



American International University-Bangladesh >>>

FACULTY OF SCIENCE & TECHNOLOGY

INTRODUCTION TO DATA SCIENCE

Spring 2024-25

Section: D

Group: 04

Mid-Term Project Report

Supervised by

TOHEDUL ISLAM

Submitted by:

<u>Name</u>	<u>ID</u>
Ashiqur Rahman Saron	22-48697-3
Soumodip Madhu	22-48707-3
Chayti Rani Mondol	22-48731-3
MD. Rifat Khan	22-46060-1

Submission Date: April 26, 2025

Data Overview: This dataset is designed for heart disease prediction and includes a combination of traditional clinical attributes along with an advanced feature for enhanced predictive modeling. It provides 151 samples, making it a solid foundation for testing machine learning techniques.

Attributes:

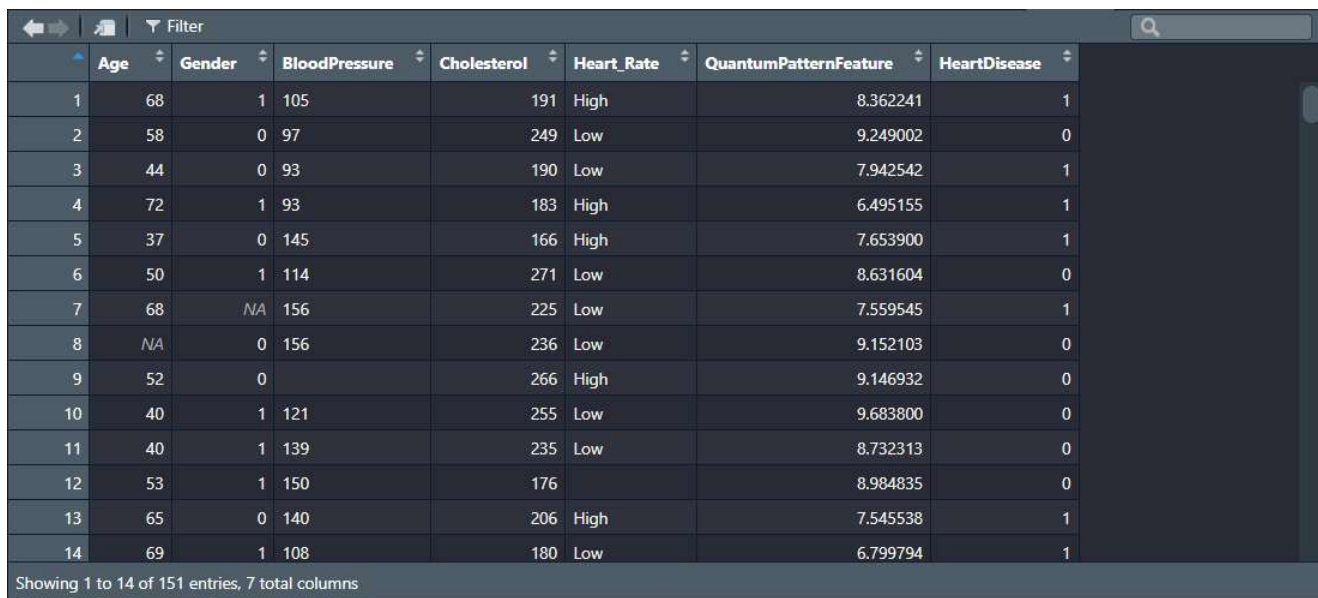
1. **Age:** Continuous variable representing the patient's age.
2. **Gender:** Binary categorical variable where “Male = 1” and “Female = 0”.
3. **BloodPressure:** Continuous variable measuring blood pressure levels.
4. **Cholesterol:** Continuous variable indicating cholesterol levels.
5. **Heart_Rate:** Continuous or categorical variable showing heart rate levels.
6. **QuantumPatternFeature:** A unique feature capturing intricate, non-linear relationships, adding complexity for advanced models.
7. **HeartDisease:** Binary categorical variable (0 or 1), representing the presence or absence of heart disease.

Importing and Viewing the Dataset: This step loads the CSV file into R, ensuring proper column headers and delimiter settings, then opens it for inspection in RStudio's data viewer.

