Heart Disease Prediction App using Machine Learning Models

Kanishka Arora

7th June, 2024

MSc in Astronomy and Astrophysics
Indian Institute of Space Science and Technology

Abstract

Heart disease, a leading cause of mortality worldwide, encompasses a range of conditions affecting the heart and blood vessels. It poses a significant public health challenge, with risk factors including lifestyle choices, genetic predisposition, and underlying medical conditions. Early detection and effective management are crucial in mitigating its impact.

This report talks about the idea of a **Heart Disease Prediction App** and how it can be useful to common people. We discuss the core idea, the revenue generation, the design of our product, and even the technical part involving the machine learning models and the background framework on which the app predicts whether a patient has a risk of heart disease or not. It also examines various aspects of heart disease, including risk factors, diagnostic methods, treatment options, and preventive strategies, aiming to contribute to the understanding and management of this prevalent health issue. We particularly deal with CAD in this report, which is Coronary Artery Disease, which will be discussed in detail.

Contents

List of Tables

List of Figures

float

Chapter 1

Introduction

Cardiovascular disease (CVD) is a leading cause of death worldwide, responsible for over 30% of all deaths. If current trends continue, the number of deaths could rise to 22 million by 2030. CVD includes conditions like heart attacks and strokes, often caused by plaques blocking blood flow in arteries. Risk factors such as lack of physical activity, poor diet, and the use of alcohol and tobacco increase the likelihood of heart disease. Adopting healthier habits, like eating less salt, consuming more fruits and vegetables, exercising regularly, and avoiding alcohol and tobacco, can help reduce this risk.

We have seen the application of Machine learning in various fields, and it's also beneficial in the medical field. It can help identify the type of tumors (whether the tumor is benign or malignant), Diabetes Prediction, or, in this case, the prediction of whether a patient has heart disease. Several types of Heart Diseases can be identified using machine learning models such as **Coronary Heart Disease**, **Heart failure**, **Arrhythmias**, **Hypertrophic Cardiomyopathy**, **Peripheral Artery Disease**, etc.

In recent years, it has become a powerful tool in the fight against heart disease, enabling early detection, accurate diagnosis, and personalized treatment. It can analyze large amounts of patient data, such as medical records, imaging tests, and genetic information, to identify patterns and predict the risk of heart disease. For example, machine learning algorithms can detect conditions like **arrhythmias**(as mentioned above) by examining ECG data. Additionally, these algorithms aid in creating personalized treatment plans tailored to individual patient characteristics and responses to therapies. By utilizing machine learning, healthcare professionals can enhance patient outcomes, optimize resource allocation, and improve overall cardiac care.

In this report, I will talk about Heart Disease Prediction, particularly **Coronary Heart Disease(CAD)**, which is characterized by the narrowing or blockage of the coronary arteries, leading to a reduced blood flow to the heart muscle. The data set I have used is the **UCI Heart disease dataset**.

Let us talk about Coronary Artery Disease which is what the model focuses on.

Coronary Heart Disease occurs when your coronary arteries get narrowed with time or get blocked, which is harmful as these are the arteries that supply oxygen-rich blood to your heart. This usually happens because plaque builds up in these arteries with time, which limits the amount of blood that can reach your heart muscles.

The plaque is mainly cholesterol which is a waxy, fat-like substance that the body needs in the right amounts for good health. This builds up inside the inner lining of the coronary arteries which can partially or fully block the flow of blood. This deposition also narrows the arteries as mentioned previously and this process is termed **atherosclerosis** and these fatty deposits are collectively called **antheroma**.

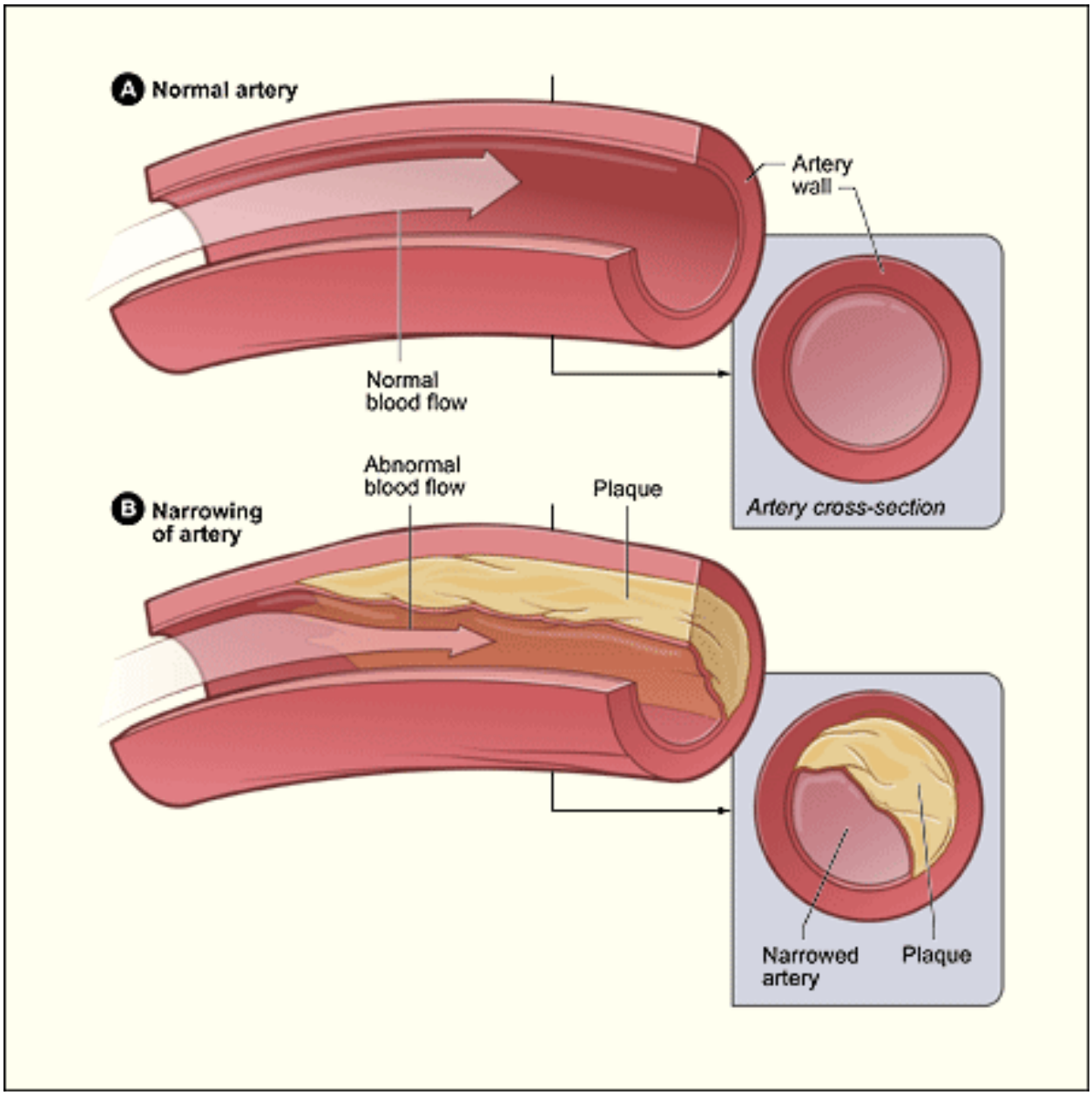


Fig. 1.1: Blockage of the Coronal Artery

The above diagram would give a better idea of what atherosclerosis is. It can be caused by lifestyle factors, such as smoking and regularly drinking excessive amounts of alcohol. You're also more at risk of getting atherosclerosis if you have conditions like **high cholesterol**, **high blood pressure (hypertension)**, or **diabetes**.

There are broadly two types of Coronary Heart Disease.

- **Stable ischemic heart disease:** Stable ischemic heart disease (SIHD), also known as chronic coronary artery disease, occurs when the coronary arteries gradually narrow over many years due to the buildup of plaque. This plaque consists of fatty deposits, cholesterol, and other substances. The progressive narrowing restricts blood flow to the heart muscle, leading to a reduction in oxygen-rich blood supply.
- **Acute Coronary Syndrome:**Acute coronary syndrome (ACS) is a sudden, severe condition that requires immediate medical attention. It occurs when an atherosclerotic plaque in a coronary artery ruptures, leading to the formation of a blood clot that significantly obstructs or completely blocks blood flow to part of the heart muscle. This sudden blockage can result in a heart attack (myocardial infarction).

The build up of plaque happens over many years and decades in fact, and you might notice mild symptoms over time. These mild symptoms signify that you heart is working harder to pump oxygen-rich blood into your heart. Below are the main symptoms of CAD.

- **Angina:** When your coronal arteries become partially blocked, you experience a chest pain which is known an **Angina**. This can be a mild, uncomfortable feeling similar to indigestion. However, a severe angina attack can cause a painful feeling of heaviness or tightness, usually in the centre of the chest, which may spread to the arms, neck, jaw, back or stomach. Angina is often triggered by physical activity or stressful situations. Symptoms usually pass in less than 10 minutes, and can be relieved by resting or using a nitrate tablet or spray.

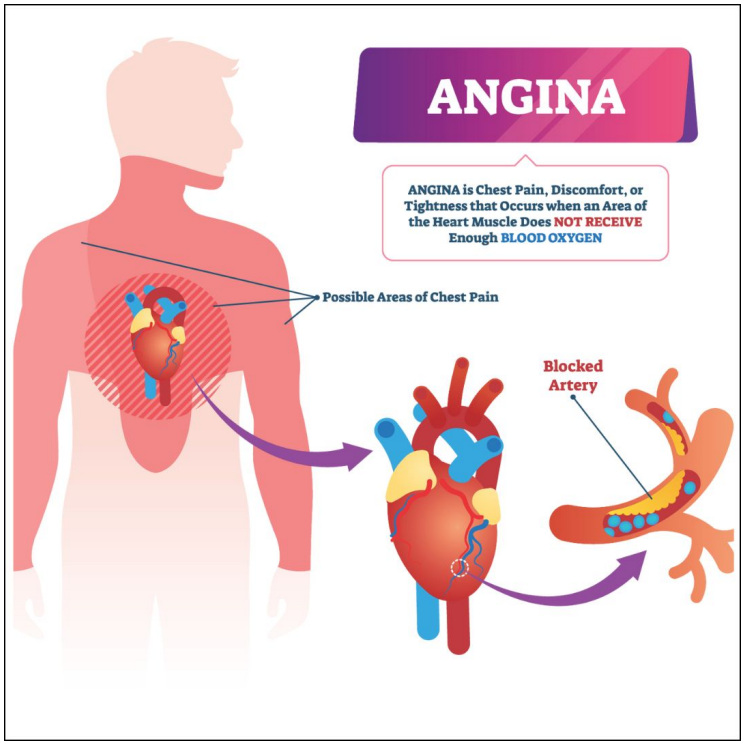


Fig. 1.2: Angina

- **Heart Attack:** If your arteries become partly or completely blocked, it can lead to a **Heart Attack (myocardial infarction)**, a condition that can cause permanent damage to the heart muscle and be fatal if not treated immediately. During a heart attack, you might experience pain that radiates from your chest to your arms, jaw, neck, back, or stomach, along with symptoms like **lightheadedness**, **sweating**, **nausea**, and **breathlessness**. These symptoms can sometimes resemble indigestion, presenting as heaviness in the chest, a stomach ache, or heartburn. Unlike angina, the discomfort from a heart attack typically isn't relieved by a nitrate tablet or spray.

  Heart attacks can occur at any time, even while resting, and if chest pain persists for more than 15 minutes, it may indicate a heart attack. Additionally, some heart attacks, known as **silent myocardial infarctions**, can occur without any noticeable symptoms, particularly in older adults and people with diabetes.

