= 12)
     plt.title("C5.0 METHOD")
     plt.show()
```



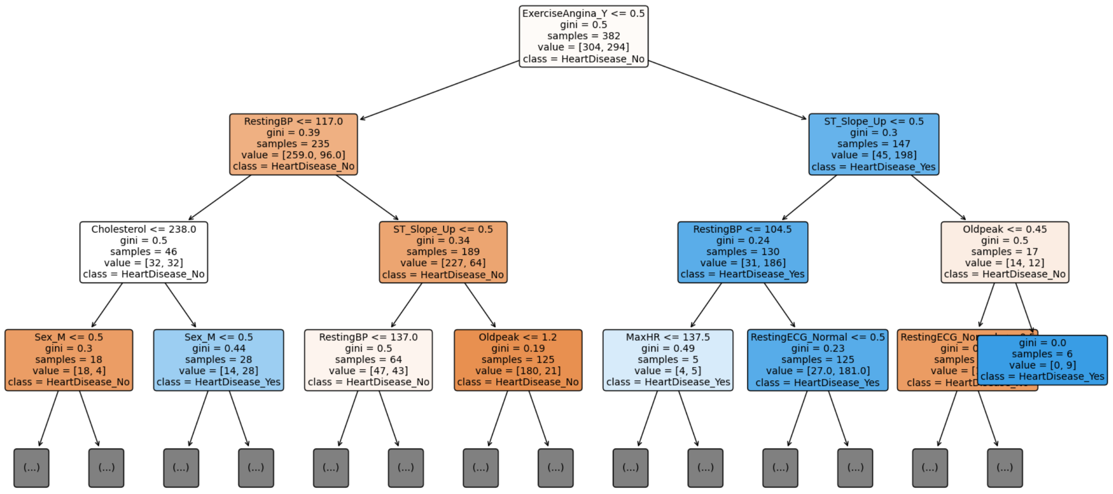
Predictions using c5.0

```
[ ]: predHeartDiseasec50 = c50_01.predict(X_test)

     # Results
     print("Confusion Matrix:")
     print(confusion_matrix(y_test, predHeartDiseasec50))
     print("\nClassification Report:")
     print(classification_report(y_test, predHeartDiseasec50, target_names=["0",␣
      ↪"1"]))
     print("\nAccuracy Score:")
     print(accuracy_score(y_test, predHeartDiseasec50))
```

```
Confusion Matrix:
[[100  11]
 [ 29  90]]

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.90      0.83       111
           1       0.89      0.76      0.82       119

    accuracy                           0.83       230
   macro avg       0.83      0.83      0.83       230
weighted avg       0.84      0.83      0.83       230


Accuracy Score:
0.8260869565217391
```

Using Random Forest Method

```
[ ]: # Requires a response variable formatted as a one-dimensional array
     rfy = np.ravel(y)

     # Training RandomForest
     rf01 = RandomForestClassifier(n_estimators= 100, random_state=42, criterion=␣
      ↪'gini').fit(X,rfy)

     # Visualizing Random Forests
     estimator = rf01.estimators_[0]

     plt.figure(figsize=(20, 10))
     tree.plot_tree(estimator,
                    feature_names=X.columns,
                    class_names=["HeartDisease_No", 'HeartDisease_Yes'],
                    filled=True,
                    rounded=True,
                    proportion=False,
                    precision=2,
                    fontsize=10,
                    max_depth=3)
     plt.show()
```



Predicting using RandomForest method

```
[ ]: predHeartDiseaseRandomForest = rf01.predict(X_test)

     # Results
     print("Confusion Matrix:")
     print(confusion_matrix(y_test, predHeartDiseaseRandomForest))
     print("\nClassification Report:")
     print(classification_report(y_test, predHeartDiseaseRandomForest,␣
      ↪target_names=["0", "1"]))
     print("\nAccuracy Score:")
     print(accuracy_score(y_test, predHeartDiseaseRandomForest))
```

