# Using Combined Naive Bayes to Predict Heart Failure

**Wilson Feng**

May 2025

# Contents

# 1 Introduction

## 1.1 Rationale

Personal health has always been a conversation of interest to me; however, I'm often left with disorientation as I'm lost in a sea of anecdotal life experiences, or common cookie-cutter advice, devoid of rigorous data analysis or imperial explanation.

Alongside this, 2 years ago, I "developed" a machine-learning project to classify abnormalities in chest X-rays, which I proceeded to demonstrate at my local science fair. However, in reality, I only had a surface-level understanding of what I was doing, as the project was pretty much carried through preexisting guides and introductory courses that I applied to a chest x-ray dataset. This meant I glossed over all the fascinating math that had progressed through decades of research.

## 1.2 Aim

Ultimately, I want to explore the underlying mathematics of one of these machine learning techniques, naive Bayes, and apply it to the possible indicators of heart health, such that meaningful insights on what constitutes heart health are revealed. Therefore, this Math IA will **determine the extent to which combined naive Bayes can predict heart failure.**

Furthermore, this is a significant topic to investigate as according to Savarese et al., 2022, 64 million people are affected by heart failure globally, and thus, it has become a major public health priority.

## 1.3 Data

The data used stems from fedesoriano, 2021, where 5 heart health datasets were combined with 11 clinical measures. Each dataset was publicly available independently online; however, by combining them, greater insights may be observed. From my education in IB Psychology, I know this is because a larger sample size from a vari-

ety of locations around the world may show more generalizable correlations between lifestyle metrics and heart health.

Going further into the data as seen in Figure 1.1, we see that there are exactly 11 health metrics, and the result - the presence of heart disease.

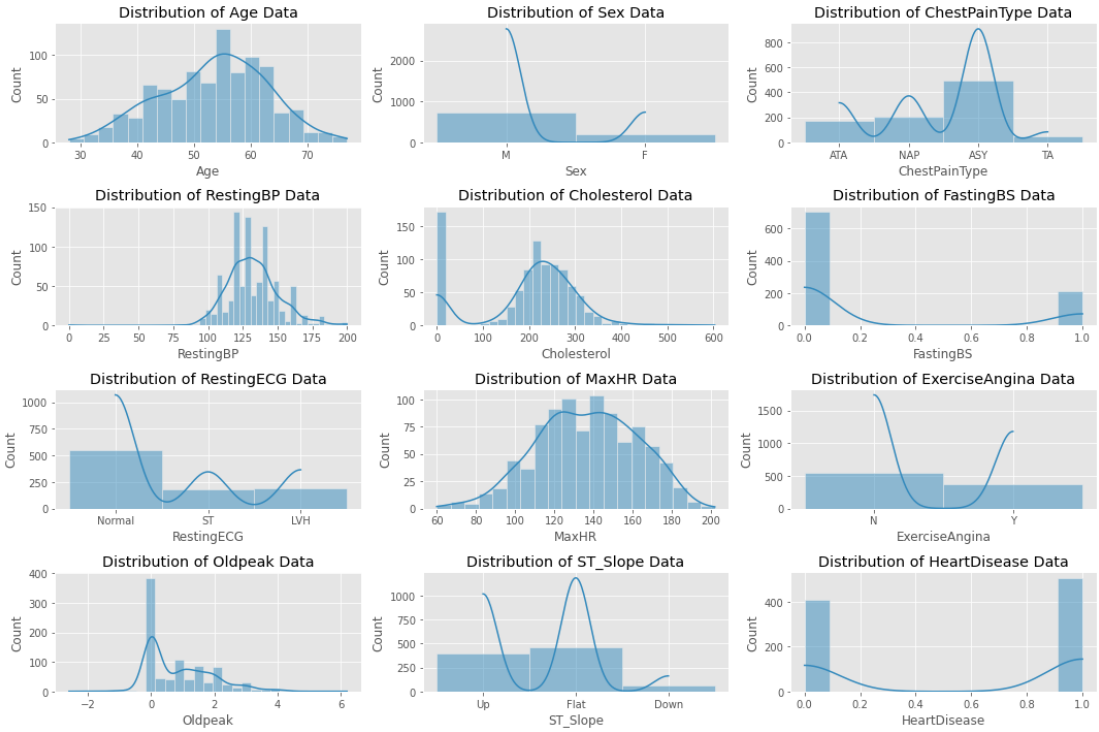


**Figure 1.1:** *Visualization of the Dataset, Source: Gaur, 2021*

## 1.4 Methodology

As the name entails, my approach to analyzing data stems from Bayes' theorem, taught in the IB Math AA HL syllabus. From this, I will use conditional probability to derive and modify the original Bayes' theorem to predict the heart failure dataset. As seen in Figure 1.1, the 11 features of the dataset are in the forms of both continuous data, such as age, and categorical data, such as gender. Therefore, to accurately generate probabilities according to the dataset, I will need to ensure that my usage of Bayes' theorem incorporates both data types.

Using Gaussian Naive Bayes for continuous data, and Multinomial Naive Bayes for categorical data, I will combine both approaches such that I can accurately generate insights by accounting for all types of data. Afterward, I will compare the predicted result with the actual result to evaluate the effectiveness of my model.

To make the calculations, I will use various technologies:

- Ti-84 Plus CE for basic calculations Instruments, n.d.
- Python 3 for scripting to execute on the Kaggle Notebook PYTHON, 2019

- Kaggle Notebook for a platform to execute Python scripts to manipulate data *Notebooks Documentation* n.d.

### 1.4.1 Variables

To proceed further with the math behind these techniques, I must define a series of notations and variables first. First, the term *feature* refers to the input clinical metrics that we are using to predict. Examples include age, sex, or maximum heart rate.

Conversely, the term *class* refers to the output condition that we are explicitly trying to predict. In my case, this is binary, where I am predicting the presence of heart disease.

Next, let $\mathbf{x}$ be the vector of features of a single data point whose class we know, where $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, and $x_i$ represents the value of each feature. For my case, $n = 11$ as there are 11 metrics included in the dataset. For example, one data point might look like:

$$\mathbf{x} = \begin{bmatrix} 40 \\ \text{M} \\ \text{ATA} \\ 140 \\ \vdots \\ \text{Up} \end{bmatrix} \tag{1.1}$$

On the other hand, let $\mathbf{y}$ be the output classes involved. Generally, $y_0$ or $y_k$ where $k = 0$ mean no heart disease while $y_1$ or $y_k$ where $k = 1$ mean there is heart disease.

Furthermore, let $\{z_1, z_2, \ldots, z_n\}$ be the vector of features of a data point whose class is unknown. It is the same as $\mathbf{x}$ except that we do not know its class and we need to instead predict it.

Moving on to the variables to manipulate the dataset, let $\phi_i$ be the $\text{i}^{\text{th}}$ feature in the dataset. This is different from $x_i$ where it is the $\text{i}^{\text{th}}$ known data point in the dataset. Let $\phi_{i_j}$ be the $\text{j}^{\text{th}}$ data point of the $\text{i}^{\text{th}}$ feature of the dataset. Furthermore, let $\phi_{i_{k_j}}$ be the $\text{j}^{\text{th}}$ value of the $\text{i}^{\text{th}}$ feature given the $\text{k}^{\text{th}}$ class in the dataset.

For example, looking at Figure 1.2, $\phi_1$ would be the entirety of the Age column, while $\phi_{1_0}$ would be the entire age column, except those that have heart disease, since $k = 0$. Finally, $\phi_{1_{1_1}}$ would be equal to 49. Note that it is not 40, which is the first value, as $k = 1$, therefore the Heart Disease column would need to be 1 too.