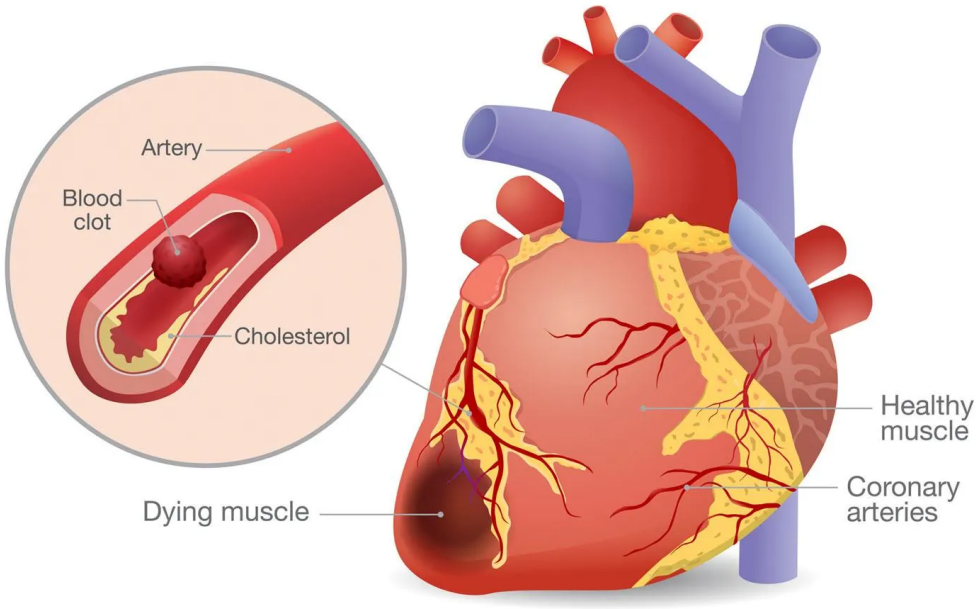


Fig. 1.3: What a Heart attack looks from inside

Blood clots, also known as **thrombi**, can form in arteries when plaque ruptures within narrowed arterial passages. This occurs due to **arteriosclerosis**(as mentioned before), where fatty deposits cause the artery walls to thicken. As the artery narrows, the blood flow is forced through the constricted opening, which can cause the plaque to rupture. This rupture releases molecules that trigger the body's response, leading to an unnecessary clot in the artery.

If this clot blocks blood flow to the brain or heart, it can result in a stroke or **Heart Attack**.

Our app will tell the patient if he has a chance of having CAD, with the help of data sets from reliable sources.

Chapter 2

Product Idea

As discussed, the implementation of ML can be very helpful in identifying medical conditions and in this case, Heart Disease.

The basic idea is the implementation of an app that can help in the prediction of whether a patient has heart disease or not. The patient will have to create an account and input his data which will be helpful for the app as it would make sense of the data with the Machine Learning Framework in the background. This data would be fed as input in the Machine Learning Model which would then be used to determine if the patient has any chance of a heart disease or not.

If the patient does have a Heart Disease, he would be immediately recommended to go to a hospital for a doctor consultation. The app would recommend nearby doctors and hospitals for the same depending on the location of the patient and the availability of the doctors at that particular time. The basic idea of the app would be to guide the patient through the proper channels if he is at risk of heart disease.

Other than that, if the patient has any queries, there will be a chat-based system available for the user, where chat would be available 24/7. If the patient also has some medicines that he has to purchase prescribed for his heart condition, it can be done by uploading the prescription and purchasing the required drugs which can be home delivered. Of course, this is secondary to the Heart Disease Detection system.

Below I will be showing a basic diagram that neatly outlines the idea.

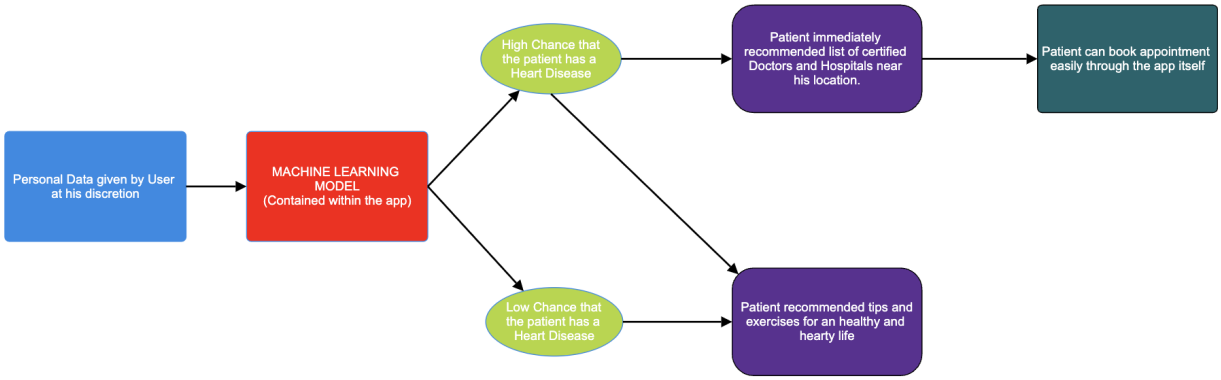


Fig. 2.1: Flowchart depicting basic Idea of App

The app will work on an easy-to-use interface. It will be equipped with a User-Friendly Interface and simplistic in its format.

This is not only what the app has to offer though. Other than this the app can monitor your other vital functions through wearable(smartwatches) as well, and it can detect some unusual activity in periodic trends if any are present. Even after consultation with your doctor, you can put in tasks in the app for intake of your medicine or any prescribed drug, or any physical activity that needs to be done etc. All this data will be recorded and stored for the patient.

The next time the patient consults their doctor, they can simply check whether the patient has been up to date with their medications or not, or perhaps any other task they had prescribed them to do.

Below I will summarize and highlight the key features of the app.

- **User Registration and Profile Creation:** Users can create accounts and provide personal information such as age, gender, medical history, lifestyle habits, and family history of heart disease. The app securely stores user data and ensures compliance with privacy regulations.
- **Data Input and Integration:** Users can input their health metrics manually or integrate with wearable devices, fitness trackers, or electronic health records (EHR) systems to automatically sync data. Supported metrics include age, gender, blood pressure, cholesterol levels, blood sugar levels, BMI(Body Mass Index), exercise habits, smoking status, and more.
- **Risk Assessment and Recommendations:** Users receive personalized risk assessments for heart disease, along with actionable recommendations to mitigate risk factors and improve cardiovascular health. Recommendations may include lifestyle modifications (e.g., diet, exercise), medication adherence, regular check-ups, and consultations with healthcare providers.
- **Progress Tracking and Alerts:** The app allows users to track their health metrics over time and visualize trends using interactive charts and graphs. Users receive alerts and notifications for important milestones, reminders for medication intake, upcoming appointments, or recommended screenings.
- **Educational Resources and Content:** The app provides educational resources, articles, and tips on heart health, prevention strategies, treatment options, and lifestyle changes. Users can access curated content from reputable sources, healthcare professionals, and organizations specializing in cardiovascular health.
- **Community Engagement and Support:** The app fosters a supportive community where users can share experiences, ask questions, and provide encouragement to others on their health journeys. Features include forums, discussion boards, live chat support, and peer-to-peer networking.
- **Security and Privacy:** The app prioritizes security measures to protect user data, including encryption, secure authentication, and regular security audits. User consent and data transparency are emphasized, with clear policies on data usage, sharing, and opt-in/opt-out preferences.

Rather than going to the hospital every time, the patient has a convenient method by using the app. The app provides continuous access to heart health monitoring and risk assessment tools, allowing users to track their health metrics and receive recommendations from the comfort of their homes or on the go. It is a **Cost Effective Method** as there will be less frequent hospital visits that will save time and money for the patient.

Doctors, Clinics, Hospitals, and Pharmacies can set up their portfolios for patients. The patient can view their profiles, their medical experience, qualifications, pricing, etc, and can book appointments for consultations with their desired choice. Patients can give their reviews on the Doctors and Hospitals in the app itself for other patients to view and make a better judgement for whom they want to choose.

So as we can see other than defining the Machine Learning Framework to detect whether a patient has Heart Disease or not(which is the central task of the app) the app also provides other useful features that make it easier and straightforward for the patient to do what's necessary.

This is the outline of my Product Idea.

Chapter 3

Business Model

Let us now discuss the business model for the app.

3.1 Target Audience

The target audience involves the following:

- **Primary Users:**
  People who are concerned about their heart health, particularly those with risk factors such as age, family history of heart disease, high blood pressure, high cholesterol levels, obesity, diabetes, or a sedentary lifestyle. These users seek preventive measures and early detection to manage their health better.

- **Healthcare Providers:**
  This involves clinics, hospitals, and medical facilities looking for tools to assist in patient risk assessment and enhance their preventive care strategies. By integrating the app into their systems, they can provide more comprehensive care and monitor patient health more effectively. They can even set up their profile in the app for patients to view so that they have a better idea of which hospital, medical care facility, or doctor to go to.

- **Insurance Companies:**
  Companies are interested in assessing client health risks more accurately. Using the app, they can offer tailored insurance plans based on individual risk profiles, promote preventive care, and potentially reduce claims by encouraging healthier lifestyles among policyholders. Thus they will directly be linked to patients through the app.

3.2 Revenue Streams

3.2.1 Subscription Models

- **Individual Plans**
  I have come up with the following subscription models.

  1. **Monthly Subscriptions:** Users can subscribe to monthly plans that provide access to premium features such as detailed health reports, continuous health

monitoring, personalized coaching, and advanced analytics. This allows users to benefit from the app's full potential without a long-term commitment.

2. **Yearly Subscriptions:** Offer discounted rates for yearly subscriptions, encouraging users to commit long-term. This provides more stable revenue and better user retention.

3. **Tiered Subscriptions:** Create different subscription tiers (e.g., Basic, Standard, Premium) to cater to varying user needs and budgets. Each tier can offer progressively more features, such as more frequent health check-ins, in-depth analytics, personalized diet and exercise plans, and direct access to health professionals for consultations.

- **Family Plans:**
  Offer discounted rates for family plans where multiple family members can subscribe under a single account. This is particularly useful for households where multiple members are concerned about heart health or likely have a genetic problem with heart health.

- **Corporate Plans:**
  Partner with companies to offer subscription plans to their employees as part of their corporate wellness programs. This can include bulk discounts and tailored features for employee health monitoring.

3.2.2 Business-2-Business Licensing

– **Healthcare Providers**

1. **Clinics and Hospitals:** License the technology to medical facilities for integration into their patient management systems. This can involve a one-time setup fee and ongoing maintenance or usage fees, allowing them to use the app for patient assessments and monitoring.

2. **Telehealth Services:** Integrate the app with telehealth providers, offering remote patient monitoring and risk assessment tools to enhance their service offerings.

– **Insurance Companies**

1. **Risk Assessment Tools:** Provide customized solutions for insurance firms to integrate the app's risk assessment tools into their underwriting processes. This can help in accurately assessing client health risks and offering personalized insurance plans.

2. **Preventive Care Programs:** Collaborate with insurance companies to promote preventive care programs using the app. This can help in reducing claims and long-term healthcare costs for insurers.

– **Freemium Model:**

1. **Basic Features for Free**: Offer essential heart disease risk assessment tools at no cost. This can include basic risk scoring and general health

tips. The goal is to attract a large user base and encourage them to upgrade to premium features.

2. **Premium Services:** Unlock advanced features such as comprehensive health analytics, personalized recommendations, regular health monitoring, and detailed reports through paid subscriptions. This model allows users to see the value of the app before committing to a paid plan.

- **Affiliate Marketing:**
  1. **Health and Wellness Products:** Form partnerships with companies that sell health-related products such as fitness trackers, dietary supplements, healthy food products, and exercise equipment. Recommend these products through the app and earn a commission on sales generated through these recommendations.

  2. **Healthcare Services:** Collaborate with health service providers like gyms, nutritionists, and health coaches. Promote their services through the app and earn referral fees for each user who signs up.

  3. **Health Insurance Plans:** Partner with health insurance companies to offer users tailored insurance plans based on their health data. Earn commissions or referral fees for each user who subscribes to an insurance plan through the app.
- **Data Insights and Analytics:**
  1. **Research and Development:** Anonymized data collected through the app can be valuable for medical research and pharmaceutical companies. Offer insights and analytics services to these organizations for a fee. Ensure compliance with data privacy regulations when sharing data.

  2. **Healthcare Providers:** Provide detailed analytics and insights to healthcare providers to help them better understand patient populations and improve care strategies. This can be offered as a subscription-based service or a one-time fee for detailed reports.
- **Sponsored Content:**
  1. **Educational Content:** Partner with healthcare brands to create and distribute sponsored educational content within the app. This can include articles, videos, and webinars on heart health, diet, and exercise, with sponsorship fees paid by the brands.

  2. **Product Reviews and Recommendations:**Feature sponsored reviews and recommendations for health-related products. This can generate additional revenue while providing users with useful information.

Hence, the above sources are methods of collection, the below flowchart will make it easier to grasp.

