

## Major Project Report

Predicting Chronic Heart Disease (CHD) Risk Using Machine Learning Algorithms



Semester-2

Four Year Undergraduate Program-Design Your Degree

Submitted to

University of Jammu, Jammu

**By**

Diya Rani	DYD-23-05
Kashvi Vaid	DYD-23-09
Narayan Choudhary	DYD-23-12
Pawandeep Singh	DYD-23-15

UNDER THE MENTORSHIP OF

Prof. K.S. Charak	Dr. Jatinder Manhas
Dr. Sandeep Arya	Dr. Sunil Kumar

On 10 September, 2024

---

## Certificate

The report titled “Predicting Chronic Heart Disease (CHD) Risk Using Machine Learning Algorithms” was completed by the group members: Diya, Kashvi, Narayan, and Pawan. This project was a key component of Semester 2 of their academic program, conducted under the supervision and guidance of Prof. KS Charak, Dr. Jatinder Manhas, Dr. Sandeep Arya, and Dr. Sunil Kumar in partial fulfillment of the requirements for the Design Your Degree, Four Year Undergraduate Program at the University of Jammu, Jammu and Kashmir. This project report is original and has not been submitted elsewhere for any academic recognition.

### Signature of Students:

- 1) Diya Rani
- 2) Kashvi Vaid
- 3) Narayan Choudhary
- 4) Pawandeep Singh

### Signature of Mentors:

- 1) Prof. K.S. Charak
- 2) Dr. Jatinder Manhas
- 3) Dr. Sandeep Arya
- 4) Dr. Sunil Kumar

Prof. Alka Sharma

Director SIIDEC, University of Jammu

---

## **Acknowledgement**

We would like to express our sincere gratitude to all those who have contributed to the successful completion of this project on “Predicting Chronic Heart Disease (CHD) Risk Using Machine Learning Algorithms”.

First and foremost, we are deeply grateful to Padam Shri Prof. Dinesh Singh, Vice Chairman of Jammu and Kashmir Higher Education, for his invaluable guidance and support throughout this endeavor. His insights and vision have been instrumental in shaping the direction of this project. We would also like to extend our heartfelt thanks to Prof. Umesh Rai, Vice Chancellor of Jammu University, for providing the opportunity and resources to pursue this work, and to Prof. Alka Sharma, Director of SIIEDC, for her continued encouragement and mentorship, which greatly enhanced the quality of this research.

A special mention goes to Prof. KS Charak, Dr. Jatinder Manhas, Dr. Sandeep Arya and Dr. Sunil Kumar for their technical expertise, constructive feedback, and support in refining the methodology and ensuring the accuracy of the results. In particular, we wish to express our deep appreciation to Dr. Jatinder Manhas Sir for his invaluable guidance throughout the study, which provided clarity and direction during the course of this research.

Finally, we would like to thank our colleagues and fellow researchers for their collaboration and insightful discussions, which were instrumental in overcoming the challenges encountered during this project. This project would not have been possible without the collective efforts and support of all these esteemed individuals, and we are profoundly grateful for their contributions.

---

## **Abstract**

The prediction of chronic heart disease (CHD) is a critical area of research in healthcare, where early detection can significantly enhance patient outcomes. This study leverages machine learning algorithms to predict the likelihood of individuals developing chronic heart disease within the next ten years, based on a comprehensive dataset collected via sensors. The dataset includes key variables such as gender, age, smoking status, daily cigarette consumption, blood pressure medication usage, history of stroke, hypertension, diabetes, cholesterol levels, systolic and diastolic blood pressure, body mass index (BMI), heart rate, and glucose levels. Our approach involves preprocessing the data to address missing values and normalize continuous variables, followed by feature selection to identify the most significant predictors of CHD. Various machine learning models, including logistic regression, decision trees, K- Nearest Neighbor and random forests, are then trained and validated to assess their predictive performance. The outcome variable, Ten Year CHD, indicates whether an individual is likely to develop CHD within a decade. The study's findings aim to contribute to the field of preventive cardiology by offering a robust predictive model that can aid in the early identification of high-risk individuals, thereby enabling timely intervention and reducing the incidence of chronic heart disease.

---

## Contents

S. No.	Chapter	Page No.
1	Introduction	1 - 12
2	Literature Survey	13 - 17
3	Methodology	18 - 58
4	Data Collection	59 – 61
5	Experimentation	62 - 110
6	Conclusion, Recommendations and Future scope	111 - 113
7	References	114 - 115

---

# Chapter 1: Introduction

## 1.1. Predicting Chronic Heart Disease (CHD) Risk Using Machine Learning

Heart disease is a catch-all phrase for a variety of conditions that affect the heart's structure and how it works. Heart disease is the leading cause of death in the United States. Coronary heart disease is a type of heart disease where the arteries of the heart cannot deliver enough oxygen-rich blood to the heart. It is also sometimes called coronary artery disease or ischemic heart disease. About 20.5 million U.S. adults have coronary artery disease, making it the most common type of heart disease in the United States, according to the Centers for Disease Control and Prevention [1].

Chronic Heart Disease (CHD) is a leading cause of mortality globally, and early prediction can be crucial in preventing its onset and managing patient outcomes. In this project, we utilized machine learning models to predict the likelihood of a person developing CHD within the next ten years based on their current health parameters. The goal was to develop a predictive model that could assist healthcare professionals in identifying high-risk individuals and intervening before the disease progresses.

## 1.2. Machine learning

### A. What Is Machine Learning?

Machine Learning (ML) is a branch of artificial intelligence (AI) that focuses on developing algorithms and models that enable computers to learn from and make decisions based on data. Unlike traditional programming, where a programmer explicitly writes code to perform a specific task, machine learning involves training a model using data so that it can automatically learn patterns, make predictions, or take actions without being explicitly programmed for each individual task.

Here is a slightly more general definition:

*[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.*

—Arthur Samuel, 1959

And a more engineering-oriented one:

*A computer program is said to learn from experience  $E$  with respect to some task  $T$  and some performance measure  $P$ , if its performance on  $T$ , as measured by  $P$ , improves with experience  $E$ .*

—Tom Mitchell, 1997

For example, your spam filter is a Machine Learning program that can learn to flag spam given examples of spam emails (e.g., flagged by users) and examples of regular (non-spam, also called “ham”) emails. The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample). In this case, the task T is to flag spam for new emails, the experience E is the training data, and the performance measure P needs to be defined; for example, you can use the ratio of correctly classified emails. This particular performance measure is called accuracy and it is often used in classification tasks.

## B. Why Use Machine Learning?

When considering the implementation of a spam filter using traditional programming techniques, the process typically involves the following steps:

- 1) **Pattern Identification:** Initially, you would analyze spam emails to identify common characteristics. For example, you might notice that certain words or phrases, such as “4U,” “credit card,” “free,” and “amazing,” frequently appear in the subject lines. Additional patterns might be observed in the sender’s name, the body of the email, and other elements.
- 2) **Rule Creation:** Based on these observations, you would develop a detection algorithm that flags emails containing a certain number of these patterns as spam
- 3) **Iteration and Testing:** The program would then undergo testing and further refinement through repeated iterations of steps 1 and 2 until the spam detection is sufficiently accurate.

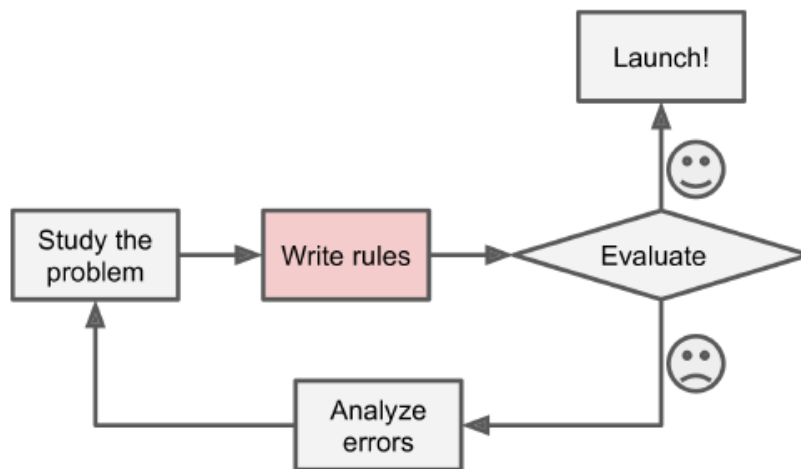


Figure 2-1. The traditional approach [2]

However, because spam filtering is a complex problem, this traditional approach often results in a program with a long and intricate list of rules, making it difficult to maintain. In contrast, a spam filter based on Machine Learning (ML) techniques can automatically learn which words and phrases are strong predictors of spam by analyzing patterns in spam examples compared to non-spam (ham) examples. This results in a shorter, more maintainable, and likely more accurate program.

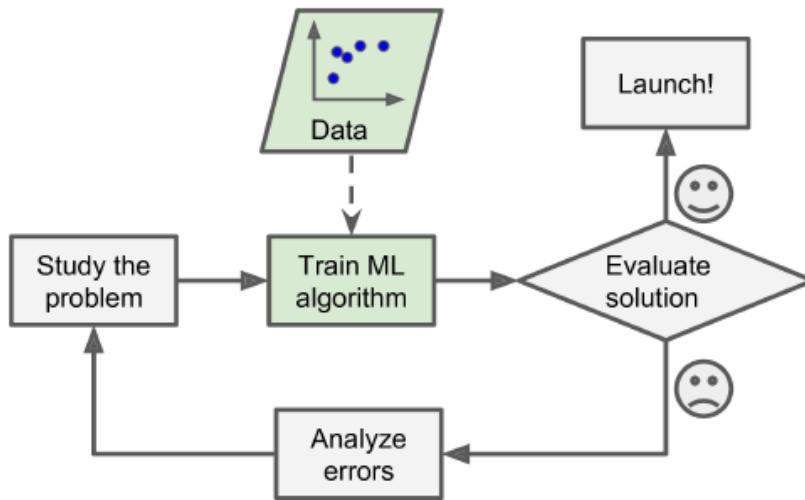


Figure 2-2. The machine learning approach [3]

Moreover, traditional programming techniques require constant updates to address evolving spam tactics. For instance, if spammers change "4U" to "For U," the code must be updated to detect the new pattern. In contrast, an ML-based spam filter can automatically detect that "For U" has become more common in spam flagged by users and adjust accordingly, without requiring manual intervention.

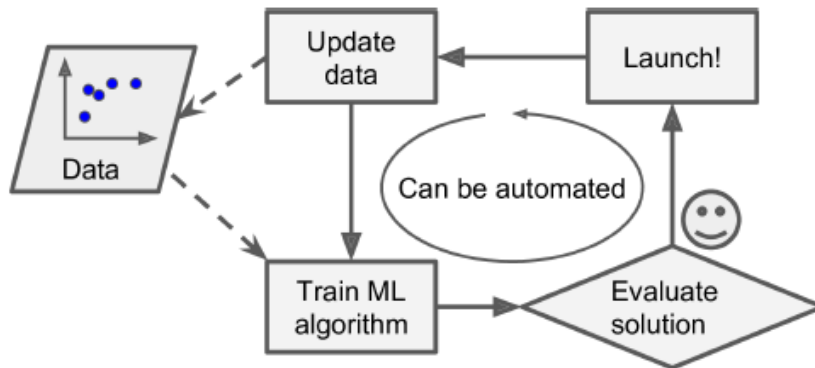


Figure 2-3. Automatically adapting to change [4]

Machine Learning is also particularly effective for problems that are too complex for traditional approaches or lack a known algorithmic solution. For example, in speech recognition, trying to distinguish between spoken words like "one" and "two" using hardcoded rules based on pitch or sound intensity is impractical. However, an ML algorithm can learn from a large dataset of spoken examples, enabling it to generalize across different accents, languages, and noisy environments.



Additionally, ML algorithms can be inspected to reveal what they have learned. For example, after training a spam filter on enough data, it can be examined to identify the words and word combinations it considers the best predictors of spam. This process can uncover unexpected correlations or trends, leading to a deeper understanding of the problem. When applied to large datasets, ML techniques can also facilitate data mining, revealing patterns that might not be immediately apparent.

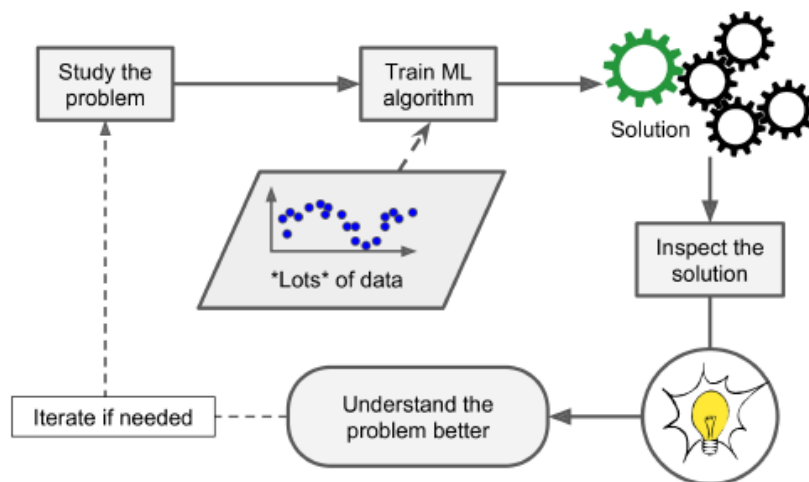


Figure 2-4. Machine learning can help humans learn [5]

### Summary of Machine Learning Advantages:

- **Simplification of Complex Rules:** ML algorithms can often simplify code and enhance performance in cases where traditional solutions require extensive hand-tuning or long lists of rules.
- **Solving Complex Problems:** ML can provide solutions for complex problems where traditional methods fall short.
- **Adaptability:** ML systems can adapt to changing environments and new data, ensuring continuous effectiveness.
- **Insight Generation:** ML can uncover valuable insights from large and complex datasets, aiding in the understanding and resolution of intricate problems.

### C. Types of Machine Learning Systems

There are so many different types of Machine Learning systems that it is useful to classify them in broad categories based on:

- Whether or not they are trained with human supervision (supervised, unsupervised, semisupervised, and Reinforcement Learning)
- Whether or not they can learn incrementally on the fly (online versus batch learning)

- Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model, much like scientists do (instance-based versus model-based learning)

## 1) Supervised/Unsupervised Learning

Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semisupervised learning, and Reinforcement Learning.

### Supervised learning

In *supervised learning*, the training data you feed to the algorithm includes the desired solutions, called *labels*

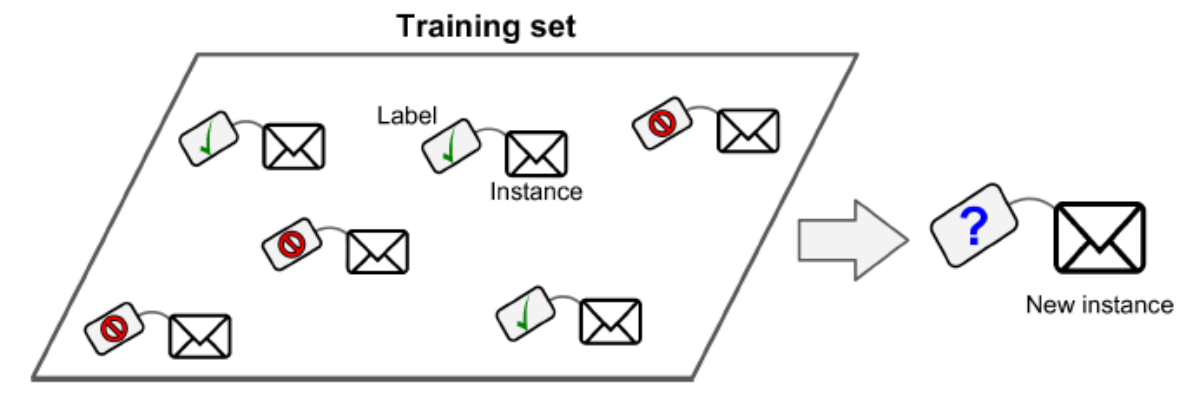


Figure 2-5. A labeled training set for supervised learning e.g., spam classification [6]

A typical supervised learning task is *classification*. The spam filter is a good example of this: it is trained with many examples' emails along with their *class* (spam or ham), and it must learn how to classify new emails. Another typical task is to predict a *target* numeric value, such as the price of a car, given a set of *features* (mileage, age, brand, etc.) called *predictors*. This sort of task is called *regression* (Figure 1-6).<sup>1</sup> To train the system, you need to give it many examples of cars, including both their predictors and their labels (i.e., their prices).



Figure 2-6. Regression [7]

Note that some regression algorithms can be used for classification as well, and vice versa. For example, *Logistic Regression* is commonly used for classification, as it can output a value that corresponds to the probability of belonging to a given class (e.g., 20% chance of being spam).

Here are some of the most important supervised learning algorithms which we used I our project:

- a. k-Nearest Neighbors
- b. Logistic Regression
- c. Decision Trees
- d. Random Forests

## 2) Unsupervised learning

In *unsupervised learning*, as you might guess, the training data is unlabeled. The system tries to learn without a teacher.

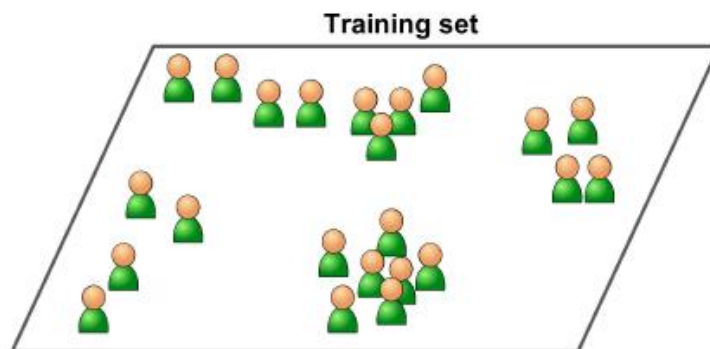


Figure 2-7. An unlabeled training set for supervised learning [8]

Here are some of the most important unsupervised learning algorithms

- Clustering
  - K-Means
  - DBSCAN
  - Hierarchical Cluster Analysis (HCA)
- Anomaly detection and novelty detection
  - One-class SVM
  - Isolation Forest
- Visualization and dimensionality reduction
  - Principal Component Analysis (PCA)
  - Kernel PCA
  - Locally-Linear Embedding (LLE)
  - t-distributed Stochastic Neighbor Embedding (t-SNE)
- Association rule learning
  - Apriori
  - Eclat

For example, say you have a lot of data about your blog's visitors. You may want to run a *clustering* algorithm to try to detect groups of similar visitors. At no point do you tell the algorithm which group a visitor belongs to: it finds those connections without your help. For example, it might notice that 40% of your visitors are males who love comic books and generally read your blog in the evening, while 20% are young sci-fi lovers who visit during the weekends, and so on. If you use a *hierarchical clustering* algorithm, it may also subdivide each group into smaller groups. This may help you target your posts for each group.

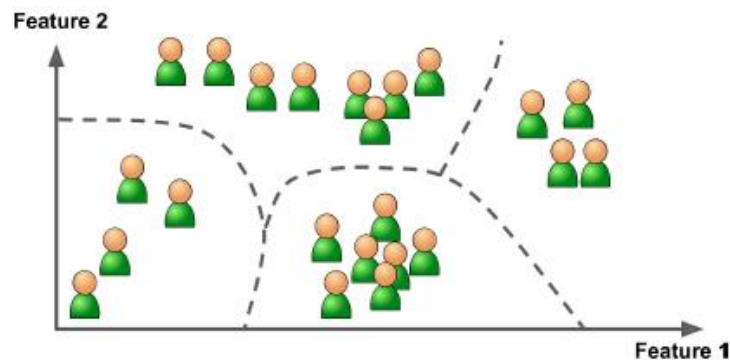


Figure 2-8. Clustering [9]

*Visualization* algorithms are also good examples of unsupervised learning algorithms: you feed them a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted. These algorithms try to preserve as much structure as they can (e.g., trying to keep separate clusters in the input space from overlapping in the visualization), so you can understand how the data is organized and perhaps identify unsuspected patterns.

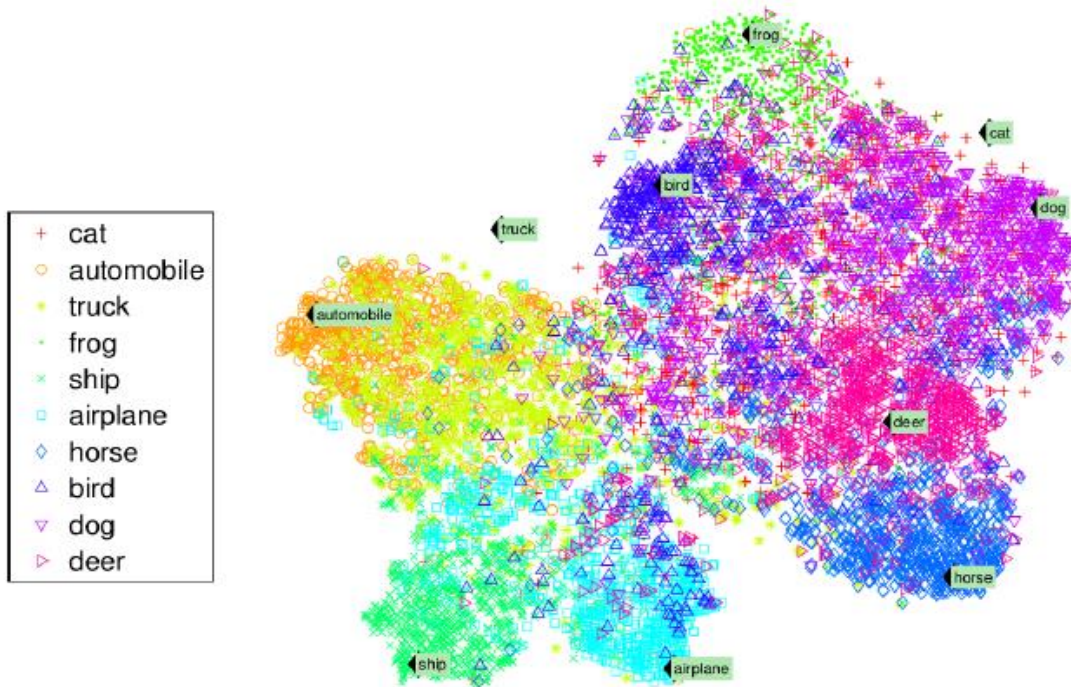


Figure 2-9. Example of a t-SNE visualization highlighting semantic clusters [10]

A related task is *dimensionality reduction*, in which the goal is to simplify the data without losing too much information. One way to do this is to merge several correlated features into one. For example, a car's mileage may be very correlated with its age, so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear. This is called *feature extraction*.

### 3) Semisupervised learning

Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. This is called *semisupervised learning*.

Some photo-hosting services, such as Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically recognizes that the same person A shows up in photos 1, 5, and 11, while another person B shows up in photos 2, 5, and 7. This is the unsupervised part of the algorithm (clustering). Now all the system needs is for you to tell it who these people are. Just one label per person,<sup>4</sup> and it is able to name everyone in every photo, which is useful for searching photos. Most semisupervised learning algorithms are combinations of unsupervised and supervised algorithms.

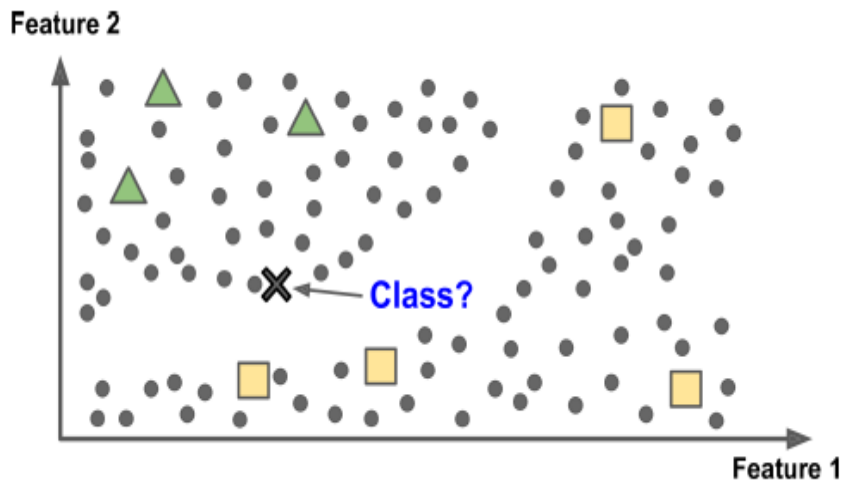


Figure 2-10. Semisupervised learning [11]

## D. Main Challenges of Machine Learning

In short, since your main task is to select a learning algorithm and train it on some data, the two things that can go wrong are “bad algorithm” and “bad data.” Let’s start with examples of bad data.

### 1) Insufficient Quantity of Training Data

For a toddler to learn what an apple is, all it takes is for you to point to an apple and say “apple” (possibly repeating this procedure a few times). Now the child is able to recognize apples in all sorts of colors and shapes. Genius. Machine Learning is not quite there yet; it takes a lot of data for most Machine Learning algorithms to work properly. Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples (unless you can reuse parts of an existing model).

### 2) Poor-Quality Data

Obviously, if your training data is full of errors, outliers, and noise (e.g., due to poor-quality measurements), it will make it harder for the system to detect the underlying patterns, so your system is less likely to perform well. It is often well worth the effort to spend time cleaning up your training data. The truth is, most data scientists spend a significant part of their time doing just that. For example:

- If some instances are clearly outliers, it may help to simply discard them or try to fix the errors manually.
- If some instances are missing a few features (e.g., 5% of your customers did not specify their age), you must decide whether you want to ignore this attribute altogether, ignore these instances, fill in the missing values (e.g., with the median age), or train one model with the feature and one model without it, and so on.

### 3) Irrelevant Features

As the saying goes: garbage in, garbage out. Your system will only be capable of learning if the training data contains enough relevant features and not too many irrelevant ones. A

critical part of the success of a Machine Learning project is coming up with a good set of features to train on. This process, called *feature engineering*, involves:

- *Feature selection*: selecting the most useful features to train on among existing features.
- *Feature extraction*: combining existing features to produce a more useful one (as we saw earlier, dimensionality reduction algorithms can help).
- Creating new features by gathering new data.