### 1.4.2 Naive Baye's Classifier

The Naive Baye's Classifier, one of the simplest Bayesian Network models *Naïve Bayes Model - an overview* n.d., is a method that allows us to transform Bayes' Theorem, denoted by

| Age | Sex | ChestPain... | RestingBP | Cholesterol | FastingBS | RestingECG | MaxHR | ExerciseA... | Oldpeak | ST_Slope | HeartDise... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | M | ATA | 140 | 289 | 0 | Normal | 172 | N | 0 | Up | 0 |
| 49 | F | NAP | 160 | 180 | 0 | Normal | 156 | N | 1 | Flat | 1 |
| 37 | M | ATA | 130 | 283 | 0 | ST | 98 | N | 0 | Up | 0 |
| 48 | F | ASY | 138 | 214 | 0 | Normal | 108 | Y | 1.5 | Flat | 1 |
| 54 | M | NAP | 150 | 195 | 0 | Normal | 122 | N | 0 | Up | 0 |
| 39 | M | NAP | 120 | 339 | 0 | Normal | 170 | N | 0 | Up | 0 |
| 45 | F | ATA | 130 | 237 | 0 | Normal | 170 | N | 0 | Up | 0 |
| 54 | M | ATA | 110 | 208 | 0 | Normal | 142 | N | 0 | Up | 0 |
| 37 | M | ASY | 140 | 207 | 0 | Normal | 130 | Y | 1.5 | Flat | 1 |
| 48 | F | ATA | 120 | 284 | 0 | Normal | 120 | N | 0 | Up | 0 |
| 37 | F | NAP | 130 | 211 | 0 | Normal | 142 | N | 0 | Up | 0 |
| 58 | M | ATA | 136 | 164 | 0 | ST | 99 | Y | 2 | Flat | 1 |
| 39 | M | ATA | 120 | 204 | 0 | Normal | 145 | N | 0 | Up | 0 |
| 49 | M | ASY | 140 | 234 | 0 | Normal | 140 | Y | 1 | Flat | 1 |
| 42 | F | NAP | 115 | 211 | 0 | ST | 137 | N | 0 | Up | 0 |
| 54 | F | ATA | 120 | 273 | 0 | Normal | 150 | N | 1.5 | Flat | 0 |
| 38 | M | ASY | 110 | 196 | 0 | Normal | 166 | N | 0 | Flat | 1 |
| 43 | F | ATA | 120 | 201 | 0 | Normal | 165 | N | 0 | Up | 0 |

**Figure 1.2:** *The Dataset, Source: Photo by Author*

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(A)} \tag{1.2}$$

However, faced with the unknown data point features of $\mathbf{z}$ and the output class of $y_k$ defined in Section 1.4.1, we need to modify the equation to suit these features.

Substituting these values into the equation, it becomes

$$P(y_k \mid z_1, z_2, \ldots, z_n) = \frac{P(y_k)P(z_1, z_2, \ldots, z_n \mid y_k)}{P(z_1, z_2, \ldots, z_n)} \tag{1.3}$$

However, we want an equation that allows us to compute values, therefore we want to rewrite the equation in regards to the definition of conditional probability.

$$P(y_k \mid z_1, z_2, \ldots, z_n) = \frac{P(y_k, z_1, z_2, \ldots, z_n)}{P(z_1, z_2, \ldots, z_n)}$$

Rewriting in terms of the chain rule of the repeated application of condition probability,

$$= \frac{P(z_1, z_2, \ldots, z_n, y_k)}{P(z_1, z_2, \ldots, z_n)}$$

$$= \frac{P(z_1 \mid z_2, \ldots, z_n, y_n)P(z_2, \ldots, z_n, y_k)}{P(z_1, z_2, \ldots, z_n)}$$

$$= \frac{P(z_1 \mid z_2, \ldots, z_n, y_k)P(z_2 \mid z_3, \ldots, z_n, y_k)P(z_3, \ldots, z_n)}{P(z_1, z_2, \ldots, z_n)}$$

$$= \ldots$$

$$= \frac{P(z_1 \mid z_2, \ldots, z_n, y_k)P(z_2 \mid z_3, \ldots, z_n, y_k) \ldots P(z_{n-2} \mid z_{n-1}, y_k)P(z_{n-1} \mid z_n, y_k)P(z_n \mid y_k)P(y_k)}{P(z_1, z_2, \ldots, z_n)}$$

However, since this approach utilizes Naive Bayes, the probabilities are "naive". Therefore, we assume that each feature is independent of each other. This means that $P(z_i \mid z_{i+1}, \ldots, z_n, y_k) = P(z_i \mid y_k)$. In doing so, we essentially negate any notion of

interplay between the features. For example, in my case, this will compute age, resting heart rate, and blood pressure separately from each other regardless of whether one factor influences the other in any way. In the real world, this is hardly the case, as with age, a noticeable change will occur in resting heart rate and blood pressure Clinic, 2020. Nevertheless, Naive Bayes is still viable as it is very computationally efficient and simple to calculate *Naïve Bayes Model - an overview* n.d.

We can thus express the equation as

$$P(y_k \mid z_1, z_2, \ldots, z_n) = \frac{P(z_1 \mid y_k)P(z_2 \mid y_k)P(z_3 \mid y_k)\ldots P(z_n \mid y_k)P(y_k)}{P(z_1, z_2, \ldots, z_n)}$$

Rewriting this in Pi product notation, we get

$$P(y_k \mid z_1, z_2 \ldots, z_n) = \frac{P(y_k)\prod_{i=1}^{n}(z_i \mid y_k)}{P(z_1, z_2, \ldots, z_n)}$$

What we have right now gives us the probability of the $k^{\text{th}}$ class of $y$. Substituting for each $y_0$, and $y_1$, we are left with two probabilities. Additionally, since the denominator is effectively the same for all $y$, it is constant and we can ignore it. Finally, we implement a decision rule, by taking the *argmax* of the two probabilities. In essence, to make sure the probability of misclassification is at a minimum, this takes the higher value of the two, also known as the *minimum a posteriori* decision rule. Pishro-Nik, n.d.

$$P(y_k \mid z_1, z_2, \ldots, z_n) \propto P(y_k)\prod_{i=1}^{n}(z_i \mid y_k)$$

$$y = \underset{k\in\{k_0,k_1\}}{\operatorname{argmax}} P(y_k)\prod_{i=1}^{n} P(z_i \mid y_k) \tag{1.4}$$

To recap what this equation does,

1. Calculates the following with $k = 0$ and $k = 1$.
2. Loops through each feature $z_i$ of the data point and gets the probability of seeing it, given a class $y_k$, (e.g. The probability of a normal resting ECG, given the presence of heart disease).
3. Multiplies these values together, along with the corresponding $P(y_k)$.
4. Returns the class at the higher value of $k = 0$ or $k = 1$.

While this gives us a helpful binary prediction of heart failure, there are still unknown variables we have to calculate. First, each class's prior probability, or $P(y_k)$, needs to be calculated. Next, the probability of seeing a feature, or $P(z_i \mid y_k)$ given a class also needs to be calculated.

The class's prior probability can be fulfilled through two methods: assuming that that the classes are equiprobable, as in there is an equal probability among all classes, or calculating an estimate for class probability, based on the dataset. Equiprobable

class among $k_0$ and $k_1$ would assume a prior of 0.5 for both classes, as denoted by $P(y_k) = \frac{1}{\text{number of classes}}$; however, it may not reflect real life in that there are a lot more people that do not have heart failure than those who do. Even so, as seen in Figure 1.1, the dataset also does not reflect real conditions as there are more data points with heart disease than those who do not. Therefore, both approaches would not model a realistic scenario. Even so, I chose to estimate using the dataset ratio. The reason for this is because of how I test the performance of my model: randomly sampling the dataset. What this means is that the environment I am simulating when I'm testing the model would have heart disease rates proportional to the rates in the dataset. This ratio is calculated by

$$P(y_k) = \frac{\text{number of data points in the class}}{\text{number of data points in total}}$$

This information is given by the data set provider; however, one may generate these values by utilizing an indicator function.

$$c(y_k) = \sum_{j=1}^{|x|} 1[x_j \in y_k] \tag{1.5}$$

This expression counts the number of data points in $y_k$ by summing through all known data points of $x$, and then checking if they are in $y_k$. If they are, the indicator function returns 1, and 0 if not. Note that $|x|$ is the total amount of data points.

Now, we can calculate our class priors with

$$P(y_k) = \frac{c(y_k)}{|x|} \tag{1.6}$$

Substituting our values in, our priors are:

$$P(y_0) = \frac{410}{918} \approx 0.447$$
$$P(y_1) = \frac{508}{918} \approx 0.553$$

Now, for $P(z_i \mid y_k)$, we need to separate $z_i$ into two categories: continuous and categorical data. The reason we need to split this problem into these two data types is that we need to handle these two data types differently. Furthermore, because of the limited generalizability for continuous probabilities, we will need to use probability densities.

For example, say that we want to calculate the probability of a continuous value such as age. One might think of counting the number of times the age, for example, 60, occurred and there was the presence of heart disease, divided by the total number of data points with heart disease. In doing so, if we see 60 as an input, we would simply use the precomputed probability to make our predictions. However, this suffers as it wouldn't give meaningful insights to other data points. If there was another data point

with age 61, the probability would be very low even though the values are very close together, and thus, its probability should be around the same. This is unfortunately not the case as there may not be many data points with an age of exactly 61. Therefore, we can not use exact probabilities anymore and instead, the need for probability densities. This is where we need to use the Gaussian distribution to adjust our naive Bayes equation.