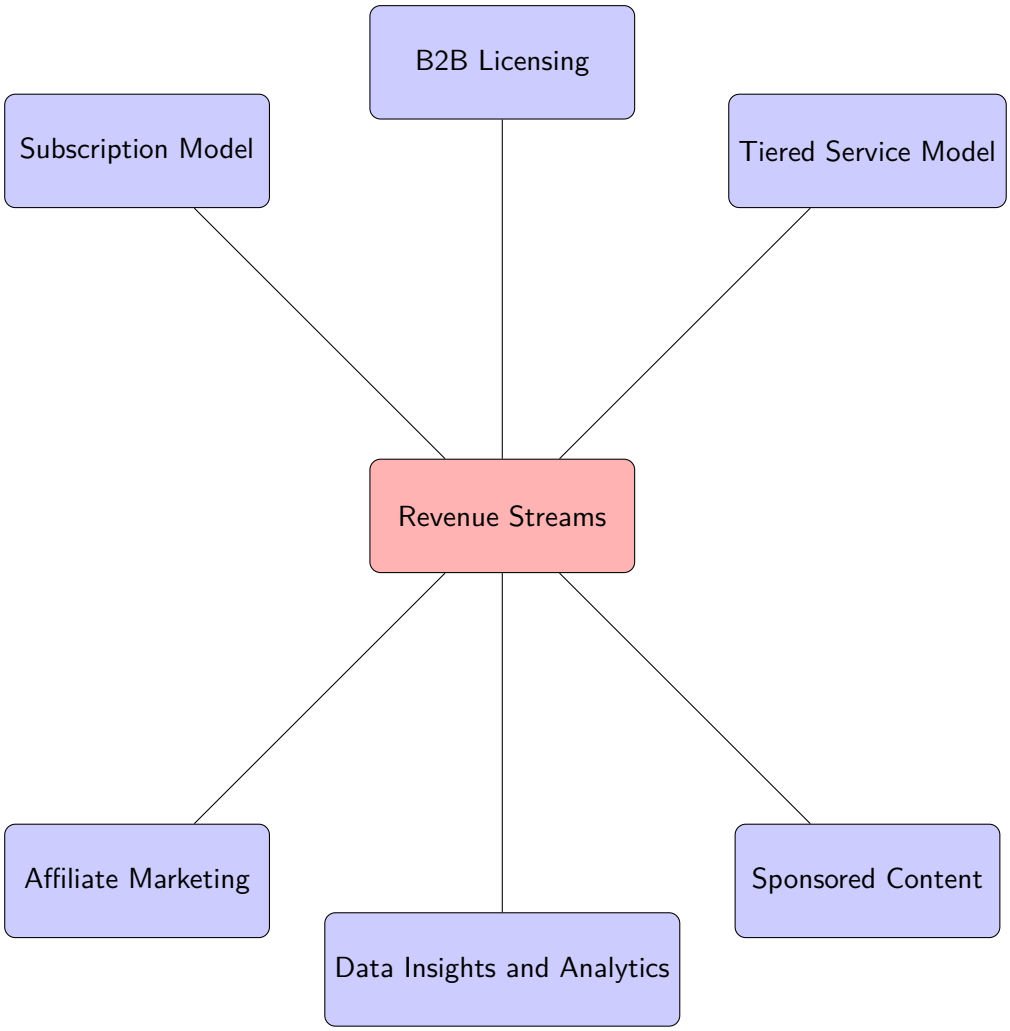


Fig. 3.1: Flowchart representing Revenue Streams

Chapter 4

Product Design

Let us discuss how the product design of the idea will be.

4.1 Data Collection and Preprocessing

4.1.1 Dataset

I'll be using the **UCI Cleveland Heart Disease Dataset**, which is a multivariate dataset containing various medical attributes of patients. This dataset comprises 14 key attributes, including **age**, **sex**, **chest pain type**, **resting blood pressure**,**serum cholesterol**, **fasting blood sugar**, **resting electrocardiographic results**, **maximum heart rate achieved**, **exercise-induced angina**, **oldpeak (ST depression induced by exercise relative to rest)**, **the slope of the peak exercise ST segment**,**number of major vessels colored by fluoroscopy** and **thalassemia status**. Don't worry these attributes will be listed again for easier reference.
Although the complete database contains 76 attributes, most studies and machine learning models focus on this subset of 14 attributes. The primary objective using this dataset is to predict whether a patient has heart disease based on these attributes.

4.1.2 Parameters of the Datsaet

Table 4.1: Description of Parameters

Parameter	Description
id	Unique identifier for each patient
age	Age of the patient in years
origin	Place of study
sex	Gender of the patient (Male/Female)
cp	Chest pain type (typical angina, atypical angina, non-anginal, asymptomatic)
trestbps	Resting blood pressure (mm Hg)
chol	Serum cholesterol (mg/dl)
fbs	Fasting blood sugar level (>120 mg/dl)
restecg	Resting electrocardiographic results (normal, ST-T abnormality, LV hypertrophy)
thalach	Maximum heart rate achieved
exang	Exercise-induced angina (True/False)
oldpeak	ST depression induced by exercise relative to rest
slope	The slope of the peak exercise ST segment
ca	Number of major vessels (0-3) colored by fluoroscopy
thal	Thalassemia status (normal, fixed defect, reversible defect)
num	Predicted attribute indicating the presence of heart disease

These are the parameters that are utilized in the Machine Learning Model as well. I have obtained the dataset from Kaggle, and the link to the same can be found here. The dataset contains the data of 920 patients. The dataset consisted of a .csv file which can be viewed here.

4.1.2.1 Preprocessing Steps for the Dataset:

1. **Handling Missing Values:** Use imputation techniques to fill in any missing data points, ensuring the dataset is complete and usable. Managing missing values is crucial when analyzing data and building models for a few key reasons:
   - **Data Integrity:** When data points are missing, it can throw off the accuracy and overall quality of the dataset, leading to flawed analyses and conclusions.
   - **Avoiding Bias:** Ignoring missing values might skew results, as the available data might not accurately represent the entire dataset. This can introduce bias into statistical inferences and machine learning models.
   - **Improving Model Performance:** Many machine learning algorithms struggle with missing data. By addressing these gaps, models can be trained more effectively on complete datasets, leading to better performance and more reliable results.
   - **Improving Model Performance:** Missing values reduce the sample size, weakening the statistical power of analyses. Dealing with missing data helps ensure that studies remain robust and trustworthy.
   - **Preventing Misinterpretation:** Incomplete data can lead analysts to draw incorrect conclusions or make faulty assumptions. Handling missing values appropriately makes insights derived from the data more accurate and dependable.
   - **Supporting Decision-Making:** In both business and research contexts, decisions based on incomplete or inaccurate data can have significant consequences. Managing missing values ensures that decisions are based on complete and reliable information.
2. **Normalization:** Normalize numerical features to maintain consistent performance across different scales, which is crucial for the proper functioning of many machine learning algorithms.
3. **Categorical Encoding:** Convert categorical variables into a numerical format using one-hot encoding. Machine learning models typically require input data to be in numerical format. By using one-hot encoding, one can transform categorical variables into a format that these models can study. Each category becomes a separate feature with a binary representation, allowing the model to learn relationships and patterns in the data. This transformation allows machine learning models to process categorical data effectively.

Fig. 4.1: ECG

4.2 Machine Learning Models

The following machine models can be applied to study the dataset, which will help us predict a patient's heart disease risk.

1. **Logistic Regression**
   Logistic regression is a widely used supervised machine learning algorithm designed for predicting categorical outcomes based on a set of independent variables. Ideal for classification tasks, logistic regression predicts the likelihood of a binary dependent variable, making it well-suited for applications where the result is either yes or no, true or false, or 0 or 1. Unlike linear regression, which addresses regression problems, logistic regression employs an "S" shaped logistic or sigmoid function to generate probabilistic values between 0 and 1 rather than exact binary values.
   The sigmoid function is defined as:

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

   where z represents the input value. This function maps any input to a value within this range, effectively modeling the probability that a given input belongs to a certain class. This capability allows logistic regression to classify observations using continuous or discrete datasets and to identify the most influential variables for classification. The logistic function's curve demonstrates the likelihood of an event, such as the probability of cells being cancerous or a mouse being obese based on weight. Consequently, logistic regression is a powerful tool for binary classification, providing interpretable results and handling diverse data types effectively.

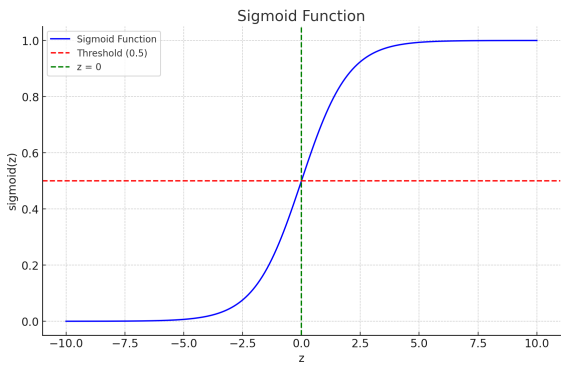


Fig. 4.2: Graph of a Sigmoid Function

2. **Random Forest**
   Random Forest is a versatile machine learning technique used for tasks like classification and regression. It works by creating multiple decision trees during training and combining their predictions to enhance accuracy and prevent overfitting, where the model learns too much from the training data and performs poorly on new data.

   One of its main strengths is its ability to handle large datasets with many features (high dimensionality) effectively. It achieves this by building each decision tree on a random subset of the training data, ensuring diversity among the trees. This diversity helps the model capture different aspects of the data and prevents it from relying too heavily on any particular subset. It also has the advantage of being able to handle missing values in the data, which is common in real-world datasets. By averaging the predictions of multiple trees, it can still provide reliable predictions even if some data points have missing values.

   It's a powerful tool used for both regression and classification tasks in machine learning. It operates by constructing multiple decision trees, each trained on a subset of the data. Before training, we need to set three key hyperparameters: **node size, the number of trees, and the number of features sampled**.

   Each decision tree in the ensemble is built using a random sample of the training data, with replacement, known as the bootstrap sample. Additionally, to inject further randomness and ensure diversity among the trees, a subset of features is randomly selected for each tree, a process called feature bagging. During prediction, the individual decision trees make their predictions, which are then combined to produce the final result. For regression tasks, the predictions of all trees are averaged, while for classification tasks, the class with the most votes among the trees is chosen.

   One unique aspect of Random Forest is the use of **out-of-bag (OOB) samples**. These are data points not included in the bootstrap sample for each tree and serve as a validation set during training. After the model is trained, the OOB samples are used for cross-validation, ensuring the robustness of the predictions.

   In summary, Random Forest harnesses the power of multiple decision trees trained on random subsets of data to provide accurate and robust predictions for both regression and classification problems. It introduces randomness through bootstrap sampling and feature bagging, utilizing OOB samples for validation and cross-validation to enhance the model's reliability.
   The below diagram and flowchart will give you a better idea.