#### 4) **Overfitting the Training Data**

Say you are visiting a foreign country and the taxi driver rips you off. You might be tempted to say that *all* taxi drivers in that country are thieves. Overgeneralizing is something that we humans do all too often, and unfortunately machines can fall into the same trap if we are not careful. In Machine Learning this is called *overfitting*: it means that the model performs well on the training data, but it does not generalize well.

#### 5) **Underfitting the Training Data**

As you might guess, *underfitting* is the opposite of overfitting: it occurs when your model is too simple to learn the underlying structure of the data. For example, a linear model of life satisfaction is prone to underfit; reality is just more complex than the model, so its predictions are bound to be inaccurate, even on the training examples.

The main options to fix this problem are:

- Selecting a more powerful model, with more parameters
- Feeding better features to the learning algorithm (feature engineering)
- Reducing the constraints on the model (e.g., reducing the regularization hyperparameter)

### **E. Testing and Validating**

The only way to know how well a model will generalize to new cases is to actually try it out on new cases. One way to do that is to put your model in production and monitor how well it performs. This works well, but if your model is horribly bad, your users will complain—not the best idea. A better option is to split your data into two sets: the *training set* and the *test set*. As these names imply, you train your model using the training set, and you test it using the test set. The error rate on new cases is called the *generalization error* (or *out-of-sample error*), and by evaluating your model on the test set, you get an estimate of this error. This value tells you how well your model will perform on instances it has never seen before. If the training error is low (i.e., your model makes few mistakes on the training set) but the generalization error is high, it means that your model is overfitting the training data.

### **F. Problem Statement**

The objective of this project was to predict whether an individual would develop Chronic Heart Disease (CHD) within the next ten years based on their present health metrics. This task was formulated as a binary classification problem, where the model would output a probability or a binary prediction (0 or 1) indicating the likelihood of CHD occurrence.

## G. Objective

The primary objective of this project is to evaluate and compare the performance of various machine learning algorithms in predicting the risk of Chronic Heart Disease (CHD) over a ten-year period. By utilizing different models, the goal is to identify the most accurate and reliable algorithm for predicting CHD based on the given health parameters. This involves:

1. Understanding and implementing multiple machine learning algorithms.
2. Applying these algorithms to the CHD dataset to make predictions.
3. Evaluating the performance of each model using appropriate metrics such as accuracy, precision, recall, and F1 score.
4. Comparing the results of the models to determine which algorithm performs best in predicting CHD risk.

The insights gained from this comparison will help in selecting the most suitable model for CHD risk prediction, which could ultimately contribute to better early detection and preventive healthcare.

## H. Supervised Machine Learning

### 1) Introduction

Supervised machine learning is a type of machine learning in which a model is trained on a labeled dataset, meaning that the input data is paired with the correct output. The goal of supervised learning is to learn a mapping from inputs to outputs that can be used to predict the output for new, unseen inputs. This approach is widely used in various applications, including classification, regression, and prediction tasks.

#### a) Key Concepts

1. **Labeled Data:** In supervised learning, the dataset consists of input-output pairs, where the input is often referred to as features, and the output as labels. The model learns from these pairs to make predictions on new data.
2. **Training and Testing:** The dataset is typically split into two parts: a training set and a testing set. The model is trained on the training set and then evaluated on the testing set to assess its performance. This ensures that the model generalizes well to new data.
3. **Types of Algorithms:**
  - **Classification:** In classification tasks, the model predicts a discrete label from a set of categories. Examples include spam detection in emails and handwriting recognition.
  - **Regression:** Regression involves predicting a continuous value. Common applications include predicting house prices based on features like location and size.
4. **Evaluation Metrics:** The performance of supervised learning models is often evaluated using metrics such as accuracy, precision, recall, F1-score for classification, and mean squared error or R-squared for regression.



## 2) Applications of Supervised Learning

Supervised learning has numerous applications across various industries:

- a) **Healthcare:** Predicting the likelihood of diseases such as chronic heart disease based on patient data.
- b) **Finance:** Fraud detection by identifying unusual transactions that deviate from normal patterns.
- c) **Marketing:** Customer segmentation to target specific groups with personalized campaigns.

## 3) Challenges and Considerations

Despite its widespread use, supervised learning faces several challenges. One major issue is the need for large amounts of labeled data, which can be time-consuming and expensive to obtain. Additionally, the model's performance can be heavily influenced by the quality of the data and the choice of features.

Moreover, there is the risk of overfitting, where the model performs well on the training data but poorly on new, unseen data. Techniques such as cross-validation and regularization are often used to mitigate this issue.

In conclusion, supervised machine learning is a powerful tool for making predictions and decisions based on data. Its effectiveness depends on the quality of the data, the choice of model, and the careful tuning of parameters to ensure the model generalizes well to new data.

## Chapter 2: Literature Review

### 2.1. HISTORY OF MACHINE LEARNING ALGORITHMS:

#### 1) Early Beginnings (1950s-1960s):

- **1950:** **Alan Turing**, a key figure in computing, suggests a test to check if a machine can think like a human. This idea sparks interest in creating intelligent machines.
- **1957:** **Frank Rosenblatt** created the **Perceptron**, an early kind of computer program that can learn to make simple decisions by itself, like sorting things into two categories.

#### 2) Growing Interest (1960s-1970s)

- **1960s:** Researchers start using the term "machine learning" and develop early techniques to make computers smarter using rules and logic.
- **1970s:** The **ID3** algorithm is created, which helps computers make decisions based on a series of questions, like a flowchart.

#### 3) Setbacks (1970s-1980s)

- **1970s-1980s:** Interest in AI and machine learning drops because early systems didn't live up to their promises and computing power was still limited. This period is known as the "AI Winter."

#### 4) Revival and New Methods (1980s-1990s)

- **1986:** Geoffrey Hinton introduces a method called **backpropagation** that helps computers learn more effectively from their mistakes, making it possible to train more complex learning systems.
- **1989:** **Support Vector Machines (SVMs)** are developed. These are powerful tools for sorting data into different categories by finding the best line or boundary that separates them.

#### 5) Breakthroughs (2000s-2010s)

- **2006:** Geoffrey Hinton and his team show that deep learning, a method where computers learn through multiple layers of processing, can be very effective. This leads to big improvements in how machines understand complex data.
- **2012:** **AlexNet**, a deep learning system, wins a major competition by a large margin, proving how powerful these new techniques can be for recognizing images and driving wider use of deep learning.

## 6) Recent Developments (2010s-Present)

- **2014: Generative Adversarial Networks (GANs)** are introduced. These are special types of learning systems where two networks compete to create realistic images or data, leading to impressive results in generating new content.
- **2018:** The **Transformer** model is introduced. This approach revolutionizes how machines understand and generate human language, leading to advanced tools like chatbots and language translation systems.

### 2.2. History of Decision Tree:

The Decision Tree algorithm is rooted in the fields of statistics and decision theory. Its origins trace back to the 1960s and 1970s, with the development of algorithms like CHAID (Chi-squared Automatic Interaction Detector) and ID3 (Iterative Dichotomiser 3) by Ross Quinlan in 1986. ID3 was one of the first algorithms designed to generate a Decision Tree from a dataset by iteratively selecting the feature that provided the best split according to Information Gain.

Later developments included Quinlan's C4.5, an improvement over ID3, which could handle continuous attributes and missing values. Another significant advancement was the CART (Classification and Regression Trees) algorithm introduced by Breiman et al. in 1984, which offered a robust framework for both classification and regression tasks. These early innovations laid the foundation for modern Decision Trees and ensemble methods like Random Forests and Gradient Boosting.

Decision Trees gained popularity due to their interpretability, as the decisions made at each node can be easily understood and visualized, making them particularly useful in fields requiring transparent decision-making.

### 2.3. History of Logistic Regression Algorithm:

#### 1) Early Concepts (1950s-1960s)

- **1950s:** The idea of logistic regression starts with the need to predict the likelihood of an event occurring. Researchers are looking for ways to model binary outcomes (yes/no, success/failure) based on input data.

#### 2) Introduction of Logistic Function (1960s)

- **1960s: David Cox**, a statistician, develops the **logistic function**. This function is important because it can model probabilities (values between 0 and 1) and is used to predict outcomes like whether an email is spam or not.

### 3) Formalizing Logistic Regression (1970s-1980s)

- **1970s:** Logistic regression is formally introduced as a statistical technique. It uses the logistic function to model the probability of a certain class or event. Researchers use it in various fields, like medicine, to predict disease presence based on test results.

### 4) Growth in Popularity (1990s-2000s)

- **1990s:** Logistic regression becomes popular in various fields like marketing and social sciences for its simplicity and effectiveness in predicting binary outcomes.
- **2000s:** With the rise of computers and more data, logistic regression becomes a standard tool in data analysis and machine learning due to its interpretability and efficiency.

### 5) Modern Usage (2010s-Present)

- **2010s:** Logistic regression is widely used in machine learning for classification tasks. It remains popular because it's straightforward to implement, understand, and interpret.
- **Present:** It's used in various applications, from predicting customer churn to medical diagnoses, thanks to its ease of use and effectiveness.

## 2.4. History of KNN Algorithm

### 1. Early Concepts (1960s-1970s)

- **What Happened:** The idea of classifying data based on similarity starts to take shape. Early computer scientists and statisticians are exploring ways to make predictions based on the idea that similar items should be grouped together.

### 2. Introduction of K-Nearest Neighbors (KNN) (1970s)

- **What Happened:** The **K-Nearest Neighbors (KNN)** algorithm is formally introduced. It's a straightforward method where you classify a new data point based on the majority class of its nearest neighbors. For example, if you want to classify a new fruit, you look at the most similar fruits in your dataset and see which category they belong to.

### 3. Early Applications and Growth (1980s-1990s)

- **What Happened:** KNN begins to be applied in various fields like pattern recognition and data mining. Its simplicity and ease of understanding make it a popular choice for initial classification tasks. During this time, researchers start to explore different ways to optimize and improve the algorithm.

### 4. Integration with Machine Learning (2000s)

- **What Happened:** KNN becomes widely used in the machine learning community. With the rise of more powerful computers and larger datasets, KNN is used for a variety of

applications, including image recognition and recommendation systems. Researchers continue to refine the algorithm to handle larger and more complex datasets efficiently.

## **5. Modern Usage (2010s-Present)**

- **What Happened:** KNN remains a popular and useful algorithm in machine learning for tasks that involve classification and regression. It's appreciated for its simplicity and effectiveness, especially in situations where the relationship between data points is not easily captured by more complex models. Improvements and variations, like weighted KNN, are developed to enhance performance.

## **2.5. History of Random Forest Algorithm:**

The Random Forest algorithm is a relatively recent development in the field of machine learning, but its roots go back to earlier ideas in decision tree learning.

### **1. Decision Trees:**

The concept of decision trees, which are the building blocks of Random Forest, was first introduced in the 1960s and 1970s. A decision tree is a simple yet powerful tool that makes decisions by splitting data into different branches based on certain conditions. Each branch represents a possible decision path, leading to a final outcome. While decision trees are easy to understand and use, they have a tendency to overfit, meaning they might perform very well on training data but poorly on new, unseen data.

### **2. Ensemble Learning:**

To overcome the limitations of individual decision trees, researchers began exploring ensemble methods in the 1980s and 1990s. Ensemble learning involves combining multiple models to improve overall performance. The idea is that a group of models, when combined, can make more accurate predictions than any single model on its own. This concept laid the groundwork for the development of the Random Forest algorithm.

### **3. Birth of Random Forest:**

The Random Forest algorithm was developed by Leo Breiman, a statistician at the University of California, Berkeley, in 2001. Breiman built on the idea of "bagging" (short for bootstrap aggregating), which involves training multiple versions of a model on different subsets of the data and then averaging their predictions. Random Forest took this idea further by adding randomness to the process: each decision tree in the forest is trained on a random subset of the data and a random subset of the features. This randomness helps to create a diverse set of trees, reducing the risk of overfitting and improving the model's generalization to new data.

### **4. Impact and Adoption:**

Since its introduction, Random Forest has become one of the most popular machine learning algorithms due to its high accuracy, robustness, and ability to handle large datasets with many features. It is widely used in various fields, including healthcare, finance, and marketing, for

tasks such as classification, regression, and feature selection. Its ability to provide feature importance scores has also made it valuable for understanding which factors are most important in making predictions

## Chapter 3: Methodology

In this section, we detail the machine learning algorithms employed to predict chronic heart disease. The models chosen include Logistic Regression, k-Nearest Neighbors (KNN), Random Forest, and Decision Tree. These algorithms were selected due to their distinct approaches to classification, which allowed us to compare their effectiveness and robustness in handling the dataset.

### 3.1. Decision Tree Classifier for Predicting Coronary Heart Disease (CHD)

#### 1. Introduction:

In this project, we utilized a **Decision Tree** classifier to predict the likelihood of developing CHD within the next ten years. The Decision Tree algorithm is a widely used machine learning technique, appreciated for its simplicity, transparency, and ability to handle both categorical and numerical data. By leveraging this algorithm, we aim to build a model that can effectively predict CHD risk based on a variety of health-related features. The dataset employed for this project is comprehensive, encompassing a range of health indicators that are closely associated with cardiovascular health. These indicators include but are not limited to, age, cholesterol levels, blood pressure, and body mass index (BMI), among others. The dataset, collected over time, provides a rich source of information that enables the model to learn and identify patterns that could signal an increased risk of CHD. This report provides a detailed documentation of the entire process involved in implementing and evaluating the Decision Tree algorithm for CHD prediction. We begin by introducing the theoretical foundation of the Decision Tree model, followed by a thorough explanation of the steps taken to preprocess the data, select relevant features, and train the model. The report also presents the Python code used in this project, offering insights into how the model was developed and tuned for optimal performance. In the latter sections of the report, we analyse the results obtained from the model, including key performance metrics such as accuracy, precision, recall, and the F1-score. These metrics provide a comprehensive evaluation of the model's effectiveness in predicting CHD. We also discuss the strengths and limitations of the Decision Tree classifier in this specific context, highlighting areas where the model performed well and areas that may require further refinement.

Through this project, we aim to demonstrate the potential of machine learning, specifically the Decision Tree algorithm, in predicting CHD risk. The findings could contribute to the broader efforts in the medical community to harness the power of data-driven approaches for improving cardiovascular health outcomes.

#### 2. History and Development of the Decision Tree Algorithm

The Decision Tree algorithm has a rich history rooted in the fields of statistics, machine learning, and decision theory. Its development spans several decades, during which it evolved

from a simple concept into a sophisticated tool widely used in various domains for both classification and regression tasks.

### **1) Early Beginnings: 1960s and 1970**

The concept of decision trees can be traced back to the early 1960s, emerging from the need to create models that could simulate human decision-making processes. During this period, the field of statistics was heavily focused on finding efficient methods to segment data, classify observations, and make predictions. The earliest decision tree models were simplistic and primarily used in experimental settings. One of the pioneering algorithms from this era was CHAID (Chi-squared Automatic Interaction Detector), developed in the late 1960s by Gordon V. Kass. CHAID was designed to automatically select the best predictors and their interactions by using chi-square statistics to split the data at each node. It was one of the first algorithms to introduce the concept of statistically driven splitting, which became a fundamental principle in the development of more advanced decision tree algorithms.

### **2) ID3 and Ross Quinlan's Contributions: 1980s**

The 1980s marked a significant period in the evolution of decision tree algorithms, primarily due to the work of Ross Quinlan, a researcher in machine learning and artificial intelligence. In 1986, Quinlan introduced the ID3 (Iterative Dichotomiser 3) algorithm, which became one of the first widely recognized methods for constructing decision trees. ID3 was designed to create a decision tree from a dataset by iteratively selecting the feature that provided the best split according to the concept of Information Gain. Information Gain, derived from the entropy of the dataset, measures how well a particular feature separates the data into distinct classes. ID3 was innovative because it applied this statistical concept to automatically build a decision tree that could classify data with minimal human intervention. However, ID3 had some limitations, particularly its inability to handle continuous attributes or missing values effectively. These limitations led Quinlan to further develop the algorithm.

### **3) Advancements: C4.5 and Beyond**

In response to the limitations of ID3, Ross Quinlan introduced an improved version called C4.5 in 1993. C4.5 addressed several shortcomings of its predecessor by incorporating the ability to handle continuous attributes through the use of thresholds that split data at certain points. It also introduced methods to manage missing values, prune the tree to avoid overfitting, and apply a post-pruning process to improve the tree's generalizability. C4.5 quickly became one of the most popular and widely used decision tree algorithms, solidifying Quinlan's place in the history of machine learning. The algorithm's ability to handle real-world datasets with varying data types and missing values made it a versatile tool in the field. The C4.5 algorithm eventually led to the development of C5.0, a commercial version that further improved performance and efficiency, and became the foundation for many modern decision tree implementations.



#### **4) The Emergence of CART: 1980s**

Another significant milestone in the development of decision trees was the introduction of the CART (Classification and Regression Trees) algorithm by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone in 1984. Unlike ID3 and C4.5, which were primarily focused on classification, CART provided a robust framework for both classification and regression tasks. CART introduced the concept of Gini impurity as a criterion for splitting nodes, as opposed to the Information Gain used by ID3 and C4.5. Gini impurity measures the likelihood of a randomly chosen element being incorrectly classified, which made CART particularly effective for certain types of data. Additionally, CART was designed to create binary trees, where each internal node had exactly two branches, simplifying the tree structure and improving computational efficiency. The introduction of CART also brought about the notion of tree pruning to prevent overfitting, a common issue where a model becomes too complex and performs poorly on unseen data. Pruning involves removing branches of the tree that have little importance, thereby improving the model's ability to generalize to new data.

#### **5) Modern Evolution and Ensemble Methods**

The foundational work on decision trees by researchers like Quinlan and Breiman set the stage for more advanced machine learning techniques. Decision trees, despite their simplicity and interpretability, had limitations, particularly in terms of variance and bias. To overcome these limitations, researchers developed ensemble methods, which combine multiple decision trees to create more robust models. One of the most notable advancements in this area is the Random Forest algorithm, introduced by Leo Breiman in 2001. Random Forests build multiple decision trees using different subsets of the data and features, and then aggregate their predictions to improve accuracy and reduce overfitting. Another key development is Gradient Boosting, which sequentially builds decision trees where each tree corrects the errors made by the previous ones. These ensemble methods have significantly enhanced the performance and applicability of decision trees in modern machine learning tasks.

### **3. About the Decision Tree Algorithm**

The Decision Tree algorithm is one of the most widely used supervised learning techniques in machine learning and data science. It is capable of performing both classification and regression tasks, making it a versatile tool for various predictive modelling challenges. The structure of a Decision Tree resembles a flowchart, where each internal node represents a decision point based on a specific feature, each branch corresponds to the outcome of that decision, and each leaf node signifies a final prediction—either a class label in classification problems or a continuous value in regression tasks.

## 1) Structure and Functionality

At its core, a Decision Tree operates by recursively splitting the dataset into subsets based on the value of an attribute (feature). This splitting continues until the subsets belong to a single class (in classification) or have a homogenous value (in regression). The goal of each split is to partition the data in such a way that each resulting subset is as pure as possible, meaning that the data points within a subset are as similar as possible in terms of the target variable.

- a. **Root Node:** The root of the tree represents the entire dataset. The first split, made at this level, is based on the feature that provides the best separation according to a chosen metric, such as Information Gain, Gini impurity, or variance reduction.
- b. **Internal Nodes:** Each internal node represents a decision point where the data is split further based on the value of one of the features. The selection of features at each node is crucial for the performance of the tree, as it determines the accuracy and efficiency of the model.
- c. **Branches:** The branches of the tree represent the possible outcomes of the decisions made at each node. Each branch leads to another node or to a leaf, depending on whether further splitting is required.
- d. **Leaf Nodes:** The leaves of the tree represent the final output of the model. In a classification tree, the leaves correspond to the predicted class labels, while in a regression tree, they represent the predicted continuous values.

## 2) Key Characteristics

- a. **Simplicity and Interpretability:** One of the most compelling features of Decision Trees is their simplicity and ease of interpretation. Unlike many other machine learning models, Decision Trees can be visualized graphically, making it straightforward to understand the logic behind the predictions. Each path from the root to a leaf can be interpreted as a set of conditions leading to a decision, which is especially useful in fields that require transparency, such as healthcare and finance.
- b. **Non-linearity:** Decision Trees are capable of capturing complex, non-linear relationships between features and the target variable. Unlike linear models that assume a straight-line relationship between input features and the target, Decision Trees can model intricate patterns by recursively splitting the data based on different features at different levels of the tree.
- c. **No Assumptions About Data:** Decision Trees do not require the data to follow any specific distribution. This makes them highly adaptable and suitable for a wide variety of datasets, including those with complex or irregular structures.

### 3) Advantages

- a. Decision Trees are easy to interpret and explain, as they provide a clear and visual representation of the decision-making process.
- b. They can handle both numerical and categorical data, making them versatile for different types of datasets.
- c. They require little data preparation, such as normalization or scaling, allowing for a straightforward implementation.

### 4) Disadvantages

- a. Decision Trees can be prone to overfitting, especially when they grow too complex. Overfitting occurs when the model becomes too tailored to the training data, reducing its ability to generalize to new, unseen data.
- b. They can be sensitive to small changes in the data, leading to instability in predictions. Slight variations in the data can result in a completely different tree being generated.[12]

## 4. Mathematical Model

The Decision Tree algorithm's power lies in its ability to select the optimal features to split the data at each node, ensuring that the dataset is partitioned in a way that maximizes the model's predictive accuracy. This selection process is grounded in well-established mathematical principles that guide the tree-building process. Here, we will explore the key concepts involved: Entropy, Information Gain, and Gini Impurity.

### A. Entropy and Information Gain

Entropy is a fundamental concept in information theory, introduced by Claude Shannon. In the context of Decision Trees, entropy is used as a measure of impurity or uncertainty in a dataset. It quantifies the level of disorder or randomness in the target variable's distribution. The more homogeneous a dataset is, the lower its entropy, and conversely, the more mixed the dataset, the higher its entropy.

#### a. Mathematically:

$$Entropy(S) = - \sum_{i=1}^c p_i \log_2(p_i)$$

Where:

- $(p_i)$  represents the proportion of elements in class  $(i)$  relative to the total number of elements in the dataset  $(S)$ .
- $(c)$  is the number of distinct classes in the dataset.
- $(\log_2)$  is the logarithm base 2, which ensures that entropy is measured in bits.

For example, if we have a binary classification problem where the dataset is perfectly balanced (i.e., 50% of the instances belong to one class and 50% to the other), the entropy would be 1, indicating maximum uncertainty. On the other hand, if all instances belong to a single class, the entropy would be 0, indicating complete certainty and no disorder.

### **b. Information Gain:**

Information Gain (IG) is the metric used to select the best feature for splitting the data at each node in a Decision Tree. It measures the reduction in entropy achieved by partitioning the dataset based on a particular feature. The feature that results in the highest Information Gain is chosen for the split, as it most effectively separates the dataset into pure subsets.

Information Gain is calculated as:

$$\text{Information Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \times \text{Entropy}(S_v)$$

where  $S_v$  is the subset of  $S$  where attribute  $A$  has value  $v$ .

Where:

- $H(S)$  is the entropy of the original dataset ( $S$ ).
- $A$  is the attribute (feature) used for splitting.
- $\text{Values}(A)$  represents the set of all possible values for attribute ( $A$ ).
- $S_v$  is the subset of ( $S$ ) where attribute ( $A$ ) has value ( $v$ ).
- $\frac{|S_v|}{|S|}$  is the proportion of instances in ( $S$ ) that have the value ( $v$ ) for attribute ( $A$ ).
- $H(S_v)$  is the entropy of the subset ( $S_v$ ).

The attribute with the highest Information Gain is selected because it best reduces the uncertainty (entropy) in the dataset, leading to more distinct and pure child nodes.

## **B. Gini Impurity**

Gini Impurity is another popular criterion used to evaluate the quality of a split in Decision Trees. Unlike entropy, which measures disorder in terms of information theory, Gini Impurity measures the probability of incorrectly classifying a randomly chosen instance if it were randomly labeled according to the distribution of labels in the dataset. Lower Gini Impurity values indicate purer nodes.

The formula for Gini Impurity ( $G$ ) is:

$$Gini(S) = 1 - \sum_{i=1}^c (p_i)^2$$

Where:

- $(p_i)$  is the proportion of elements in class  $(i)$  in the dataset  $(S)$ .
- $(c)$  is the number of classes.

For example, in a binary classification problem, if all instances in a node belong to a single class, the Gini Impurity would be 0, indicating a pure node. If the instances are evenly split between the two classes, the Gini Impurity would be 0.5, indicating maximum impurity for a binary classification.

### **Decision-Making Process in a Decision Tree**

At each step in building a Decision Tree, the algorithm evaluates each feature's potential splits using one of the criteria mentioned above (Information Gain or Gini Impurity). The feature and corresponding split that result in the highest Information Gain (or lowest Gini Impurity) are chosen. The dataset is then divided according to this split, and the process is repeated recursively for each subset, creating a hierarchical structure of nodes and branches.

This recursive splitting continues until one of the following conditions is met:

- All instances in a node belong to a single class (pure node).
- There are no remaining features to split on.
- A predefined stopping criterion, such as maximum tree depth or minimum samples per node, is satisfied.

The result is a fully grown Decision Tree, where each path from the root node to a leaf node represents a unique decision rule [13].

## **5. How the Decision Tree Algorithm Works**

The Decision Tree algorithm is a popular and powerful method in machine learning, particularly valued for its simplicity, interpretability, and ability to handle both categorical and continuous data. The process of constructing a Decision Tree can be broken down into several key steps, which together define how the algorithm builds a model that can make predictions based on input data. Here's a detailed walkthrough of each step:

### **1) Initialization**

The process begins with the entire dataset, which contains a mixture of input features and corresponding target labels. At this stage, the algorithm treats the whole dataset as a single, undivided entity, often referred to as the root node of the tree. The goal is to split this dataset into smaller subsets in a way that maximizes the purity of each subset, meaning that each subset should ideally contain instances that belong to the same class.