# 2

# Exploration

## 2.1 Gaussian Naive Bayes

To handle continuous values, we first need to take a look at the 5 continuous categories that we see in Figure 1.1: Age, Resting Blood Pressure, Cholesterol, Maximum Heart Rate, and Oldpeak. Gaussian Naive Bayes allows us to handle these values by assuming our data is normally distributed, then computing the mean and variance of every data point, separating by class and feature. The formula for this is:

$$P(x = v \mid y_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}} \tag{2.1}$$

However, when applied to my variables and use cases, it becomes

$$P(z_i \mid y_k) = \frac{1}{\sqrt{2\pi\sigma_{i_k}^2}} e^{-\frac{(x_i-\mu_{i_k})^2}{2\sigma_{i_k}^2}} \tag{2.2}$$

$z_i$ references the i$^{\text{th}}$ feature, and $y_k$ references the k$^{\text{th}}$ class of the unknown data point, which should correspond with the i$^{\text{th}}$ feature and k$^{\text{th}}$ class in the dataset. This means that $\sigma_{i_k}^2$ and $\mu_{i_k}$ are the variance of the mean of the i$^{\text{th}}$ feature and k$^{\text{th}}$ class of the dataset respectively.

For example, suppose I wanted to find the probability a data point is 65 years old, given they have heart disease. How we would calculate this is first set up the variables, $z_i = 65$, and $k = 1$. Next, we want to find the mean and variance of age, given whether there is heart disease or not. To do this, we would need to look at our original dataset and calculate from the data. How I do the calculations will be demonstrated later. As for now, the calculations are seen in Figure 2.1.

```
Mean Age for people without Heart Disease: 50.551219512195125
Variance of Age for people without Heart Disease: 89.2064166020633
Mean Age for people with Heart Disease: 55.8996062992126
Variance of Age for people with Heart Disease: 76.16149885850054
```

**Figure 2.1:** *Mean and Variance of Age Given Heart Disease Conditions*

Substituting these values into the equation, we get

$$P(\text{Age} = 65 \mid y_0) = \frac{1}{\sqrt{2\pi(89.2)}}\exp(-\frac{(65-58.6)^2}{2(89.2)}) \approx 0.0336$$

$$P(\text{Age} = 65 \mid y_1) = \frac{1}{\sqrt{2\pi(76.2)}}\exp(-\frac{(65-55.9)^2}{2(76.2)}) \approx 0.0265$$

In this case, since $y_0$ or the no heart disease condition is higher, there is a slightly higher likelihood that the data point does not have heart disease. However, this probability is almost useless on its own as it's only one small fraction of the 10 other features, along with the prior probability of each of the classes. Nevertheless, this gives a demonstration of how one of the features of a continuous data point may be calculated, such that it can be used in the Naive Bayes Classifier formula derived in 1.4.

However, before continuing to calculate the mean and variance of each feature given each class, there is a concern that I want to address: we can visually see that Cholesterol Data and Oldpeak data may not follow a perfect normal distribution, as demonstrated in Figure 2.2. Recall that at the beginning of Section 2.1, we needed to assume that the data followed a normal distribution.

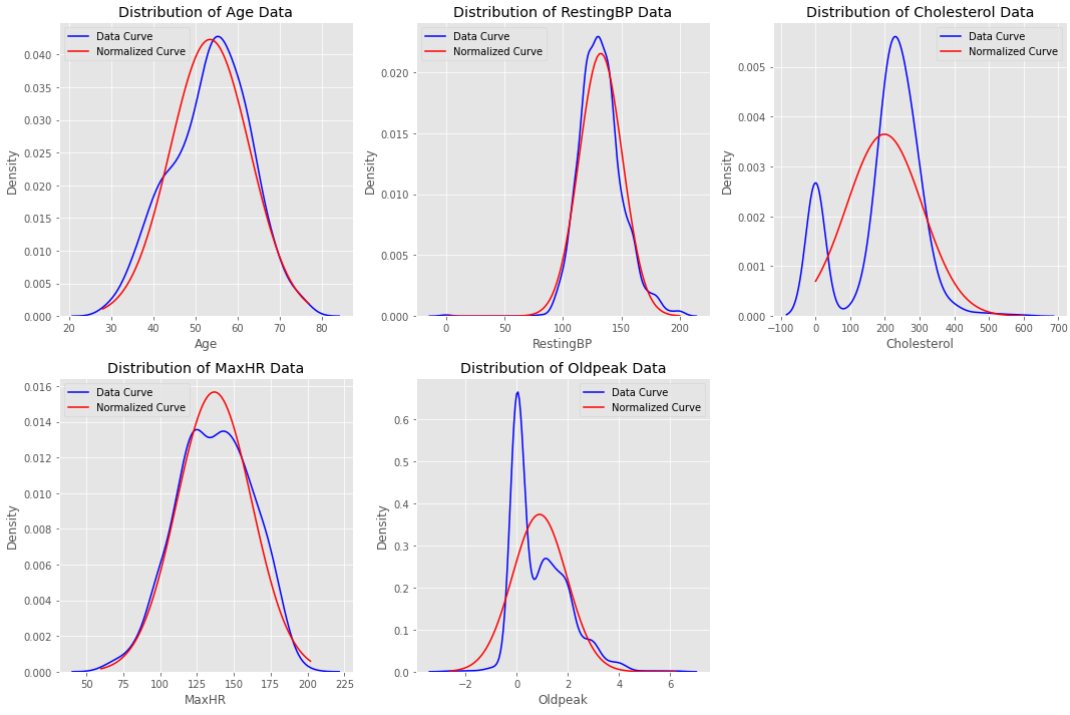


**Figure 2.2:** *Visualization of the Continuous Data, and its Normalized Counterpart, Source: Photo by author*

For Cholesterol, we know that the skewness on the left is caused by erroneous values as a cholesterol level of 0 is preposterous. Therefore, we can discard those values when computing the mean and variance of Cholesterol data. By doing this, we see that it eventually corrects towards a normalized distribution, exemplified in Figure 2.3.

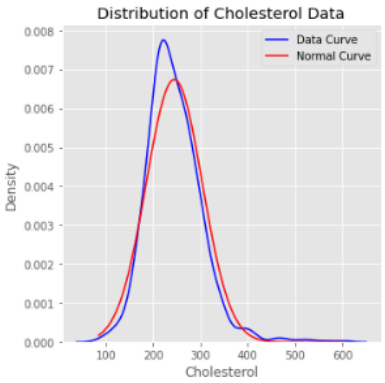


**Figure 2.3:** *Corrected Cholesterol Graph, Source: Photo by Author*

As for Oldpeak, there is a sharp spike in the value 0, and then another hump at 1.5. More specifically, as seen in the differences between the two curves at Oldpeak in Figure 2.2, due to the long tail skewed to the right, the variance, or $\sigma_k$, may be overestimated, leading to broader likelihood curves. Additionally, this means the mean, $\mu_k$ is brought towards the tail, which may lead to an overestimation in ranges of $x > 0$ and underestimation in $x \leq 0$. Nonetheless, I chose to disregard this and carry on with the Gaussian distribution despite the possibility of a slightly less accurate model, as the other option of using a kernel density estimate to model the two peaks was beyond the scope of this project. As for the other three features: Age, Resting Blood Pressure, and Maximum heart rate, they follow an approximate normal distribution, indicated by the fact that we can see the curve generated by the data is similar to the normalized curve. For these values, we can almost confidently assume that they are normal for Gaussian Naive Bayes.

Now that the previous concern is alleviated, the next step is to actually find the values for the mean and variance of each feature, given in each class. To find the mean, we simply use the common formula of adding up all the samples, and then dividing by the number of samples.

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

To apply this to my use case, I need to add a few variables, notably $\phi$ to access the data from the dataset.

$$\mu_{i_k} = \frac{\sum_{j=1}^{n} \phi_{i_{k_j}}}{n}$$

This gets the mean of feature $i$, given class $k$ by summing over all $j$ data points of the $i^{th}$ feature in the $k^{th}$ class in the dataset, and then dividing by the amount of data points, $n$. For example, going back to Figure 2.1, knowing that Age is the 1st feature in the dataset, $i = 1$, we see that

$$\mu_{1_0} = \frac{\sum_{j=1}^{n} \phi_{1_{0_j}}}{n} \approx 50.6$$

$$\mu_{1_1} = \frac{\sum_{j=1}^{n} \phi_{1_{1_j}}}{n} \approx 55.9$$

This means that the sum of all n values in feature $i = 1$, and class $k = 0$, divided by n is approximately 50.6. In other words, the sum of all the ages that do not have heart disease, divided by the number of ages observed is approximately 50.6. On the other hand, the sum of all the values in $i = 1$ (ages) in class $k = 1$ (have heart disease) divided by $n$ (number of ages observed) is approximately 55.9.

Next, we need to get the variance. This is somewhat similar to how it is done for the mean, except we use a different formula,

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n} \tag{2.3}$$

Now, applied to my dataset,

$$\sigma_{i_k}^2 = \frac{\sum_{j=1}^{n}(\phi_{i_{k_j}} - \mu_{i_k})^2}{n} \tag{2.4}$$

Again, $\phi_{i_{k_j}}$ is each value of the i$^{\text{th}}$ feature given the k$^{\text{th}}$ class. $\mu_{i_k}$ is the mean of this feature-class combination, which we just calculated previously.