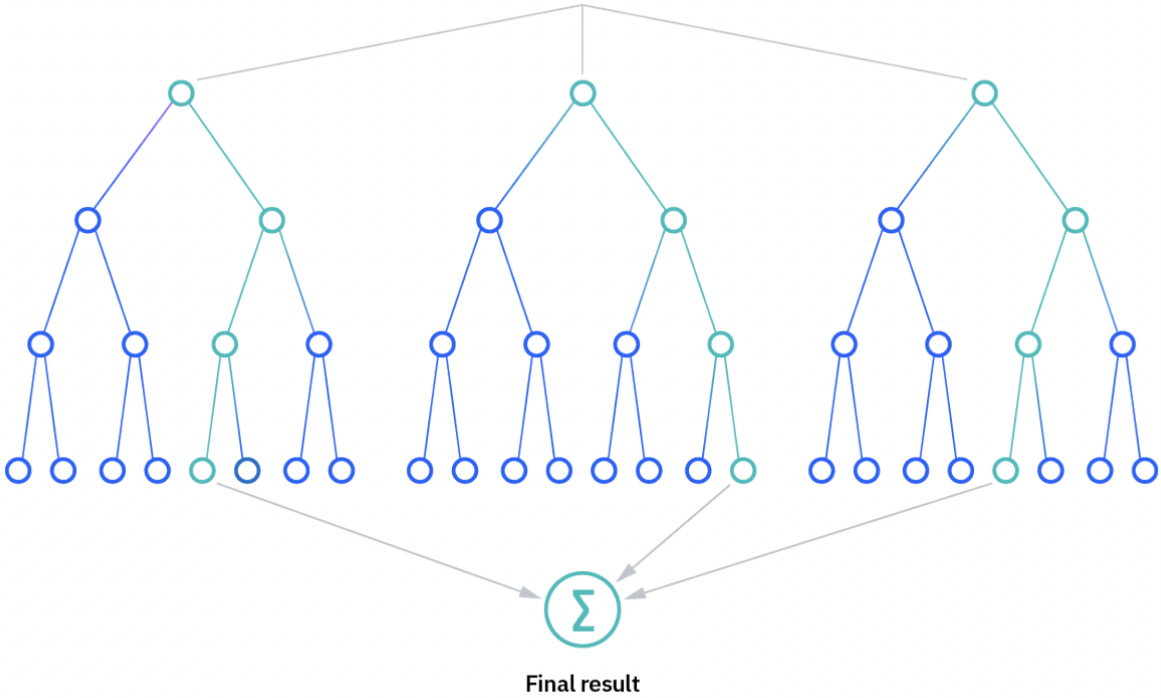


Fig. 4.3: Random Forest

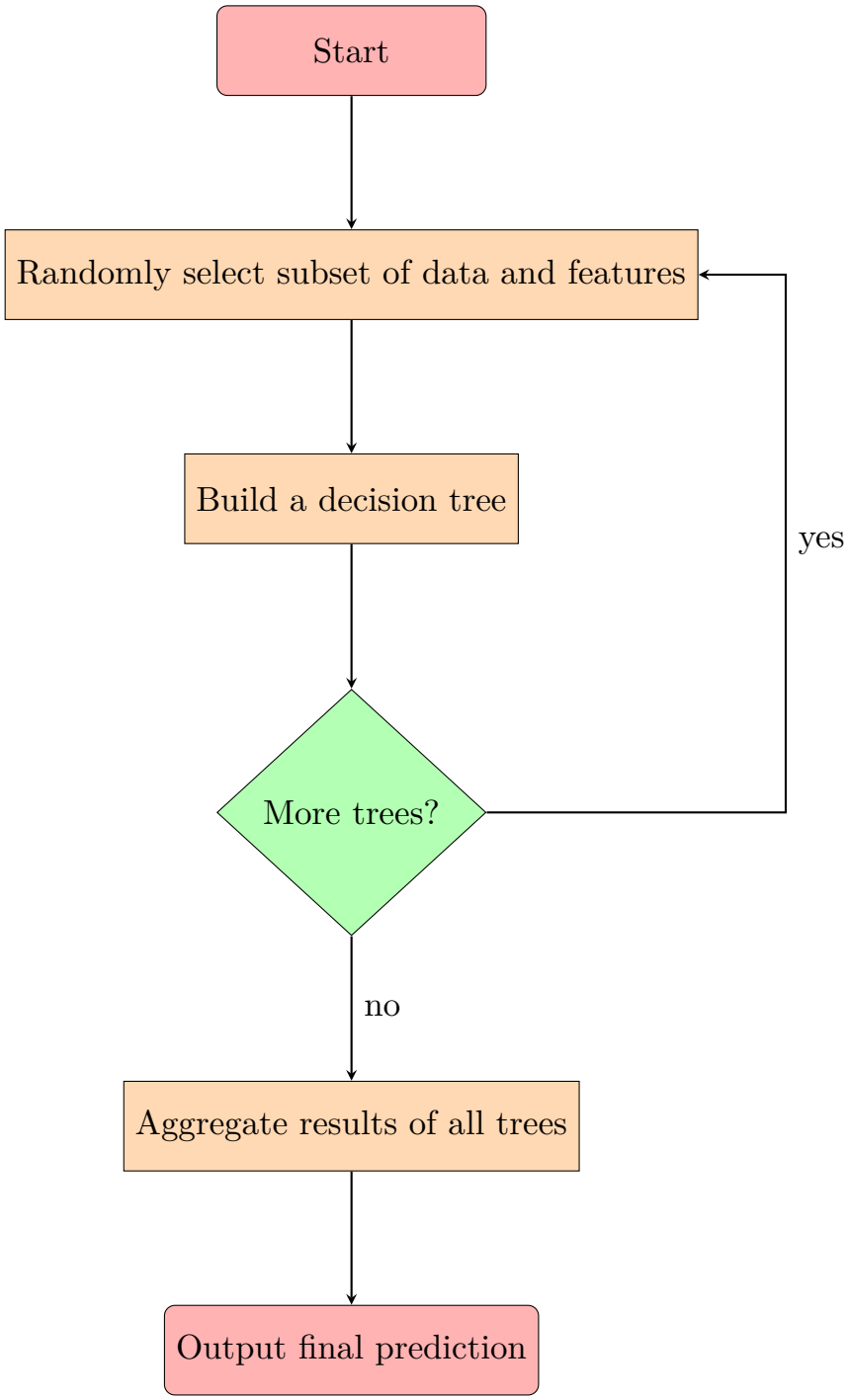


Fig. 4.4: Flowchart representing Random Forest Algorithm

Bibliography

These are the following links and papers I referred to make this report.

1. Kaggle: https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data/data
2. NHS: https://www.nhs.uk/conditions/coronary-heart-disease/
3. NHLBI: https://www.nhlbi.nih.gov/health/coronary-heart-disease/symptoms
4. NCBI: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8898839/
5. Coursera: Link to Andrew NG course on Machine Learning
6. GeekForGeeks: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/
7. IBM: https://www.ibm.com/topics/random-forest
8. Machine Learning Technology-Based Heart Disease Detection Models: https://onlinelibrary.wiley.com/doi/10.1155/2022/7351061
9. International application of a new probability algorithm for the diagnosis of coronary artery disease: https://www.ajconline.org/article/0002-9149(89)90524-9/abstract

I also referred to previous reports of students to see how they have done their projects, and I took some input on the format as well.

code

June 17, 2024

1 Importing Important Packages

[33]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.figure_factory as ff
import plotly.graph_objects as go
from sklearn.experimental import enable_iterative_imputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder,␣
↪KBinsDiscretizer, PolynomialFeatures
from sklearn.impute import SimpleImputer, KNNImputer, IterativeImputer
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
↪GradientBoostingClassifier, RandomForestRegressor
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report, mean_absolute_error, mean_squared_error, r2_score
import warnings
```

2 Loading the Important Packages

[34]:
```
# Loading the data set from the .csv file

column_names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',␣
↪'thalch', 'exang', 'oldpeak',
                'slope', 'ca', 'thal', 'target']

Data = pd.read_csv("/Users/kanishka.arora/Downloads/heart_disease_uci 2.csv",␣
↪names=column_names, header=0)
```

Variable	Description
age	Age of the patient in years
sex	Gender of the patient (0 = male, 1 = female)
cp	Chest pain type: 0: Typical angina, 1: Atypical angina, 2: Non-anginal pain, 3: Asymptomatic
trestbps	Resting blood pressure in mm Hg
chol	Serum cholesterol in mg/dl
fbs	Fasting blood sugar level, categorized as above 120 mg/dl (1 = true, 0 = false)
restecg	Resting electrocardiographic results: 0: Normal, 1: Having ST-T wave abnormality, 2: Showing probable or definite left ventricular hypertrophy
thalach	Maximum heart rate achieved during a stress test
exang	Exercise-induced angina (1 = yes, 0 = no)
oldpeak	ST depression induced by exercise relative to rest
slope	Slope of the peak exercise ST segment: 0: Upsloping, 1: Flat, 2: Downsloping
ca	Number of major vessels (0-4) colored by fluoroscopy
thal	Thalium stress test result: 0: Normal, 1: Fixed defect, 2: Reversible defect, 3: Not described
target(num)	Heart disease status (0 = no disease, 1 = presence of disease)

[35]:
```
print(Data['target'])
```

```
0      0
1      2
2      1
3      0
4      0
      ..
915    1
916    0
917    2
918    0
919    1
Name: target, Length: 920, dtype: int64
```

3 Converting Columns into 'Numerical' or 'Categorical' type

[36]:
```
# Manually convert specific columns to categorical based on domain knowledge

categorical_columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca',␣
↪'thal', 'target']
```

```
for column in categorical_columns:
    Data[column] = Data[column].astype('category')

# Convert other columns to numeric, coercing errors to NaN

numerical_columns = [col for col in Data.columns if col not in␣
 ↪categorical_columns]
for column in numerical_columns:
    Data[column] = pd.to_numeric(Data[column], errors='coerce')
    Data[column] = Data[column].astype('float')
```

[37]:
```
# We print this to see and confirm that the first row of Data is filled with␣
 ↪non-numeric values, and hence delete it

print(Data.head(5))
```

```
    age     sex               cp  trestbps   chol    fbs         restecg  \
0  63.0    Male   typical angina     145.0  233.0   True  lv hypertrophy
1  67.0    Male     asymptomatic     160.0  286.0  False  lv hypertrophy
2  67.0    Male     asymptomatic     120.0  229.0  False  lv hypertrophy
3  37.0    Male      non-anginal     130.0  250.0  False          normal
4  41.0  Female  atypical angina     130.0  204.0  False  lv hypertrophy

   thalch  exang  oldpeak        slope   ca               thal target
0   150.0  False      2.3  downsloping  0.0       fixed defect      0
1   108.0   True      1.5         flat  3.0             normal      2
2   129.0   True      2.6         flat  2.0  reversable defect      1
3   187.0  False      3.5  downsloping  0.0             normal      0
4   172.0  False      1.4    upsloping  0.0             normal      0
```

[38]:
```
# We delete the first row for the reasons mentioned in the previous cell

data = Data.drop(0).reset_index(drop=True)
print(data.head())
```

```
    age     sex               cp  trestbps   chol    fbs         restecg  \
0  67.0    Male     asymptomatic     160.0  286.0  False  lv hypertrophy
1  67.0    Male     asymptomatic     120.0  229.0  False  lv hypertrophy
2  37.0    Male      non-anginal     130.0  250.0  False          normal
3  41.0  Female  atypical angina     130.0  204.0  False  lv hypertrophy
4  56.0    Male  atypical angina     120.0  236.0  False          normal

   thalch  exang  oldpeak        slope   ca               thal target
0   108.0   True      1.5         flat  3.0             normal      2
1   129.0   True      2.6         flat  2.0  reversable defect      1
2   187.0  False      3.5  downsloping  0.0             normal      0
3   172.0  False      1.4    upsloping  0.0             normal      0
```

```
4   178.0  False      0.8    upsloping  0.0             normal      0
```

4 Exploratory Data Analysis

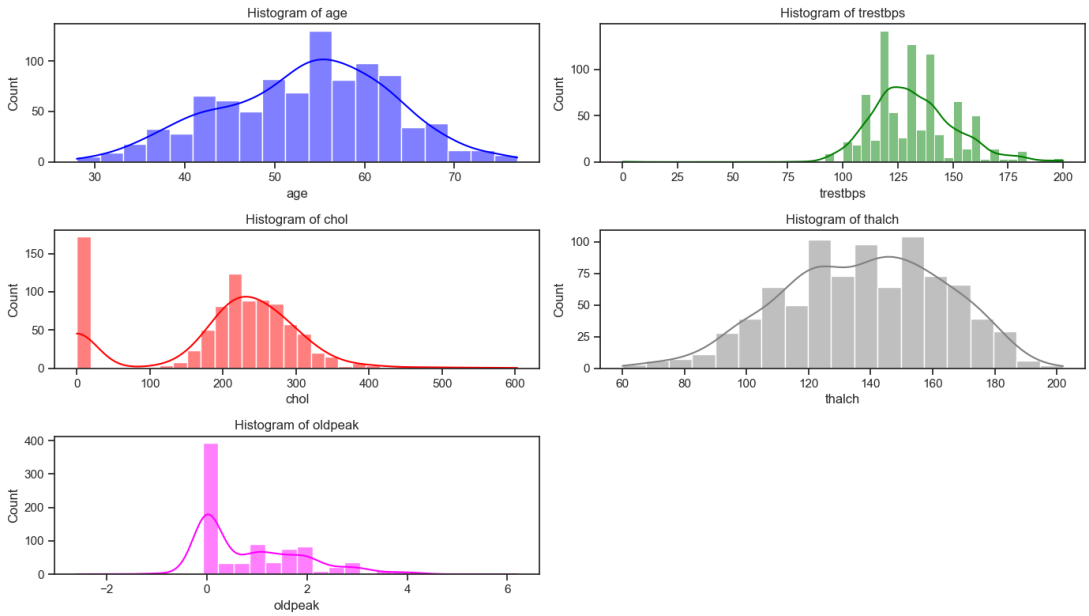
In this section we explore the dataset to find patterns and important information.

[39]:
```
categorical_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca',␣
 ↪'thal']
numeric_cols = [col for col in data.columns if col not in categorical_cols and␣
 ↪col != 'target']  # Assuming 'target' is your label column
```

4.0.1 1. Histograms for Numerical Columns

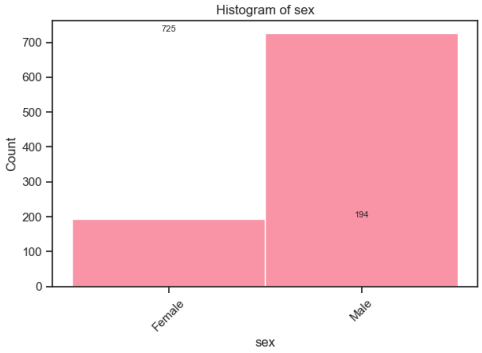
-

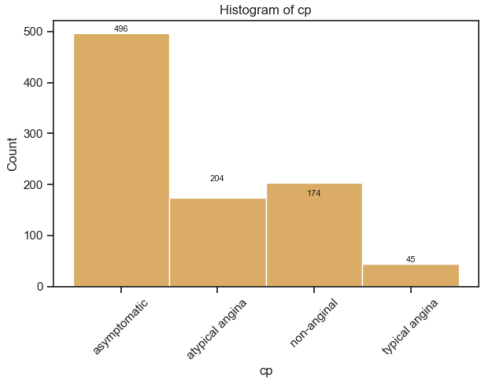
[40]:
```
# Assuming numeric_cols contains the list of numeric column names
numeric_cols = data.select_dtypes(include=['number']).columns