## 2) Feature Selection

At each node in the Decision Tree, the algorithm needs to decide which feature to use to split the data. This decision is made by evaluating all available features and selecting the one that best separates the data into distinct classes.

To make this selection, the algorithm calculates a metric, such as Information Gain or Gini Impurity, for each feature.

- Information Gain measures how much uncertainty (entropy) is reduced by splitting the data based on that feature. The feature with the highest Information Gain is chosen because it most effectively reduces the entropy, leading to purer subsets.
- Gini Impurity is another measure used to evaluate the quality of a split. It quantifies the likelihood of incorrect classification if a random instance from the subset was labeled randomly according to the class distribution.

The feature that results in the highest Information Gain (or lowest Gini Impurity) is selected for the split.

## 3) Splitting

Once the best feature is selected, the dataset is divided into subsets based on the values of this feature. For example, if the chosen feature is "age" and the decision rule is "age > 50," the dataset will be split into two branches: one for instances where the condition holds true (age > 50) and one where it does not (age ≤ 50).

Each resulting subset represents a branch of the tree, with the original dataset's root node now leading to these branches. This step reduces the complexity of the data at each node, focusing on smaller, more homogeneous subsets.

## 4) Recursive Partitioning

The process of feature selection and splitting is repeated recursively for each branch or subset, creating sub-nodes within the tree. At each sub-node, the algorithm again evaluates all available features and selects the best one to split the data further. This recursive partitioning continues, creating an increasingly complex tree structure with multiple levels of nodes.

The recursive process continues until one of the following stopping criteria is met:

- Maximum Tree Depth: The tree reaches a predefined maximum number of levels, beyond which no further splits are allowed.
- Minimum Samples per Node: The number of samples in a node falls below a predefined threshold, preventing further splits.
- Pure Node: All samples in a node belong to the same class, meaning no further splitting is necessary. These criteria help prevent the tree from growing too complex and overfitting the training data.

## 5) Prediction

Once the Decision Tree is fully grown, it can be used to make predictions on new, unseen data. The prediction process involves following the decision rules from the root of the tree down to a leaf node.

For a given new data point, the model starts at the root node and evaluates the feature at that node. Depending on the value of the feature, the model follows the corresponding branch to the next node. This process continues, moving down the tree, until it reaches a leaf node, which contains the predicted class label or value. For instance, if the Decision Tree was trained to predict whether a person is at risk of CHD, a new patient's data would be input into the model, and the model would follow the decision rules down the tree until it arrives at a leaf node that predicts either "at risk" or "not at risk" based on the patient's health indicators.[14]

### 3.2. Introduction to K-Nearest Neighbors (KNN) Algorithm

The K-Nearest Neighbors (KNN) algorithm is a fundamental and versatile machine learning technique used for both classification and regression tasks. As one of the simplest yet effective algorithms in the field, KNN has found applications in various domains, from image recognition to recommendation systems.

At its core, KNN is based on the principle that similar data points tend to exist in close proximity to each other. This intuitive concept is leveraged to make predictions about new, unseen data points based on their similarity to known, labeled data points in a dataset.

The "K" in KNN refers to the number of nearest neighbors that the algorithm considers when making a prediction. This parameter is crucial as it directly influences the algorithm's behavior and performance. A smaller K value makes the model more sensitive to local patterns but potentially more susceptible to noise, while a larger K value can make the model more robust but may overlook nuanced patterns in the data.

One of KNN's distinctive features is its nature as a non-parametric and instance-based learning algorithm. Unlike many other machine learning algorithms, KNN doesn't learn a fixed set of parameters from the training data. Instead, it memorizes the entire training dataset and uses it directly for making predictions. This characteristic makes KNN a "lazy learner," as it defers the bulk of its computational effort until the prediction phase rather than during training.[15]

## A. History of KNN

### The Evolution of K-Nearest Neighbors (KNN): A Comprehensive History

The K-Nearest Neighbors (KNN) algorithm has a rich and fascinating history that spans over six decades, with its roots deeply embedded in statistical estimation and pattern recognition theory. From its humble beginnings to its current status as a fundamental algorithm in machine learning, KNN has undergone significant transformations, shaped by the contributions of numerous researchers and advancements in computational power. In this detailed account, we will delve into the key milestones, developments, and breakthroughs that have defined the evolution of KNN.

#### 1) 1951: Fix and Hodges' Non-Parametric Method - The Precursor to KNN

Eugene Fix and Joseph Hodges introduced a non-parametric method for pattern classification, which is considered a precursor to KNN. Their work, titled "Discriminatory analysis: Nonparametric discrimination: Consistency properties," laid the foundation for the development of KNN. Fix and Hodges' method employed a distance-based approach to classify patterns, which would later become a cornerstone of the KNN algorithm.

#### 2) 1967: Cover and Hart's K-Nearest Neighbor Rule - The Theoretical Foundation

T. M. Cover and P. E. Hart formally introduced the K-Nearest Neighbor rule in their seminal paper "Nearest Neighbor Pattern Classification". This paper established the theoretical foundation for KNN and proved several important properties of the algorithm. Cover and Hart demonstrated that the KNN rule is optimal for certain types of distributions, and their work set the stage for the widespread adoption of KNN in various fields.

#### 3) 1970s-1980s: KNN's Growing Popularity - Exploring Applications

KNN gained popularity as researchers explored its applications in various fields, including statistics, data mining, and pattern recognition. Researchers like R. O. Duda and P. E. Hart contributed to the growth of KNN during this period. They investigated the use of KNN in image recognition, speech recognition, and other areas, further solidifying its position as a versatile algorithm.

#### 4) 1990s-2000s: Computational Advancements - Optimizations and Variations

With the advent of more powerful computers, KNN became increasingly practical for larger datasets. Researchers developed variations and optimizations of the algorithm to improve its efficiency and effectiveness [16]. Techniques like k-d trees, ball trees, and locality-sensitive hashing were introduced to accelerate KNN computations, enabling its application to more complex problems.

#### 5) Present Day: KNN's Continued Relevance - A Fundamental Algorithm

KNN remains a fundamental algorithm in machine learning, often used as a benchmark for more complex algorithms and as a practical solution for various classification and regression tasks. Its simplicity, interpretability, and flexibility make it an attractive choice



for many applications. Researchers continue to explore new ways to improve KNN, incorporating techniques like kernel methods, ensemble learning, and deep learning. [17]

## **B. KNN Classifier Explained**

### **1) K-Nearest Neighbors (KNN) in Classification**

K-Nearest Neighbors (KNN) is a simple, yet powerful machine learning algorithm used for both classification and regression tasks. The algorithm is intuitive, as it classifies a data point based on how its neighbors are classified. KNN is particularly useful in scenarios where the decision boundary is irregular or not well defined by linear methods.

Historically, KNN has been used in various fields such as handwriting recognition, image classification, and even in finance for predicting stock movements. Its simplicity lies in its lazy learning approach, where the algorithm doesn't build an explicit model but rather memorizes the training dataset, making predictions only when required.

### **2) Working Mechanism of KNN in Classification**

#### **a. Computing KNN Distance Metrics**

The key to the KNN algorithm is determining the distance between the query point and the other data points. Determining distance metrics enables decision boundaries. These boundaries create different data point regions. There are different methods used to calculate distance:

- **Euclidean distance** is the most common distance measure, which measures a straight line between the query point and the other point being measured.
- **Manhattan distance** is also a popular distance measure, which measures the absolute value between two points. It is represented on a grid, and often referred to as taxicab geometry — how do you travel from point A (your query point) to point B (the point being measured)
- **Minkowski distance** is a generalization of Euclidean and Manhattan distance metrics, which enables the creation of other distance metrics. It is calculated in a normed vector space. In the Minkowski distance,  $p$  is the parameter that defines the type of distance used in the calculation. If  $p=1$ , then the Manhattan distance is used. If  $p=2$ , then the Euclidean distance is used.
- **Hamming distance**, also referred to as the overlap metric, is a technique used with Boolean or string vectors to identify where vectors do not match. In other words, it measures the distance between two strings of equal length. It is especially useful for error detection and error correction.

### b. What is Euclidean Distance?

- Euclidean Distance is a way to measure how far apart two points are in space. Imagine it like drawing a straight line between two points on a graph and then measuring the length of that line.

### c. How is it Calculated?

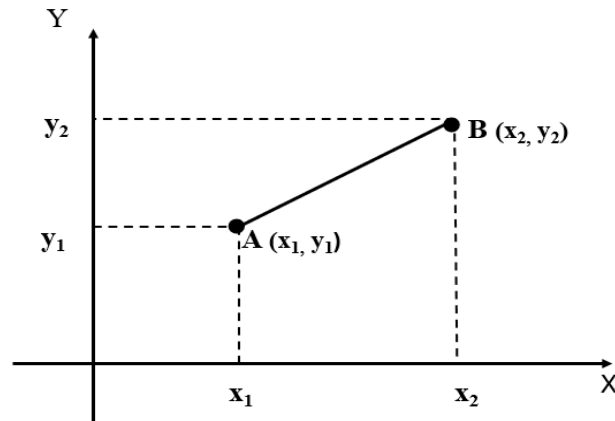


Figure 3-1

- **Formula:** The Euclidean Distance between two points A and B is calculated using this formula:

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

**x<sub>1</sub>, x<sub>2</sub>:** These are the coordinates (like positions) of the first point A.

**y<sub>1</sub>, y<sub>2</sub>:** These are the coordinates of the second point B.

**(x<sub>1</sub> - y<sub>1</sub>)<sup>2</sup>, (x<sub>2</sub> - y<sub>2</sub>)<sup>2</sup>:** For each coordinate, find the difference between the two points, square it (multiply it by itself), and add all these squares together.

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

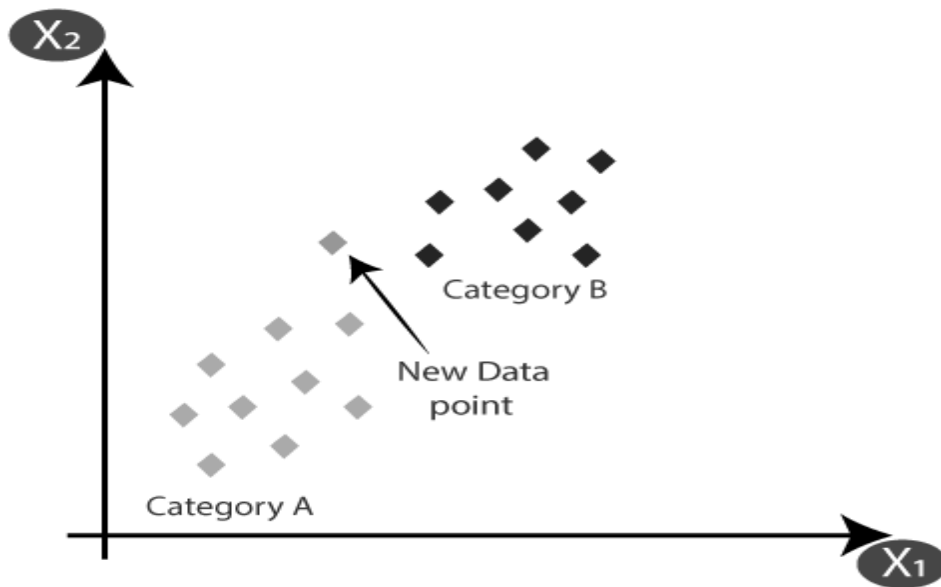
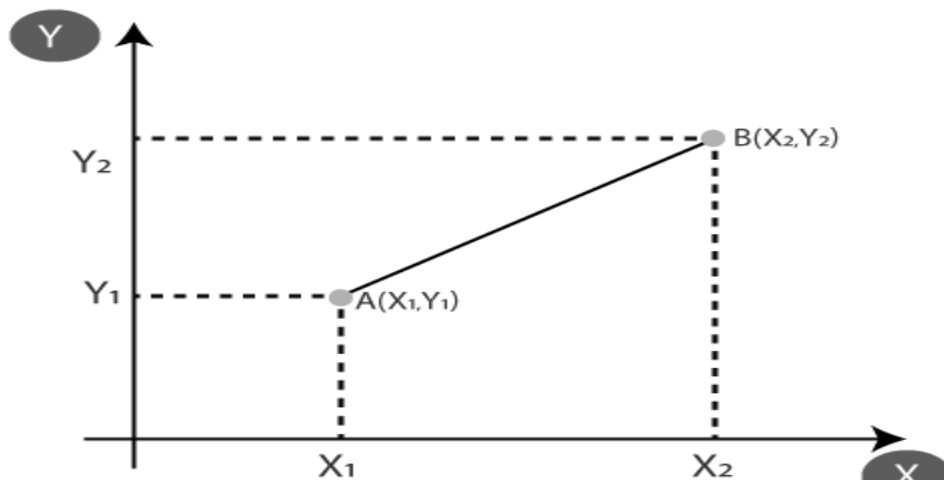


Figure 3-2

- Firstly, we will choose the number of neighbors, so we will choose the  $k=3$ .
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



$$\text{Euclidean Distance between } A_1 \text{ and } B_2 = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

Figure 3-3

By calculating the Euclidean distance, we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

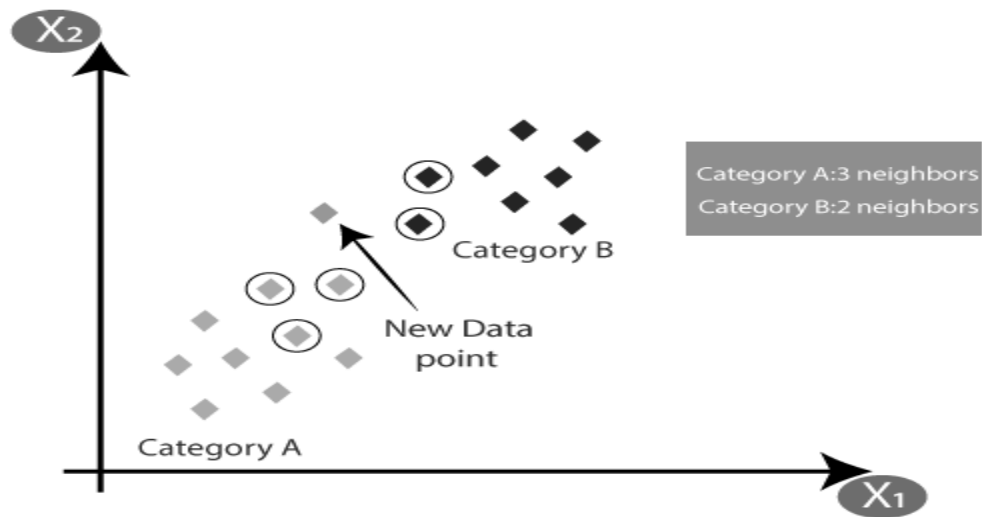


Figure 3-4

- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

### C. Data Preprocessing for KNN Classification

Data preprocessing is crucial for KNN, as the algorithm is sensitive to the scale of the data, missing values, and noise. Effective preprocessing steps include handling missing values, managing duplicates, and normalizing features.

#### 1) Handling Missing Values

Missing values can significantly affect the performance of KNN. Techniques like KNN imputation, where missing values are filled in based on the nearest neighbors, or simpler methods like mean, median, or mode imputation, are commonly used.

#### 2) Managing Duplicates

Duplicated data points can distort the neighborhood structure in KNN. Removing duplicates ensures that the algorithm's predictions are not biased by repeated data points.

#### 3) Feature Scaling and Normalization

Feature scaling is vital in KNN because the algorithm relies on distance metrics. Unscaled features can dominate the distance calculation, leading to biased results. Common scaling methods include Min-Max scaling and Z-score normalization.

### D. Handling Categorical Data in KNN

KNN requires numerical input for distance calculations. Categorical data should be encoded into numerical values using techniques like label encoding to ensure the algorithm performs accurately.

### **1) Dealing with Imbalanced Data in KNN Classification**

Imbalanced data is a common challenge in classification tasks. When one class is underrepresented, the KNN algorithm might be biased towards the majority class, leading to poor performance on the minority class.

### **2) Techniques for Handling Imbalanced Classes**

Several techniques can mitigate the issue of imbalanced data. SMOTE (Synthetic Minority Over-sampling Technique) is a popular method that generates synthetic samples for the minority class by interpolating between existing samples. Other techniques include ADASYN, which focuses on generating more samples for harder-to-classify instances, and random under-sampling, which balances the class distribution by reducing the size of the majority class.

### **3) Overfitting and Underfitting in KNN Classification**

Overfitting occurs when the model is too complex and captures noise in the data, while underfitting happens when the model is too simple and fails to capture the underlying patterns. In KNN, overfitting can occur with a very low 'K' value, and underfitting with a very high 'K' value.

#### **a. Strategies to Prevent Overfitting**

Preventing overfitting in KNN involves selecting an optimal 'K' value, typically through cross-validation. Regularization methods, though less common in KNN, can also be used. Feature selection and dimensionality reduction techniques like PCA can help by reducing the number of features, thereby simplifying the model.

## **E. Hyperparameter Tuning for KNN Classification**

Finding the optimal 'K' value is crucial for balancing bias and variance in KNN. Cross-validation is a standard approach to tune 'K'. Grid Search and Random Search are popular methods for hyperparameter tuning, allowing the exploration of different 'K' values and other parameters like distance metrics.

### **1) What is a confusion matrix?**

It is a matrix of size  $2 \times 2$  for binary classification with actual values on one axis and predicted on another.

		ACTUAL	
		Negative	Positive
PREDICTION	Negative	TRUE NEGATIVE	FALSE NEGATIVE
	Positive	FALSE POSITIVE	TRUE POSITIVE

Figure 3-5: Confusion Matrix graph

### Confusion Matrix

- **True Positive (TP)** — model correctly predicts the positive class (prediction and actual both are positive). In the above example, **10 people** who have tumors are predicted positively by the model.
- **True Negative (TN)** — model correctly predicts the negative class (prediction and actual both are negative). In the above example, **60 people** who don't have tumors are predicted negatively by the model.
- **False Positive (FP)** — model gives the wrong prediction of the negative class (predicted-positive, actual-negative). In the above example, **22 people** are predicted as positive of having a tumor, although they don't have a tumor. FP is also called a **TYPE I** error.
- **False Negative (FN)** — model wrongly predicts the positive class (predicted-negative, actual-positive). In the above example, **8 people** who have tumors are predicted as negative.

With the help of these four values, we can calculate True Positive Rate (TPR), False Negative Rate (FNR), True Negative Rate (TNR), and False Negative Rate (FNR).

$$TPR = \frac{TP}{Actual\ Positive} = \frac{TP}{TP + FN}$$

$$FNR = \frac{FN}{Actual\ Positive} = \frac{FN}{TP + FN}$$

$$TNR = \frac{TN}{Actual\ Negative} = \frac{TN}{TN + FP}$$

$$FPR = \frac{FP}{Actual\ Negative} = \frac{FP}{TN + FP}$$

Even if data is imbalanced, we can figure out that our model is working well or not. For that, **the values of TPR and TNR should be high, and FPR and FNR should be as low as possible.**

With the help of TP, TN, FN, and FP, other performance metrics can be calculated.

## 2) Accuracy, Precision, Recall

Accuracy, precision and recall are crucial for information retrieval, where positive class mattered the most as compared to negative.

While searching something on the web, the model does not care about something **irrelevant** and **not retrieved** (this is the true negative case). Therefore, only TP, FP, FN are used in Precision and Recall.

### 1. Accuracy

Accuracy is used to measure the performance of the model. It is the ratio of Total correct instances to the total instances.

$$Accuracy = \frac{TP + TN + FP + FN}{TP + TN}$$

### 2. Precision

Out of all the positive predicted, what percentage is truly positive.

$$Precision = \frac{TP}{TP + FP}$$

### 3. Recall

Out of the total positive, what percentage are predicted positive. It is the same as TPR (true positive rate).

$$Recall = \frac{TP}{TP + FN}$$

## F. Challenges and Limitations in KNN Classification

Despite its simplicity, KNN has several limitations. It can be computationally expensive, especially with large datasets, as it requires calculating the distance to every point in the training set. KNN is also sensitive to outliers and noisy data, which can lead to incorrect classifications.

The curse of dimensionality is another challenge in KNN. As the number of features increases, the distance between points becomes less meaningful, leading to poor performance. Dimensionality reduction techniques can alleviate this issue.

## G. Applications of KNN Classifier

- 1) **Medical Diagnosis**- KNN is frequently used in medical diagnosis, such as predicting whether a patient has a certain disease based on symptoms and medical history, as in your project on coronary heart disease prediction.
- 2) **Image Recognition**- KNN is applied in image classification tasks, where images are classified based on their similarity to other labeled images.
- 3) **Recommendation Systems**- KNN is used in recommendation systems to suggest items to users based on the preferences of similar users.
- 4) **Financial Market Predictions**- KNN is used to classify stock market data and predict whether the price of a stock will go up or down.
- 5) **Anomaly Detection**- KNN can identify outliers in datasets, which is useful in fraud detection and network security.

## 3.3. Logistic Regression for Predicting Coronary Heart Disease (CHD)

### A. What is Regression?

Regression [18] is a statistical approach used to analyze the relationship between a dependent variable (target variable) and one or more independent variables (predictor variables). The objective is to determine the most suitable function that characterizes the connection between these variables.

It seeks to find the best-fitting model, which can be utilized to make predictions or draw conclusions [19].

- **Regression in Machine Learning**

It is a supervised machine learning technique, used to predict the value of the dependent variable for new, unseen data. It models the relationship between the input features and the target variable, allowing for the estimation or prediction of numerical values.

Regression analysis problem works with if output variable is a real or continuous value, such as “salary” or “weight”. Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyper-plane which goes through the points [20]

### B. Linear Regression [21]

Linear regression is a basic predictive analytics technique that uses historical data to predict an output variable. It is popular for predictive modeling because it is easily understood and can be explained using plain English.



The basic idea is that if we can fit a linear regression model to observed data, we can then use the model to predict any future values. For example, let us assume that we have found from historical data that the price (P) of a house is linearly dependent upon its size (S)—in fact, we found that a house's price is exactly 90 times its size.

The equation will look like this:  $P = 90 * S$

With this model, we can then predict the cost of any house. If we have a house that is 1,500 square feet, we can calculate its price to be:  $P = 90 * 1500$

= \$135,000

There are two kinds of variables in a linear regression model:

- The input or predictor variable is the variable(s) that help predict the value of the output variable. It is commonly referred to as X.
- The output variable is the variable that we want to predict. It is commonly referred to as Y.

To estimate Y using linear regression, we assume the equation:  $Y_e = \alpha + \beta X$  where  $Y_e$  is the estimated or predicted value of Y based on our linear equation. Our goal is to find statistically significant values of the parameters  $\alpha$  and  $\beta$  that minimize the difference between Y and  $Y_e$ . If we are able to determine the optimum values of these two parameters, then we will have the line of best fit that we can use to predict the values of Y, given the value of X. So, how do we estimate  $\alpha$  and  $\beta$ ? We can use a method called ordinary least squares.

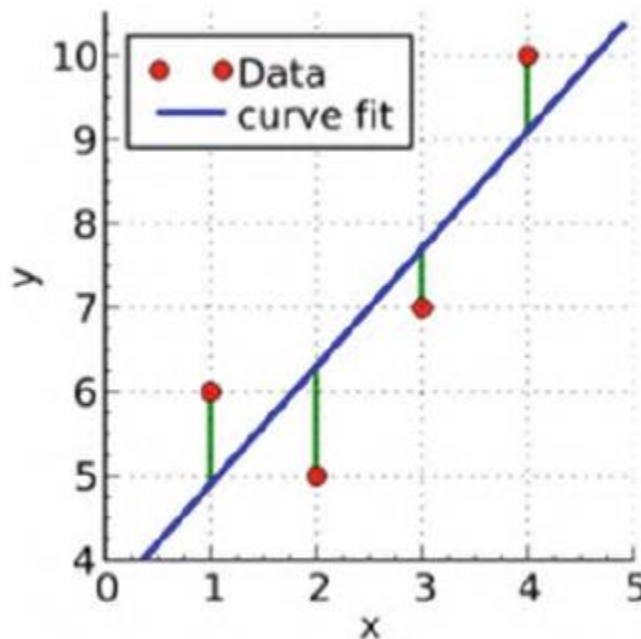


Figure 3-6: Linear function graph

### C. Classification [22]

Generally, classification is about predicting a label whereas regression is usually used to predict a quantity. Classification models approximate a mapping function from inputs to produce a class or category. Logistic regression is used instead of linear regression for binary classification problems. It produces a value ranging from 0 to 1, which can be interpreted as a probability that an event occurred. Decision Trees, Random Forests, and Neural Networks can also be used for classification tasks, similar to that in regression.

Classification is about predicting a label and regression is about predicting a quantity. Classification predictive modeling is the task of approximating a mapping function ( $f$ ) from input variables ( $X$ ) to discrete output variables ( $y$ ). The output variables are often called labels or categories. The mapping function predicts the class or category for a given observation.

For example, an email of text can be classified as belonging to one of two classes: “spam” and “not spam.” A classification can have real-valued or discrete input variables.

Here are different types of classification problem:

- A problem with two classes is often called a two-class or binary classification problem.
- A problem with more than two classes is often called a multi-class classification problem.
- A problem where an example is assigned multiple classes is called a multi-label classification problem.

It is common for classification models to predict a continuous value as the probability of a given example belonging to each output class. The probabilities can be interpreted as the likelihood or confidence of a given example belonging to each class. A predicted probability can be converted into a class value by selecting the class label that has the highest probability. For example, a specific email of text may be assigned the probabilities of 0.1 as being “spam” and 0.9 as being “not spam.” We can convert these probabilities to a class label by selecting the “not spam” label as it has the highest predicted likelihood.

There are many ways to estimate the skill of a classification predictive model, but perhaps the most common is to calculate the classification accuracy. The classification accuracy is the percentage of correctly classified examples out of all predictions made.