➤ **Code:**

```
file_path <- "C:\\Users\\ashik\\OneDrive\\Desktop\\Dataset_MIdterm_sectoin(D).csv"
df <- read.csv(file_path,header=TRUE,sep = ",")
View(df)
```

➤ **Output:**



	Age	Gender	BloodPressure	Cholesterol	Heart_Rate	QuantumPatternFeature	HeartDisease
1	68	1	105	191	High	8.362241	1
2	58	0	97	249	Low	9.249002	0
3	44	0	93	190	Low	7.942542	1
4	72	1	93	183	High	6.495155	1
5	37	0	145	166	High	7.653900	1
6	50	1	114	271	Low	8.631604	0
7	68	NA	156	225	Low	7.559545	1
8	NA	0	156	236	Low	9.152103	0
9	52	0		266	High	9.146932	0
10	40	1	121	255	Low	9.683800	0
11	40	1	139	235	Low	8.732313	0
12	53	1	150	176		8.984835	0
13	65	0	140	206	High	7.545538	1
14	69	1	108	180	Low	6.799794	1

This code imports a CSV file from the specified path and assigns it to the variable df. The header = TRUE argument indicates that the first row of the CSV contains column names, while sep = "," specifies that the file uses commas as delimiters. The View(df) command displays the imported dataset in a tabular format in RStudio's data viewer for easy inspection.

Examining the Dataset Dimensions:

➤ Code:

```
cat("Number of rows:", nrow(df),"\n")
cat("Number of columns:", ncol(df))
```

➤ Output:

```
> cat("Number of rows:", nrow(df),"\n")
Number of rows: 151
> cat("Number of columns:", ncol(df))
Number of columns: 7
```

This code outputs 151 rows, and 7 attributes columns for the dataset.

Inspecting Dataset Structure: Examining the structure and attributes of the dataset.

➤ Code:

```
str(df)
```

➤ Output:

```
> str(df)
'data.frame': 151 obs. of 7 variables:
 $ Age      : int  68 58 44 72 37 50 68 NA 52 40 ...
 $ Gender   : int  1 0 0 1 0 1 NA 0 0 1 ...
 $ BloodPressure : chr  "105" "97" "93" "93" ...
 $ Cholesterol : int  191 249 190 183 166 271 225 236 266 255 ...
 $ Heart_Rate  : chr  "High" "Low" "Low" "High" ...
 $ QuantumPatternFeature: num  8.36 9.25 7.94 6.5 7.65 ...
 $ HeartDisease : int  1 0 1 1 1 0 1 0 0 0 ...
```

The output showcases us that Age, Gender, HeartDisease and Cholesterol are integer (int), BloodPressure and Heart_Rate are character (chr), QuantumPatternFeature is numeric (num) type attribute.

Handling Missing Values:

1. **Checking For Missing Values:** The code snippet given below checks for missing value in each column by checking for NA values. For every empty character cell in converts it to NA.

➤ Code:

```
df[df==""]<- NA
num_of_missing_value_per_column <- sapply(df, function(x) sum(is.na(x)))
print(num_of_missing_value_per_column)
print(paste("Total missing values:",sum(is.na(df))))
```

➤ Output:

```

> df[df==""]<- NA
> num_of_missing_value_per_column <- sapply(df, function(x) sum(is.na(x)))
> print(num_of_missing_value_per_column)
      Age      Gender      BloodPressure
      3         3         3
Cholesterol Heart_Rate QuantumPatternFeature
      0         3         0
HeartDisease
      0
> print(paste("Total missing values:",sum(is.na(df))))
[1] "Total missing values: 12"

```

The output shows us that there are a total of 12 missing values including empty characters cell in the data frame. For attributes Age, Gender, BloodPressure, HeartRate there are total of 3 missing values. Attributes of column cholesterol, QuantamPatternFeature and HeartDisease have no missing values.

2. Graph For Missing Values:

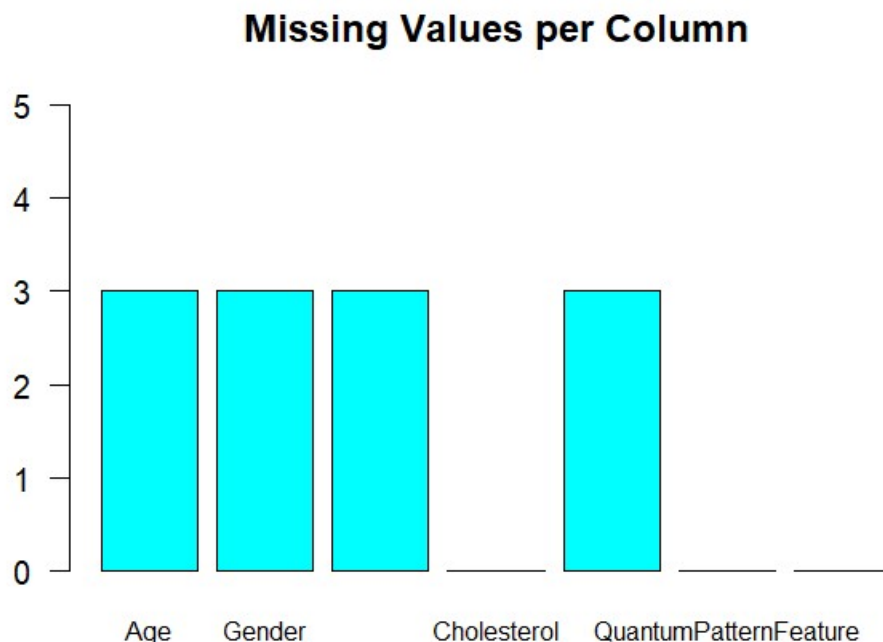
➤ Code:

```

num_of_missing_value_per_column <- sapply(df, function(x) sum(is.na(x)))
barplot(num_of_missing_value_per_column,
      main = "Missing Values per Column",
      col = "cyan",
      las = 1,
      cex.names = 0.8,
      ylim = c(0, 5))