# Define a list of colors for the histograms
colors = ['blue', 'green', 'red', 'grey', 'magenta']
plt.figure(figsize=(14, 8))
num_numeric = len(numeric_cols)
rows_numeric = num_numeric // 2 + num_numeric % 2

for i, col in enumerate(numeric_cols):
    plt.subplot(rows_numeric, 2, i + 1)
    color = colors[i % len(colors)]  # Cycle through the colors list
    sns.histplot(data[col], kde=True, color=color)
    plt.title(f'Histogram of {col}')

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

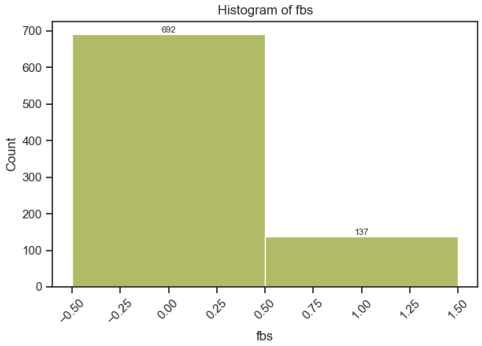


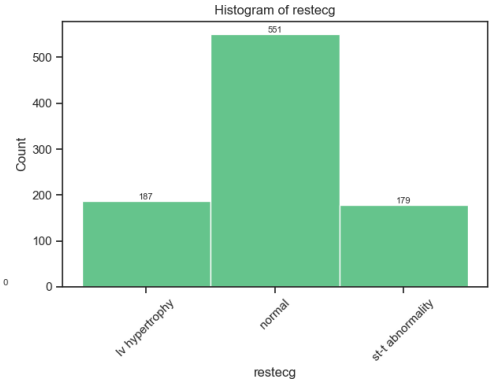
-

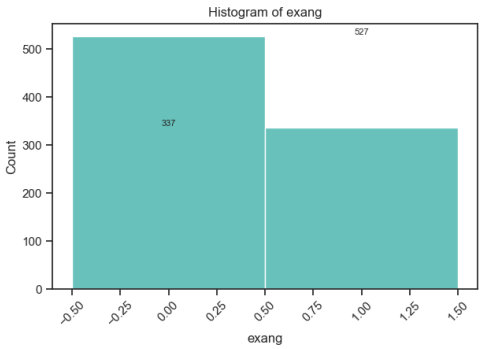
[41]:
```
# Generate a list of unique colors
colors = sns.color_palette("husl", len(categorical_cols))

# Plot histograms for categorical columns with category labels on x-axis and␣
 ↪frequency annotations
plt.figure(figsize=(14, 20))
num_categorical = len(categorical_cols)
for i, col in enumerate(categorical_cols):
    plt.subplot(num_categorical // 2 + num_categorical % 2, 2, i + 1)
    plot = sns.histplot(data[col], discrete=True, color=colors[i])
    plt.title(f'Histogram of {col}')
    plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
    labels = data[col].unique()
    for j, label in enumerate(labels):
        count = (data[col] == label).sum()
        plt.text(j, count, f'{count}', ha='center', va='bottom', fontsize=8)
plt.tight_layout()
plt.show()
```

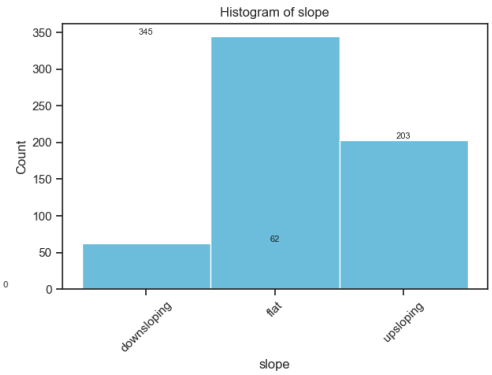

Histogram of sex
Count
sex
Female
Male
725
194

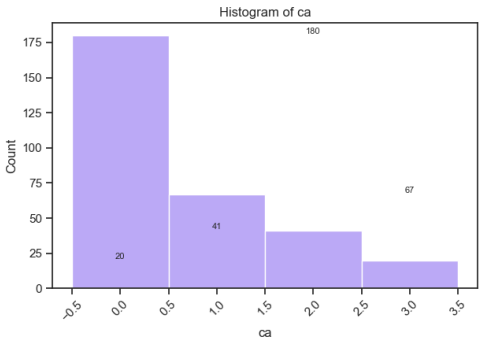

Histogram of cp
Count
cp
asymptomatic
atypical angina
non-anginal
typical angina
496
204
174
45

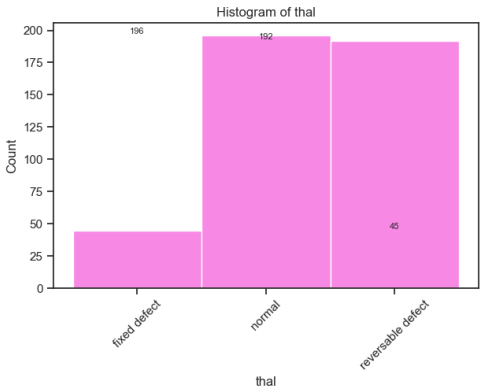

Histogram of fbs
Count
fbs
692
137


Histogram of restecg
Count
restecg
lv hypertrophy
normal
st-t abnormality
187
551
179


Histogram of exang
Count
exang
337
527

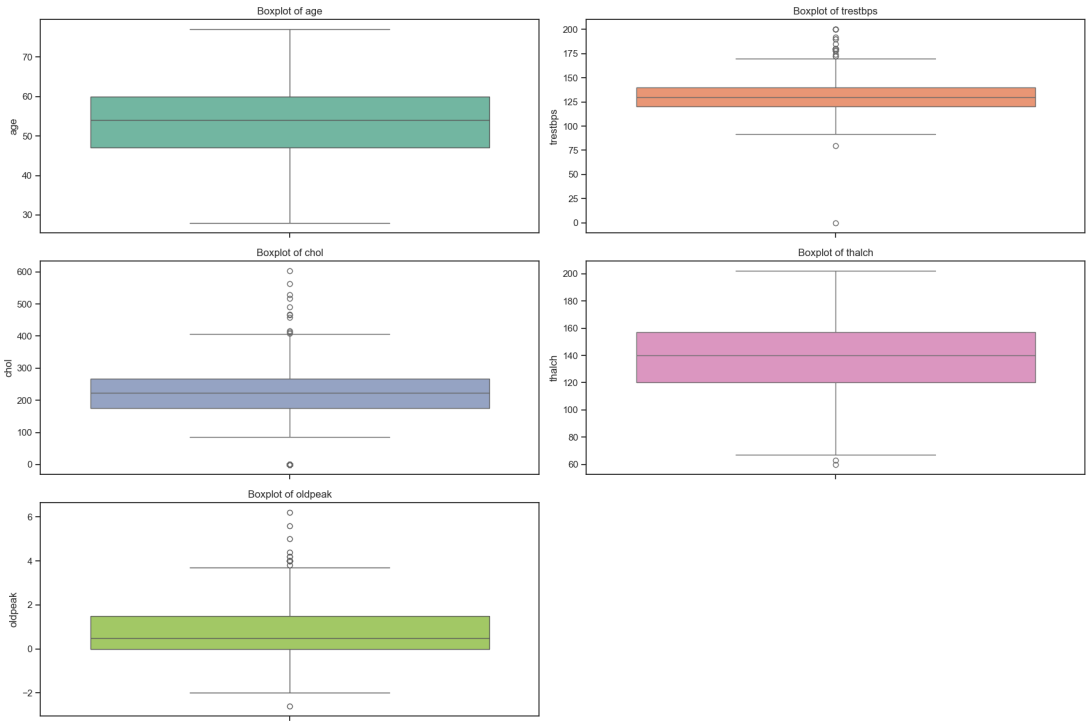

Histogram of slope
Count
slope
downsloping
flat
upsloping
345
62
203


Histogram of ca
Count
ca
180
20
41
67


Histogram of thal
Count
thal
fixed defect
normal
reversable defect
196
192
45

2. Box Plots for Numerical Columns Box plots are valuable tools in data analysis for summarizing the distribution of numerical data, highlighting key statistics like median, quartiles, and outliers. They offer a clear and concise way to visualize and compare multiple variables within a dataset, aiding in exploratory data analysis and insights generation.

[42]:
```
# Plot box plots for numeric columns with different colors
plt.figure(figsize=(18, 12))  # Larger figure size
colors = sns.color_palette('Set2')  # Generate a palette of colors
for i, col in enumerate(numeric_cols):
    plt.subplot(len(numeric_cols) // 2 + len(numeric_cols) % 2, 2, i + 1)
    sns.boxplot(y=data[col], color=colors[i % len(colors)])  # Use different␣
 ↪color for each plot
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()
```



3. Pair Plots for Numerical Columns

Pair plots, also known as scatter plot matrices, are a powerful visualization tool used to explore relationships between pairs of variables in a dataset. They provide a comprehensive view of pairwise relationships among multiple variables simultaneously.

- **Scatter Plots**
  Each subplot in a pair plot displays the relationship between two variables. For instance, if you have variables x1, x2, …, xn, the pair plot includes scatter plots like x1 vs x2, x1 vs x3, …, x1 vs xn, and so forth.
- **Diagonal Plots**
  Along the diagonal of the pair plot matrix, you typically find histograms or density plots for each variable. These diagonal plots show the distribution of each variable in the dataset.
- **Correlation**

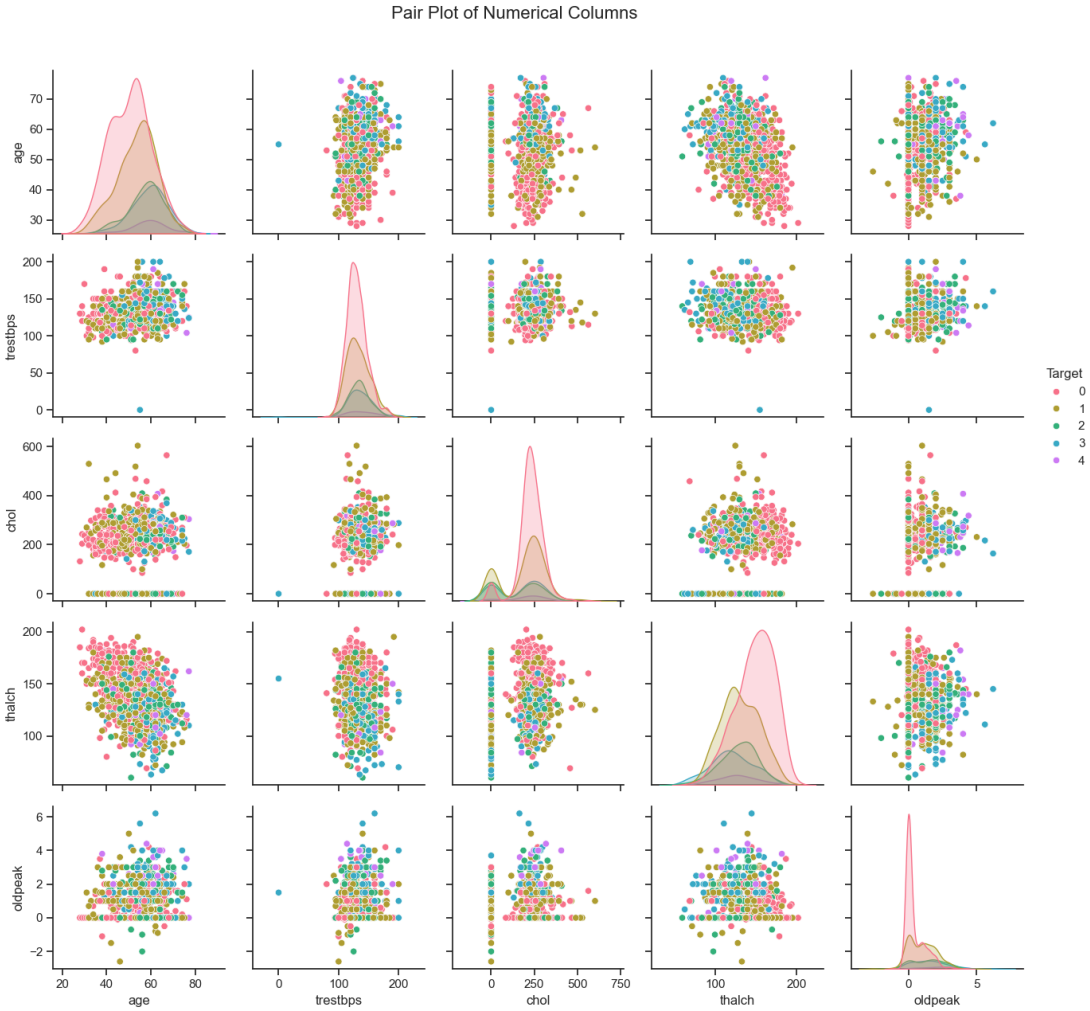
  Pair plots are effective for identifying:

Analysis Technique	Description
Linear Relationships	Determining if variables have a positive or negative linear relationship.
Non-linear Relationships	Detecting curvilinear relationships that may not be apparent from summary statistics.
Clustering	Identifying clusters or groups of data points that share similar characteristics across variables.
Outliers	Spotting outliers that stand out in scatter plots.