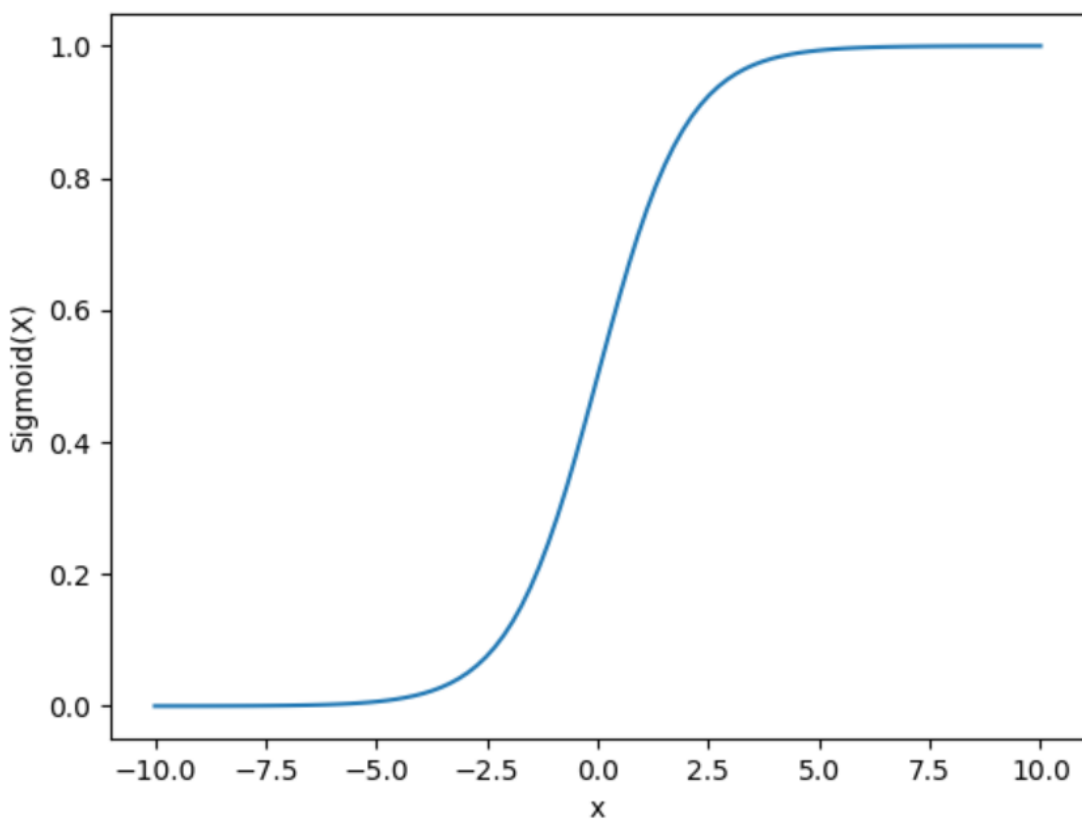
### D. Logistic regression

Logistic regression is often used to predict whether an email is spam or not spam, whether a tumor is malignant or benign, or whether a customer will buy a product or not.

A basic machine learning approach that is frequently used for binary classification tasks is called logistic regression. Logistic regression falls under **classification** rather than regression, despite its name. While the term "regression" might suggest that it's used for

predicting continuous values, logistic regression is actually used for predicting discrete outcomes, particularly binary outcomes (i.e., two possible classes). It uses the sigmoid function to simulate the likelihood of an instance falling into a specific class, producing values between 0 and 1. Logistic regression, with its emphasis on interpretability, simplicity, and efficient computation, is widely applied in a variety of fields, such as marketing, finance, and healthcare, and it offers insightful forecasts and useful information for decision making.

The model estimates the probability that a given input belongs to a particular class, and then classifies the input based on that probability (usually by applying a threshold, such as 0.5). Thus, logistic regression is a core technique in binary classification tasks.



*Figure 3-7: Sigmoid function graph*

## 1) History of Logistic Regression

Logistic regression, a fundamental statistical modeling technique, has played a pivotal role in the evolution of various scientific disciplines. Logistic regression has grown from its roots in biostatistics to become a versatile and powerful tool in fields such as artificial intelligence (AI) and machine learning. The history and development of logistic regression,

highlights its journey from early applications in biostatistics to its integration into AI and deep learning.

**a. The origins of logistic regression (early 20th century)**

**1. The Logistic Function**

The foundation of logistic regression lies in the logistic function, a mathematical model introduced by Belgian mathematician Pierre François Verhulst in the 1830s. Verhulst initially developed the logistic function to model population growth, capturing how populations grow rapidly before slowing down as they approach a maximum capacity, or carrying capacity. This S-shaped curve, characterized by an initial slow start, followed by rapid growth and eventual leveling off, became a critical tool in population dynamics. While Verhulst's work was primarily focused on ecology, the mathematical properties of the logistic function soon drew attention from statisticians. Its ability to model processes with saturation effects made it valuable for studying binary outcomes in various fields, setting the stage for logistic regression as a statistical method.

**2. Early Statistical Applications**

In the early 20th century, statisticians began to explore the logistic function's potential in modeling binary classification problems. It was used to model the probability of binary outcomes, such as success/failure or yes/no decisions. This marked the beginning of logistic regression as a method for analyzing binary data, providing valuable insights into the relationship between predictor variables and binary outcomes. The recognition that the logistic function could model the probability of an event occurring based on predictor variables laid the groundwork for the formal development of logistic regression.

**b. The formal development of logistic regression (1940s-1950s)**

**1. Joseph Berkson and the Birth of Logistic Regression**

The formalization of logistic regression is credited to Joseph Berkson, who made significant contributions in the mid-20th century. In 1944, Berkson introduced the "logit" function, representing the natural logarithm of the odds ratio. The logit function established a linear relationship between the log-odds of an event occurring and predictor variables, simplifying the estimation of logistic regression model parameters. Berkson's work laid the foundation for logistic regression as a statistical method for binary outcomes. He proposed using maximum likelihood estimation (MLE) to estimate model parameters, a method that became the standard for fitting logistic regression models. His contributions were particularly impactful in biostatistics, where logistic regression was first widely applied to study the

relationship between smoking and lung cancer, marking one of its first major applications in medical research.

## **2. Early Applications in Biostatistics**

Following Berkson's pioneering work, logistic regression quickly gained traction in biostatistics. Researchers realized its potential for analyzing the relationship between risk factors and disease likelihood. One of the earliest significant applications was in the Framingham Heart Study, launched in the 1940s to identify cardiovascular disease risk factors. Logistic regression was used to model the probability of heart disease development based on factors like age, blood pressure, cholesterol levels, and smoking. This study revolutionized cardiovascular risk understanding and established logistic regression as a powerful tool in medical research.

### **c. Expansion into social sciences (1950s-1960s)**

As logistic regression matured, its applications extended into the social sciences. Researchers in sociology, political science, and economics began using it to analyze binary outcomes in social phenomena. In the 1950s, sociologists employed logistic regression to study voter behavior and political participation, modeling binary outcomes like whether an individual would vote. The method was also used to examine party affiliation determinants, exploring how factors like socioeconomic status, education, and ethnicity influenced political preferences

### **d. Educational attainment and occupational choices**

Logistic regression also proved valuable in studying educational attainment and occupational choices. Social scientists used it to model the probability of achieving different education levels based on factors like family background, socioeconomic status, and gender. The method also helped analyze factors influencing occupational choices, providing insights into social mobility and stratification. For example, logistic regression was used to examine the relationship between parental education and children's educational outcomes, offering insights into the role of social and economic factors in educational attainment. Similarly, it was used to study the factors influencing career choices, shedding light on the social and economic determinants of occupational mobility.

### **e. Integration into Artificial Intelligence and Machine Learning (1960s-1990s)**

The mid-20th century marked a transformative period for logistic regression as it began to intersect with the emerging fields of artificial intelligence (AI) and machine learning. This era not only witnessed the expansion of logistic regression's applications but also its deep integration into foundational AI technologies, which set the stage for its critical role in contemporary machine learning.

## **1. The Perceptron and Early Neural Networks**

One of the earliest and most significant links between logistic regression and artificial intelligence was established with the development of the perceptron in the late 1950s. Frank Rosenblatt, an American psychologist, and computer scientist, introduced the perceptron in 1958 as a model inspired by the human brain. The perceptron was designed to simulate the way biological neurons process information, making it one of the first neural network models. At the core of the perceptron was the concept of neurons "firing" based on the weighted sum of inputs they received. To determine whether a neuron would fire, Rosenblatt employed an activation function—often the logistic function. The logistic function's S-shaped curve was particularly suitable for this purpose because it could map any input value to a probability between 0 and 1, effectively determining the likelihood that a neuron should activate. This use of the logistic function as an activation mechanism in the perceptron was groundbreaking. It illustrated the power of logistic regression in binary classification tasks, where the objective was to categorize inputs into one of two classes. The perceptron's success in these tasks demonstrated that logistic regression could be harnessed within neural networks to create more complex AI systems. This early integration was a precursor to the more extensive use of logistic regression in later AI and machine learning models, bridging the gap between traditional statistical methods and the emerging field of artificial intelligence. The perceptron's influence extended beyond its immediate applications. It laid the groundwork for the development of more sophisticated neural networks in the decades that followed. Although the initial excitement around the perceptron waned due to its limitations—such as its inability to solve non-linearly separable problems—its reliance on the logistic function was a key step in the evolution of AI. The perceptron set the stage for the future integration of logistic regression into a wide array of AI applications, from simple binary classifiers to complex, multi-layered neural networks.

## **2. The Rise of Machine Learning in the 1980s**

The 1980s marked a pivotal decade in the evolution of machine learning, during which logistic regression emerged as a leading tool for classification tasks. As researchers delved deeper into the mechanics of machine learning, they recognized the unique advantages offered by logistic regression, particularly its simplicity, interpretability, and effectiveness in binary classification problems. During this period, researchers such as Leo Breiman, Jerome Friedman, Charles J. Olshen, and Richard A. Stone made significant contributions to the field by integrating logistic regression into the framework of classification and regression trees (CART). Their work showcased how logistic regression could be combined with other machine learning techniques to enhance predictive accuracy and model robustness. By

integrating logistic regression into decision tree models, these researchers were able to create more versatile and powerful predictive models that could handle a wide range of classification tasks. One of the key advantages of logistic regression in the machine learning landscape of the 1980s was its ability to provide probabilistic predictions. Unlike some other classification models that simply assign a data point to a class, logistic regression could output the probability that a given data point belonged to a particular class. This feature made logistic regression particularly valuable in applications where understanding the likelihood of an outcome was crucial, such as in risk assessment, medical diagnosis, and financial forecasting. Moreover, the interpretability of logistic regression models made them an attractive choice for early machine learning practitioners. Logistic regression's clear and understandable results allowed researchers and practitioners to easily interpret the outcomes and make informed, data-driven decisions. This transparency was especially important in fields such as medicine and finance, where decision-makers needed to understand the reasoning behind model predictions.

### **3. The emergence of statistical learning in the 1990s**

The 1990s witnessed the emergence of statistical learning as a field that seamlessly blended the principles of statistics with the methodologies of machine learning. This integration brought logistic regression to the forefront as a versatile and powerful tool for analyzing complex datasets and predicting binary outcomes. Statistical learning frameworks provided a structured approach to modeling and prediction, combining the rigorous mathematical foundations of statistics with the computational power of machine learning algorithms. Within this context, logistic regression played a crucial role, offering a reliable method for modeling relationships between predictor variables and binary outcomes. One of the most significant applications of logistic regression in statistical learning was in the field of medical diagnosis. During the 1990s, researchers began using logistic regression to develop models that could predict the likelihood of a patient developing a particular disease based on various risk factors. For example, logistic regression was employed to analyze data from large-scale epidemiological studies, such as the Framingham Heart Study, to predict the probability of heart disease based on factors like age, blood pressure, cholesterol levels, and smoking status. These models provided valuable insights into the risk factors associated with heart disease and helped inform clinical decision-making. In addition to its applications in medicine, logistic regression became a popular tool in natural language processing (NLP) during the 1990s. Researchers used logistic regression to classify text documents into predefined categories, such as spam or non-spam emails, by modeling the probability that a given document belonged to a particular category based on its features. This application of logistic regression in NLP laid the foundation for the development of more advanced text classification techniques, such as support vector

machines and neural networks, in the following decades. The integration of logistic regression into statistical learning frameworks also facilitated its application in fields such as finance, marketing, and social sciences. In finance, for example, logistic regression was used to model the probability of default on loans, enabling lenders to assess the creditworthiness of borrowers and make informed lending decisions. In marketing, logistic regression was employed to predict customer behavior, such as the likelihood of making a purchase or responding to a marketing campaign, based on demographic and behavioral data.

Therefore, the success of logistic regression in these diverse applications can be attributed to its ability to provide probabilistic predictions and its relatively simple model structure. Unlike more complex models, logistic regression offered a straightforward interpretation of the relationship between predictor variables and binary outcomes, making it an attractive choice for researchers and practitioners who needed to balance predictive accuracy with model transparency.[23] [24] [25]

## E. How the Logistic Regression Algorithm Works

Logistic Regression [26] models predict the likelihood that an instance will belong to a particular class. It uses a linear equation to combine the input information and the sigmoid function to restrict predictions between 0 and 1. These coefficients produce the resulting decision boundary, which divides instances into two classes. When it comes to binary classification, logistic regression is the best choice because it is easy to understand, straightforward, and useful in a variety of settings. Generalization can be improved by using regularization. [27]

### 1. Key Concepts of Logistic Regression

Important key concepts in logistic regression include:

- **Sigmoid Function:** The main function that ensures outputs are between 0 and 1 by converting a linear combination of input data into probabilities. The sigmoid function [28] is denoted as

$$\sigma(z)$$

and is defined as:

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

Where,  $z$  is linear combination of input features and coefficients. [29]

### 2. Logistic Regression Function

**Linear Combination:** In logistic regression, the linear combination is used to compute a single value from the input features, which is then transformed into a probability through the logistic (sigmoid) function.



### Formula:

$$z = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots + \beta_nx_n$$

### Components Explanation

#### 1. $z$ (Predicted Value)

$z$  is the final result or outcome that the model predicts. It is the sum of all the influences from the independent variables ( $x_1, x_2, \dots, x_n$ ) after they have been weighted by their respective coefficients. For example; predicting a person's salary,  $z$  would represent the estimated salary based on factors like experience, education, and hours worked.

#### 2. $\beta_0$ (Intercept)

$\beta_0$  is the intercept value of  $z$  when all the independent variables ( $x_1, x_2, \dots, x_n$ ) are zero (when all the independent variables  $x_1, x_2, \dots, x_n$  are zero means that none of these factors contribute anything to the outcome  $z$ . In other words, the value of each independent variable is zero, meaning they don't add any positive or negative influence to  $z$ . In this scenario, the outcome  $z$  is entirely determined by the intercept  $\beta_0$ . The intercept  $\beta_0$  represents the baseline or starting value of  $z$  when there is no contribution from the independent variables). It's the starting point or baseline value before considering any of the other factors. For example; In the salary prediction example,  $\beta_0$  might represent the baseline salary, such as the minimum salary a person could expect with no experience, education, or work hours.

#### 3. $x_1, x_2, \dots, x_n$ (Independent Variables)

These are the factors or predictors that influence  $z$ . Each  $x_1$  corresponds to a different aspect that affects the outcome. For example;  $x_1$  could be years of experience,  $x_2$  might be education level, and so on. These are the inputs that impact the final salary prediction.

#### 4. $\beta_1, \beta_2, \dots, \beta_n$ (Coefficients/Weights)

These are the coefficients that measure how much influence each independent variable  $x_1$  has on the dependent variable  $z$ . They tell how strongly each factor impacts the outcome. For example,  $\beta_1$  might show how much an additional year of experience increases the salary, while  $\beta_2$  might indicate how much higher education boosts the salary. Positive coefficients mean the factor increases  $z$ , while negative coefficients mean it decreases  $z$ .

### 3. Sigmoid function

The sigmoid function is a mathematical formula that takes any number and squashes it in the ranges between 0 and 1. This is useful for making predictions where estimations of the probabilities, like whether an email is spam or not takes place.

**Formula:**

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

**Components Explanation**

1) The sigmoid function is denoted by the Greek letter 'σ'.

2) **z:**

- It is the input to the sigmoid function. It can be any real number. The value of x determines the output.
- If the input x to the sigmoid function is a **small negative number**, the output is very close to 0.

$$\begin{aligned}\lim_{x \rightarrow -\infty} \sigma(z) &= \lim_{x \rightarrow -\infty} \frac{1}{1 + e^{-z}} \\ &= \lim_{z \rightarrow -\infty} \frac{1}{1 + e^{-\infty}} \\ &= 0\end{aligned}$$

- For example:

$$\begin{aligned}\sigma(-4) &= \frac{1}{1 + e^{-(-4)}} \\ &= 0.01798621\end{aligned}$$

- If x is a **large positive number**, then the output is very close to 1.

$$\begin{aligned}\lim_{x \rightarrow \infty} \sigma(z) &= \lim_{x \rightarrow \infty} \frac{1}{1 + e^{-z}} \\ &= \lim_{z \rightarrow \infty} \frac{1}{1 + e^{-\infty}} \\ &= 1\end{aligned}$$

- For example:

$$\begin{aligned}\sigma(4) &= \frac{1}{1 + e^{-4}} \\ &= 0.9820138\end{aligned}$$

**3) Numerator (1):**

The numerator of the sigmoid function is simply 1. It acts as a constant and is crucial for normalizing the function's output to be between 0 and 1. The 1 ensures that the function can provide values in this range regardless of the value of z.

#### 4) $e^{-z}$

- $e^{-z}$ : This is the base of the natural logarithm, approximately equal to 2.718. It is a fundamental mathematical constant used in exponential growth.
- $z$ : The exponent in the term  $e^{-z}$ . The negation of  $z$  means that as  $z$  increases,  $e^{-z}$  decreases exponentially, and as  $z$  decreases,  $e^{-z}$  increases exponentially.
- $e^{-z}$ : This term represents an exponential decay function. It determines how rapidly the value decreases as  $z$  increases and how it grows as  $z$  decreases. This exponential term ensures the smooth, S-shaped curve of the sigmoid function.

### F. Real life examples

Logistic regression is widely used in various real-life scenarios, particularly when the goal is to predict a binary outcome (yes/no, true/false, 0/1). Here are some examples:

#### 1. Medical Diagnosis

- 1) Disease Prediction: Logistic regression is used to predict whether a patient has a particular disease (e.g., diabetes, heart disease) based on various factors like age, blood pressure, and cholesterol levels.
- 2) Cancer Detection: It can help predict the presence of cancerous cells based on imaging data or genetic markers.

#### 2. Marketing

- 1) Customer Churn Prediction: Companies use logistic regression to predict whether a customer is likely to stop using their service based on their interaction history, purchase behavior, and customer support interactions.
- 2) Email Campaign Effectiveness: Predict whether a customer will click on a link in an email or make a purchase after receiving a marketing email.

#### 3. Finance

- 1) Credit Scoring: Logistic regression models are used to assess the probability that a borrower will default on a loan. Factors like income, credit history, and debt to income ratio are often used as predictors.
- 2) Fraud Detection: It helps in identifying potentially fraudulent transactions based on patterns in transaction data.

#### 4. Human Resources

- 1) Employee Attrition: HR departments use logistic regression to predict whether an employee is likely to leave the company. Factors could include job satisfaction, salary, and work life balance.
- 2) Recruitment: Predicting whether a job applicant will accept an offer based on their application details and interview performance.

## 5. Healthcare

- 1) Hospital Readmission: Predicting the likelihood of a patient being readmitted to the hospital within a certain time frame after discharge, based on their medical history, treatment, and demographics.
- 2) Birth Outcomes: Logistic regression is used to predict the likelihood of complications during birth based on prenatal factors

## 6. Ecommerce

- 1) Purchase Prediction: Online retailers use logistic regression to predict whether a customer will make a purchase based on their browsing history, cart status, and demographic information.
- 2) Customer Segmentation: Classifying customers into different segments (e.g., highvalue, atrisk) based on their purchase history and engagement level.

## 7. Education

- 1) Student Performance: Predicting whether a student will pass or fail a course based on their attendance, assignment scores, and participation in class.
- 2) Dropout Prediction: Identifying students at risk of dropping out of a course or program, based on their academic performance and engagement.

### 3.4. Random Forest Algorithm:

#### 1. Introduction:

Heart disease is one of the biggest health problems in the world, leading to many deaths each year. To help prevent this, it's important to know who is at risk of developing heart disease in the future. In this project, we use a technique called Random Forest, which is like a group of experts working together to make decisions.

Imagine you have a panel of doctors, each looking at different health factors—like your age, cholesterol levels, and blood pressure. Each doctor gives their opinion on whether you might develop heart disease in the next ten years. The Random Forest algorithm works in a similar way: it combines the opinions (or decisions) of many different "decision trees" to make a final prediction. This method is powerful because it doesn't rely on just one decision but rather on the collective wisdom of many.

By using Random Forest, we can predict who is more likely to develop heart disease, allowing doctors to focus on preventive care for those individuals. This can ultimately lead to better health outcomes and save lives.

## **2. History and development of random forest algorithm:**

The Random Forest algorithm is a relatively recent development in the field of machine learning, but its roots go back to earlier ideas in decision tree learning.

### **1) Decision Trees:**

The concept of decision trees, which are the building blocks of Random Forest, was first introduced in the 1960s and 1970s. A decision tree is a simple yet powerful tool that makes decisions by splitting data into different branches based on certain conditions. Each branch represents a possible decision path, leading to a final outcome. While decision trees are easy to understand and use, they have a tendency to overfit, meaning they might perform very well on training data but poorly on new, unseen data.

### **2) Ensemble Learning:**

To overcome the limitations of individual decision trees, researchers began exploring ensemble methods in the 1980s and 1990s. Ensemble learning involves combining multiple models to improve overall performance. The idea is that a group of models, when combined, can make more accurate predictions than any single model on its own. This concept laid the groundwork for the development of the Random Forest algorithm.

### **3) Birth of Random Forest:**

The Random Forest algorithm was developed by Leo Breiman, a statistician at the University of California, Berkeley, in 2001. Breiman built on the idea of "bagging" (short for bootstrap aggregating), which involves training multiple versions of a model on different subsets of the data and then averaging their predictions. Random Forest took this idea further by adding randomness to the process: each decision tree in the forest is trained on a random subset of the data and a random subset of the features. This randomness helps to create a diverse set of trees, reducing the risk of overfitting and improving the model's generalization to new data.

### **4) Impact and Adoption:**

Since its introduction, Random Forest has become one of the most popular machine learning algorithms due to its high accuracy, robustness, and ability to handle large datasets with many features. It is widely used in various fields, including healthcare, finance, and marketing, for tasks such as classification, regression, and feature selection. Its ability to provide feature importance scores has also made it valuable for understanding which factors are most important in making predictions [30]

### 3. About the Random Forest Algorithm

Imagine you have a forest, and each tree in this forest is a decision-maker. When you ask the forest a question, each tree gives its own answer. The forest then combines all these answers to give you the final decision. This is what the Random Forest algorithm does in machine learning.

#### 1) Key Characteristics

- a) **Team Effort:** Random Forest uses many decision trees working together to make better predictions. This teamwork helps improve accuracy and reduces mistakes.
- b) **Strong and Flexible:** It can handle large amounts of data with many features (columns) without getting overwhelmed.
- c) **Sees the Big Picture:** Like decision trees, Random Forest can understand complex relationships between features.
- d) **Feature Importance:** It tells you which features (columns) are most important for making predictions.

#### 2) Advantages

- a) **Less Overfitting:** Unlike a single decision tree that might get too specific, Random Forest reduces the risk of overfitting by averaging multiple trees.
- b) **High Accuracy:** It often provides very accurate predictions.
- c) **Handles Missing Data:** It can still make good predictions even if some data is missing.

#### 3) Disadvantages

- a) **Complexity:** It can be harder to understand compared to a single decision tree.
- b) **Computationally Intensive:** Building many trees can take a lot of computing power and time.

### 4. Methodology

For this project, Random Forest was chosen due to its robustness and ability to handle a large number of features. The algorithm works by building multiple decision trees and aggregating their results to produce a final prediction. The model was trained on a dataset containing various features such as age, blood pressure, cholesterol levels, smoking habits, and other factors that contribute to the likelihood of developing heart disease.

## 1) How the Random Forest Algorithm Works:

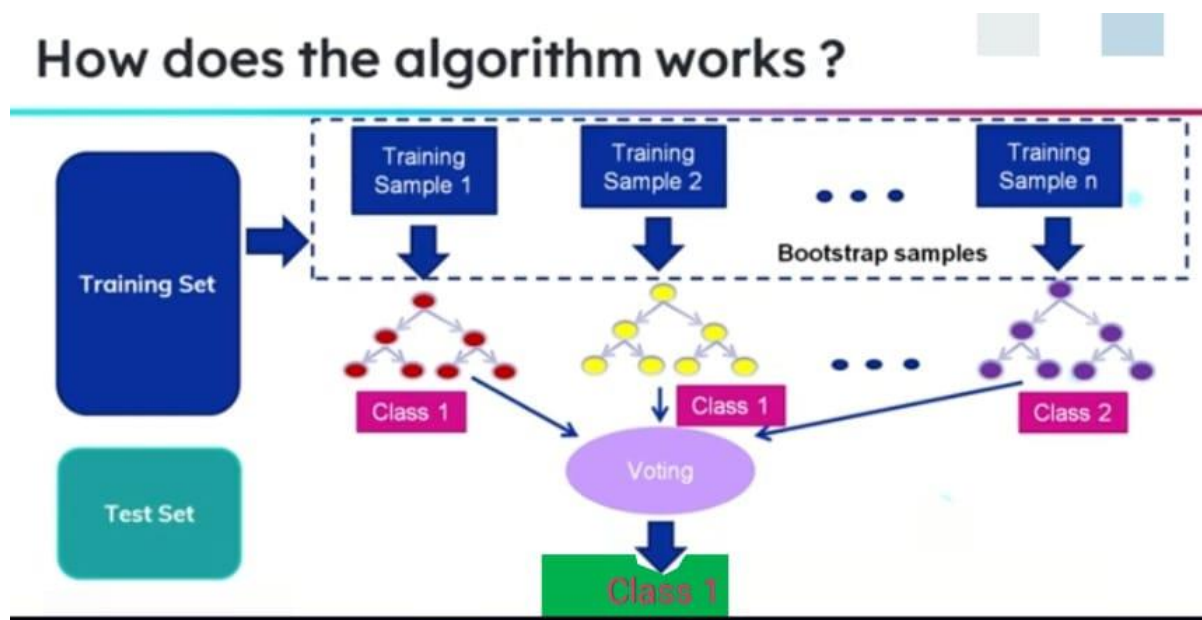


Figure 3-8: Flowchart: Working of Random Forest Algorithm

### Start

- i. **Select Random Subsets of Data**
  - a. Randomly select subsets of data from the training set.
- ii. **Build Decision Trees**
  - a. For each subset, build a decision tree.
- iii. **Repeat for Multiple Trees**
  - a. Repeat the process to create multiple decision trees.
- iv. **Make Predictions**
  - a. For classification: Each tree votes for a class.
  - b. For regression: Each tree makes a prediction.
- v. **Aggregate Results**
  - a. For classification: Take the majority vote.
  - b. For regression: Average the predictions.
- vi. **Output Final Prediction**

OR

- i. **Initialization:** For each tree, a random subset of features is chosen. Think of it as each tree getting a different set of tools to work with.
- ii. **Tree Construction:** Each tree is built using a different random subset of the training data. This diversity helps create a robust model.