Finally, using these equations, we can calculate our required values to use the Gaussian Naive Bayes Formula. Given that we have 5 continuous features, and we need to calculate mean and variance, and given that we need to calculate for both the $y_0$ and $y_1$ case, we should expect $(5)(2)(2) = 20$ values in total which we will be using. Ultimately, these values can be seen in Figure 2.4.

| | Mean (HeartDisease=0) | Variance (HeartDisease=0) |
|---|---|---|
| Age | 50.551220 | 88.988840 |
| RestingBP | 130.180488 | 271.572302 |
| Cholesterol | 227.121951 | 5556.746104 |
| MaxHR | 148.151220 | 541.011279 |
| Oldpeak | 0.408049 | 0.488399 |

| | Mean (HeartDisease=1) | Variance (HeartDisease=1) |
|---|---|---|
| Age | 55.899606 | 76.011575 |
| RestingBP | 134.185039 | 392.402768 |
| Cholesterol | 175.940945 | 15943.339032 |
| MaxHR | 127.655512 | 545.871485 |
| Oldpeak | 1.274213 | 1.324197 |

**Figure 2.4:** *Calculated Mean and Variance of Continuous Features, Source: Photo by Author*

To put this into perspective, let us use an example. Say we have the following unknown data point that has the continuous values of

$$\mathbf{z} = \begin{bmatrix} 40 \\ 140 \\ 289 \\ 172 \\ 0 \end{bmatrix} \tag{2.5}$$

Where $z_i$ denotes age, resting blood pressure, cholesterol, maximum heart rate, and old peak respectively. Using the mean and variance of each feature given a class, calculated in 2.4, we can substitute into the Gaussian Naive Bayes equation seen in 2.2.

$$P(z_1 \mid y_0) = \frac{1}{\sqrt{2\pi(89.0)}}\exp(-\frac{(40-50.6)^2}{2(89.0)}) = 0.0226$$

$$P(z_2 \mid y_0) = \frac{1}{\sqrt{2\pi(272)}}\exp(-\frac{((140)-130)^2}{2\sigma_{i_k}^2}) \approx 0.0203$$

$$P(z_3 \mid y_0) = \frac{1}{\sqrt{2\pi(5560)}}\exp(-\frac{((289)-227)^2}{2(5560)}) \approx 0.00380$$

$$P(z_4 \mid y_0) = \frac{1}{\sqrt{2\pi(541)}}\exp(-\frac{((172)-148)^2}{2(541)}) \approx 0.0101$$

$$P(z_5 \mid y_0) = \frac{1}{\sqrt{2\pi(0.488)}}\exp(-\frac{((0)-1.27)^2}{2(0.488)}) \approx 0.481$$

I then do the same except for $y = 1$, now taking from the bottom half of 2.4. As seen in Figure 2.5, since, the probability of $y_0$ is higher than $y_1$, this means that the continuous part of this data point has a higher probability that it does not have heart disease.

Gaussian Naive Bayes Calculations:

| | Feature | Value | Mean | Variance | Gaussian_Prob | Class |
|---|---|---|---|---|---|---|
| 0 | Age | 40.0 | 50.551220 | 88.988840 | 0.022625 | 0 |
| 1 | Age | 40.0 | 55.899606 | 76.011575 | 0.008675 | 1 |
| 2 | RestingBP | 140.0 | 130.180488 | 271.572302 | 0.020271 | 0 |
| 3 | RestingBP | 140.0 | 134.185039 | 392.402768 | 0.019290 | 1 |
| 4 | Cholesterol | 289.0 | 227.121951 | 5556.746104 | 0.003792 | 0 |
| 5 | Cholesterol | 289.0 | 175.940945 | 15943.339032 | 0.002116 | 1 |
| 6 | MaxHR | 172.0 | 148.151220 | 541.011279 | 0.010140 | 0 |
| 7 | MaxHR | 172.0 | 127.655512 | 545.871485 | 0.002819 | 1 |
| 8 | Oldpeak | 0.0 | 0.408049 | 0.488399 | 0.481386 | 0 |
| 9 | Oldpeak | 0.0 | 1.274213 | 1.324197 | 0.187796 | 1 |

Final Probabilities:
P(HeartDisease=0) = 8.48866278005488e-09
P(HeartDisease=1) = 1.8747803450406336e-10

**Figure 2.5:** *Calculations for a Single Data Point*

Ultimately, this means that when we are faced with an unknown data point **z**, we are now able to process the continuous part. However, we are still missing the categorical section of the data point. The following section will deal with the remaining 6 features.

## 2.2 Multinomial Naive Bayes

Now that we have dealt with continuous values using the probability densities with a Gaussian distribution, we now have to direct our attention towards the discrete or categorical values. As observed in Figure 2.6, we see that Sex, Chest Pain Type, Fasting Blood Sugar, Resting ECG, Exercise Angina, and ST Slope are all discrete and categorical. As for Fasting Blood Sugar, one may think it is continuous, as it deals with numerical values. However; upon further inspection, we see they are only either one or two. Furthermore, the dataset provider states the way they were separated: "1: if FastingBS $>$ 120 mg/dl, 0: otherwise" fedesoriano, 2021. Therefore, we are free to proceed with Multinomial Naive Bayes.

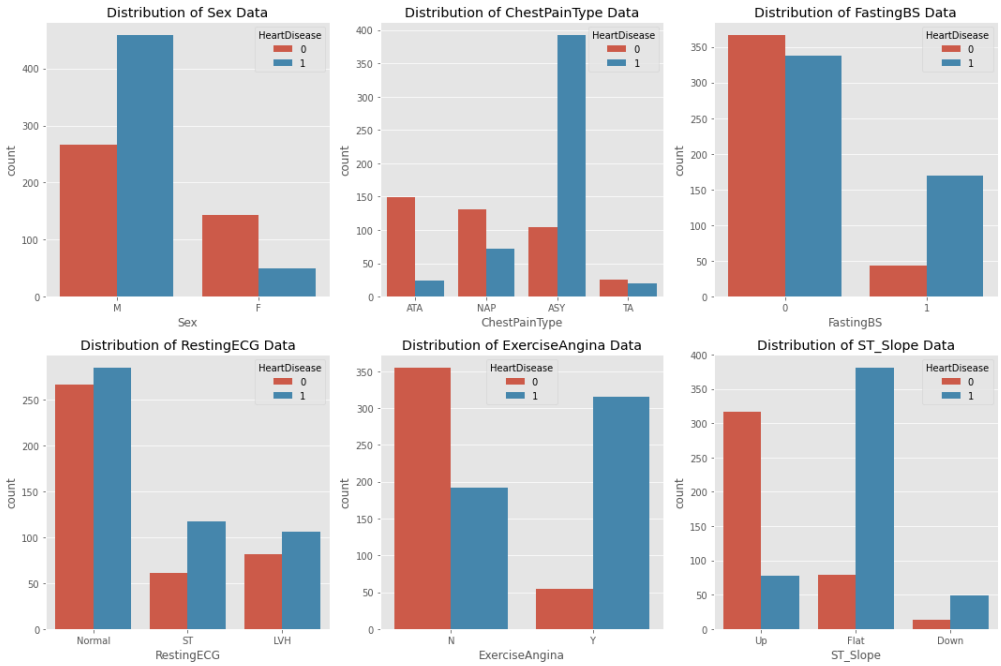


**Figure 2.6:** *Categorical Data Separated by Presence of Heart Disease, Source: Photo by Author*

To first approach this, we must process our categorical data.

Since we already computed our class priors from 1.5, and 1.6, we can focus on counting the number of data points of a feature given a class, such that we can fill in our missing puzzle piece of $P(z_i \mid y_k)$

$$c(\phi_{i_k}) = \sum_{j=1}^{|x|} 1[x_j \in y_k, x_j \in \phi_i] \tag{2.6}$$

In contrast to the previous expression seen in 1.5 counting the amount of data points in $y_k$, I add another condition, in which $x_j$ also needs to be in $\phi_i$. This effectively counts the amount of data points that are present in the i$^{\text{th}}$ feature **and** in the k$^{\text{th}}$ class. Using the previous two expressions, we can now construct the probability

of each feature given a class with

$$P(\phi_i \mid y_k) = \frac{c(\phi_{i_k})}{c(y_k)} \tag{2.7}$$

By dividing the number of data points in a feature, given a class, by the total number of data points of that class, we get the probability of feature $i$ given class $k$.

At this step, we are practically at the finish line, such that we can substitute our equation into our Naive Bayes Classifier in 1.4. For each term we need to calculate in $\prod_{i=1}^{n} P(z_i \mid y_k)$, we can use 2.7 to calculate the probability of $P(z_i \mid y_k)$, and we can use 1.6 to calculate the $P(y_k)$ term.