- **Data Distribution**
  Diagonal plots in pair plots help assess the distribution and skewness of each variable, aiding in understanding the spread and central tendency of data points.
- **Strength of Relationship**
  Scatter plots in pair plots reveal the strength and direction of relationships between variables. Tight clustering around a line indicates a strong relationship, while random scattering suggests a weak or non-existent relationship.
- **Multivariate Insights**
  Pair plots facilitate multivariate exploration by visualizing relationships across multiple dimensions simultaneously. This is particularly valuable in datasets with numerous variables to understand interactions and dependencies comprehensively.

[43]:
```
# Set the aesthetic style of the plots
sns.set(style="ticks")

# Create a pair plot for the numerical columns, color-coded by the 'target'␣
 ↪column
pairplot = sns.pairplot(Data, vars=numerical_columns, hue='target',␣
 ↪palette="husl", diag_kind='kde', markers=["o", "o", "o", "o", "o"])

# Add a legend and adjust its position
pairplot._legend.set_title("Target")
pairplot._legend.set_bbox_to_anchor((1.05, 0.6))

# Improve the layout and aesthetics
pairplot.fig.suptitle('Pair Plot of Numerical Columns', y=1.02, fontsize=16)
```

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



Legend Explanation for the above plot In the above plot,

Value	Description
0	No heart disease
1	Mild heart disease
2	Moderate heart disease
3	Severe heart disease
4	Critical heart disease

4. Correlation Heat Map for Numerical Columns A correlation heatmap is a visual representation of the correlation matrix, showing the pairwise correlations between different numerical

features in your dataset.

- **Correlation Coefficient**

  The correlation coefficient is a statistical measure that calculates the strength and direction of the relationship between two variables. It ranges from -1 to 1:

Correlation Type	Description
Perfect positive correlation (+1)	As one variable increases, the other variable increases proportionally.
Perfect negative correlation (-1)	As one variable increases, the other variable decreases proportionally.
No correlation (0)	There is no linear relationship between the variables.

- **What the Heatmap Shows**

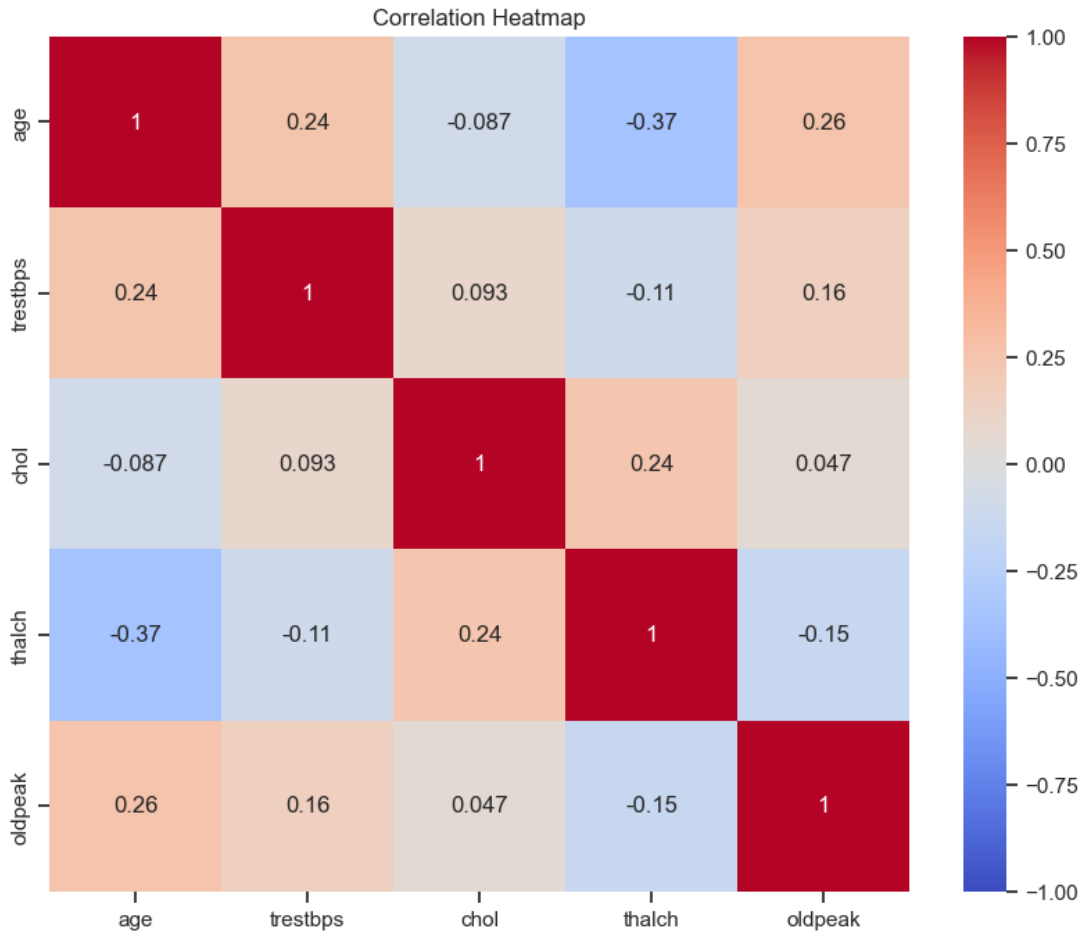
  The heatmap provides a visual way to interpret the correlation matrix:

  – Color Coding: The colors represent the strength and direction of the correlations. Typically, a color scale is used (e.g., from blue to red), where,

Color Indicator	Description
Darker shades of one color (e.g., blue)	Indicate strong negative correlations.
Darker shades of another color (e.g., red)	Indicate strong positive correlations.
Lighter shades or neutral colors	Indicate weak or no correlation.

  – Values in Cells: Each cell in the heatmap contains a numerical value representing the correlation coefficient between two variables.

[44]:
```
# Plot correlation heatmap for numeric columns

plt.figure(figsize=(10, 8))
corr = data[numeric_cols].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()
```



By looking at the heatmap, you can quickly identify:

- **Strong Positive Correlations**

  Variables that have high correlation coefficients close to +1. These variables ove together in the same direction.

- **Strong Negative Correlations**

  Variables that have high correlation coefficients close to -1. These variables move in opposite directions.

- **Weak or No Correlations**

  Variables that have correlation coefficients close to 0. There is no linear relationship between these variables.

5 Handling of Missing Values

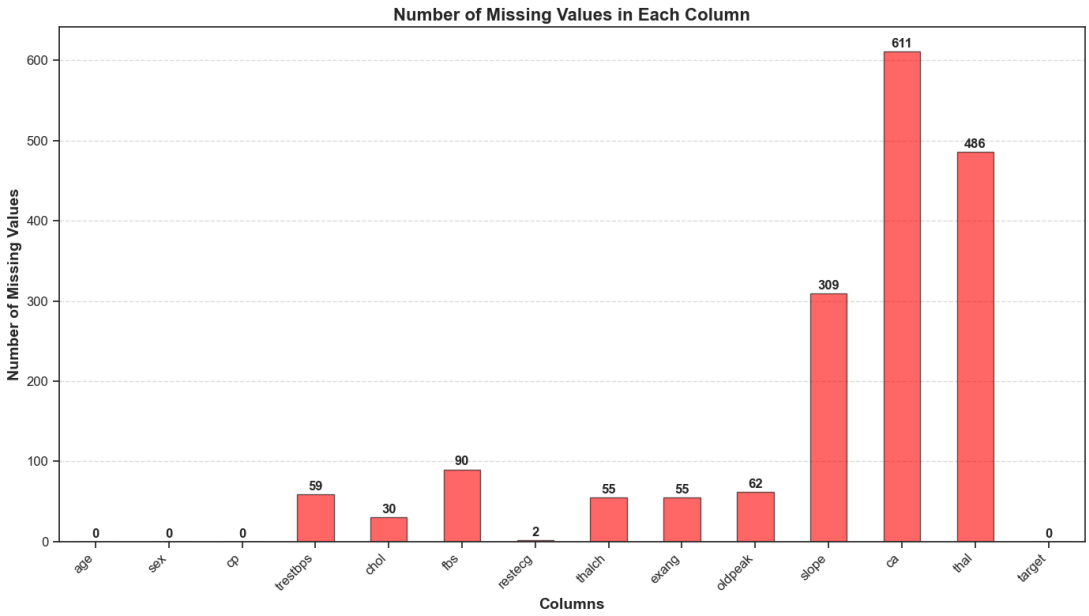
5.0.1 Checking the number of missing values in each column

[45]:
```
# Handling of missing values
# Over here we check the number of missing values that we have in each columns

missing_values = data.isnull().sum()
print(missing_values)
```

```
age           0
sex           0
cp            0
trestbps     59
chol         30
fbs          90
restecg       2
thalch       55
exang        55
oldpeak      62
slope       309
ca          611
thal        486
target        0
dtype: int64
```

5.0.2 Plotting the missing value data as a histogram

[46]:
```
# We plot the missing values as a histogram

plt.figure(figsize=(14, 8))
missing_values.plot(kind='bar', color='red', edgecolor='black', alpha=0.6)
plt.title('Number of Missing Values in Each Column', fontsize=16,␣
 ↪fontweight='bold')
plt.xlabel('Columns', fontsize=14, fontweight='bold')
plt.ylabel('Number of Missing Values', fontsize=14, fontweight='bold')
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Adding value labels on top of the bars
for i, value in enumerate(missing_values):
    plt.text(i, value + 5, str(value), ha='center', fontsize=12,␣
 ↪fontweight='bold')

plt.show()
```



5.0.3 Seperating columns which have missing values

[47]:
```
# Check for columns which have missing values

missing_data_cols = data.isnull().sum()[data.isnull().sum() > 0].index.tolist()
print(f"Columns with missing values: \n{missing_data_cols}")
```

```
Columns with missing values:
['trestbps', 'chol', 'fbs', 'restecg', 'thalch', 'exang', 'oldpeak', 'slope',
'ca', 'thal']
```

[48]:
```
# Loop through each column in the DataFrame

for column in data.columns:
    # Check if the column is categorical
    if data[column].dtype == 'object' or data[column].dtype == 'category':
        # Print the value counts of the column
        print(data[column].value_counts())
        print('-------------')
```

```
sex
Male      725
Female    194
Name: count, dtype: int64
-------------
cp
asymptomatic       496
```

```
non-anginal        204
atypical angina    174
typical angina      45
Name: count, dtype: int64
-------------
fbs
False    692
True     137
Name: count, dtype: int64
-------------
restecg
normal              551
lv hypertrophy      187
st-t abnormality    179
Name: count, dtype: int64
-------------
exang
False    527
True     337
Name: count, dtype: int64
-------------
slope
flat           345
upsloping      203
downsloping     62
Name: count, dtype: int64
-------------
ca
0.0    180
1.0     67
2.0     41
3.0     20
Name: count, dtype: int64
-------------
thal
normal               196
reversable defect    192
fixed defect          45
Name: count, dtype: int64
-------------
target
0    410
1    265
2    109
3    107
4     28
Name: count, dtype: int64
-------------
```

5.0.4 Classification of Columns and data segregation

[49]:
```
# Classifying Columns for better clarity

categorical_cols = ['thal', 'ca', 'slope', 'exang', 'restecg','fbs', 'cp',␣
 ↪'sex', 'num']
bool_cols = ['fbs', 'exang']
numeric_cols = ['oldpeak', 'thalch', 'chol', 'trestbps', 'age']

data_null = data[data.isnull()]
data_not_null = data[data.notnull()]
```

5.0.5 Defining Functions to fill Missing values in columns(Imputataion)

The imputation method used in the code employs a combination of Iterative Imputer and Random Forest models to handle missing data for both categorical and continuous variables effectively.