- iii. **Voting (for Classification):** When making a prediction, each tree votes for a class (like "yes" or "no"). The class with the most votes win.
- iv. **Averaging (for Regression):** For predicting a number, each tree gives its prediction, and the average of all these predictions is taken as the final answer.[31]

## 5. Mathematical Model of Random Forest Algorithm:

The Random Forest classification algorithm is a method used in machine learning to make predictions, such as whether an email is spam or not, or whether a patient has a particular disease. It does this by combining the results of many different decision trees.

### 1) Decision Trees

#### What is a Decision Tree?

Imagine you're trying to decide whether to go outside based on the weather. You might ask yourself a series of questions like, "Is it raining?" or "Is it windy?" Each question leads you to another question or a decision. This process is like a decision tree. The tree starts at the top with one question, and branches down based on the answers, eventually leading to a final decision (like "Yes, I'll go outside" or "No, I'll stay inside").

### 2) Gini Impurity

#### What It Is:

Gini Impurity tells you how "pure" a group of data is. In simple terms, it measures how mixed up the data is in a particular split or group. If a group has a lot of variety (some people have heart disease, some don't), the impurity is high. If everyone in the group either has heart disease or doesn't, the impurity is low.

#### For your dataset:

- Imagine you want to split your data based on the feature "Cholesterol Level." After the split, Gini Impurity will measure how mixed the groups are. If one group mostly consists of people who have heart disease and the other group mostly consists of people who don't, Gini Impurity will be low (which is good).

#### Mathematical Term (Gini Impurity)

$$Gini(t) = 1 - \sum_{i=1}^c p_i^2$$

- $p_i$ : Probability of picking a data point from class  $i$  (for example, CHD = Yes or CHD = No).
- $c$ : Number of different classes (in your case, 2 classes: CHD = Yes, CHD = No).



### Example Using Your Dataset:

Let's say we split the data by cholesterol levels into two groups. In the first group, 80% of the people do **not** have heart disease (CHD = No), and 20% do (CHD = Yes). The Gini Impurity for that group would be:

$$\text{Gini} = 1 - (0.82 + 0.22) = 1 - (0.64 + 0.04) = 1 - 0.68 = 0.32$$

$$\text{Gini} = 1 - (0.8^2 + 0.2^2) = 1 - (0.64 + 0.04) = 1 - 0.68 = 0.32$$

$$\text{Gini} = 1 - (0.82 + 0.22) = 1 - (0.64 + 0.04) = 1 - 0.68 = 0.32$$

This means that the first group is fairly "pure" (a lot of people in this group do not have heart disease). The lower the Gini, the "purer" the group is.

If a group had an equal number of people with and without heart disease (50% CHD = Yes, 50% CHD = No), the Gini would be:

$$\text{Gini} = 1 - (0.52 + 0.52) = 1 - (0.25 + 0.25) = 1 - 0.5 = 0.5$$

$$\text{Gini} = 1 - (0.5^2 + 0.5^2) = 1 - (0.25 + 0.25) = 1 - 0.5 = 0.5$$

$$\text{Gini} = 1 - (0.52 + 0.52) = 1 - (0.25 + 0.25) = 1 - 0.5 = 0.5$$

This is a less pure group because it's more mixed.

### Why Gini Impurity is Useful:

The decision tree in the Random Forest uses Gini Impurity to decide which feature (like cholesterol, age, or smoking) is the best one to split on. The goal is to find splits that make the groups as pure as possible, meaning the people in each group are mostly similar (either a lot of them have heart disease or a lot don't).

### 3) Entropy and Information Gain:

#### a) Entropy:

Entropy is a way to measure how "messy" or "uncertain" the data is. If everyone in the group is the same (either they all have heart disease or none of them do), the entropy is low. But if the group is mixed, the entropy is high.

#### For your dataset:

- Entropy helps the model understand how much uncertainty there is in the group. A high entropy means the group is very mixed (some people have heart disease, some don't). A low entropy means the group is more certain (most people in the group either have heart disease or don't).

#### Mathematical Term (Entropy):

$$\text{Entropy}(t) = - \sum_{i=1}^c p_i \log_2(p_i)$$

- $p_i$ : Probability of class  $i$  (CHD = Yes or No).
- $C$ : Number of classes (here, 2: CHD = Yes and CHD = No).

**Example Using Your Dataset:**

Let's say after splitting the data by cholesterol levels, one group has 90% of people without heart disease (CHD = No) and 10% with heart disease (CHD = Yes). The entropy of this group would be:

$$\text{Entropy} = - [0.9 \log_2 (0.9) + 0.1 \log_2 (0.1)]$$

$$\text{Entropy} = - [0.9 \log_2(0.9) + 0.1 \log_2(0.1)]$$

$$\text{Entropy} = - [0.9 \log_2(0.9) + 0.1 \log_2(0.1)]$$

**Calculating this:**

$$\text{Entropy} = - [0.9 \times (-0.152) + 0.1 \times (-3.32)] = - [-0.137 + (-0.332)] = 0.469$$

This is low entropy, meaning the group is not very mixed.

If the group had 50% CHD = Yes and 50% CHD = No, the entropy would be:

$$\text{Entropy} = - [0.5 \log_2(0.5) + 0.5 \log_2(0.5)] = - [0.5 \times (-1) + 0.5 \times (-1)] = 1$$

This is higher entropy because the group is more mixed and uncertain.

**b) Information Gain:**

Information Gain tells the model how much "certainty" is gained after making a split. If splitting the data based on cholesterol levels reduces the entropy a lot, the information gain is high, meaning the split was helpful.

**For your dataset:**

- Information Gain helps the model decide which feature is the best to split on. It chooses the feature that reduces the most uncertainty (entropy), helping it create purer groups.

**Mathematical Term (Information Gain):**

$$IG = \text{Entropy}(\text{parent}) - \sum_{k=1}^K \frac{|S_k|}{|S|} \text{Entropy}(S_k)$$

- $S_k$ : Subset after the split.
- $|S_k|/|S|$  Proportion of data in subset  $S_k$ .
- $K$ : Number of groups after the split.

**Example Using Your Dataset:**

Suppose the entropy before the split (the parent) is 0.8. After splitting by cholesterol levels, the two groups have entropies of 0.469 and 0.6. Information Gain is:

### **Example Using Your Dataset:**

Suppose the entropy before the split (the parent) is 0.8. After splitting by cholesterol levels, the two groups have entropies of 0.469 and 0.6. Information Gain is:

$$IG = 0.8 - \left( \frac{60}{100} \times 0.469 + \frac{40}{100} \times 0.6 \right)$$

$$IG = 0.8 - [0.282 + 0.24] = 0.8 - 0.522 = 0.278$$

This means splitting the data by cholesterol level helped reduce the uncertainty by 0.278, making it a good split.

## **4) Bootstrap Aggregation (Bagging)**

### **What is Bootstrap Aggregation?**

Imagine you have a deck of cards. To train the model, you shuffle the deck and draw cards multiple times, allowing repeats. Each draw gives you a different "hand" of cards. In machine learning, this is called bootstrapping. Each decision tree in a Random Forest is trained on a different "hand" of data (a subset of the original data).

### **Random Feature Selection:**

At each step of building a tree, instead of looking at all possible features (like "Is it raining?" or "Is it windy?"), the algorithm only considers a random subset of features. This randomness helps make each tree different and prevents overfitting (when a model is too closely tailored to the training data).

## **5) Tree Construction**

The tree is built by asking questions (splits) that try to make the data in each branch as "pure" as possible, meaning all the data in a branch is as similar as possible (like all red balls in a branch).

## **6) Ensemble of Trees**

### **What is an Ensemble?**

Instead of relying on just one decision tree, Random Forest combines the results of many trees. Imagine asking 100 people whether it will rain tomorrow. Each person (tree) gives an answer, and you go with the majority vote. This is what happens in Random Forest.

### **Voting Mechanism:**

For classification (like deciding if an email is spam or not), each tree gives a vote. The class with

### **7) Out-of-Bag (OOB) Error Estimation**

#### **What is Out-of-Bag Error?**

When each tree is trained, about one-third of the data is left out. This data can be used to test how well the tree performs. The average error across all trees on this left-out data is called the Out-of-Bag error. It's like a built-in way to check how good the Random Forest is at making predictions without needing extra data.

### **8) Probability Estimates**

#### **What are Probability Estimates?**

Instead of just saying "yes" or "no," Random Forest can also tell you how confident it is in its prediction. For example, it might say there's an 80% chance of rain tomorrow. This is done by looking at how many trees voted for each possible outcome and calculating the probabilities.[33]

## **G. Tools and Technology**

### **1) Introduction to Python**

**“Python has gotten sufficiently weapons grade that we don’t descend into R anymore. Sorry, R people. I used to be one of you but we no longer descend into R.”**

**– Chris Wiggins**

Python is a general-purpose programming language conceived in 1989 by Dutch programmer Guido van Rossum. Python is free and open source, with development coordinated through the Python Software Foundation.

Python has experienced rapid adoption in the last decade and is now one of the most commonly used programming languages.

#### **a. Common Uses**

Python is a general-purpose language used in almost all application domains such as

- Communications
- Web development (Flask and Django covered in the future chapters)

- CGI and graphical user interfaces
- Game development
- AI and data science (very popular)
- Multimedia, data processing, security, etc.

Python is beginner-friendly and routinely used to teach computer science and programming in the top computer science programs. Python is particularly popular within the scientific and data science communities. It is steadily replacing familiar tools like Excel in the fields of finance and banking.

#### **b. Features**

Python is a high-level language suitable for rapid development. It has a relatively small core language supported by many libraries. Multiple programming styles are supported (procedural, object-oriented, functional, etc.)

Python is interpreted rather than compiled.

#### **c. Syntax and Design**

One nice feature of Python is its elegant syntax. Elegant code might sound superfluous, but in fact it is highly beneficial because it makes the syntax easy to read and easy to remember. Closely related to elegant syntax is an elegant design. Features like iterators, generators, decorators, and list comprehensions make Python highly expressive, allowing you to get more done with less code. Namespaces improve productivity by cutting down on bugs and syntax errors.

### **2) Scientific Programming**

Python has become one of the core languages of scientific computing.

It is either the dominant player or a major player in

- Machine learning and data science
- Astronomy
- Artificial intelligence
- Chemistry
- Computational biology
- Meteorology

Its popularity in economics is also beginning to rise.

### **3) Why Python for Artificial Intelligence**

Python is very popular for Artificial Intelligence developers for a few reasons:

#### **1. It is easy to use:**

- Python is easy to use and has a fast-learning curve. New data scientists can easily learn Python with its simple to utilize syntax and better comprehensibility.

- Python additionally gives a lot of data mining tools that help in better handling of the data, for example, Rapid Miner, Weka, Orange, and so on.
- Python is significant for data scientists since it has many useful and easy to use libraries like Pandas, NumPy, SciPy, TensorFlow, and many more concepts that a skilled Python programmer must be well acquainted with.

## **2. Python is flexible:**

- Python not only lets you create software but also enables you to deal with the analysis, computing of numeric and logical data, and web development.
- Python has additionally become ubiquitous on the web, controlling various Prominent websites with web development frameworks like TurboGears, Django, and Tornado.
- It is perfect for developers who have the talent for application and web development. No big surprise, most data scientists favor this to the next programming alternatives available in the market.

## **3. Python builds better analytics tools:**

- Data analytics is a necessary part of data science. Data analytics tools give information about different frameworks that are important to assess the performance in any business. Python programming language is the best choice for building data analytics tools.
- Python can easily provide better knowledge, get examples, and correlate data from big datasets. Python is additionally significant in self-service analytics.
- Python has likewise helped the data mining organizations to all the more likely to handle the data for their sake.

## **4. Python is significant for deep learning:**

- Python has a lot of packages like TensorFlow, Keras, and Theano that are assisting data scientists with developing deep learning algorithms. Python gives superior help with regard to deep learning algorithms.
- Deep learning algorithms were inspired by the human brain architecture. It manages to build artificial neural networks that reenact the conduct of the human mind. Deep learning neural networks give weight and biasing to different input parameters and give the desired output

## **5. Huge community base:**

- Python has a gigantic community base of engineers and data scientists like Python.org, Fullstackpython.com, realpython.com, etc. Python developers can impart their issues and thoughts to the community. Python Package Index is an extraordinary place to explore the different skylines of the Python programming language. Python developers are continually making enhancements in the language that is helping it to turn out to be better over time.

## **4) Setting Up the Python Environment**

After understanding the basics of Python as a programming language, the next essential step is to set up the appropriate environment to write and execute our Python code

efficiently. The environment setup is crucial as it ensures that all necessary dependencies are installed, and the code runs smoothly.

There are several options available for setting up a Python environment:

**1. Anaconda Distribution:**

- Anaconda provides a comprehensive platform for data science and machine learning projects. It includes the Python interpreter, Jupyter notebooks, and numerous pre-installed libraries like NumPy, Pandas, and scikit-learn. It's a popular choice due to its ease of installation and robust package management via `conda`.

**2. Visual Studio Code (VS Code):**

- VS Code, a lightweight yet powerful code editor, is another excellent choice for setting up a Python environment. It offers extensions for Python, integrated terminal access, and a user-friendly interface. VS Code also supports virtual environments and provides debugging tools for Python code.

**3. Jupyter Notebooks:**

- For interactive coding, Jupyter notebooks are an excellent option. They allow you to write and execute Python code in blocks, visualize data with plots, and document your work in a readable format. This makes it easier to experiment with machine learning algorithms and track progress.

Once the environment is set up, we can proceed to write and execute our machine learning code. With the proper setup in place, we ensure that the Python interpreter, libraries, and tools are readily available for seamless development. This brings us to the next step: implementing machine learning algorithms and evaluating their performance

**5) Transition to Machine Learning Code**

With the environment ready, we can now begin coding our machine learning models. In the upcoming sections, we will walk through the process of building, training, and evaluating different machine learning algorithms, using the setup we have just prepared. The choice of algorithms, their configurations, and performance optimizations will be detailed step by step.

In this project, several machine learning models were employed to tackle the problem at hand. Each model was selected based on its suitability for the dataset and the specific characteristics of the problem. The following algorithms were implemented:

- a) **Decision Tree**
- b) **K-Nearest Neighbors (KNN)**
- c) **Logistic Regression**
- d) **Random Forest**

These models represent a mix of interpretable and complex algorithms, providing a balance between simplicity, accuracy, and computational efficiency.

## Chater 4: Data Collection

### 4.1. About dataset:

World Health Organization has estimated 12 million deaths occur worldwide every year due to heart diseases. Half the deaths in the United States and other developed countries are due to cardio vascular diseases. The early prognosis of cardiovascular diseases can aid in making decisions on lifestyle changes in high-risk patients and in turn reduce the complications [33].

#### 4.1.1. Dataset Description

The dataset used for this analysis included several key health parameters that are commonly associated with heart disease risk. The features selected for prediction were:

1. **Age:** The age of the individual.
2. **CurrentSmoker:** A binary indicator of whether the individual is a current smoker.
3. **CigsPerDay:** The average number of cigarettes smoked per day by the individual.
4. **BPMeds:** Whether the individual is on blood pressure medication.
5. **PrevalentStroke:** A binary indicator of whether the individual has had a stroke in the past.
6. **PrevalentHyp:** Whether the individual has a history of hypertension.
7. **Diabetes:** A binary indicator of whether the individual has diabetes.
8. **TotChol:** Total cholesterol level of the individual.
9. **SysBP:** Systolic blood pressure of the individual.
10. **DiaBP:** Diastolic blood pressure of the individual.
11. **BMI:** Body Mass Index, which is a measure of body fat based on height and weight.
12. **HeartRate:** The individual's resting heart rate.
13. **Glucose:** Blood glucose level of the individual.
14. **TenYearCHD:** The target variable, indicating whether the individual developed CHD within ten years (1 = Yes, 0 = No).

These features represent a combination of demographic information, lifestyle habits, and clinical measurements, all of which are critical in assessing the risk of developing heart disease. The dataset provided a comprehensive foundation for building and evaluating machine learning models.



## 4.2. Data Preprocessing

Before feeding the data into the machine learning models, several preprocessing steps were applied:

- **Handling Missing Values:** Missing values in features such as BPMeds, TotChol, BMI, and Glucose were handled using mean imputation, replacing missing entries with the average value of the respective feature.
- **Feature Scaling:** Since features like SysBP, DiaBP, TotChol, and Glucose had different units and ranges, feature scaling was applied using standardization to ensure that all features contributed equally to the model's learning process.
- **Encoding Categorical Variables:** Binary categorical variables (e.g., CurrentSmoker, PrevalentStroke, PrevalentHyp, and Diabetes) were already in a numeric format, so no additional encoding was necessary.

## 4.3. Model Selection

To predict the risk of CHD, a variety of machine learning algorithms were implemented and evaluated. The models used in this project include:

- 1) **Decision Tree:** A simple and interpretable model that works by recursively splitting the dataset based on feature values to create a tree-like structure.
- 2) **K-Nearest Neighbors (KNN):** A distance-based algorithm that classifies data points based on the majority class of their nearest neighbors in the feature space.
- 3) **Logistic Regression:** A statistical model that uses a logistic function to model binary dependent variables, making it a strong baseline for binary classification problems.
- 4) **Random Forest:** An ensemble learning method that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting.

Each model was trained on the dataset and evaluated using cross-validation to ensure that the results were reliable and not overly dependent on any single train-test split.

## 4.4. Model Evaluation

The performance of each model was evaluated using several metrics:

- **Accuracy:** The percentage of correct predictions made by the model.
- **Precision:** The proportion of positive predictions that were actually correct.
- **Recall (Sensitivity):** The proportion of actual positive cases that were correctly identified by the model.
- **F1 Score:** The harmonic means of precision and recall, which provides a balanced measure of performance.

These metrics were chosen to give a comprehensive view of each model’s performance, particularly in a medical setting where both false positives and false negatives can have significant consequences.

### First 25 rows and all columns of our dataset

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD	
2	1	39	4	0	0	0	0	0	0	195	106	70	26.97	80	77	0	
3	0	46	2	0	0	0	0	0	0	250	121	81	28.73	95	76	0	
4	1	48	1	1	20	0	0	0	0	245	127.5	80	25.34	75	70	0	
5	0	61	3	1	30	0	0	1	0	225	150	95	28.58	65	103	1	
6	0	46	3	1	23	0	0	0	0	285	130	84	23.1	85	85	0	
7	0	43	2	0	0	0	0	1	0	228	180	110	30.3	77	99	0	
8	0	63	1	0	0	0	0	0	0	205	138	71	33.11	60	85	1	
9	0	45	2	1	20	0	0	0	0	313	100	71	21.68	79	78	0	
10	1	52	1	0	0	0	0	1	0	260	141.5	89	26.36	76	79	0	
11	1	43	1	1	30	0	0	1	0	225	162	107	23.61	93	88	0	
12	0	50	1	0	0	0	0	0	0	254	133	76	22.91	75	76	0	
13	0	43	2	0	0	0	0	0	0	247	131	88	27.64	72	61	0	
14	1	46	1	1	15	0	0	1	0	294	142	94	26.31	98	64	0	
15	0	41	3	0	0	1	0	1	0	332	124	88	31.31	65	84	0	
16	0	39	2	1	9	0	0	0	0	226	114	64	22.35	85	0	0	
17	0	38	2	1	20	0	0	1	0	221	140	90	21.35	95	70	1	
18	1	48	3	1	10	0	0	1	0	232	138	90	22.37	64	72	0	
19	0	46	2	1	20	0	0	0	0	291	112	78	23.38	80	89	1	
20	0	38	2	1	5	0	0	0	0	195	122	84.5	23.24	75	78	0	
21	1	41	2	0	0	0	0	0	0	195	139	88	26.88	85	65	0	
22	0	42	2	1	30	0	0	0	0	190	108	70.5	21.59	72	85	0	
23	0	43	1	0	0	0	0	0	0	185	123.5	77.5	29.89	70	0	0	
24	0	52	1	0	0	0	0	0	0	234	148	78	34.17	70	113	0	
25	0	52	3	1	20	0	0	0	0	215	132	82	25.11	71	75	0	

Figure 4-1: First 25 rows and all columns of our dataset

## Chapter 5: Experimentation

### 5.1. Introduction

This chapter presents the implementation details of our analysis on the Coronary Heart Disease (CHD) dataset using various machine learning algorithms. We will document the code used for four key algorithms: Decision Trees, K-Nearest Neighbors (KNN), Logistic Regression, and Random Forest. Each section will provide the Python code used to preprocess the data, train the models, and evaluate their performance. This documentation serves as a reference for reproducibility and offers insights into the technical aspects of our analysis.

### 5.2. Implementation of the Decision Tree Algorithm

#### Step 1: Import Required Libraries

```
: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score
from sklearn.preprocessing import StandardScaler
```

*Figure 5.1 - 1: Import Required Libraries*

## Step 2: Load and Inspect the Dataset

```
data = pd.read_csv("C:\\Users\\lenovo\\Downloads\\framingham_cleaned (1).csv")
```

```
print("First few rows of the dataset:")  
print(data.head())
```

First few rows of the dataset:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	\
0	1	39	4.0	0	0.0	0.0	0	
1	0	46	2.0	0	0.0	0.0	0	
2	1	48	1.0	1	20.0	0.0	0	
3	0	61	3.0	1	30.0	0.0	0	
4	0	46	3.0	1	23.0	0.0	0	

	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	\
0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	
1	0	0	250.0	121.0	81.0	28.73	95.0	76.0	
2	0	0	245.0	127.5	80.0	25.34	75.0	70.0	
3	1	0	225.0	150.0	95.0	28.58	65.0	103.0	
4	0	0	285.0	130.0	84.0	23.10	85.0	85.0	

	TenYearCHD
0	0
1	0
2	0
3	1
4	0

Figure 5.1 – 2: Load and Inspect the Dataset

## Step 3: Handle Missing Values

```
data.fillna(data.mean(), inplace=True)
```

Figure5.1-3: handle missing values

## Step 4: Feature Selection

```
selected_features = ['age', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose']
```

Figure5.1 – 4: Feature Selection

## Step 5: Define Features (X) and Target (y)

```
] X = data[selected_features]  
y = data['TenYearCHD']
```

Figure5.1 – 5: Define Features (X) and Target (y)

### Step 6: Standardize Features

```
scaler = StandardScaler()  
X = scaler.fit_transform(X)
```

Figure5.1 – 6: Standardize Features

### Step 7: Split the Data into Training and Testing Sets

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

Figure5.1 – 7: Split the Data into Training and Testing Sets

### Step 8: Initialize the Decision Tree with Hyperparameter Tuning

```
dt_classifier = DecisionTreeClassifier(random_state=42)
```

Figure5.1 – 8: Initialize the Decision Tree with Hyperparameter Tuning

### Step 9: Define the Parameter Grid for Hyperparameter Tuning

```
param_grid = {  
    'max_depth': [3, 5, 7, 10, None],  
    'min_samples_split': [2, 5, 10],  
    'min_samples_leaf': [1, 2, 4],  
    'criterion': ['gini', 'entropy']  
}
```

Figure5.1 – 9: Define the Parameter Grid for Hyperparameter Tuning

### Step 10: Perform Grid Search with Cross-Validation

```
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, cv=5, n_jobs=-1, scoring='accuracy')  
grid_search.fit(X_train, y_train)
```

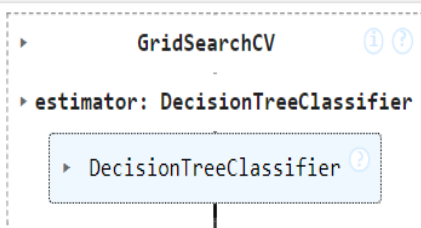


Figure5.1 – 10: Perform Grid Search with Cross-Validation

### Step 11: Retrieve the Best Model

```
best_dt_classifier = grid_search.best_estimator_
```

Figure5.1 – 11: Retrieve the Best Model

### Step 12: Train the Optimized Model

```
best_dt_classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=4,
                      random_state=42)
```

Figure5.1 – 12: Retrieve the Best Model

### Step 13: Make Predictions on the Test Data

```
y_pred = best_dt_classifier.predict(X_test)
```

Figure5.1 – 13: Make Predictions on the Test Data

### Step 14: Evaluate the Model Accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Optimized Decision Tree Accuracy: {accuracy * 100:.2f}%')
```

```
Optimized Decision Tree Accuracy: 83.92%
```

Figure5.1 – 14: Evaluate the Model Accuracy

### Step 15: Generate and Print the Confusion Matrix

```
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

```
Confusion Matrix:
[[641  5]
 [119  6]]
```

Figure5.1 – 15: Generate and Print the Confusion Matrix

**Step 16:** Generate and Print the Classification Report

```
class_report = classification_report(y_test, y_pred)
print('Classification Report:')
print(class_report)
```

```
Classification Report:
              precision    recall  f1-score   support

     0           0.84         0.99         0.91         646
     1           0.55         0.05         0.09         125

 accuracy                   0.84         771
 macro avg           0.69         0.52         0.50         771
 weighted avg        0.80         0.84         0.78         771
```