However, to use this, we need to compute all the possible values for 2.7. By automating the counting functions in 1.5 for each feature given a class to get our values to substitute into 2.6, we get our necessary values seen in Figure 2.7. This essentially looped through each combination of a given feature and its number of occurrences when it had heart disease and when it did not. It then divides by the total amount of occurrences present within its corresponding class, finally converting into a percentage for easier readability.

| Feature | Feature_Value | Heart_Disease_Status | Count | Proportion (%) |
|---|---|---|---|---|
| Sex | F | 0 | 143 | 34.878049 |
| Sex | F | 1 | 50 | 9.842520 |
| Sex | M | 0 | 267 | 65.121951 |
| Sex | M | 1 | 458 | 90.157480 |
| ChestPainType | ASY | 0 | 104 | 25.365854 |
| ChestPainType | ASY | 1 | 392 | 77.165354 |
| ChestPainType | ATA | 0 | 149 | 36.341463 |
| ChestPainType | ATA | 1 | 24 | 4.724409 |
| ChestPainType | NAP | 0 | 131 | 31.951220 |
| ChestPainType | NAP | 1 | 72 | 14.173228 |
| ChestPainType | TA | 0 | 26 | 6.341463 |
| ChestPainType | TA | 1 | 20 | 3.937008 |
| FastingBS | 0 | 0 | 366 | 89.268293 |
| FastingBS | 0 | 1 | 338 | 66.535433 |
| FastingBS | 1 | 0 | 44 | 10.731707 |
| FastingBS | 1 | 1 | 170 | 33.464567 |
| RestingECG | LVH | 0 | 82 | 20.000000 |
| RestingECG | LVH | 1 | 106 | 20.866142 |
| RestingECG | Normal | 0 | 267 | 65.121951 |
| RestingECG | Normal | 1 | 285 | 56.102362 |
| RestingECG | ST | 0 | 61 | 14.878049 |
| RestingECG | ST | 1 | 117 | 23.031496 |
| ExerciseAngina | N | 0 | 355 | 86.585366 |
| ExerciseAngina | N | 1 | 192 | 37.795276 |
| ExerciseAngina | Y | 0 | 55 | 13.414634 |
| ExerciseAngina | Y | 1 | 316 | 62.204724 |
| ST_Slope | Down | 0 | 14 | 3.414634 |
| ST_Slope | Down | 1 | 49 | 9.645669 |
| ST_Slope | Flat | 0 | 79 | 19.268293 |
| ST_Slope | Flat | 1 | 381 | 75.000000 |
| ST_Slope | Up | 0 | 317 | 77.317073 |
| ST_Slope | Up | 1 | 78 | 15.354331 |

**Figure 2.7:** *Calculated Probabilities of each Feature Given Class*

At this stage, utilizing this data is relatively straightforward. For example, a data point may have the following discrete values:

$$\mathbf{z} = \begin{bmatrix} \text{M} \\ \text{ASY} \\ 0 \\ \text{Normal} \\ \text{N} \\ \text{Flat} \end{bmatrix} \tag{2.8}$$

Next, referencing the table in Figure 2.7, we can calculate the probabilities given that the data point has heart disease or not.

$$P(y_0 \mid \mathbf{z}) = (0.65) * (0.25) * (0.89) * (0.65) * (0.86) * (0.19) \approx 0.01536$$

$$P(y_1 \mid \mathbf{z}) = (0.90) * (0.77) * (0.66) * (0.56) * (0.37) * (0.75) \approx 0.07107$$

Note that we are only substituting numbers pre-calculated in Figure 2.7. For instance, 0.65 corresponds to the probability of Male, given no heart disease. Conversely, 0.90 corresponds to the probability of Male, given heart disease.

Therefore, since $0.01536 < 0.07107$, we take $y_1$ as the result, which means that this data point likely has heart disease. We see some of the significant values contributing towards this result due to the data point being male, having ASY chest pain, and having a flat ST Slope, which all have a probability of above 0.7

Again, we can apply this process generally for any time we need to calculate the probabilities of a data point's categorical data.

## 2.3 Modeling

Now that we have our methods of generating probabilities suited to our data points, we still have a problem: how do we combine the probability generated by Gaussian Naive Bayes and the probability generated by Multinomial Naive Bayes? To remedy this issue, I see three possible options.

1. Use another Gaussian Naive Bayes and substitute the two values from the previous Gaussian and Multinomial to generate a new distribution to make predictions off of.
2. Derive a new equation that allows for both the usage of continuous and categorized data simultaneously in the same equation, allowing us to multiply across different data types.
3. Multiply the two probabilities together and compare the results of the two classes.

While the first option might be the more straightforward and intuitive approach, it would require computer science and significant computational power. To create the new Gaussian distribution, it would require me to recursively iterate through each

data point, run through a naive Bayes computation based on its respective data type, and then run another Gaussian naive Bayes computation on top of that. Though feasible at my scale - around 900 data points, with a decent computer capable of the task - I know that it would not be viable at a much larger scale, hence its computational inefficiency.

The next approach, however, appealed to me as it would only require one round of computation. This meant that my model would run more cleanly, which is important in real-world situations when millions of data points are being required to process, and the quality of lives of those in need is at stake. Moreover, this equation has already been explored by Farhangi, 2024. That being said, the mathematics that it requires is beyond my current level of understanding, which unfortunately makes it unviable for me.

This leads to my last option, which is what I ultimately chose. The reason I chose this is first because of its simplicity, and also since this is Naive Bayes, and I'm allowed to multiply across these data types as the features are independent of each other. One concern that may arise is determining the weight of either the categorical or continuous data. Since the Gaussian Naive Bayes equation usually outputs values quite low, one may think that multiplying this on the output of the multinomial distribution may be erroneous. However, we need to remember that I will take the *argmax* of the two values and that the values are relative to each other. Therefore, the continuous data outputting significantly lower values than the categorical data would not matter as we are only looking at the difference between our two classes, heart disease or no heart disease. Therefore, to determine my final prediction, I will use the following equation:

$$y \propto \underset{k \in \{0,1\}}{\operatorname{argmax}} P(y_k) P(\mathbf{z}_{\text{discrete}} \mid y_k) P(\mathbf{z}_{\text{continuous}} \mid y_k)$$

By taking the products of $\prod_{i=1}^{n} P(\mathbf{z}_{\text{discrete}_i} \mid y_k)$ and $\prod_{i=1}^{n} P(\mathbf{z}_{\text{continuous}_i} \mid y_k)$, we can calculate the total probability of both the discrete data and the continuous data.

Therefore, the generalized equation that I will use is

$$y \propto \underset{k \in \{0,1\}}{\operatorname{argmax}} P(y_k) \prod_{i=1}^{n} P(\mathbf{z}_{discrete_i} \mid y_k) \prod_{i=1}^{n} P(\mathbf{z}_{continuous_i} \mid y_k) \tag{2.9}$$

### 2.3.1 Testing

Now, we can evaluate the performance of the equation seen in 2.9. By randomly sampling 20 percent of the dataset, each one being $\mathbf{x}$, I can substitute all my values and compare the result of the *argmax* to the real expected result. By adding up the successful data points and dividing by the total amount of data points taken, I can get the success rate of my model.

$$\text{success rate} = \frac{\sum_{i=1}^{|x|} 1[y \in y_k]}{|x|}$$

where $|x|$ is the number of data points sampled from the dataset and $y$ is the predicted class, taken from substituting the values of $\mathbf{x}$ into 2.9.

By doing this, the result I get is seen in Figure 17, with a success rate of 76.09%.

# 3

# Conclusion

## 3.1 Restated Aim

To recap, my aim for this project was to evaluate the extent to which a combined Naive Bayes Classifier can predict heart disease.

## 3.2 Evaluation

As seen in the results of my test in Figure 17, the extent to which the Naive Bayes Classifier can be applied to predicting Heart Disease is to a moderate extent. While a 76.09% accuracy represents a significant margin for improvement, the Naive Bayes Classifier is still a simple and interpretable method of generating insights on sets of data. Furthermore, because of its relatively straightforward sets of calculations, the model can handle complex datasets comprising a variety of data types in a computationally efficient manner.

Because of this, I believe this model has the potential to serve as an initial screening tool in low-resource environments. Using its data-driven approach, high-risk patients may be able to be efficiently identified for further testing and detailed evaluation. Moreover, since we can see which features are weighed more heavily, the Naive Bayes approach may be able to help clinicians and researchers identify which health indicators are the most predictive of heart disease. Ultimately, a model such as this may be applied to the medical field by complementing expert clinical judgment, providing a probabilistic estimate that can aid decision-making.