1. **Iterative Imputer**

   The code utilizes the IterativeImputer from scikit-learn, which iteratively models each feature with missing values as a function of other features. This method estimates the missing values through multiple iterations until a convergence criterion is met. In this code, a RandomForestRegressor serves as the estimator for the imputer, enabling the model to capture complex, non-linear relationships between features, thereby enhancing the imputation's accuracy and reliability.

2. **Random Forest for Prediction**

   For categorical missing data, the code trains a RandomForestClassifier on the non-missing data to predict the missing values. Similarly, for continuous missing data, it trains a RandomForestRegressor on the non-missing data to predict and impute the missing values. This approach leverages the power of ensemble learning, where the random forest model, consisting of multiple decision trees, provides robust predictions by reducing overfitting and improving generalization.

[50]:
```
# define the function to impute the missing values in thal column

import warnings
warnings.filterwarnings('ignore')

warnings.filterwarnings('ignore')

def encode_columns(df, bool_cols, label_encoder):
    """
    Encode categorical and boolean columns in a DataFrame.
    """
    for col in df.columns:
        if df[col].dtype == 'object' or df[col].dtype == 'category':
            df[col] = label_encoder.fit_transform(df[col])
```

```
        elif col in bool_cols:
            df[col] = label_encoder.fit_transform(df[col].astype(str))
    return df

def impute_missing_columns(df, columns, iterative_imputer):
    """
    Impute missing values in the specified columns using iterative imputer.
    """
    for col in columns:
        if df[col].isnull().sum() > 0:
            df[col] = iterative_imputer.fit_transform(df[col].values.
 ↪reshape(-1, 1))[:, 0]
    return df

def impute_categorical_missing_data(data, passed_col, bool_cols=[],␣
 ↪missing_data_cols=[]):
    # Split data into null and not-null parts for the passed column
    data_null = data[data[passed_col].isnull()]
    data_not_null = data[data[passed_col].notnull()]

    # Separate features and target variable
    X = data_not_null.drop(passed_col, axis=1)
    y = data_not_null[passed_col]

    # Initialize label encoder and iterative imputer
    label_encoder = LabelEncoder()
    iterative_imputer =␣
 ↪IterativeImputer(estimator=RandomForestRegressor(random_state=42),␣
 ↪add_indicator=True)

    # Encode columns in X
    X = encode_columns(X, bool_cols, label_encoder)

    # Encode target column if it is boolean
    if passed_col in bool_cols:
        y = label_encoder.fit_transform(y)

    # Impute other columns with missing values
    X = impute_missing_columns(X, missing_data_cols, iterative_imputer)

    # Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

    # Train RandomForest classifier
    rf_classifier = RandomForestClassifier()
    rf_classifier.fit(X_train, y_train)
```

```
    # Predict and calculate accuracy
    y_pred = rf_classifier.predict(X_test)
    acc_score = accuracy_score(y_test, y_pred)
    print(f"The feature '{passed_col}' has been imputed with {round((acc_score␣
 ↪* 100), 2)}% accuracy\n")
    print("_________________________________")

    # Prepare the null part of the data for prediction
    X_null = data_null.drop(passed_col, axis=1)
    X_null = encode_columns(X_null, bool_cols, label_encoder)
    X_null = impute_missing_columns(X_null, missing_data_cols,␣
 ↪iterative_imputer)

    # Predict the missing values for the passed column
    if len(data_null) > 0:
        data_null[passed_col] = rf_classifier.predict(X_null)
        if passed_col in bool_cols:
            data_null[passed_col] = data_null[passed_col].map({0: False, 1:␣
 ↪True})

    # Combine the imputed data
    data_combined = pd.concat([data_not_null, data_null])
    return data_combined[passed_col]



from sklearn.model_selection import train_test_split

def impute_continuous_missing_data(data, passed_col):
    data_null = data[data[passed_col].isnull()]
    data_not_null = data[data[passed_col].notnull()]

    X = data_not_null.drop(passed_col, axis=1)
    y = data_not_null[passed_col]

    other_missing_cols = [col for col in data.columns if col != passed_col and␣
 ↪data[col].isnull().sum() > 0]

    label_encoder = LabelEncoder()

    for col in X.columns:
        if col != 'trestbps' and (X[col].dtype == 'object' or X[col].dtype ==␣
 ↪'category'):
            X[col] = label_encoder.fit_transform(X[col])

    if 'trestbps' in X.columns:
```

```
        X['trestbps'] = pd.to_numeric(X['trestbps'], errors='coerce')

    iterative_imputer = SimpleImputer(strategy='mean')  # Impute with mean value

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values = iterative_imputer.
↪fit_transform(col_with_missing_values)
            if imputed_values.size > 0:  # Check if imputed_values is not empty
                X[col] = imputed_values[:, 0]
            else:
                print(f"No imputed values for column {col}")
        else:
            pass

    if len(data_not_null) < 2:
        print("Not enough non-null values to perform train-test split.")
        return data[passed_col]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

    if len(X_train) == 0 or len(X_test) == 0:
        print("Train or test set is empty.")
        return data[passed_col]

    rf_regressor = RandomForestRegressor()

    rf_regressor.fit(X_train, y_train)

    y_pred = rf_regressor.predict(X_test)

    print("MAE(Mean Ansolute Error) =", mean_absolute_error(y_test, y_pred),␣
↪"\n")
    print("RMSE(Root Mean Square Error) =", mean_squared_error(y_test, y_pred,␣
↪squared=False), "\n")
    print("R2(R - Squared) =", r2_score(y_test, y_pred), "\n")
    print("--------------------------------")

    X_null = data_null.drop(passed_col, axis=1)

    for col in X_null.columns:
        if col != 'trestbps' and (X_null[col].dtype == 'object' or X_null[col].
↪dtype == 'category'):
            X_null[col] = label_encoder.fit_transform(X_null[col])
```

```
    if 'trestbps' in X_null.columns:
        X_null['trestbps'] = pd.to_numeric(X_null['trestbps'], errors='coerce')

    for col in other_missing_cols:
        if X_null[col].isnull().sum() > 0:
            col_with_missing_values = X_null[col].values.reshape(-1, 1)
            imputed_values = iterative_imputer.
↪fit_transform(col_with_missing_values)
            if imputed_values.size > 0:  # Check if imputed_values is not empty
                X_null[col] = imputed_values[:, 0]
            else:
                print(f"No imputed values for column {col}")
    else:
        pass

    if len(data_null) > 0:
        X_null_imputed = iterative_imputer.fit_transform(X_null)
        if X_null_imputed.shape[1] < X_train.shape[1]:  # Check if number of␣
↪features match
            X_null_imputed = np.hstack((X_null_imputed, np.
↪zeros((X_null_imputed.shape[0], X_train.shape[1] - X_null_imputed.
↪shape[1]))))
        data_null[passed_col] = rf_regressor.predict(X_null_imputed)
    else:
        pass

    data_combined = pd.concat([data_not_null, data_null])

    return data_combined[passed_col]
```

[51]:
```
data.isnull().sum()[data.isnull().sum() > 0].sort_values(ascending=False)
```

[51]:
```
ca          611
thal        486
slope       309
fbs          90
oldpeak      62
trestbps     59
thalch       55
exang        55
chol         30
restecg       2
dtype: int64
```

[52]:
```
print(missing_data_cols)
```

```
['trestbps', 'chol', 'fbs', 'restecg', 'thalch', 'exang', 'oldpeak', 'slope',
```

'ca', 'thal']

5.0.6 Imputing the Missing Values by calling the function

[53]:
```
# impute missing values using our functions

for col in missing_data_cols:
    print("Missing Values", col, ":", str(round((data[col].isnull().sum() /␣
 ↪len(data)) * 100, 2))+"%")
    if col in categorical_cols:
        data[col] = impute_categorical_missing_data(data, col)
    elif col in numeric_cols:
        data[col] = impute_continuous_missing_data(data, col)
    else:
        pass
```

```
Missing Values trestbps : 6.42%
MAE(Mean Ansolute Error) = 13.403372093023256

RMSE(Root Mean Square Error) = 17.190973023315813

R2(R - Squared) = 0.06101402404519085


--------------------------------
Missing Values chol : 3.26%
MAE(Mean Ansolute Error) = 56.72455056179776

RMSE(Root Mean Square Error) = 80.74859634568116

R2(R - Squared) = 0.4152477581817188


--------------------------------
Missing Values fbs : 9.79%
The feature 'fbs' has been imputed with 83.73% accuracy


--------------------------------
Missing Values restecg : 0.22%
The feature 'restecg' has been imputed with 60.33% accuracy


--------------------------------
Missing Values thalch : 5.98%
MAE(Mean Ansolute Error) = 16.632369942196533

RMSE(Root Mean Square Error) = 21.870794578417343

R2(R - Squared) = 0.28681072425138476


--------------------------------
```

```
Missing Values exang : 5.98%
The feature 'exang' has been imputed with 73.41% accuracy


--------------------------------
Missing Values oldpeak : 6.75%
MAE(Mean Ansolute Error) = 0.5712906976744186

RMSE(Root Mean Square Error) = 0.8229173359824575

R2(R - Squared) = 0.36543512880027995


--------------------------------
Missing Values slope : 33.62%
The feature 'slope' has been imputed with 66.39% accuracy


--------------------------------
Missing Values ca : 66.49%
The feature 'ca' has been imputed with 69.35% accuracy


--------------------------------
Missing Values thal : 52.88%
The feature 'thal' has been imputed with 64.37% accuracy


--------------------------------
```

5.0.7 Checking if the values have been imputed now or not

[54]:
```
# Checking the columns with Num Values now
# We see that the missing values have been imputed

data.isnull().sum()
```

[54]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalch      0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

[55]:
```
print(data['target'])
```

```
0      2
1      1
2      0
3      0
4      0
      ..
914    1
915    0
916    2
917    0
918    1
Name: target, Length: 919, dtype: category
Categories (5, int64): [0, 1, 2, 3, 4]
```

5.0.8 Re-writing the Target column or the num column values

The values in the Target(or num) column take the following values.

Value	Description
0	No heart disease
1	Mild heart disease
2	Moderate heart disease
3	Severe heart disease
4	Critical heart disease

As mentioned before,

We will convert the values 1,2,3,4 to 1 itself and 0 remains as 0. So now we only have two categories 0 or 1. Where,

Value	Description
0	No heart disease
1	Heart disease

This is done because any stage of heart disease is harmful and should be dispayed as same.

[56]:
```
# Ensure the target column is numeric
data['target'] = pd.to_numeric(data['target'], errors='coerce')

# Convert the target column to binary
data['target'] = data['target'].apply(lambda x: 1 if x > 0 else 0)
print(data['target'])
```

```
0      1
```

```
1      1
2      0
3      0
4      0
      ..
914    1
915    0
916    1
917    0
918    1
Name: target, Length: 919, dtype: int64
```

[57]:
```
print(data.dtypes)
```

```
age           float64
sex          category
cp           category
trestbps      float64
chol          float64
fbs            object
restecg        object
thalch        float64
exang          object
oldpeak       float64
slope          object
ca            float64
thal           object
target          int64
dtype: object
```

6 Running Logistic Regression Model

[58]:
```
# Define your categorical and numeric columns
categorical_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'thal']
numeric_cols = [col for col in data.columns if col not in categorical_cols and␣
 ↪col != 'target']  # Assuming 'target' is your label column

# Encode categorical columns
label_encoders = {}    # This is a dictionary
for col in categorical_cols:
    le = LabelEncoder() # object of LabelEncoder class
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

# Split the data into features and target
X = data.drop('target', axis=1)
y = data['target']
```

```
# Generate polynomial and interaction features
# Generate a new feature matrix consisting of all polynomial,
# combinations of the features with degree less than or equal to the specified␣
 ↪degree.
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
X_poly = poly.fit_transform(X)

# Split the new dataset
# 30% of data set for testing pourposes and 70% of the dataset for training
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.3,␣
 ↪random_state=42)


# Normalize numeric columns
# This is basically feature scaling to resize values between 0 and 1
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the logistic regression model with Grid Search for hyperparameter␣
 ↪tuning
# grid of hyperparameters that will be used for tuning a machine learning model
# This is a dictionary
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'saga'],
    'class_weight': [None, 'balanced']
}

# Initialize the grid search
# performs hyperparameter optimization through an exhaustive search over a␣
 ↪specified grid of hyperparameters.
grid_search = GridSearchCV(LogisticRegression(random_state=42, max_iter=10000),␣
 ↪param_grid, cv=5, scoring='accuracy')

# Perform the grid search on the training data
# it tries every combination of hyperparameters and finds the best set for this␣
 ↪data
grid_search.fit(X_train, y_train)

# Best parameters and score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
best_estimator = grid_search.best_estimator_
```

```
print(f'Best Parameters: {best_params}')
print(f'Best Cross-Validation Accuracy: {best_score:.2f}')
print(f'Best Estimator: {best_estimator}')

# Train the final model with best parameters
logreg = LogisticRegression(random_state=42, max_iter=10000, **best_params)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

# Evaluate the model
# We compute the accuracy with the test variable y and the prediction variable y
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:')
print(classification_report(y_test, y_pred))
```

```
Best Parameters: {'C': 1, 'class_weight': None, 'solver': 'liblinear'}
Best Cross-Validation Accuracy: 0.85
Best Estimator: LogisticRegression(C=1, max_iter=10000, random_state=42,
solver='liblinear')
Accuracy: 0.85
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.88      0.82       112
           1       0.91      0.82      0.87       164

    accuracy                           0.85       276
   macro avg       0.84      0.85      0.85       276
weighted avg       0.86      0.85      0.85       276
```