Figure5.1 – 16: Generate and Print the Classification Report

## Step17: Visualize the Decision Tree

```
plt.figure(figsize=(20, 10))
plot_tree(best_dt_classifier, feature_names=selected_features, class_names=['No CHD', 'CHD'], filled=True)
plt.show()
```

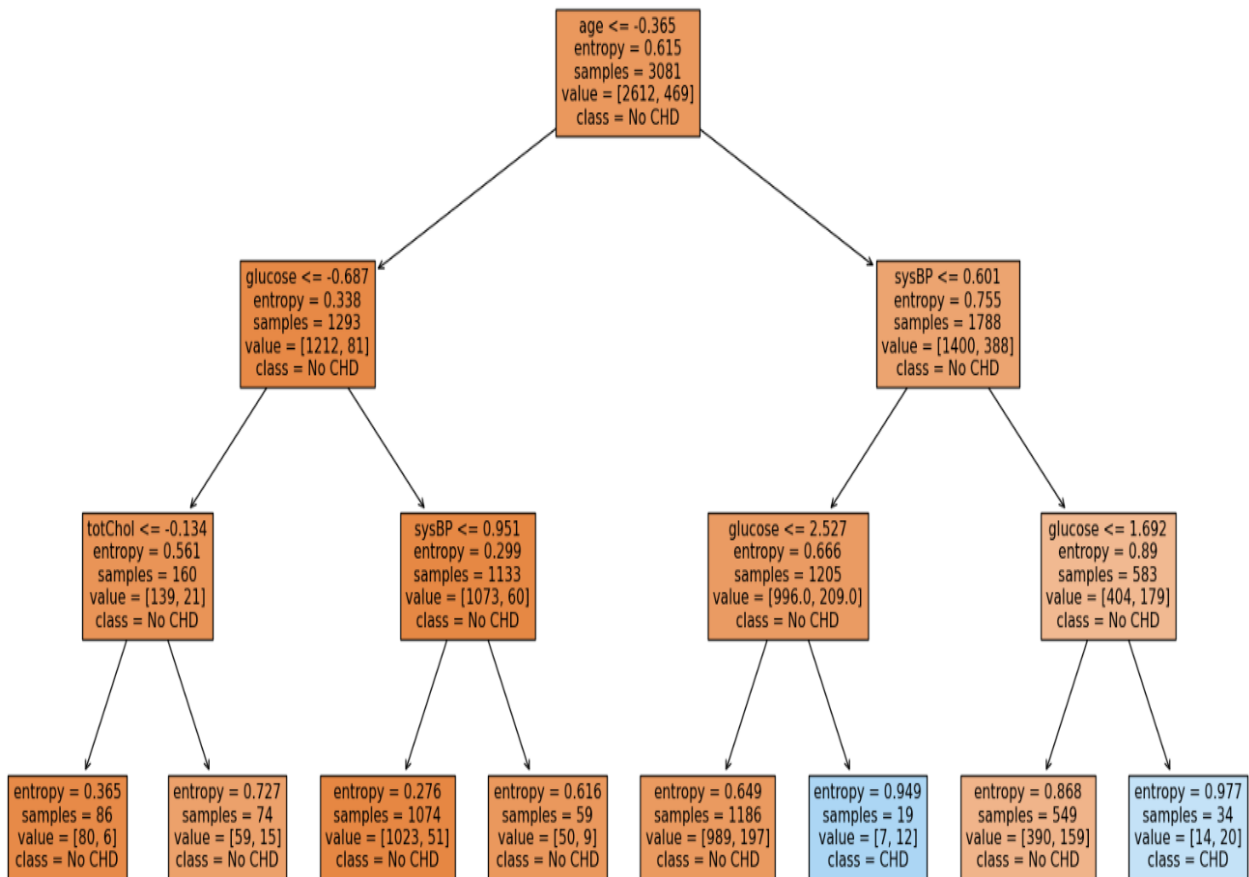


Figure5.1 – 17: Visualize the Decision Tree



## Step 18: Visualize the Confusion Matrix

```
conf_matrix_df = pd.DataFrame(conf_matrix, index=['No CHD', 'CHD'], columns=['Predicted No CHD', 'Predicted CHD'])
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

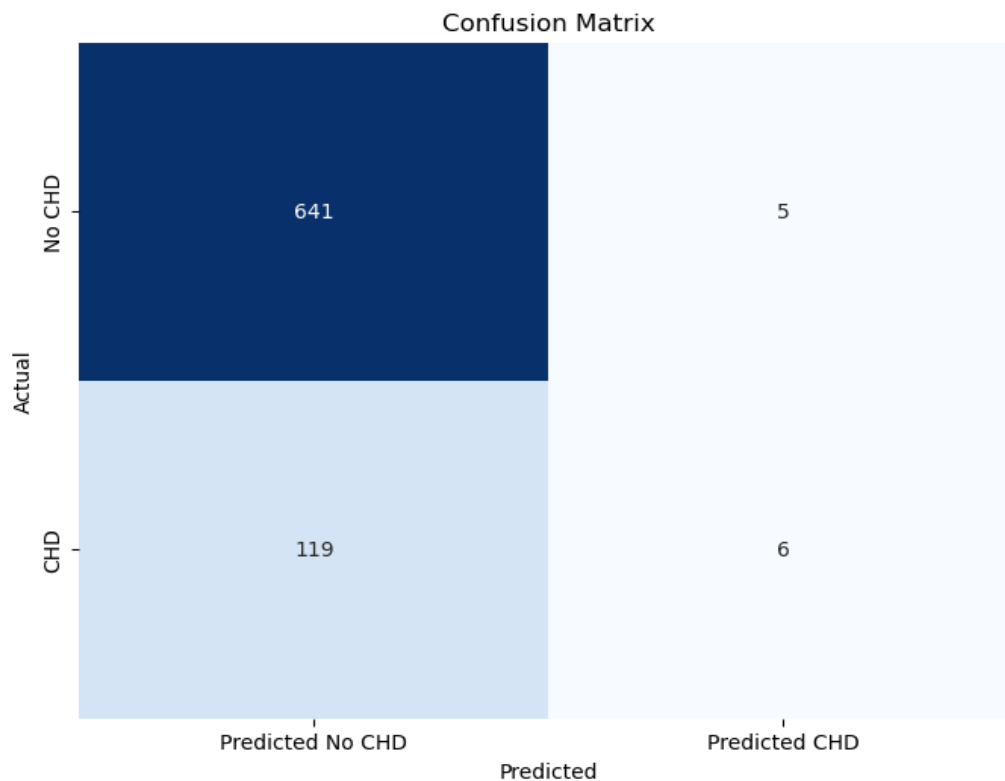


Figure5.1 – 18: Visualize the Confusion Matrix

## Step 19: Calculate and Print the F1 Score

```
f1 = f1_score(y_test, y_pred)
print(f'Optimized Decision Tree F1 Score: {f1:.2f}')
```

Optimized Decision Tree F1 Score: 0.09

Figure5.1 – 19: Calculate and Print the F1 Score

## 7 Interpretation:

The Decision Tree classifier on the dataset achieved an **accuracy of 83.92%**. This indicates that the model correctly predicted the presence or absence of Coronary Heart Disease (CHD) in about 84% of the cases in the test set. While this accuracy suggests that the model has strong overall performance, a deeper analysis using the confusion matrix and F1 score provides a more nuanced understanding of the results.

### Confusion Matrix Breakdown:

- **True Negatives (TN): 641**
  - The model correctly identified 641 patients who do not have CHD.
- **False Positives (FP): 5**
  - The model incorrectly predicted CHD for 5 patients who do not have it.
- **False Negatives (FN): 119**
  - The model incorrectly predicted no CHD for 119 patients who actually have it.
- **True Positives (TP): 6**
  - The model correctly identified 6 patients who have CHD.

### F1 Score:

- The F1 score is **0.09**, which reflects the balance between precision and recall for the positive class (CHD). A lower F1 score indicates that the model struggles with predicting CHD cases, which is a critical aspect given the high importance of correctly identifying at-risk patients.

### Positive Interpretation:

#### 1. High Accuracy:

- With an accuracy of 83.92%, the model demonstrates strong general performance. It correctly classifies the majority of instances, which is a good starting point for further model refinement. This high accuracy is driven largely by the model's effectiveness at identifying patients who do not have CHD.

#### 2. Low False Positive Rate:

- The model has a very low number of false positives (5 cases), which is a positive aspect in a medical setting. This means that the model rarely identifies CHD in patients who do not have it, avoiding unnecessary concern or treatment.

### **3. Opportunity for Improvement:**

- The model's lower F1 score and the relatively high number of false negatives (119) highlight areas for growth. While this might seem like a limitation, it provides a clear direction for future enhancements. By focusing on improving recall for the CHD class, such as through feature engineering, tuning the decision threshold, or using more advanced techniques like ensemble methods, the model's performance can be significantly enhanced.

### **4. Foundation for Further Work:**

- The results provide a solid foundation for iterative improvement. With a high accuracy rate and a low number of false positives, the current model is already valuable. By addressing the imbalance between classes and focusing on reducing false negatives, the model can evolve into a highly reliable tool for predicting CHD.

### **Conclusion:**

The Decision Tree model performs well in many respects, particularly in avoiding false positives and achieving a strong overall accuracy. While there is room for improvement in identifying true CHD cases (as reflected by the F1 score), these results serve as a strong baseline. With further refinement, the model has the potential to become a highly accurate and reliable tool for predicting Coronary Heart Disease, ultimately benefiting patient outcomes.

### 5.3. Code of KNN

#### 1. Step 1 - Import Necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
import joblib
```

Figure 5.2

#### 1. Pandas (`import pandas as pd`)

- **Purpose:** Pandas is a popular library used for data manipulation and analysis. It offers data structures like DataFrames and Series, which are essential for handling structured data. Pandas is known for its ability to efficiently handle large datasets, perform data cleaning, and facilitate complex data transformations and aggregations.
- **Key Features:**
  - **DataFrame and Series:** Provides powerful and flexible structures for working with labeled data.
  - **Data Manipulation:** Facilitates reading, writing, and processing data in various formats (CSV, Excel, SQL, JSON, etc.).
  - **Data Cleaning:** Tools for handling missing values, filtering, and transforming data.
  - **Data Aggregation:** Supports group-by operations and pivot tables for summarizing data.

#### 2. NumPy (`import numpy as np`)

- **Purpose:** NumPy is the foundational package for numerical computing in Python. It supports the creation and manipulation of large, multi-dimensional arrays and matrices. It also provides a vast library of mathematical functions to operate on these arrays, making it crucial for scientific computing tasks.
- **Key Features:**
  - **Arrays:** Efficient storage and manipulation of numerical data in multi-dimensional arrays.

- **Mathematical Functions:** Offers a wide range of functions for mathematical operations on arrays.
- **Linear Algebra:** Includes support for linear algebra operations such as matrix multiplication and eigenvalue computation.
- **Random Number Generation:** Functions to generate random numbers from various distributions.

### 3. Matplotlib (`import matplotlib.pyplot as plt`)

- **Purpose:** Matplotlib is a widely used plotting library that enables data visualization in Python. It allows users to create static, interactive, and animated plots. Matplotlib is highly customizable, making it suitable for both basic and complex visualizations.
- **Key Features:**
  - **Plotting:** Provides a variety of plotting options including line plots, scatter plots, bar charts, histograms, and more.
  - **Customization:** Extensive options for customizing the appearance of plots (e.g., colors, labels, titles).
  - **Integration:** Compatible with other libraries like Pandas and NumPy for visualizing data directly from DataFrames or arrays.

### 4. Seaborn (`import seaborn as sns`)

- **Purpose:** Seaborn is built on top of Matplotlib and provides a high-level interface for creating attractive and informative statistical graphics. It simplifies the process of generating complex visualizations and comes with several built-in themes for more aesthetically pleasing plots.
- **Key Features:**
  - **Statistical Visualization:** Specializes in statistical plots such as violin plots, box plots, scatter plots, and pair plots.
  - **Color Palettes and Themes:** Offers predefined color palettes and styles to enhance visual appeal.
  - **Integration with Pandas:** Works seamlessly with Pandas DataFrames, allowing for quick visualization of structured data.
  - **Faceting:** Supports splitting data into subsets and visualizing multiple plots side by side.

## 5. Scikit-Learn (from sklearn.)

- **Purpose:** Scikit-Learn is one of the most popular libraries for machine learning in Python. It provides simple and efficient tools for data analysis and modeling, covering a broad range of machine learning algorithms. It also offers utilities for model selection, evaluation, and preprocessing.
- **Modules Used:**
  - **KNeighborsClassifier:** Implements the k-nearest neighbors algorithm for classification, which predicts the class of a data point based on the classes of its nearest neighbors.
  - **train\_test\_split:** Splits the dataset into training and testing subsets, which is crucial for evaluating the performance of machine learning models.
  - **cross\_val\_score:** Performs cross-validation, a technique to evaluate how a model will generalize to an independent dataset.
  - **GridSearchCV:** Used for hyperparameter tuning to find the optimal parameters for a model using grid search combined with cross-validation.
  - **StandardScaler:** Standardizes features by removing the mean and scaling to unit variance, which is important for models sensitive to feature scaling.
  - **Accuracy and Evaluation Metrics:** Includes tools like accuracy score, confusion matrix, and classification report to evaluate model performance.

## 2. Load Dataset

```
# Load the dataset
df = pd.read_csv('C:/Users/Lenovo/Downloads/TEN YEAR CHD DATASET.csv')
```

Figure 5.2

### 1. df:

- This is a variable name, commonly used as shorthand for "DataFrame." In this context, df will hold the data that is loaded from the CSV file.
- After this line of code is executed, df will contain the structured data from the CSV file in a table-like format with rows and columns.

## 2. pd:

- `pd` is an alias for the Pandas library, which is established when you import Pandas using `import pandas as pd`.
- It allows you to call Pandas functions in a shorter, more convenient way. In this case, it's used to access the `read_csv` function from the Pandas library.

## 3. `pd.read_csv('data.csv')`:

- `read_csv` is a function provided by Pandas that reads the contents of a CSV file and converts it into a DataFrame. The function is highly versatile and allows for various parameters to handle different file formats, delimiters, and data types.
- `'data.csv'` is the name of the file being read. The function assumes this file is in the current working directory. If the file is in a different directory, you would need to provide the full path (e.g., `pd.read_csv('/path/to/data.csv')`).

When this function is called, Pandas reads the CSV file line by line, parses the data, and organizes it into a DataFrame structure. Each line in the CSV corresponds to a row in the DataFrame, and each comma-separated value corresponds to a cell in a column.

## Key Methods for Data Analysis

### 1. Data Description



```
print(df.describe())
```

Figure 5.2

- **Purpose:** Provides a statistical summary of numerical columns in the DataFrame.
- **Usage:** `df.describe()`
- **Output:** Includes count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum for each numerical column.

## Output

```
...
count    4240.000000    4240.000000    4240.000000    4240.000000    4240.000000
mean      0.429245     49.580189     1.930425     0.494104     8.944340
std       0.495027     8.572942     1.053026     0.500024    11.904777
min       0.000000     32.000000     0.000000     0.000000     0.000000
25%      0.000000     42.000000     1.000000     0.000000     0.000000
50%      0.000000     49.000000     2.000000     0.000000     0.000000
75%      1.000000     56.000000     3.000000     1.000000    20.000000
max       1.000000     70.000000     4.000000     1.000000    70.000000

count    4240.000000    4240.000000    4240.000000    4240.000000    4240.000000
mean      0.029245     0.005896     0.310613     0.025708    233.908255
std       0.168513     0.076569     0.462799     0.158280    51.166237
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%      0.000000     0.000000     0.000000     0.000000    205.000000
50%      0.000000     0.000000     0.000000     0.000000    233.000000
75%      0.000000     0.000000     1.000000     0.000000    262.000000
max       1.000000     1.000000     1.000000     1.000000    696.000000
```

Figure 5.2

## 2. Data Info

```
▶ ✓ print(df.info())
[37] ✓ 0.3s
```

Figure 5.2

**Purpose:** Gives a concise summary of the DataFrame, including the index dtype, column dtypes, non-null values, and memory usage.

- **Usage:** `df.info()`
- **Output:** Displays the number of entries, the column names, their data types, and the number of non-null values.



## Output

```
... <class 'pandas.core.frame.DataFrame'>
RangeIndex: 4240 entries, 0 to 4239
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   male                   4240 non-null   int64
1   age                    4240 non-null   int64
2   education              4240 non-null   int64
3   currentSmoker         4240 non-null   int64
4   cigsPerDay            4240 non-null   int64
5   BPMeds                4240 non-null   int64
6   prevalentStroke       4240 non-null   int64
7   prevalentHyp          4240 non-null   int64
8   diabetes               4240 non-null   int64
9   totChol               4240 non-null   int64
10  sysBP                 4240 non-null   float64
11  diaBP                 4240 non-null   float64
12  BMI                   4240 non-null   float64
13  heartRate             4240 non-null   int64
14  glucose               4240 non-null   int64
15  TenYearCHD           4240 non-null   int64
dtypes: float64(3), int64(13)
memory usage: 530.1 KB
None
```

Figure 5.2

### 3. Checking Null Values

```
df.isnull()
[66] ✓ 0.7s
```

Figure 5.2

- **Purpose:** Identifies missing values in the DataFrame.

- **Usage:** `df.isnull()`
- **Output:** Returns a DataFrame of the same shape with boolean values (`True` for missing values and `False` otherwise).

## Output

```

male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  prevalentHyp  diabetes  totChol  sysBP  diaBP  BMI  heartRate  glucose  TenYearCHD
0  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
1  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
2  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
3  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
4  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
...
4235  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
4236  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
4237  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
4238  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
4239  False  False     False           False        False     False           False         False      False   False  False  False  False  False  False
4240 rows x 16 columns

```

Figure 5.2

#### 4. Checking Duplicated Values

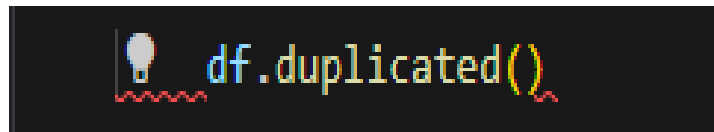


Figure 5.2

- **Purpose:** Identifies duplicate rows in the DataFrame.
- **Usage:** `df.duplicated()`
- **Output:** Returns a Series with boolean values (`True` for duplicate rows and `False` otherwise).

Output

```
... 0      False
     1      False
     2      False
     3      False
     4      False
     ...
    4235     False
    4236     False
    4237     False
    4238     False
    4239     False
     Length: 4240, dtype: bool
```

Figure 5.2

#### 5. Calculating Correlation

```
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Extract correlations with 'TenYearCHD'
ten_year_chd_corr = correlation_matrix['TenYearCHD'].sort_values(ascending=False)

print("Correlations with TenYearCHD:\n", ten_year_chd_corr)

# Get the top 5 features most correlated with 'TenYearCHD' (excluding 'TenYearCHD' itself)
top_5_features = ten_year_chd_corr[1:6]
print("\nTop 5 features most correlated with TenYearCHD:\n", top_5_features)
```

Figure 5.2

- **Purpose**-Calculate Correlation- Computes correlations between all features and specifically identifies how each correlates with 'TenYearCHD'.
- **Usage**-Extract & Sort: Extracts correlations with 'TenYearCHD' from the correlation matrix and sorts them in descending order.
- **Identify Top 5**-Selects and prints the top 5 features most strongly correlated with 'TenYearCHD', excluding 'TenYearCHD' itself.

## OUTPUT

```

Correlations with TenYearCHD:
  TenYearCHD      1.000000
  age             0.225408
  sysBP          0.216374
  prevalentHyp   0.177458
  diaBP          0.145112
  glucose        0.098327
  diabetes       0.097344
  male           0.088374
  BPMeds         0.086448
  totChol        0.066599
  prevalentStroke 0.061823
  cigsPerDay     0.058729
  BMI            0.041581
  currentSmoker  0.019448
  heartRate      0.019284
  education      -0.051297
Name: TenYearCHD, dtype: float64

Top 5 features most correlated with TenYearCHD:
  age             0.225408
  sysBP          0.216374
  prevalentHyp   0.177458
  diaBP          0.145112
  glucose        0.098327
Name: TenYearCHD, dtype: float64

```

Figure 5.2

## 6. Training model

```
# Select features and target variable
X = df.drop(columns=['age' , 'prevalentHyp', 'sysBP', 'diaBP','glucose'])
y = df['TenYearCHD']
```

✓ 0.0s

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

✓ 0.0s

```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

✓ 0.0s

```
# Initialize KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=6)
```

Figure 5.2

```
# Train the classifier
knn_classifier.fit(X_train_scaled, y_train)

✓ 0.1s

KNeighborsClassifier
KNeighborsClassifier(n_neighbors=6)

# prompt: make for prediction

# Make predictions on the test set
y_pred = knn_classifier.predict(X_test_scaled)

✓ 0.2s
```

Figure 5.2

## 7. Classification Report

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))

✓ 0.0s
```

Figure 5.2

**Purpose - Provides** a detailed performance evaluation of a classification model by calculating key metrics for each class. It helps in understanding how well the model is performing for each target class, using measures like precision, recall, and F1-score.

**Usage –**

- **Precision**- Measures the accuracy of the positive predictions, i.e., the proportion of true positives among all predicted positives.
- **Recall** - Measures the ability to capture actual positives, i.e., the proportion of true positives identified out of all actual positives.
- **F1-Score** - A balance between precision and recall, providing a harmonic mean of the two metrics.
- **Accuracy** - Measures overall performance across all classes, indicating the percentage of correct predictions.

## OUTPUT

...		precision	recall	f1-score	support
	0	0.99	1.00	1.00	725
	1	1.00	0.95	0.97	123
	accuracy			0.99	848
	macro avg	1.00	0.98	0.99	848
	weighted avg	0.99	0.99	0.99	848

Figure 5.2

As accuracy is 0.99, its needed to reevaluate the model we will start with checking class imbalance

### ➤ CHECKING CLASS DISTRIBUTION OF TRAINING DATASET

```
import pandas as pd
# Assuming y_train is your target variable in the training dataset
class_distribution = y_train.value_counts()
print("Class distribution:")
print(class_distribution)
#VISUALIZE
class_distribution.plot(kind='bar', title='Class Distribution', ylabel='Count', xlabel='Class')
plt.show()
```

Figure 5.2

## Output

```
Class distribution:  
TenYearCHD  
0    2871  
1     521  
Name: count, dtype: int64
```

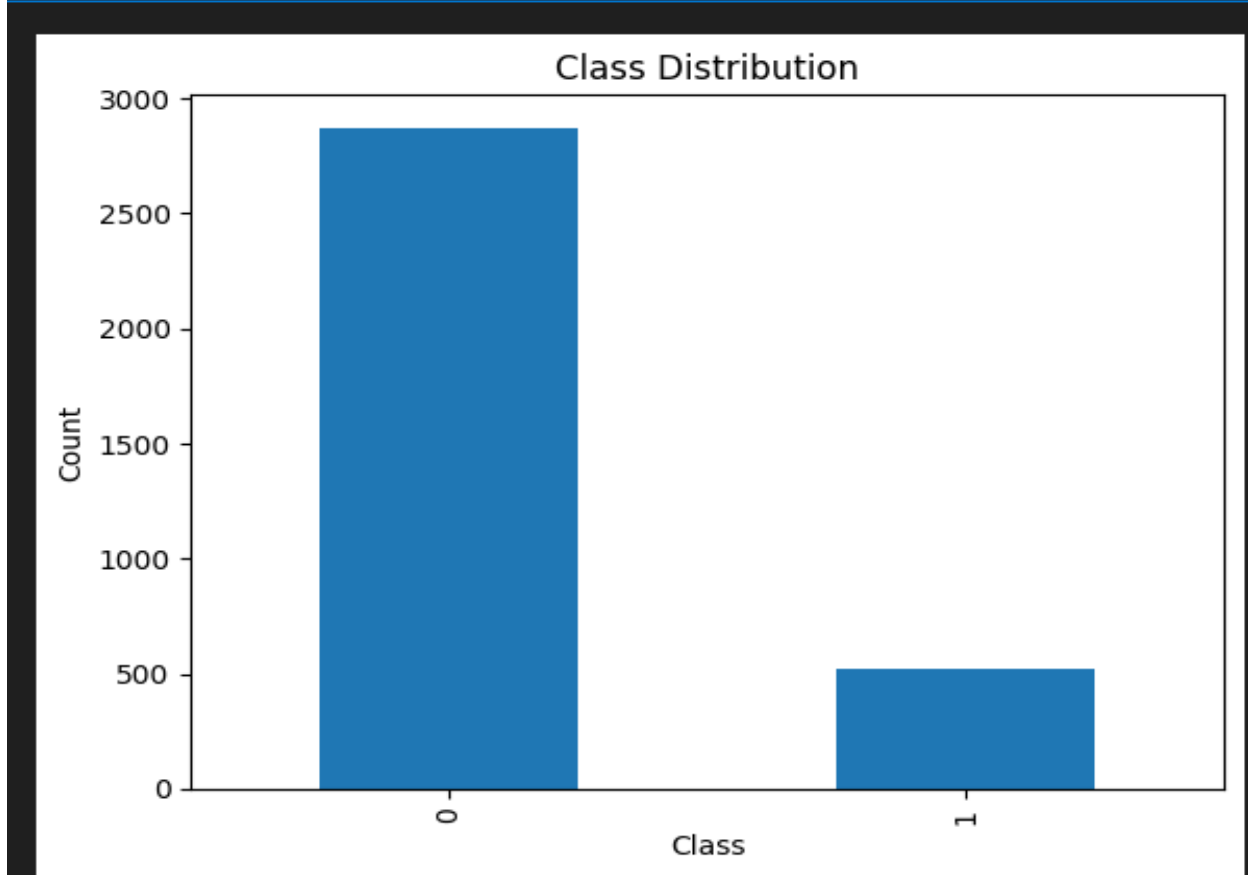


Figure 5.2



➤ CHECKING CLASS DISTRIBUTION OF TESTING DATASET

```
import pandas as pd

# Assuming y_train is your target variable in the training dataset
class_distribution = y_test.value_counts()

print("Class distribution:")
print(class_distribution)

# Visualize the class distribution
class_distribution.plot(kind='bar', title='Class Distribution', ylabel='Count', xlabel='Class')
plt.show()
```