```
Confusion Matrix:
[[ 91  20]
 [  9 110]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.82      0.86       111
           1       0.85      0.92      0.88       119
```

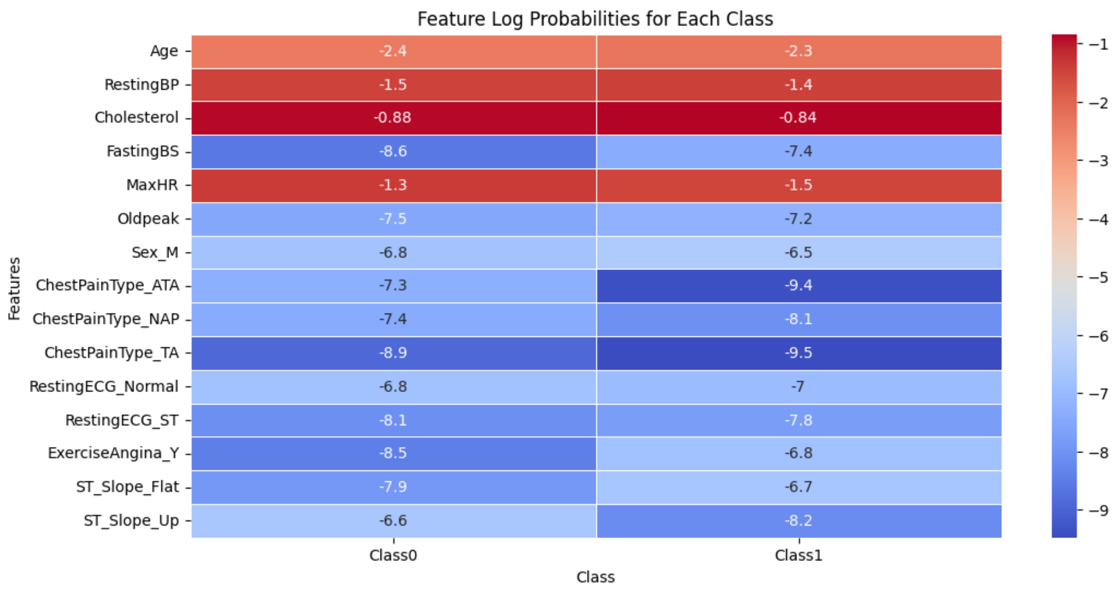
```
    accuracy                           0.87       230
   macro avg       0.88      0.87      0.87       230
weighted avg       0.88      0.87      0.87       230


Accuracy Score:
0.8739130434782608
```

Naïve Bayes Method First, handle negative numbers in column "Oldpeak"

```
[ ]: # Initializing the MinMaxScaler function
     scaler = MinMaxScaler()

     X['Oldpeak'] = scaler.fit_transform(X[['Oldpeak']])
```

Training Naïve Bayes

```
[ ]: # Training Naive Bayes model
     nb_01 = MultinomialNB().fit(X,rfy)

     # Get feature log probabilities
     feature_log_probs = nb_01.feature_log_prob_

     # Convert to DataFrame for easier plotting
     feature_log_probs_df = pd.DataFrame(feature_log_probs, columns=X.columns,␣
      ↪index=['Class0', 'Class1'])

     # Plot the feature log probabilities as a heatmap
     plt.figure(figsize=(12, 6))
     sns.heatmap(feature_log_probs_df.T, annot=True, cmap='coolwarm', cbar=True,␣
      ↪linewidths=0.5)
     plt.title('Feature Log Probabilities for Each Class')
     plt.xlabel('Class')
     plt.ylabel('Features')
     plt.show()
```



Predicting using Naïve Bayes method

```
[ ]: predHeartDiseasecNB = nb_01.predict(X_test)