## 3.3 Overview

An overview of what I did throughout this paper is as follows:

1. Acquired a dataset on heart disease from Kaggle
2. Modified Bayes' Theorem to suit the dataset

3. Rearranged and simplified the equation in regards to the Naive Bayes' assumption
4. Determined an expression for predicting an unknown data point's class with the *minimum a posteriori* decision rule
5. Determined the prior $P(y_k)$ with a counting function
6. Split the dataset by continuous and discrete data
7. Used the formula for mean and variance to compute needed values in Gaussian Naive Baye's to hand continuous data
8. Used another counting function to compute probabilities of categorical data
9. Substituted the output of both Gaussian and Multinomial Bayes into the original Naive Bayes' equation, taking the higher value of $k = 0$ and $k = 1$
10. Simulated this process with randomly sampled data points from the dataset, comparing the calculated result with the actual result, obtaining its success rate

## 3.4 Limitations

There are a few limitations to my investigation, with the most notable being in the name: Naive Bayes is "naive". By multiplying probabilities together, assuming their independence, we miss crucial details of how one factor compares to another.

In the context of my medical data, this is often unrealistic as health indicators such as age, blood pressure, and cholesterol are often correlated. Next, going back to the Section 2.1, recall that to make probability densities, we needed to assume that our data was normal. Seeing that there were slight deviations from normality in most of the continuous features and a very large deviation when observing Oldpeak, the model may have suffered from a significant decrease in accuracy.

Additionally, coupled with the assumption of independence from Naive Bayes, if highly correlated features were to deviate from normality, they would have an exacerbated effect where accuracy is impacted.

## 3.5 Extension to Investigation

A big improvement can be made through choosing a more sophisticated model, such as logistic regression, decision trees, or neural networks which can capture feature-feature interactions and improve accuracy. On the same line, using Bayesian networks that extend Bayes' Theorem so that dependencies between features can be explicitly modeled, certain features may be able to exert influence on others, increasing accuracy. Additionally, to address the normality issue for Gaussian Naive Bayes, applying log transforms to skewed continuous data can make distributions more symmetrical and thus, more normal Feng et al., 2014.

In summary, while the model's accuracy is acceptable, significant enhancements can be made to further its applicability to high-stakes environments such as healthcare

or finance. By furthering development and testing on larger datasets, its reliability, and impacts on real-world scenarios can be enhanced.

# Bibliography


Clinic, Mayo (Nov. 2020). *Aging: What to expect*. Mayo Clinic. URL: https://www.mayoclinic.org/healthy-lifestyle/healthy-aging/in-depth/aging/art-20046070.

Farhangi, Ashkon (2024). *ashkonf/HybridNaiveBayes: A Generalized Implementation of the Naive Bayes Classifier in Python*. GitHub. URL: https://github.com/ashkonf/HybridNaiveBayes (visited on 12/12/2024).

fedesoriano (Sept. 2021). *Heart Failure Prediction Dataset*. Kaggle. URL: https://www.kaggle.com/fedesoriano/heart-failure-prediction (visited on 12/07/2024).

Feng, C et al. (Apr. 2014). "Log-transformation and its implications for data analysis." In: *Shanghai archives of psychiatry* 26, pp. 105–109. DOI: 10.3969/j.issn.1002-0829.2014.02.009. URL: http://europepmc.org/articles/PMC4120293.

Gaur, Durgance (Sept. 2021). *A Guide to Any Classification Problem*. kaggle.com. URL: https://www.kaggle.com/code/durgancegaur/a-guide-to-any-classification-problem.

Instruments, Texas (n.d.). *TI-84 Plus CE Python| Graphing Calculator| Texas Instruments*. education.ti.com. URL: https://education.ti.com/en/products/calculators/graphing-calculators/ti-84-plus-ce-python.

*Naïve Bayes Model - an overview* (n.d.). www.sciencedirect.com. URL: https://www.sciencedirect.com/topics/computer-science/naive-bayes-model.

*Notebooks Documentation* (n.d.). www.kaggle.com. URL: https://www.kaggle.com/docs/notebooks.

Pishro-Nik, Hossein (n.d.). *Maximum A Posteriori (MAP) Estimation*. www.probabilitycourse.com. URL: https://www.probabilitycourse.com/chapter9/9_1_2_MAP_estimation.php.

PYTHON (May 2019). *Python*. Python.org. URL: https://www.python.org/.

Savarese, Gianluigi et al. (Feb. 2022). "Global Burden of Heart failure: a Comprehensive and Updated Review of Epidemiology". In: *Cardiovascular Research* 118. DOI: 10.1093/cvr/cvac013.

# Appendices

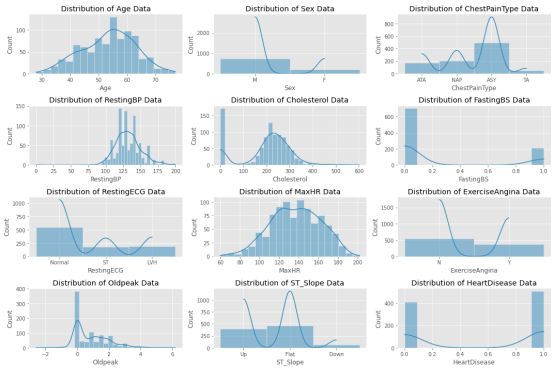


**Figure 1:** *Visualization of the Dataset, Source: Gaur, 2021*

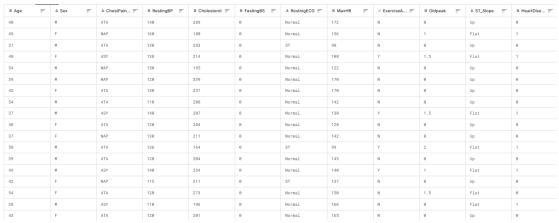

**Figure 2:** *The Dataset, Source: Photo by Author*

```
Mean Age for people without Heart Disease: 50.551219512195125
Variance of Age for people without Heart Disease: 89.2064166020633
Mean Age for people with Heart Disease: 55.8996062992126
Variance of Age for people with Heart Disease: 76.16149885850054
```

**Figure 3:** *Mean and Variance of Age Given Heart Disease Conditions, Source: Photo by Author*

```
df_no_heart_disease = df[df['HeartDisease'] == 0]

mean_no_heart_disease = df_no_heart_disease['Age'].mean()
variance_no_heart_disease = df_no_heart_disease['Age'].var()

print("Mean Age for people without Heart Disease:", mean_no_heart_disease)
print("Variance of Age for people without Heart Disease:", variance_no_heart_disease)

df_heart_disease = df[df['HeartDisease'] == 1]

mean_heart_disease = df_heart_disease['Age'].mean()
variance_heart_disease = df_heart_disease['Age'].var()

print("Mean Age for people with Heart Disease:", mean_heart_disease)
print("Variance of Age for people with Heart Disease:", variance_heart_disease)
```

**Figure 4:** *Script for Mean and Variance of Age Given Heart Disease Conditions, Source: Photo by Author*

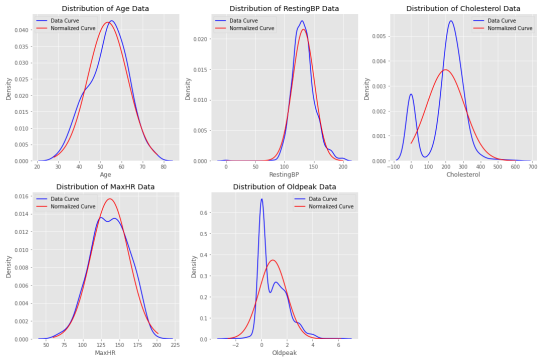


**Figure 5:** *Visualization of the Continuous Data, and its Normalized Counterpart, Source: Photo by author*

```
from scipy.stats import norm
plt.figure(figsize=(15,10))
continuousColumns = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"]

for i in range(1, 6):
    plt.subplot(2, 3, i)
    plt.title(f"Distribution of {continuousColumns[i-1]} Data")

    sns.kdeplot(df[continuousColumns[i-1]], label='Data Curve', color='blue')

    mean = df[continuousColumns[i-1]].mean()
    std = df[continuousColumns[i-1]].std()

    x = np.linspace(df[continuousColumns[i-1]].min(), df[continuousColumns[i-1]].max(), 100)

    normal_curve = norm.pdf(x, mean, std)

    plt.plot(x, normal_curve, label='Normalized Curve', color='red')

    plt.legend()
    plt.tight_layout()

plt.show()
```