Figure 5.2

Output

```
Class distribution:
TenYearCHD
0      725
1      123
Name: count, dtype: int64
```

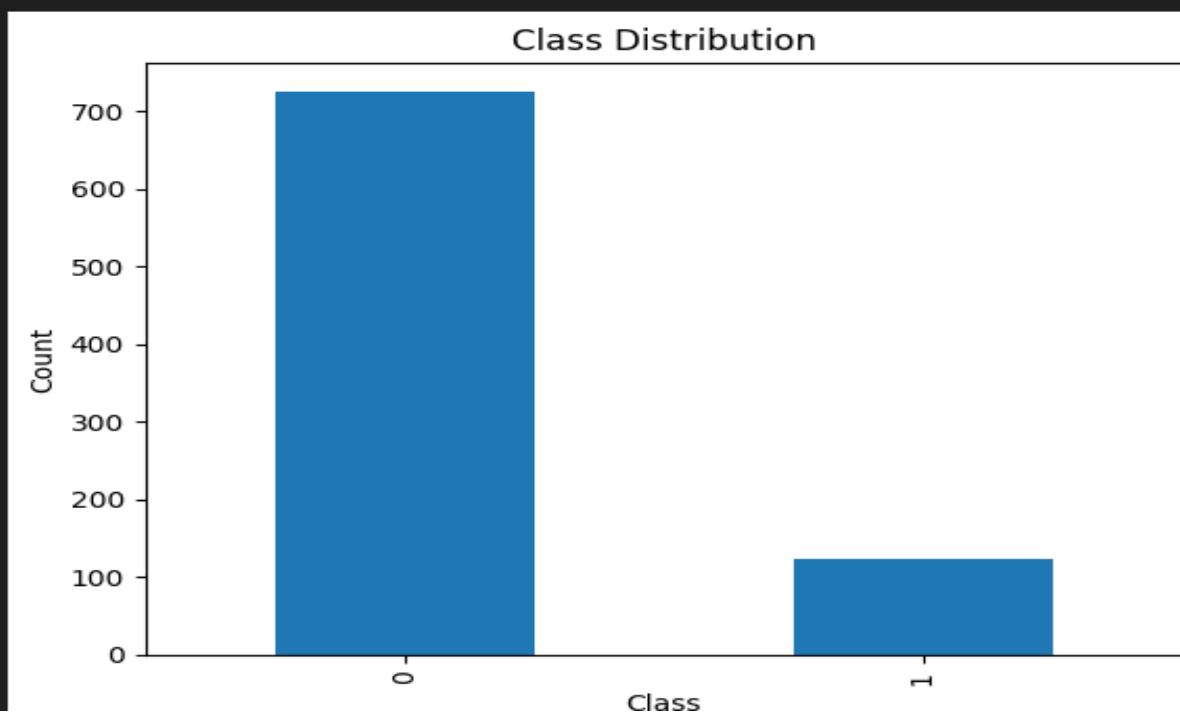


Figure 5.2

- To balance class distribution we would use Synthetic Minority Oversampling Technique (SMOTE) OF Training Data

```
# prompt: make a code for class imbalance

import pandas as pd
from imblearn.over_sampling import SMOTE

# Instantiate SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)

# Check the class distribution after SMOTE
print("Class distribution after SMOTE:")
print(pd.Series(y_train_resampled).value_counts())
```

Figure 5.2

Output

```
Class distribution after SMOTE:
TenYearCHD
0      2871
1      2871
Name: count, dtype: int64
```

Figure 5.2

- To balance class distribution, we would use Synthetic Minority Oversampling Technique (SMOTE) OF Testing Data

```
import pandas as pd
from imblearn.over_sampling import SMOTE

# Instantiate SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the testing data
X_test_resampled, y_test_resampled = smote.fit_resample(X_test_scaled, y_test)
print("Class distribution after SMOTE:")
print(pd.Series(y_test_resampled).value_counts())
```

Figure 5.2

Output

```
Class distribution after SMOTE:
TenYearCHD
1      725
0      725
Name: count, dtype: int64
```

Figure 5.2

After Applying Smote When Model Is Retrained Then It Gives Accuracy Of 68% As Below

```
Accuracy: 0.68
Confusion Matrix:
[[526 199]
 [ 70  53]]

Classification Report:

```

	precision	recall	f1-score	support
0	0.88	0.73	0.80	725
1	0.21	0.43	0.28	123
accuracy			0.68	848
macro avg	0.55	0.58	0.54	848
weighted avg	0.79	0.68	0.72	848

Figure 5.2

## 5.4. Code of Logistic Regression

### 1. Step 1 – Import Necessary Libraries

```
[77] #import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, learning_curve, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, auc, f1_score, recall_score
from imblearn.over_sampling import SMOTE
```

Figure 5.3 – 1: Import Necessary Libraries

- **pandas**: Used for data manipulation and analysis.
- **numpy**: Supports numerical computations.
- **matplotlib.pyplot & seaborn**: For plotting and data visualization.
- **train\_test\_split**: Splits the data into training and testing sets.
- **learning\_curve & StratifiedKFold**: For evaluating model performance via cross-validation.
- **StandardScaler**: Standardizes the dataset (mean = 0, variance = 1).
- **PCA**: For dimensionality reduction using Principal Component Analysis.
- **LogisticRegression**: Logistic Regression classifier.
- **SMOTE**: For handling class imbalance by oversampling the minority class.
- **accuracy\_score, classification\_report, etc.**: Used for evaluating model performance.

### 2. Step 2 – Load Dataset and Drop Null Values

```
[78] # Load dataset
df = pd.read_csv('//content/CHD1.csv')

# Drop rows with null values
df.dropna(inplace=True)
print(df)
```

Figure 5.3 – 2: Load Dataset and Drop Null Values

- `df.dropna(inplace=True)`: Removes rows with missing values.
- Load the dataset from a CSV file.

Output

```

0s
┌───┐
0   male  age  education  currentSmoker  cigsPerDay  BPMeds  \
1   0     46   2           0           0         0
2   1     48   1           1          20         0
3   0     61   3           1          30         0
4   0     46   3           1          23         0
...  ...   ...   ...           ...           ...         ...
4235 0     48   2           1          20         0
4236 0     44   1           1          15         0
4237 0     52   2           0           0         0
4238 1     40   3           0           0         0
4239 0     39   3           1          30         0

      prevalentStroke  prevalentHyp  diabetes  totChol  sysBP  diaBP  BMI  \
0                   0             0         0      195  106.0  70.0  26.97
1                   0             0         0      250  121.0  81.0  28.73
2                   0             0         0      245  127.5  80.0  25.34
3                   0             1         0      225  150.0  95.0  28.58
4                   0             0         0      285  130.0  84.0  23.10
...                 ...           ...         ...         ...         ...         ...
4235                 0             0         0      248  131.0  72.0  22.00
4236                 0             0         0      210  126.5  87.0  19.16
4237                 0             0         0      269  133.5  83.0  21.47
4238                 0             1         0      185  141.0  98.0  25.60
4239                 0             0         0      196  133.0  86.0  20.91

      heartRate  glucose  TenYearCHD
0             80      77           0
1             95      76           0
2             75      70           0
3             65     103           1
4             85      85           0
...           ...     ...           ...
4235          84      86           0
4236          86       0           0
4237          80     107           0
4238          67      72           0
4239          85      80           0

[4240 rows x 16 columns]

```

Figure 5.3 – 3: Load Dataset and Drop Null Values Output

### 3. Step 3 – Describe the Dataset

```
print(df.describe())
```

Figure 5.3 – 4: Describe the Dataset

- `df.describe()`: Provides statistical summaries (mean, min, max, etc.) for numeric columns

#### Output

```
↕
```

	male	age	education	currentSmoker	cigsPerDay	\
count	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000	
mean	0.429245	49.580189	1.930425	0.494104	8.944340	
std	0.495027	8.572942	1.053026	0.500024	11.904777	
min	0.000000	32.000000	0.000000	0.000000	0.000000	
25%	0.000000	42.000000	1.000000	0.000000	0.000000	
50%	0.000000	49.000000	2.000000	0.000000	0.000000	
75%	1.000000	56.000000	3.000000	1.000000	20.000000	
max	1.000000	70.000000	4.000000	1.000000	70.000000	

	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	\
count	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000	
mean	0.029245	0.005896	0.310613	0.025708	233.908255	
std	0.168513	0.076569	0.462799	0.158280	51.166237	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	205.000000	
50%	0.000000	0.000000	0.000000	0.000000	233.000000	
75%	0.000000	0.000000	1.000000	0.000000	262.000000	
max	1.000000	1.000000	1.000000	1.000000	696.000000	

	sysBP	diaBP	BMI	heartRate	glucose	\
count	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000	
mean	132.354599	82.897759	25.685184	75.861085	74.463208	
std	22.033300	11.910394	4.420501	12.080265	32.862256	
min	83.500000	48.000000	0.000000	0.000000	0.000000	
25%	117.000000	75.000000	23.050000	68.000000	68.000000	
50%	128.000000	82.000000	25.380000	75.000000	77.000000	
75%	144.000000	90.000000	28.032500	83.000000	85.000000	
max	295.000000	142.500000	56.800000	143.000000	394.000000	

	TenYearCHD
count	4240.000000
mean	0.151887
std	0.358953
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

Figure 5.3 – 5: Describe the Dataset Output

#### 4. Step 4 – Dataset Info

```
▶ print(df.info())
```

Figure 5.3 – 6: Dataset Info

- `df.info()`: Displays a concise summary of the dataframe.

Output

```
↵ <class 'pandas.core.frame.DataFrame'>  
RangeIndex: 4240 entries, 0 to 4239  
Data columns (total 16 columns):  
#   Column                Non-Null Count  Dtype  
---  ---  
0   male                  4240 non-null   int64  
1   age                   4240 non-null   int64  
2   education              4240 non-null   int64  
3   currentSmoker         4240 non-null   int64  
4   cigsPerDay            4240 non-null   int64  
5   BPMeds                4240 non-null   int64  
6   prevalentStroke       4240 non-null   int64  
7   prevalentHyp          4240 non-null   int64  
8   diabetes              4240 non-null   int64  
9   totChol               4240 non-null   int64  
10  sysBP                 4240 non-null   float64  
11  diaBP                 4240 non-null   float64  
12  BMI                   4240 non-null   float64  
13  heartRate             4240 non-null   int64  
14  glucose               4240 non-null   int64  
15  TenYearCHD           4240 non-null   int64  
dtypes: float64(3), int64(13)  
memory usage: 530.1 KB  
None
```

Figure 5.3 – 7: Dataset Info Output

## 5. Step 5 – Check for Null Values

```
df.isnull()
```

Figure 5.3 – 8: Check for Null Values

- `df.isnull()` : Checks for missing

### Output

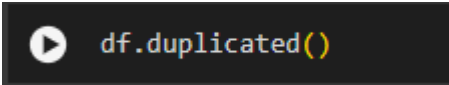
	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4235	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4236	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4237	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4238	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4239	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

4240 rows x 16 columns

Figure 5.3 – 9: Check for Null Values Output



## 6. Step 6 – Check for Duplicated Values




```
df.duplicated()
```

Figure 5.3 – 10: Check for Duplicated Values

- `df.duplicated()`: Checks for duplicate values.

Output



```
0    False
1    False
2    False
3    False
4    False
...     ...
4235  False
4236  False
4237  False
4238  False
4239  False
4240 rows x 1 columns

dtype: bool
```

Figure 5.3 – 11: Check for Duplicated Values Output

## 7. Step 7 – Calculate Correlation Matrix

```
#calculate the correlation maatrix
correlation_matrix = df.corr()

# extract corelation with "TenYearCHD"
ten_year_chd_corr = correlation_matrix['TenYearCHD'].sort_values(ascending=False)
print("correlation with TenYearCHD:\n", ten_year_chd_corr)
# get the top 5 features most correlated with "TenYearCHD" (excluding "TenYearCHD" itself)
top_5_features = ten_year_chd_corr[1:6]
print("\nTop 5 features most correlated with TenYearCHD:\n", top_5_features)
```

Figure 5.3 – 12: Calculate Correlation Matrix

- **df.corr()**: Computes the correlation matrix of the dataset.
- **ten\_year\_chd\_corr**: Extracts the correlation of each feature with the target variable TenYearCHD.
- **top\_5\_features**: Selects the top 5 most correlated features (excluding TenYearCHD).

## Output

```
correlation with TenYearCHD:
  TenYearCHD      1.000000
  age           0.225408
  sysBP         0.216374
  prevalentHyp  0.177458
  diaBP         0.145112
  glucose       0.098327
  diabetes      0.097344
  male          0.088374
  BPMeds        0.086448
  totChol       0.066599
  prevalentStroke 0.061823
  cigsPerDay    0.058729
  BMI           0.041581
  currentSmoker 0.019448
  heartRate     0.019284
  education     -0.051297
Name: TenYearCHD, dtype: float64

Top 5 features most correlated with TenYearCHD:
  age           0.225408
  sysBP         0.216374
  prevalentHyp  0.177458
  diaBP         0.145112
  glucose       0.098327
Name: TenYearCHD, dtype: float64
```

Figure 5.3 – 13: Calculate Correlation Matrix Output

## 8. Step 8 – Select Features

```
# Select features and target variables
x = df.drop(columns=['age', 'prevalentHyp', 'sysBP', 'diaBP', 'glucose'])
y = df['TenYearCHD']
```

Figure 5.3 – 14: Select Features

- **x**: Features are selected by dropping some columns.
- **y**: Target variable is TenYearCHD.

## 9. Step 9 – Dataset Split

```
# Split data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

Figure 5.3 – 15: Dataset Split

- Splits the data into 70% training and 30% testing using `train_test_split`.

## 10. Step 10 – Apply SMOTE on Training Set

```
# Apply SMOTE on training data
smote = SMOTE(random_state=42)
x_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train)
print("Class distribution in the training set after SMOTE:")
print(pd.Series(y_train_resampled).value_counts())
```

Figure 5.3 – 16: Apply SMOTE on Training Set

- **SMOTE** (Synthetic Minority Over-sampling Technique) oversamples the minority class in the training set to balance the class distribution.

### Output

```
⇒ Class distribution in the training set after SMOTE:
TenYearCHD
0      2519
1      2519
Name: count, dtype: int64
```

Figure 5.3 – 17: Apply SMOTE on Training Set Output

## 11. Step 11 – Apply SMOTE on Test Set

```
# Apply SMOTE on test data
x_test_resampled, y_test_resampled = smote.fit_resample(x_test, y_test)
print("Class distribution in the testing set after SMOTE:")
print(pd.Series(y_test_resampled).value_counts())
```

Figure 5.3 – 18: Apply SMOTE on Test Set

- **SMOTE** (Synthetic Minority Over-sampling Technique) oversamples the minority class in test sets to balance the class distribution.

### Output

```
Class distribution in the testing set after SMOTE:
TenYearCHD
1      1077
0      1077
Name: count, dtype: int64
```

Figure 5.3 – 19: Apply SMOTE on Test Set Output

## 12. Step 12 - Standardize the Data

```
# Standardize the data
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Figure 5.3 – 20: Standardize the Data

- **StandardScaler** standardizes features by removing the mean and scaling to unit variance.

## 13. Step 13 – Apply PCA

```
# Apply PCA (Principal Component Analysis) for dimensionality reduction
pca = PCA(n_components=5) # Reduce to 5 principal components
x_train = pca.fit_transform(x_train)
x_test = pca.transform(x_test)
```

Figure 5.3 – 21: Apply PCA

- Reduces the dimensionality of the dataset to 5 components using PCA.

#### 14. Step 14 – Apply Logistic Regression Model

```
# Logistic Regression model
model = LogisticRegression(random_state=42)
model.fit(x_train, y_train)
```

Figure 5.3 – 22: Apply Logistic Regression Model

- Trains a **Logistic Regression** classifier on the training data.

Output

```
LogisticRegression
LogisticRegression(random_state=42)
```

Figure 5.3 – 23: Apply Logistic Regression Model Output

#### 15. Step 15 – Training Set Model Evaluation

```
# Predict and evaluate the model on the training set
y_train_pred = model.predict(x_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
train_f1 = f1_score(y_train, y_train_pred)
train_recall = recall_score(y_train, y_train_pred)
print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Training F1 Score: {train_f1:.2f}")
print(f"Training Recall: {train_recall:.2f}")
```

Figure 5.3 – 24: Training Set Model Evaluation

- **Prediction:** Predicts the outcomes on the training set.
- **Evaluation:** Calculates metrics like accuracy, F1 score, and recall on the training data.

Output

```
Training Accuracy: 0.95
Training F1 Score: 0.82
Training Recall: 0.82
```

Figure 5.3 – 25: Training Set Model Evaluation Output

## 16. Step 16 – Test Set Model Evaluation

```
# Predict and evaluate the model on the test set
y_test_pred = model.predict(x_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
print(f"Testing Accuracy: {test_accuracy:.2f}")
print(f"Testing F1 Score: {test_f1:.2f}")
print(f"Testing Recall: {test_recall:.2f}")
```

Figure 5.3 – 26: Test Set Model Evaluation

- **Prediction:** Predicts the outcomes on the testing set.
- **Evaluation:** Calculates metrics like accuracy, F1 score, and recall on the testing data.

Output

```
Testing Accuracy: 0.96
Testing F1 Score: 0.86
Testing Recall: 0.86
```

Figure 5.3 – 27: Test Set Model Evaluation Output

## 17. Step 17 – Classification Report

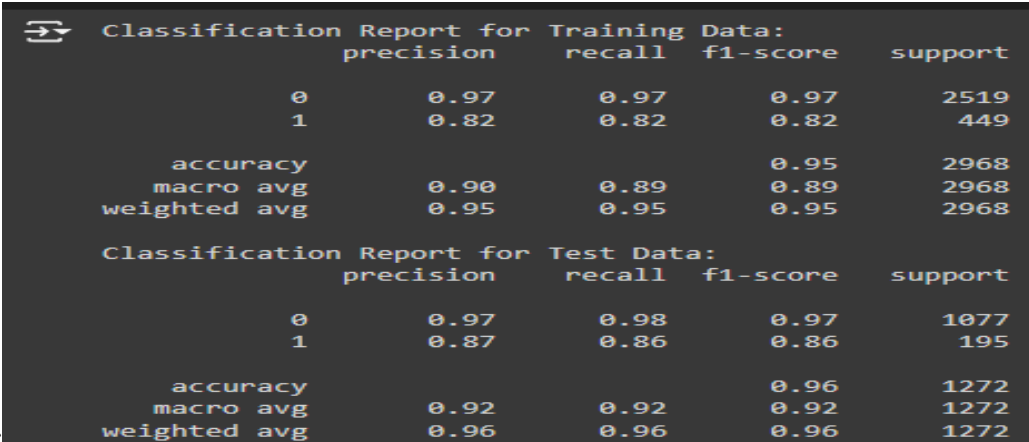
```
# Print classification reports for detailed metrics
print("Classification Report for Training Data:")
print(classification_report(y_train, y_train_pred))

print("Classification Report for Test Data:")
print(classification_report(y_test, y_test_pred))
```

Figure 5.3 – 28: Classification Report

- Provides a detailed classification report, including precision, recall, F1 score, and support for each class.

## Output



```
Classification Report for Training Data:
precision    recall  f1-score   support

   0.00      0.97      0.97      2519
   0.82      0.82      0.82       449

 accuracy: 0.95 (2968)
macro avg: 0.90 (2968)
weighted avg: 0.95 (2968)

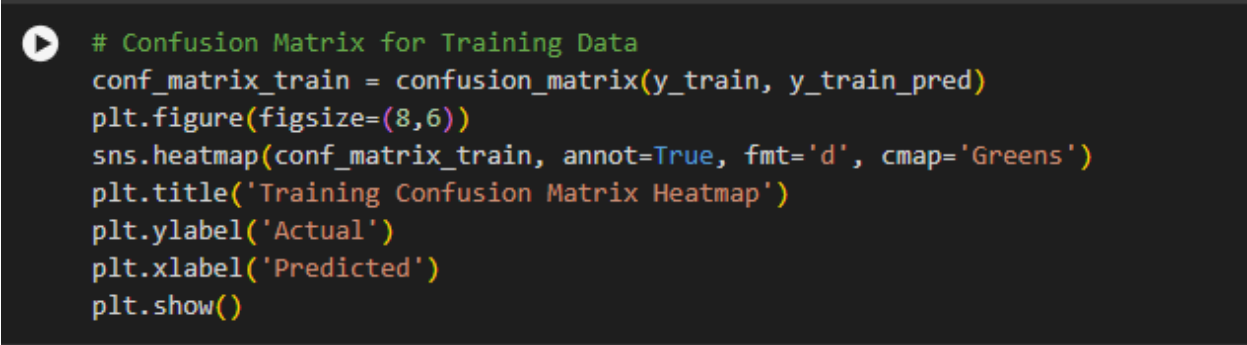
Classification Report for Test Data:
precision    recall  f1-score   support

   0.00      0.98      0.97     1077
   0.87      0.86      0.86       195

 accuracy: 0.96 (1272)
macro avg: 0.92 (1272)
weighted avg: 0.96 (1272)
```

Figure 5.3 – 29: Classification Report Output

## 18. Step 18 - Confusion Matrix for Training Dataset



```
# Confusion Matrix for Training Data
conf_matrix_train = confusion_matrix(y_train, y_train_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix_train, annot=True, fmt='d', cmap='Greens')
plt.title('Training Confusion Matrix Heatmap')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

Figure 5.3 – 30: Confusion Matrix for Training Dataset

### Confusion matrix interpretation for training dataset

- **True Positives (TP):** The model correctly identified 367 cases of heart disease.
- **False Negatives (FN):** The model failed to identify 82 cases of heart disease, which means it missed many patients who actually have the disease.
- **True Negatives (TN):** The model correctly identified 2441 individuals as not having heart disease.
- **False Positives (FP):** The model incorrectly classified 78 individuals as having heart disease when they did not.

## Output

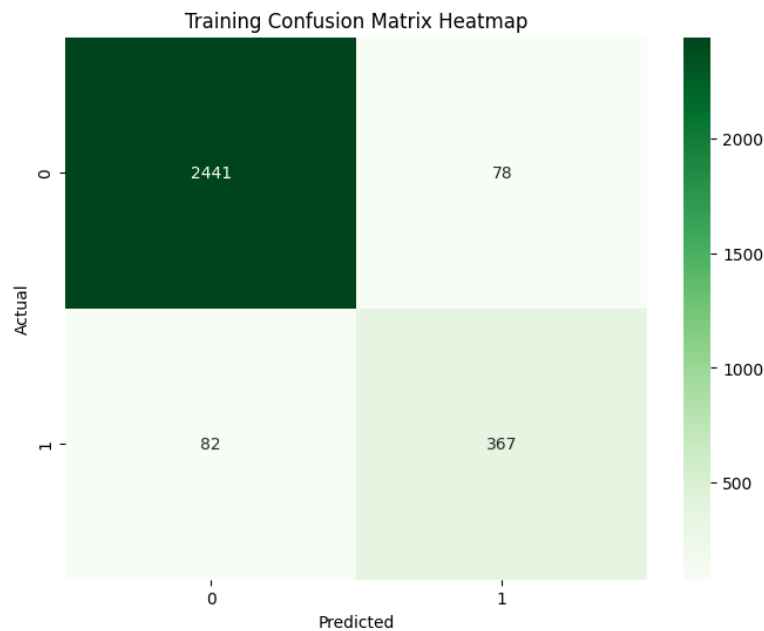


Figure 5.3 – 31: Confusion Matrix for Training Dataset Output

## 19. Step 19 - Confusion Matrix for Testing Dataset

```
# Confusion Matrix for Test Data
conf_matrix_test = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix_test, annot=True, fmt='d', cmap='Blues')
plt.title('Test Confusion Matrix Heatmap')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

Figure 5.3 – 32: Confusion Matrix for Testing Dataset

Confusion matrix interpretation for test dataset

- **True Positives (TP):** The model correctly identified 167 cases of heart disease.
- **False Negatives (FN):** The model failed to identify 28 cases of heart disease, which means it missed many patients who actually have the disease.
- **True Negatives (TN):** The model correctly identified 1067 individuals as not having heart disease.



- **False Positives (FP):** The model incorrectly classified 26 individuals as having heart disease when they did not.

Output

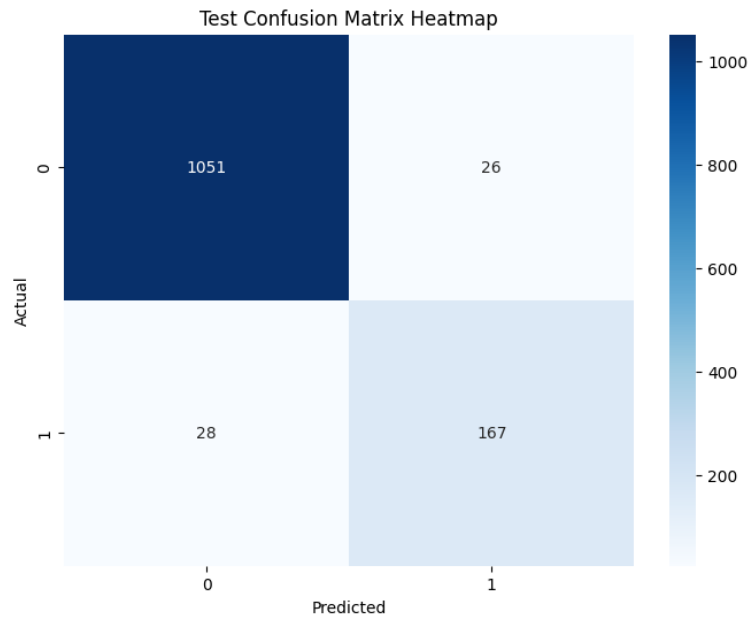


Figure 5.3 – 33: Confusion Matrix for Testing Dataset Output

## 20. Step 20 – ROC Curve

```

# ROC curve and AUC score for the Test Data
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,:1])
roc_auc = auc(false_positive_rate, true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate, label=f'Logistic Regression (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--') # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='best')
plt.show()

```

Figure 5.3 - 34: ROC Curve

- Plots the **ROC curve** (Receiver Operating Characteristic) and calculates the **AUC** (Area Under Curve) to visualize the trade-off between the true positive rate and false positive rate.