     # Results
     print("Confusion Matrix:")
     print(confusion_matrix(y_test, predHeartDiseasecNB))
     print("\nClassification Report:")
     print(classification_report(y_test, predHeartDiseasecNB, target_names=["0",␣
      ↪"1"]))
     print("\nAccuracy Score:")
     print(accuracy_score(y_test, predHeartDiseasecNB))
```

```
Confusion Matrix:
[[88 23]
 [20 99]]

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.79      0.80       111
           1       0.81      0.83      0.82       119

    accuracy                           0.81       230
   macro avg       0.81      0.81      0.81       230
weighted avg       0.81      0.81      0.81       230
```

```
Accuracy Score:
0.8130434782608695
```

Showing the contingency table of actual versus predicted outcomes

```
[ ]: ypred = pd.crosstab(y_test, predHeartDiseasecNB, rownames= ['Actual'],␣
      ↪colnames= ['Predicted'])

     ypred['Total'] = ypred.sum(axis=1)
     ypred.loc['Total'] = ypred.sum()
     ypred
```

```
[ ]: Predicted  False  True  Total
     Actual
     0             88    23    111
     1             20    99    119
     Total        108   122    230
```

88 cases: The actual value is 0, and the model correctly predicted 0 (True Negative).

23 cases: The actual value is 0, but the model incorrectly predicted 1 (False Positive).

20 cases: The actual value is 1, but the model incorrectly predicted 0 (False Negative).

99 cases: The actual value is 1, and the model correctly predicted 1 (True Positive).

Neural Networks

```
[ ]: X_train = X

     # Performing min-max standarization on all numerical variables
     X_train.loc[:, numerical_columns] = scaler.
      ↪fit_transform(X_train[numerical_columns])

     # Building keras model
     model = Sequential()
     model.add(Input(shape=(15,)))
     model.add(Dense(10, activation='relu'))
     model.add(Dense(1, activation='sigmoid'))


     # Compile keras model
     model.compile(optimizer='adam', loss='binary_crossentropy',␣
      ↪metrics=['accuracy'])

     # Train the model
     model.fit(X_train, y, epochs=150, verbose=0)

     # evaluate the keras model
     _, accuracy = model.evaluate(X_train, y, verbose=0)
     print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 85.79
```

Making predictions

```
[ ]: # Performing min-max standarization on all numerical variables
     X_test.loc[:, numerical_columns] = scaler.transform(X_test[numerical_columns])

     # Predictions
     prednnet01 = model.predict(X_test)
```

```
8/8              0s 857us/step
8/8              0s 857us/step
```

```
[ ]: # Convert probabilities to binary predictions (0 or 1)
     threshold = 0.5
     prednnet01_binary = (prednnet01 > threshold).astype(int)

     # Reshape to match y_test if necessary
     prednnet01_binary = prednnet01_binary.reshape(-1)

     # Print confusion matrix
     print("Confusion Matrix:")
     print(confusion_matrix(y_test, prednnet01_binary))

     # Print classification report
     print("\nClassification Report:")
     print(classification_report(y_test, prednnet01_binary, target_names=['No Heart␣
      ↪Disease', 'Heart Disease']))

     # Print accuracy score
     print("\nAccuracy Score:")
     print(accuracy_score(y_test, prednnet01_binary))
```

```
Confusion Matrix:
[[ 83  28]
 [ 10 109]]

Classification Report:
                  precision    recall  f1-score   support

No Heart Disease       0.89      0.75      0.81       111
   Heart Disease       0.80      0.92      0.85       119

        accuracy                           0.83       230
       macro avg       0.84      0.83      0.83       230
    weighted avg       0.84      0.83      0.83       230


Accuracy Score:
```

```
0.8347826086956521
```

Conclusion CART model gives the accuracy of : 80%

C5.0 model gives the accuracy of : 83%

Random forest gives the accuracy of : 86%

Naïve Bayes gives the accuracy of : 81%

Neural Network gives the accuracy of : 87%