**Figure 6:** *Script for Visualization of the Continuous Data, and its Normalized Counterpart, Source: Photo by author*

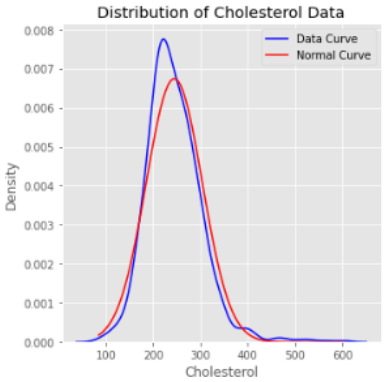


**Figure 7:** *Corrected Cholesterol Graph, Source: Photo by Author*

```
from scipy.stats import norm
import numpy as np

df_filtered = df.df['Cholesterol'] != 0]

plt.figure(figsize=(15,10))
continuousColumns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']

for i in range(1, 6):
    plt.subplot(2, 3, i)
    plt.title(f'Distribution of {continuousColumns[i-1]} Data')

    column_data = df_filtered[continuousColumns[i-1]] if continuousColumns[i-1] == 'Cholesterol' else df[continuousColumns[i-1]]

    sns.kdeplot(column_data, label='Data Curve', color='blue')

    mean = column_data.mean()
    std = column_data.std()

    x = np.linspace(column_data.min(), column_data.max(), 100)

    normal_curve = norm.pdf(x, mean, std)

    plt.plot(x, normal_curve, label='Normal Curve', color='red')

    plt.legend()
    plt.tight_layout()

plt.show()
```

**Figure 8:** *Script for Corrected Cholesterol Graph, Source: Photo by Author*

| | Mean (HeartDisease=0) | Variance (HeartDisease=0) |
|---|---|---|
| Age | 50.551220 | 88.988840 |
| RestingBP | 130.180488 | 271.572302 |
| Cholesterol | 227.121951 | 5556.746104 |
| MaxHR | 148.151220 | 541.011279 |
| Oldpeak | 0.408049 | 0.488399 |

| | Mean (HeartDisease=1) | Variance (HeartDisease=1) |
|---|---|---|
| Age | 55.899606 | 76.011575 |
| RestingBP | 134.185039 | 392.402768 |
| Cholesterol | 175.940945 | 15943.339032 |
| MaxHR | 127.655512 | 545.871485 |
| Oldpeak | 1.274213 | 1.324197 |

**Figure 9:** *Calculated Mean and Variance of Continuous Features, Source: Photo by Author*

```
continuousColumns = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"]

mean_variance_data = {}

for column in continuousColumns:
    sum_0 = 0
    sum_squared_0 = 0
    count_0 = 0

    sum_1 = 0
    sum_squared_1 = 0
    count_1 = 0

    for _, row in df.iterrows():
        if row['HeartDisease'] == 0:
            sum_0 += row[column]
            sum_squared_0 += row[column] ** 2
            count_0 += 1
        elif row['HeartDisease'] == 1:
            sum_1 += row[column]
            sum_squared_1 += row[column] ** 2
            count_1 += 1

    mean_0 = sum_0 / count_0
    var_0 = (sum_squared_0 / count_0) - (mean_0 ** 2)

    mean_1 = sum_1 / count_1
    var_1 = (sum_squared_1 / count_1) - (mean_1 ** 2)

    mean_variance_data[column] = {
        'Mean (HeartDisease=0)': mean_0,
        'Variance (HeartDisease=0)': var_0,
        'Mean (HeartDisease=1)': mean_1,
        'Variance (HeartDisease=1)': var_1
    }

mean_variance_table = pd.DataFrame(mean_variance_data).T
print(mean_variance_table)
```

**Figure 10:** *Script for Calculated Mean and Variance of Continuous Features, Source: Photo by Author*

```
Gaussian Naive Bayes Calculations:
       Feature  Value        Mean      Variance  Gaussian_Prob  Class
0          Age   40.0   50.551220     88.988840       0.022625      0
1          Age   40.0   55.899606     76.011575       0.008675      1
2    RestingBP  140.0  130.180488    271.572302       0.020271      0
3    RestingBP  140.0  134.185039    392.402768       0.019290      1
4  Cholesterol  289.0  227.121951   5556.746104       0.003792      0
5  Cholesterol  289.0  175.940945  15943.339032       0.002116      1
6        MaxHR  172.0  148.151220    541.011279       0.010140      0
7        MaxHR  172.0  127.655512    545.871485       0.002819      1
8      Oldpeak    0.0    0.408049      0.488399       0.481386      0
9      Oldpeak    0.0    1.274213      1.324197       0.187796      1

Final Probabilities:
P(HeartDisease=0) = 8.48866278005488e-09
P(HeartDisease=1) = 1.8747803450406336e-10
```

**Figure 11:** *Calculations for a Single Data Point*

```python
continuousColumns = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"]

mean_variance_data = {}

for column in continuousColumns:
    mean_0 = df[df['HeartDisease'] == 0][column].mean()
    var_0 = df[df['HeartDisease'] == 0][column].var()

    mean_1 = df[df['HeartDisease'] == 1][column].mean()
    var_1 = df[df['HeartDisease'] == 1][column].var()

    mean_variance_data[column] = {
        'Mean (No Heart Disease)': mean_0,
        'Variance (No Heart Disease)': var_0,
        'Mean (Heart Disease)': mean_1,
        'Variance (Heart Disease)': var_1
    }

mean_variance_table = pd.DataFrame(mean_variance_data).T
print(mean_variance_table)
```

**Figure 12:** *Script for Calculations for a Single Data Point*

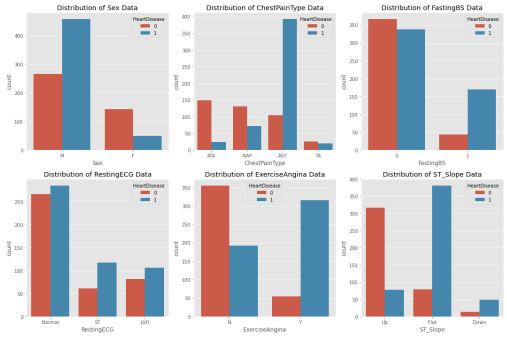


**Figure 13:** *Categorical Data Separated by Presence of Heart Disease, Source: Photo by Author*

```python
import matplotlib.pyplot as plt
import seaborn as sns

discreteColumns = ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'ExerciseAngina', 'ST_Slope']

plt.figure(figsize=(15, 10))

for i in range(1, len(discreteColumns) + 1):
    plt.subplot(2, 3, i)
    plt.title(f'Distribution of {discreteColumns[i - 1]} Data')
    sns.countplot(x=df[discreteColumns[i - 1]], hue=df['HeartDisease'])  # Use countplot with hue for HeartDisease
    plt.tight_layout()

plt.show()
```

**Figure 14:** *Script for Categorical Data Separated by Presence of Heart Disease, Source: Photo by Author*

| Feature | Feature_Value | Heart_Disease_Status | Count | Proportion (%) |
|---|---|---|---|---|
| Sex | F | 0 | 143 | 34.878049 |
| Sex | F | 1 | 50 | 9.842520 |
| Sex | M | 0 | 267 | 65.121951 |
| Sex | M | 1 | 458 | 90.157480 |
| ChestPainType | ASY | 0 | 104 | 25.365854 |
| ChestPainType | ASY | 1 | 392 | 77.165354 |
| ChestPainType | ATA | 0 | 149 | 36.341463 |
| ChestPainType | ATA | 1 | 24 | 4.724409 |
| ChestPainType | NAP | 0 | 131 | 31.951220 |
| ChestPainType | NAP | 1 | 72 | 14.173228 |
| ChestPainType | TA | 0 | 26 | 6.341463 |
| ChestPainType | TA | 1 | 20 | 3.937008 |
| FastingBS | 0 | 0 | 366 | 89.268293 |
| FastingBS | 0 | 1 | 338 | 66.535433 |
| FastingBS | 1 | 0 | 44 | 10.731707 |
| FastingBS | 1 | 1 | 170 | 33.464567 |
| RestingECG | LVH | 0 | 82 | 20.000000 |
| RestingECG | LVH | 1 | 106 | 20.866142 |
| RestingECG | Normal | 0 | 267 | 65.121951 |
| RestingECG | Normal | 1 | 285 | 56.102362 |
| RestingECG | ST | 0 | 61 | 14.878049 |
| RestingECG | ST | 1 | 117 | 23.031496 |
| ExerciseAngina | N | 0 | 355 | 86.585366 |
| ExerciseAngina | N | 1 | 192 | 37.795276 |
| ExerciseAngina | Y | 0 | 55 | 13.414634 |
| ExerciseAngina | Y | 1 | 316 | 62.204724 |
| ST_Slope | Down | 0 | 14 | 3.414634 |
| ST_Slope | Down | 1 | 49 | 9.645669 |
| ST_Slope | Flat | 0 | 79 | 19.268293 |
| ST_Slope | Flat | 1 | 381 | 75.000000 |
| ST_Slope | Up | 0 | 317 | 77.317073 |
| ST_Slope | Up | 1 | 78 | 15.354331 |