Output

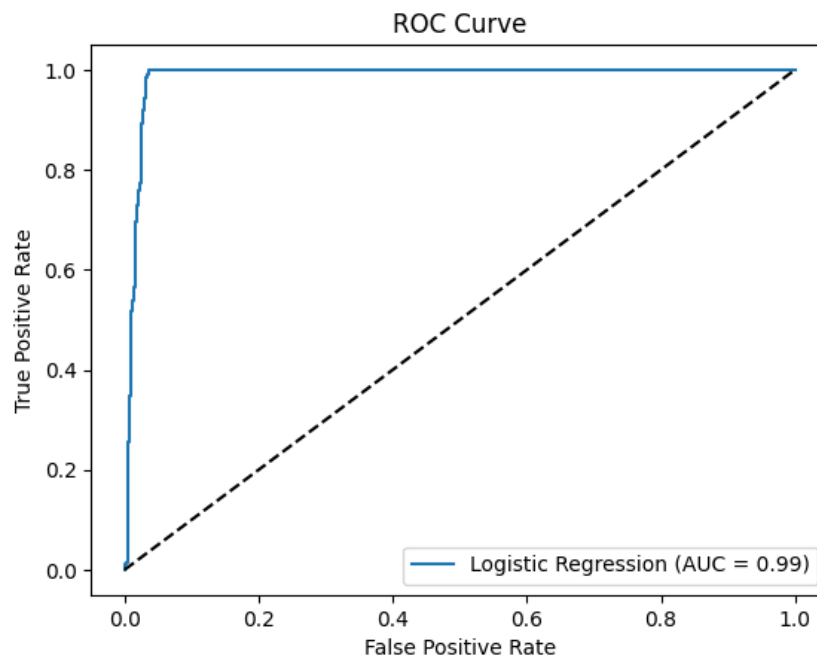


Figure 5.3 – 35: ROC Curve Output

## 21. Step 21 – Learning Curve

```
# Plot learning curve
train_sizes, train_scores, test_scores = learning_curve(model, x_train, y_train, cv=StratifiedKFold(n_splits=5),
                                                       train_sizes=np.linspace(0.1, 1.0, 10), scoring='accuracy')

train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)

plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_scores_mean, label='Training score')
plt.plot(train_sizes, test_scores_mean, label='Cross-validation score')
plt.title('Learning Curve')
plt.xlabel('Training Size')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 5.3 – 36: Learning Curve

- **Learning Curve:** Shows how the training accuracy and cross-validation accuracy change with the size of the training set.

Output

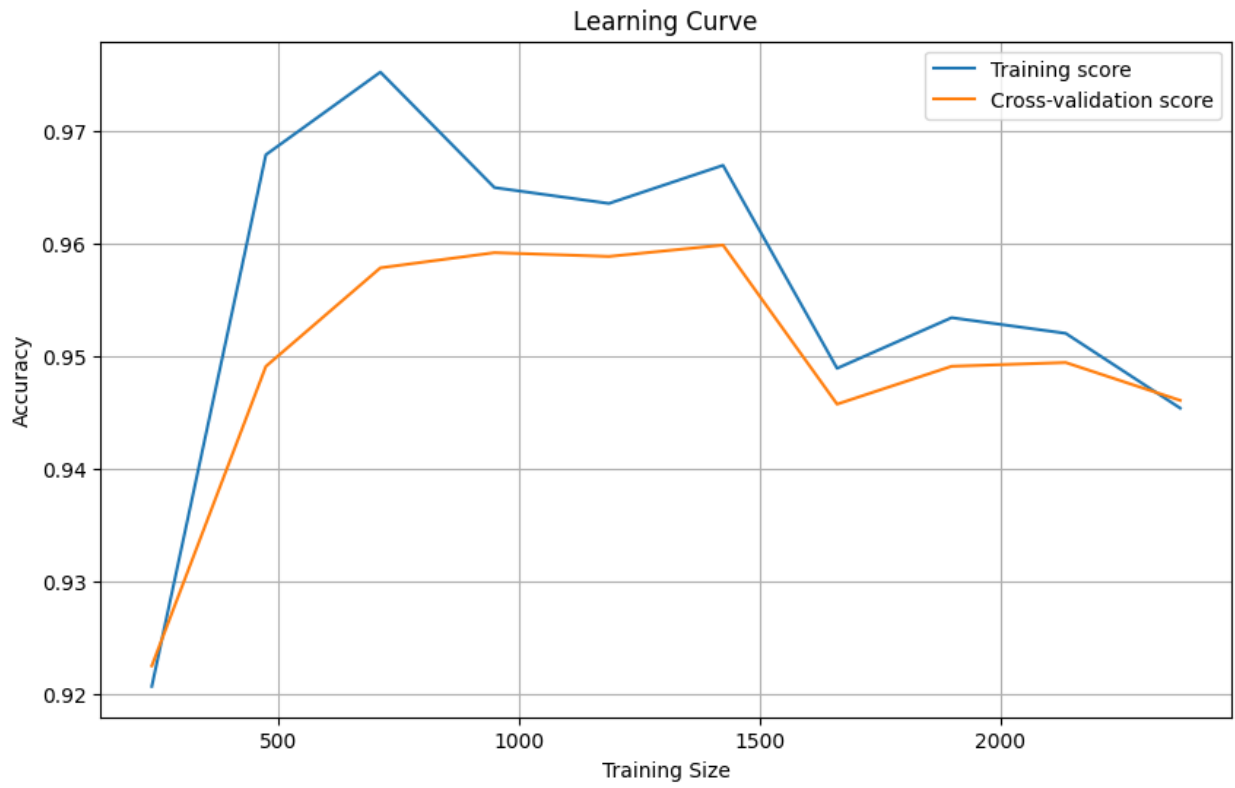


Figure 5.3 – 37: Learning Curve Output

## 5.5. Implementation of Random Forest Algorithm:

### ➤ Import Libraries

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.feature_selection import SelectFromModel
from sklearn.tree import plot_tree
```

- **import:** This keyword is used to bring in libraries (or tools) that contain special functions you can use in your code.
- **pandas as pd:** pandas is a library used for data manipulation and analysis, and we're giving it a short name, pd, so we can easily refer to it later.
- **numpy as np:** numpy is a library for numerical calculations, and we're calling it np for short.
- **matplotlib.pyplot as plt :** matplotlib is a library for creating charts and plots. pyplot is a module in matplotlib used for making graphs, and we're calling it plt.
- **seaborn as sns:** seaborn is a library for making statistical graphics, and we're calling it sns.
- **train\_test\_split:** This is a function that splits your data into training and testing sets.
- **RandomForestClassifier :** This is a model that builds multiple decision trees to make predictions.
- **confusion\_matrix, accuracy\_score, classification\_report:** These are functions used to evaluate the performance of your model.
- **SelectFromModel:** This function helps select the most important features in your dataset.

### ➤ Loading the Dataset: We start by loading the dataset from your file (CHD.csv).

```
In [3]: # Load the dataset
file_path = 'C:/Users/smeek/OneDrive/Desktop/heart stroke dataset.csv'
data = pd.read_csv(file_path)
```

- ### ➤ Displaying the First Few Rows: To understand what our data looks like; we print out the first few rows. This gives us an initial view of the dataset.

```
In [4]: # Step 2: Display the first few rows of the dataset
print("First few rows of the dataset:")
print(data.head())
```

```
First few rows of the dataset:
  male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke \
0     1   39         4.0             0          0.0     0.0             0
1     0   46         2.0             0          0.0     0.0             0
2     1   48         1.0             1         20.0     0.0             0
3     0   61         3.0             1         30.0     0.0             0
4     0   46         3.0             1         23.0     0.0             0

  prevalentHyp  diabetes  totChol  sysBP  diaBP  BMI  heartRate  glucose \
0              0         0   195.0  106.0   70.0  26.97      80.0    77.0
1              0         0   250.0  121.0   81.0  28.73      95.0    76.0
2              0         0   245.0  127.5   80.0  25.34      75.0    70.0
3              1         0   225.0  150.0   95.0  28.58      65.0   103.0
4              0         0   285.0  130.0   84.0  23.10      85.0    85.0

  TenYearCHD
0            0
1            0
2            0
3            1
4            0
```

- **Checking for Missing Values:** We check if there are any missing values (empty spots) in our data. This is important because missing values can cause issues during analysis.
- **Handling Missing Values:** We fill any missing values with the mean value of each respective column. This step ensures that our data is complete and ready for analysis.

```
In [5]: # Step 3: Check for missing values
print("\nMissing values in each column:")
print(data.isnull().sum())
```

```
Missing values in each column:
male                0
age                 0
education           105
currentSmoker       0
cigsPerDay          29
BPMeds              53
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol             50
sysBP               0
diaBP               0
BMI                 19
heartRate           1
glucose             388
TenYearCHD          0
dtype: int64
```

```
In [6]: # Step 4: Handle missing values by filling them with the mean of each column
data.fillna(data.mean(), inplace=True)
```

- **Verifying Missing Values:** After filling missing values, we check again to confirm that there are no more missing values left in the dataset.
- **Defining Features and Target:** We separate the dataset into features (**X**) and the target variable (**y**). Features are all the columns except **TenYearCHD**, which is the outcome we want to predict.

```
In [7]: # Step 5: Verify that there are no more missing values
print("\nMissing values after filling them:")
print(data.isnull().sum())
```

```
Missing values after filling them:
male                0
age                 0
education           0
currentSmoker       0
cigsPerDay          0
BPMeds              0
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol             0
sysBP               0
diaBP               0
BMI                 0
heartRate           0
glucose             0
TenYearCHD          0
dtype: int64
```

```
In [28]: # Step 6: Define the features (X) and target (y)
X = data.drop(columns=['TenYearCHD']) # All columns except 'TenYearCHD' are features
y = data['TenYearCHD'] # 'TenYearCHD' is the target variable we want to predict
```

- **Splitting the Data:** We split the data into two sets: one for training the model and the other for testing it. This helps us evaluate how well the model performs on unseen data.
- **Initializing the Random Forest Model:** We create a Random Forest model. This model is made up of multiple decision trees that work together to make predictions.
- **Training the Model:** We train the Random Forest model using the training data. During this step, the model learns from the data to make predictions.
- **Feature Selection:** We use the trained Random Forest model to identify which features are most important in predicting CHD. This helps in reducing the number of features to the most relevant ones.

```
In [29]: # Step 7: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
In [30]: # Step 8: Initialize the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
In [31]: # Step 9: Train the Random Forest model
rf_model.fit(X_train, y_train)
```

```
Out[31]:
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
In [32]: # Step 10: Feature selection - Identify the most important features
selector = SelectFromModel(rf_model, threshold="median")
selector.fit(X_train, y_train)
```

```
Out[32]:
SelectFromModel
  estimator: RandomForestClassifier
RandomForestClassifier(random_state=42)
  RandomForestClassifier
```

- **Creating the Feature Matrix:** We create a new dataset that only contains the most important features identified in the previous step. This reduces the complexity of the model.

- **Training the Model Again:** We train the Random Forest model again, but this time only with the selected important features. This often improves the model's performance.
- **Making Predictions:** We use the newly trained model to make predictions on the test data. This tells us how well the model performs.
- **Calculating Accuracy, Confusion Matrix, and Classification Report:** We evaluate the model's performance by calculating accuracy (how many predictions were correct), confusion matrix (a table that shows how well the model classified each class), and a classification report (which gives precision, recall, and F1-score).

```
In [33]: # Step 11: Create the feature matrix with the best features
X_train_selected = selector.transform(X_train)
X_test_selected = selector.transform(X_test)
```

```
In [38]: # Step 12: Train the Random Forest model again with selected features
rf_model_selected = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model_selected.fit(X_train_selected, y_train)
```

```
Out[38]: ▼ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
In [39]: # Step 13: Make predictions on the test data with the selected features
y_pred_selected = rf_model_selected.predict(X_test_selected)
```

```
In [35]: # Step 14: Calculate accuracy, confusion matrix, and classification report
accuracy = accuracy_score(y_test, y_pred_selected)
print(f"\nAccuracy: {accuracy:.2f}")

conf_matrix = confusion_matrix(y_test, y_pred_selected)
print("\nConfusion Matrix:")
print(conf_matrix)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_selected))
```



```
Accuracy: 0.85

Confusion Matrix:
[[1067  10]
 [ 184  11]]

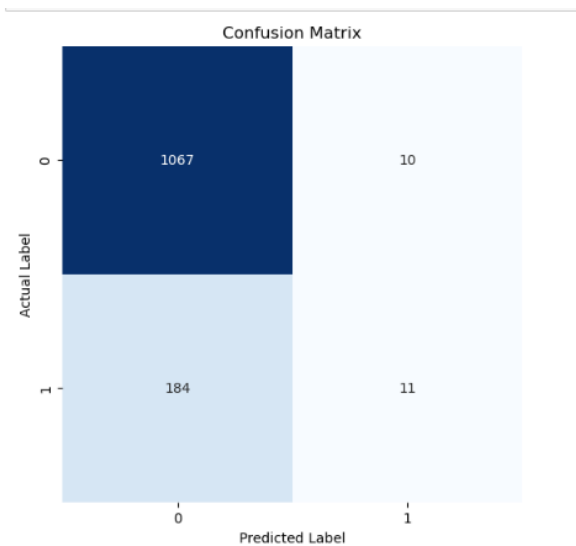
Classification Report:
              precision    recall  f1-score   support

     0       0.85         0.99     0.92     1077
     1       0.52         0.06     0.10      195

 accuracy          0.85         1272
 macro avg         0.69         0.52         0.51     1272
 weighted avg      0.80         0.85         0.79     1272
```

```
In [36]: # Step 15: Make a graph of the confusion matrix
plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.show()
```

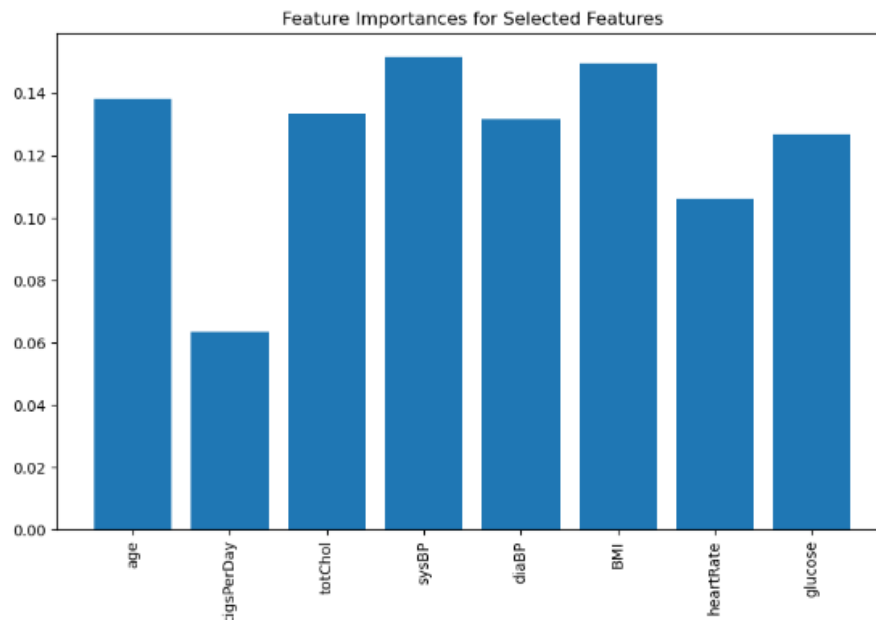
- **Making a Graph of the Confusion Matrix:** We create a heatmap to visually represent the confusion matrix. This makes it easier to see where the model made correct and incorrect predictions.



- **Plotting Feature Importance: Finally, we create a bar chart that shows the importance of each selected feature. This tells us which factors are most crucial in predicting CHD.**

```
In [37]: # Step 16: Plot feature importance for the selected features
selected_features = X.columns[selector.get_support()]
importances = rf_model_selected.feature_importances_

plt.figure(figsize=(10,6))
plt.title("Feature Importances for Selected Features")
plt.bar(range(X_train_selected.shape[1]), importances, align="center")
plt.xticks(range(X_train_selected.shape[1]), selected_features, rotation=90)
plt.show()
```



- **Calculate F1 Score**

```
In [24]: from sklearn.metrics import f1_score

# Make predictions on the test data
y_pred = rf_model.predict(X_test)

# Calculate the F1 score
f1 = f1_score(y_test, y_pred)

print(f"F1 Score: {f1:.2f}")
```

F1 Score: 0.08

## Analysis and Challenges of Using Random Forest for Heart Disease Prediction

### Model Performance

The model's performance was evaluated using a variety of metrics, including accuracy, the confusion matrix, precision, recall, and the F1-score.

1. **Accuracy:** The model achieved an accuracy of 85%, indicating that 85% of the predictions made by the model were correct.
2. **Confusion Matrix:**
  - **True Positives (TP):** The model correctly identified 11 cases of heart disease.
  - **False Negatives (FN):** The model failed to identify 184 cases of heart disease, which means it missed many patients who actually have the disease.
  - **True Negatives (TN):** The model correctly identified 1067 individuals as not having heart disease.
  - **False Positives (FP):** The model incorrectly classified 10 individuals as having heart disease when they did not.

The confusion matrix reveals a significant challenge: while the model is good at identifying non-disease cases (TN), it struggles to accurately identify those who do have the disease (high FN), which is critical in a medical context.

3. **Classification Report:**
  - **Precision:** The precision for predicting heart disease (1) was 52%, meaning that just over half of the individuals predicted to have heart disease actually did.
  - **Recall:** The recall for heart disease was only 6%, reflecting the model's poor performance in identifying true heart disease cases.
  - **F1-score:** The F1-score for heart disease prediction was 10%, indicating that the model struggles to balance precision and recall effectively.

## Chapter 6: Conclusion, Results and Future Scope

### 6.1. Conclusion

In this study, we aimed to develop and evaluate machine learning models for predicting the likelihood of Chronic Heart Disease (CHD) development in patients over the next ten years. The dataset, consisting of 4240 entries and 16 features, was analyzed using four different algorithms: Decision Tree, K-Nearest Neighbors (KNN), Logistic Regression, and Random Forest. To address the issue of class imbalance, SMOTE (Synthetic Minority Over-sampling Technique) was applied specifically to the KNN and Logistic Regression models.

### Key Findings

- 1) Decision Tree: The Decision Tree model demonstrated a moderate accuracy of 83.92%. However, it struggled with the positive class (CHD-positive), achieving a low F1 score of 0.09 and a recall of only 0.05. This indicates that while the model was reasonably good at identifying patients without CHD, it performed poorly in predicting patients who would develop CHD.
- 2) K-Nearest Neighbors (KNN): The KNN model, after applying SMOTE, had an accuracy of 68%. Despite the SMOTE intervention, which aimed to balance the class distribution, the KNN model still showed limited effectiveness in predicting the positive class. The F1 score for class 1 (CHD-positive) was 0.97, but the recall was 0.43, suggesting that while it could identify a significant proportion of CHD-positive cases, it missed many true cases.
- 3) Logistic Regression: With the application of SMOTE, the Logistic Regression model achieved the highest accuracy of 96%. It demonstrated balanced performance with an F1 score of 0.97 for class 0 (no CHD) and 0.82 for class 1 (CHD-positive). The recall for class 1 was notably high at 0.86, indicating that the model was effective in identifying most of the patients who were likely to develop CHD.
- 4) Random Forest: The Random Forest model achieved an accuracy of 85%. Similar to the Decision Tree, it faced challenges in predicting the positive class, with an F1 score of 0.10 and a recall of 0.06 for class 1. This suggests that while the Random Forest model was generally robust, it still struggled with the imbalanced nature of the dataset.

The results highlight the effectiveness of SMOTE in improving the performance of KNN and Logistic Regression models by addressing class imbalance. However, challenges remain in achieving balanced performance across all models, particularly in predicting the minority class (CHD-positive).

## 6.2. Results

### Summary of Performance Metrics

#### 1) Accuracy

- i. Decision Tree: 83.92%
- ii. KNN (with SMOTE): 68%
- iii. Logistic Regression (with SMOTE): 96%
- iv. Random Forest: 85%

#### 2) F1 Score for Class 0 (No CHD) / Class 1 (CHD Positive):

- i. Decision Tree: 0.91 / 0.09
- ii. KNN (with SMOTE): 1.00 / 0.97
- iii. Logistic Regression (with SMOTE): 0.97 / 0.82
- iv. Random Forest: 0.92 / 0.10

#### 3) Recall for Class 0 / Class 1

- i. Decision Tree: 0.99 / 0.05
- ii. KNN (with SMOTE): 0.73 / 0.43
- iii. Logistic Regression (with SMOTE): 0.98 / 0.86
- iv. Random Forest: 0.99 / 0.06

#### 4) Precision for Class 0 / Class 1

- i. Decision Tree: 0.84 / 0.55
- ii. KNN (with SMOTE): 0.88 / 0.21
- iii. Logistic Regression (with SMOTE): 0.97 / 0.87
- iv. Random Forest: 0.85 / 0.52

### Analysis

- Logistic Regression stood out as the most effective model, achieving high accuracy and a balanced performance across classes. The use of SMOTE contributed to its ability to identify CHD-positive cases more effectively.
- KNN, despite using SMOTE, showed a lower accuracy and recall for the positive class compared to Logistic Regression. This suggests that KNN might require further tuning or different approaches to handle imbalanced datasets effectively.
- Decision Tree and Random Forest models exhibited reasonable overall accuracy but struggled significantly with the positive class. Their performance could be improved by further addressing class imbalance and tuning model parameters.

### 6.3. Future Scope

To enhance the performance and address the limitations observed in this study, the following future directions are recommended:

- 1) **Advanced Resampling Techniques:** Beyond SMOTE, other resampling methods such as ADASYN or Cluster-Based Over-Sampling could be explored to improve model performance for the minority class. Combining multiple resampling techniques may also provide better results.
- 2) **Feature Engineering and Selection:** Improving feature selection through techniques like Recursive Feature Elimination (RFE) or Feature Importance Analysis can enhance model performance. Creating new features based on domain knowledge or interaction terms might also lead to better predictive power.
- 3) **Hyperparameter Tuning:** Extensive hyperparameter tuning using methods like Grid Search or Random Search could help optimize model performance, particularly for Decision Tree and Random Forest models. Adjusting parameters like tree depth, number of estimators, and learning rate can significantly impact performance.
- 4) **Ensemble Methods:** Implementing ensemble techniques such as Boosting (e.g., AdaBoost or Gradient Boosting) or Stacking could improve predictions by combining the strengths of multiple models. These methods often handle imbalanced datasets more effectively.
- 5) **Deep Learning Approaches:** Exploring neural network models or deep learning techniques could offer significant improvements, especially for complex and large datasets. Techniques like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) might be beneficial for feature extraction and classification.
- 6) **Cross-Validation and External Validation:** Employing cross-validation techniques and testing models on external or unseen datasets can help assess the generalizability of the models. This approach will ensure that the models perform well across different data distributions and real-world scenarios.

In conclusion, while the study demonstrated the effectiveness of SMOTE in enhancing the performance of KNN and Logistic Regression models, additional refinements and alternative methods are required to achieve balanced and robust predictions for CHD. The proposed future work aims to address these challenges and improve the overall predictive capability of the models.

## References

1. <https://www.nhlbi.nih.gov/health/coronary-heart-disease>
2. figure 1-1, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
3. figure 1-2, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
4. figure 1-3, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
5. figure 1-4, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
6. figure 1-5, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
7. figure 1-6, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
8. figure 1-7, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
9. figure 1-8, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
10. figure 1-9, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
11. figure 1-11, Aurélien Géron, Hands-On-Machine-Learning-with-Scikit-Learn-and-TensorFlow, O'REILLY, June 2019: Second Edition
12. <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
13. <https://www.datascienceprophet.com/understanding-the-mathematics-behind-the-decision-tree-algorithm-part->
14. <https://www.geeksforgeeks.org/decision-tree-introduction-example/>
15. <https://www.ibm.com/docs/en/ias?topic=procedures-k-nearest-neighbors-knn>
16. <https://www.elastic.co/what-is/knn#how-does-knn-work>
17. <https://www.ibm.com/docs/en/ias?topic=procedures-k-nearest-neighbors-knn>
18. <https://www.geeksforgeeks.org/types-of-regression-techniques/>
19. [https://www.geeksforgeeks.org/regression-in-machine-learning/?ref=header\\_ind](https://www.geeksforgeeks.org/regression-in-machine-learning/?ref=header_ind)
20. [https://www.geeksforgeeks.org/regression-in-machine-learning/?ref=header\\_ind](https://www.geeksforgeeks.org/regression-in-machine-learning/?ref=header_ind)
21. T. Teoh, Z. Rong, Artificial Intelligence with Python. Machine Learning Foundations, Methodologies, and Applications (2022) (Z-Lib.io)
22. T. Teoh, Z. Rong, Artificial Intelligence with Python. Machine Learning Foundations, Methodologies, and Applications (2022) (Z-Lib.io)
23. Berkson, J. (1944). "Statistical concepts in cancer research: The relation between smoking and lung cancer." *Journal of the American Statistical Association*, 39(227), 363-372.
24. Rosenblatt, F. (1958). "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, 65(6), 386-408.
25. Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth Publishing Company.
26. <https://www.geeksforgeeks.org/understanding-logistic-regression/>

27. [https://www.geeksforgeeks.org/ml-logistic-regression-using-python/?ref=gcse\\_ind](https://www.geeksforgeeks.org/ml-logistic-regression-using-python/?ref=gcse_ind)
28. <https://www.geeksforgeeks.org/derivative-of-the-sigmoid-function/>
29. [https://www.geeksforgeeks.org/ml-logistic-regression-using-python/?ref=gcse\\_ind](https://www.geeksforgeeks.org/ml-logistic-regression-using-python/?ref=gcse_ind)
30. <https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning>
31. <https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
32. [Random Forest - an overview | ScienceDirect Topics](#)
33. <https://www.kaggle.com/code/maushamjha/10-years-chronic-heart-disease#treating-missing-values>