6.0.1 We print the encoded values to see what numeric value is assigned to each variable

[59]:
```
# Function to display encoded values
for col, le in label_encoders.items():
    print(f"Column: {col}")
    for class_value in le.classes_:
        encoded_value = le.transform([class_value])[0]
        print(f"  {class_value}: {encoded_value}")
    print()
```

```
Column: sex
  Female: 0
  Male: 1

Column: cp
```

```
  asymptomatic: 0
  atypical angina: 1
  non-anginal: 2
  typical angina: 3

Column: fbs
  False: 0
  True: 1

Column: restecg
  lv hypertrophy: 0
  normal: 1
  st-t abnormality: 2

Column: exang
  False: 0
  True: 1

Column: slope
  downsloping: 0
  flat: 1
  upsloping: 2

Column: thal
  fixed defect: 0
  normal: 1
  reversable defect: 2
```

7 Running Random Forest Model

[60]:
```python
def train_random_forest(data, target, categorical_cols, numeric_cols):
    # Split the data into X and y
    X = data.drop(target, axis=1)
    y = data[target]

    # Dictionary to store LabelEncoders for each categorical column
    label_encoders = {}

    # Create a new LabelEncoder for each categorical column
    for col in categorical_cols:
        if col in X.columns:
            le = LabelEncoder()
            X[col] = le.fit_transform(X[col])
            label_encoders[col] = le

    # Split the dataset
```

```
    # 30% of data set for testing pourposes and 70% of the dataset for training
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=0)

    # Scaling Data
    # Normalize numeric columns
    # This is basically feature scaling to resize values between 0 and 1
    scaler = MinMaxScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Define the Random Forest model
    rf_model = RandomForestClassifier(random_state=0, class_weight='balanced')

    # Define hyperparameters for tuning
    param_grid = {
        'n_estimators': [50, 100, 150], # n_estimators: More trees can increase␣
↪accuracy but also increase computation time.
        'max_depth': [None, 10, 20], # max_depth: Limiting tree depth can␣
↪prevent overfitting.
        'min_samples_split': [2, 5, 10], # min_samples_split: Ensures nodes␣
↪have a minimum number of samples before splitting, preventing overly␣
↪specific splits.
        'min_samples_leaf': [1, 2, 4] # Ensures leaf nodes have a minimum␣
↪number of samples, which helps smooth the model and reduce overfitting.
    }

    # Perform GridSearchCV for hyperparameter tuning
    # performs hyperparameter optimization through an exhaustive search over a␣
↪specified grid of hyperparameters
    # it tries every combination of hyperparameters and finds the best set for␣
↪this data
    grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)

    # Get the best model and parameters
    best_params = grid_search.best_params_
    best_rf_model = grid_search.best_estimator_
    best_score = grid_search.best_score_

    # Print the best hyperparameters, Score, Estimator
    print(f'Best Hyperparameters: \n{best_params}')
    print(f'\nBest Score: {best_score}')
    print(f'\nBest Estimator: {best_estimator}')

    # Train the model on the full training set
```

```
    best_rf_model.fit(X_train, y_train)
    y_pred = best_rf_model.predict(X_test)

    # Accuracy Calculation
    accuracy = accuracy_score(y_test, y_pred)
    print(f'\nAccuracy on Test Set: {accuracy:.2f}')

    # Return the best model, parameters, and accuracy without inverse␣
 ↪transforming the entire data
    return best_rf_model, best_params, accuracy

best_model, best_params, test_accuracy = train_random_forest(data, 'target',␣
 ↪categorical_cols, numeric_cols)
```

```
Best Hyperparameters:
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5,
'n_estimators': 100}

Best Score: 0.8739946705426356

Best Estimator: LogisticRegression(C=1, max_iter=10000, random_state=42,
solver='liblinear')

Accuracy on Test Set: 0.85
```

8 Running XGBoost Model

[61]:
```
def train_xgb_classifier(data, target, categorical_cols, numeric_cols):
    # Dictionary to store LabelEncoders for each categorical column
    label_encoders = {}

    # Split the data into X and y
    X = data.drop(target, axis=1)
    y = data[target]

    # Encode X data using separate label encoder for all categorical columns␣
 ↪and save it for inverse transform
    for col in categorical_cols:
        if col in X.columns:
            le = LabelEncoder()
            X[col] = le.fit_transform(X[col])
            label_encoders[col] = le

    # Ensure y is encoded as integers
    label_encoder_y = LabelEncoder()
    y = label_encoder_y.fit_transform(y)
```

```
    # Split the data into train and test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)

    # Scaling Data
    scaler = MinMaxScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Define the XGBClassifier model
    xgb_model = XGBClassifier(random_state=0)

    # Define hyperparameters for tuning
    param_grid = {
        'n_estimators': [50, 100, 150],
        'max_depth': [3, 5, 7],
        'learning_rate': [0.01, 0.1, 0.2],
        'subsample': [0.8, 1.0],
        'colsample_bytree': [0.8, 1.0],
        'gamma': [0, 1, 2]
    }

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

    # Get the best model and parameters
    best_xgb_model = grid_search.best_estimator_
    best_params = grid_search.best_params_

    # Print the best hyperparameters
    print('Best Hyperparameters:')
    print(best_params)

    # Train the model on the full training set
    best_xgb_model.fit(X_train, y_train)

    # Evaluate the model on the test set
    y_pred = best_xgb_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'\nAccuracy on Test Set: {accuracy:.2f}')

    return best_xgb_model, best_params
# Example usage
best_model, best_params = train_xgb_classifier(data, 'target',␣
 ↪categorical_cols, numeric_cols)
```

```
Best Hyperparameters:
{'colsample_bytree': 0.8, 'gamma': 1, 'learning_rate': 0.2, 'max_depth': 3,
'n_estimators': 50, 'subsample': 0.8}

Accuracy on Test Set: 0.87
```

Additional Code

In this section, we will deal with the code that leads to the Heart Disease Prediction. I have performed some methods of Exploratory Data Analysis, Handled missing Values, and used models such as Logistic Regression, Random Forest, and XGBoost Algorithm for the prediction.

I have mentioned the accuracy of each model as well. The following table below will summarize that.

Model	Accuracy
Logistic Regression Model	0.85
Random Forest Model	0.85
XGBoost Model	0.87

Table 4.2: Accuracy of Different Models

I have done binary classification for this data set and although multi-classification is possible as well, it was taking more computational time and the accuracy had decreased to 0.65.

If the reader wishes to access the Python notebook, it can be accessed through the Github repository here.

Below is the code for the data that the patient will input. This will then be fed into the model. I will also show a sample output indicating the result.

```
def get_user_input():
    user_input = {}

    user_input['age'] = int(input("Enter age: "))
    print(f"Enter age: {user_input['age']}")

    user_input['sex'] = int(input("Enter sex (0 = female, 1 = male): "))
    print(f"Enter sex (0 = female, 1 = male): {user_input['sex']}")

    user_input['cp'] = int(input(
        "Enter chest pain type (choose the corresponding number):\n"
        "  0: asymptomatic\n"
        "  1: atypical angina\n"
        "  2: non-anginal pain\n"
        "  3: typical angina\n"
        "Your choice: "
    ))
    print(f"Enter chest pain type: {user_input['cp']}")

    user_input['trestbps'] = int(input("Enter resting blood pressure: "))
```

```
    print(f"Enter resting blood pressure: {user_input['trestbps']}")

    user_input['chol'] = int(input("Enter serum cholesterol in mg/dl: "))
    print(f"Enter serum cholesterol in mg/dl: {user_input['chol']}")

    user_input['fbs'] = int(input("Enter fasting blood sugar > 120 mg/dl (0 = no, 1 = yes): "))
    print(f"Enter fasting blood sugar > 120 mg/dl (0 = no, 1 = yes): {user_input['fbs']}")

    user_input['restecg'] = int(input(
        "Enter resting electrocardiographic results (choose the corresponding number):\n"
        "  0: lv hypertrophy\n"
        "  1: normal\n"
        "  2: st-t abnormality\n"
        "Your choice: "
    ))
    print(f"Enter resting electrocardiographic results: {user_input['restecg']}")

    user_input['thalch'] = int(input("Enter maximum heart rate achieved: "))
    print(f"Enter maximum heart rate achieved: {user_input['thalch']}")

    user_input['exang'] = int(input("Enter exercise induced angina (0 = no, 1 = yes): "))
    print(f"Enter exercise induced angina (0 = no, 1 = yes): {user_input['exang']}")

    user_input['oldpeak'] = float(input("Enter ST depression induced by exercise relative to rest: "))
    print(f"Enter ST depression induced by exercise relative to rest: {user_input['oldpeak']}")

    user_input['slope'] = int(input(
        "Enter the slope of the peak exercise ST segment (choose the corresponding number):\n"
        "  0: downsloping\n"
        "  1: flat\n"
        "  2: upsloping\n"
        "Your choice: "
    ))
    print(f"Enter the slope of the peak exercise ST segment: {user_input['slope']}")

    user_input['ca'] = int(input("Enter number of major vessels (0-3) colored by fluoroscopy: "))
    print(f"Enter number of major vessels (0-3) colored by fluoroscopy: {user_input['ca']}")

    user_input['thal'] = int(input(
```

```
60          "Enter thalassemia (choose the corresponding number):\n
                "
61          "  0: fixed defect\n"
62          "  1: normal\n"
63          "  2: reversable defect\n"
64          "Your choice: "
65      ))
66      print(f"Enter thalassemia: {user_input['thal']}")
67
68      return user_input
69
70  user_input = get_user_input()
71
72  def predict_heart_disease(user_input):
73      # user_input is a dictionary with the user's feature values
74      user_df = pd.DataFrame([user_input])
75
76      # Generate polynomial and interaction features
77      user_poly = poly.transform(user_df)
78
79      # Normalize features
80      user_scaled = scaler.transform(user_poly)
81
82      # Predict using the trained model
83      prediction = logreg.predict(user_scaled)
84      prediction_proba = logreg.predict_proba(user_scaled)
85
86      return prediction[0], prediction_proba[0]
87
88
89  try:
90      prediction, prediction_proba = predict_heart_disease(
            user_input)
91      print(f'Prediction: {"\n THE PATIENT HAS HEART DISEASE !!!"
            if prediction == 1 else "THE PATIENT DOES NOT HAVE
            HEART DISEASE !!!"}')
92  except ValueError as e:
93      print(f'Error: {e}')
```

Below are some images of what the output looks like.

```
Enter age: 60
Enter sex (0 = female, 1 = male): 1
Enter chest pain type: 3
Enter resting blood pressure: 150
Enter serum cholesterol in mg/dl: 240
Enter fasting blood sugar > 120 mg/dl (0 = no, 1 = yes): 1
Enter resting electrocardiographic results: 2
Enter maximum heart rate achieved: 160
Enter exercise induced angina (0 = no, 1 = yes): 1
Enter ST depression induced by exercise relative to rest: 2.6
Enter the slope of the peak exercise ST segment: 2
Enter number of major vessels (0-3) colored by fluoroscopy: 3
Enter thalassemia: 0
Prediction:
 THE PATIENT HAS HEART DISEASE !!!
```

Fig. 4.5: Sample Output 1

```
Enter age: 45
Enter sex (0 = female, 1 = male): 0
Enter chest pain type: 2
Enter resting blood pressure: 130
Enter serum cholesterol in mg/dl: 233
Enter fasting blood sugar > 120 mg/dl (0 = no, 1 = yes): 1
Enter resting electrocardiographic results: 1
Enter maximum heart rate achieved: 150
Enter exercise induced angina (0 = no, 1 = yes): 0
Enter ST depression induced by exercise relative to rest: 1.5
Enter the slope of the peak exercise ST segment: 1
Enter number of major vessels (0-3) colored by fluoroscopy: 2
Enter thalassemia: 2
Prediction: THE PATIENT DOES NOT HAVE HEART DISEASE !!!
```

Fig. 4.6: Sample Output 2

Hence the code can predict with good accuracy whether the patient has Heart Disease or not. The model I have used for this output is XGBoost as it has the highest accuracy among all.

Thank you for viewing this report. Best wishes.