**Figure 15:** *Calculated Probabilities of each Feature Given Class*

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

discreteColumns = ["Sex", "ChestPainType", "FastingBS", "RestingECG", "ExerciseAngina", "ST_Slope"]

heart_disease_totals = {0: 410, 1: 508}  # Total samples for HeartDisease = 0 and 1

summary_data = []

for column in discreteColumns:
    grouped = df.groupby([column, 'HeartDisease']).size().reset_index(name='Count')

    grouped['Proportion'] = grouped.apply(
        lambda row: row['Count'] / heart_disease_totals[row['HeartDisease']], axis=1
    )

    grouped.rename(columns={
        column: 'Feature_Value',
        'HeartDisease': 'Heart_Disease_Status',
        'Count': 'Count',
        'Proportion': 'Proportion (%)'
    }, inplace=True)
    grouped['Proportion (%)'] = grouped['Proportion (%)'] * 100

    grouped['Feature'] = column
    summary_data.append(grouped)

table = pd.concat(summary_data)[['Feature', 'Feature_Value', 'Heart_Disease_Status', 'Count', 'Proportion (%)']]

print(table)
```

**Figure 16:** *Script for Calculated Probabilities of each Feature Given Class*

Prediction Results:

| Sample_Index | Actual_Class | Predicted_Class | P(HeartDisease=0) | P(HeartDisease=1) |
|---|---|---|---|---|
| 99 | 0 | 0 | 1.996496e-09 | 1.071859e-12 |
| 731 | 1 | 0 | 6.275118e-10 | 2.387041e-11 |
| 544 | 0 | 0 | 5.765076e-12 | 3.732530e-13 |
| 363 | 1 | 1 | 1.116527e-12 | 5.660443e-11 |
| 594 | 1 | 1 | 5.812393e-13 | 2.170152e-11 |
| 514 | 1 | 1 | 1.017060e-11 | 2.059762e-11 |
| 42 | 0 | 0 | 3.193650e-10 | 6.250439e-14 |
| 488 | 0 | 0 | 1.779424e-10 | 4.084305e-12 |
| 463 | 0 | 1 | 5.805278e-12 | 2.932474e-11 |
| 55 | 0 | 0 | 2.527218e-10 | 6.580610e-14 |
| 694 | 0 | 0 | 1.286793e-09 | 9.905000e-13 |
| 213 | 0 | 0 | 2.989811e-10 | 3.186907e-13 |
| 802 | 0 | 0 | 1.496437e-10 | 1.878979e-11 |
| 475 | 1 | 1 | 1.910228e-16 | 6.177189e-12 |
| 818 | 1 | 0 | 3.705076e-11 | 3.177396e-11 |
| 720 | 1 | 0 | 1.437478e-10 | 4.207587e-11 |
| 543 | 1 | 1 | 2.282679e-17 | 1.870449e-12 |
| 684 | 1 | 0 | 1.330207e-09 | 3.233420e-12 |
| 221 | 1 | 1 | 5.748279e-12 | 3.088163e-11 |
| 452 | 1 | 1 | 1.512749e-12 | 4.683806e-10 |
| 20 | 0 | 0 | 5.094321e-11 | 4.571863e-14 |
| 888 | 1 | 1 | 9.862759e-12 | 4.111943e-10 |
| 400 | 1 | 1 | 2.276292e-14 | 3.655627e-12 |
| 102 | 1 | 1 | 5.701039e-13 | 3.601947e-12 |
| 282 | 0 | 0 | 2.350862e-11 | 2.117582e-12 |
| 872 | 1 | 1 | 1.176684e-11 | 7.575773e-10 |
| 34 | 0 | 0 | 6.539826e-10 | 1.208109e-13 |
| 863 | 1 | 1 | 1.417960e-13 | 1.213929e-11 |
| 317 | 1 | 1 | 3.055390e-14 | 6.747995e-11 |
| 3 | 1 | 1 | 4.022438e-12 | 9.767428e-11 |
| 896 | 0 | 0 | 1.333322e-09 | 1.113846e-12 |
| 157 | 0 | 0 | 1.510382e-09 | 9.779154e-11 |
| 727 | 1 | 0 | 3.465817e-11 | 5.319181e-13 |
| 160 | 1 | 1 | 9.377675e-13 | 1.392942e-10 |
| 566 | 0 | 1 | 5.908295e-12 | 3.454876e-10 |
| 173 | 0 | 0 | 1.547765e-09 | 2.976737e-12 |
| 252 | 0 | 1 | 1.315586e-11 | 4.072573e-11 |
| 886 | 0 | 0 | 2.161360e-09 | 4.474203e-12 |
| 30 | 1 | 1 | 2.145648e-13 | 2.610157e-12 |
| 808 | 1 | 0 | 2.480564e-11 | 1.504766e-12 |
| 908 | 1 | 1 | 8.986705e-17 | 5.156401e-12 |
| 613 | 0 | 1 | 6.979014e-12 | 7.636179e-12 |
| 562 | 0 | 1 | 4.432954e-12 | 6.028003e-10 |
| 526 | 1 | 1 | 2.691212e-13 | 1.710155e-10 |
| 504 | 1 | 1 | 9.798600e-17 | 1.284449e-11 |
| 917 | 0 | 0 | 5.978276e-10 | 4.688923e-13 |

Model Accuracy: 76.09%

**Figure 17:** *Model Performance, Source: Photo by Author*

```
discreteColumns = ["Sex", "ChestPainType", "FastingBS", "RestingECG", "ExerciseAngina", "ST_Slope"]
continuousColumns = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"]
heart_disease_totals = {0: 410, 1: 508}
summary_data = []
for column in discreteColumns:
    grouped = df.groupby([column, 'HeartDisease']).size().reset_index(name='Count')
    grouped['Proportion'] = grouped.apply(
        lambda row: row['Count'] / heart_disease_totals[row['HeartDisease']], axis=1
    )
    grouped.rename(columns={
        column: 'Feature_Value',
        'HeartDisease': 'Heart_Disease_Status'
    }, inplace=True)
    grouped['Feature'] = column
    summary_data.append(grouped)
table = pd.concat(summary_data)
mean_variance_data = {}
for column in continuousColumns:
    sum_0, sum_squared_0, count_0 = 0, 0, 0
    sum_1, sum_squared_1, count_1 = 0, 0, 0
    for _, row in df.iterrows():
        if row['HeartDisease'] == 0:
            sum_0 += row[column]
            sum_squared_0 += row[column] ** 2
            count_0 += 1
        elif row['HeartDisease'] == 1:
            sum_1 += row[column]
            sum_squared_1 += row[column] ** 2
            count_1 += 1
    mean_0 = sum_0 / count_0
    var_0 = (sum_squared_0 / count_0) - (mean_0 ** 2)
    mean_1 = sum_1 / count_1
    var_1 = (sum_squared_1 / count_1) - (mean_1 ** 2)
    mean_variance_data[column] = {
        'Mean (HeartDisease=0)': mean_0,
        'Variance (HeartDisease=0)': var_0,
        'Mean (HeartDisease=1)': mean_1,
        'Variance (HeartDisease=1)': var_1
    }
sampled_df = df.sample(frac=0.05, random_state=69)
correct_predictions = 0
results = []
for _, sample in sampled_df.iterrows():
    probabilities_discrete = {0: 1.0, 1: 1.0}
```

**Figure 18:** *Script for Model Performance 1/2, Source: Photo by Author*

```
for _, sample in sampled_df.iterrows():
    probabilities_discrete = {0: 1.0, 1: 1.0}
    for column in discreteColumns:
        value = sample[column]
        for label in [0, 1]:
            proportion = table[(table['Feature'] == column) &
                               (table['Feature_Value'] == value) &
                               (table['Heart_Disease_Status'] == label)]['Proportion'].values
            if len(proportion) == 0:
                proportion = [0.01]
            prob = proportion[0]
            probabilities_discrete[label] *= prob
    probabilities_continuous = {0: 1.0, 1: 1.0}
    for column in continuousColumns:
        x = sample[column]
        for label in [0, 1]:
            mean = mean_variance_data[column][f'Mean (HeartDisease={label})']
            variance = mean_variance_data[column][f'Variance (HeartDisease={label})']
            std_dev = math.sqrt(variance)
            exponent = math.exp(-((x - mean) ** 2) / (2 * variance))
            gaussian_prob = (1 / (math.sqrt(2 * math.pi) * std_dev)) * exponent
            probabilities_continuous[label] *= gaussian_prob
    final_probabilities = {label: probabilities_discrete[label] * probabilities_continuous[label] for label in [0, 1]}
    predicted_class = max(final_probabilities, key=final_probabilities.get)
    actual_class = sample['HeartDisease']
    if predicted_class == actual_class:
        correct_predictions += 1
    results.append({
        'Sample_Index': sample.name,
        'Actual_Class': actual_class,
        'Predicted_Class': predicted_class,
        'P(HeartDisease=0)': final_probabilities[0],
        'P(HeartDisease=1)': final_probabilities[1],
    })
accuracy = correct_predictions / len(sampled_df)
results_df = pd.DataFrame(results)
print("\nPrediction Results:")
print(results_df.to_string(index=False))
print(f"\nModel Accuracy: {accuracy * 100:.2f}%")
```

**Figure 19:** *Script for Model Performance 2/2, Source: Photo by Author*