```

➤ Output:



Here the code calculates the number of missing values in each column of the dataset df using sums and is.na function. It then creates a bar plots to visualize the missing values per column.

3. Replacing Missing Value Using Mean:

- i. **Age:** Age is an continuous attribute thus replacing its missing value with mean value of the column is the most suitable approach.

➤ **Code:**

```
floor_or_ceil <- function(mean) {  
  if (mean %% 1 == 0) {  
    return(mean)  
  } else if (mean %% 1 < 0.5) {  
    return(floor(mean))  
  } else {  
    return(ceiling(mean))  
  }  
}  
mean_age <- mean(df$Age, na.rm = TRUE)  
df$Age[is.na(df$Age)] <- floor_or_ceil(mean_age)  
print(paste("Total missing values:",sum(is.na(df[1]))))
```

➤ **Output:**

```
> df[df==""]<- NA  
> num_of_missing_value_per_column <- sapply(df, function(x) sum(is.na(x)))  
> print(num_of_missing_value_per_column)  
      Age      Gender      BloodPressure  
      3         3         3  
Cholesterol      Heart_Rate QuantumPatternFeature  
      0         3         0  
HeartDisease  
      0  
> print(paste("Total missing values:",sum(is.na(df))))  
[1] "Total missing values: 12"
```

The purpose of the floor_or_ceil function here is to handle a mean value by rounding it to either the nearest lower integer (floor) or nearest higher integer (ceiling) based on its fractional part. This ensures precise replacement for missing values. The mean of the Age column was calculated, excluding missing values, and then missing values in the Age column were replaced using the rounded mean. It was confirmed that all missing values were handled as the output showed zero missing values remaining.

- ii. **Blood Pressure:** Blood pressure is a character type value so we need to convert it to numeric type and handle the missing values after that.

➤ **Code:**

```
library(dplyr)  
df <- df %>% mutate(BloodPressure = as.numeric(BloodPressure))  
mean_blood_pressure <- mean(df$BloodPressure, na.rm = TRUE)  
df$BloodPressure[is.na(df$BloodPressure)] <- floor_or_ceil(mean_blood_pressure)  
print(paste("Total missing values:",sum(is.na(df[3]))))
```

➤ **Output:**


```
> mean_blood_pressure <- mean(df$BloodPressure, na.rm = TRUE)
> df$BloodPressure[is.na(df$BloodPressure)] <- floor_or_ceil(mean_blood_pressure)
> print(paste("Total missing values:", sum(is.na(df[3]))))
[1] "Total missing values: 0"
```

Using the dplyr library, the BloodPressure column was converted from a string to a numeric data type. A warning occurred because some values could not be converted and were replaced with NA. The mean of the BloodPressure column was then calculated, excluding missing values, and used to replace the missing values. Finally, it was confirmed that there were no remaining missing values in the column, as the output showed zero missing values.

4. Replacing Missing Values Using Mode:

- i. **Gender:** Gender is a categorical value column thus replacing the missing values with the most frequent value is the most suitable option.

➤ Code:

```
most_frequent_gender <- as.numeric(names(sort(table(df$Gender), decreasing = TRUE)[1]))
df$Gender[is.na(df$Gender)] <- most_frequent_gender
print(paste("Total missing values:", sum(is.na(df[2]))))
```

➤ Output:

```
> most_frequent_gender <- as.numeric(names(sort(table(df$Gender), decreasing = TRUE)[1]))
> df$Gender[is.na(df$Gender)] <- most_frequent_gender
> print(paste("Total missing values:", sum(is.na(df[2]))))
[1] "Total missing values: 0"
```

The most frequent value in the Gender column was determined, and missing values in this column were replaced with that value. It was then confirmed that there were no remaining missing values, as the output indicated zero missing values in the Gender column.

- ii. **Heart Rate:** Heart rate is also a categorical value and replacing missing value with most frequent value is the best possible option here.

➤ Code:

```
most_frequent_heart_rate <- names(sort(table(df$Heart_Rate), decreasing = TRUE)[1])
df$Heart_Rate[is.na(df$Heart_Rate)] <- most_frequent_heart_rate
print(paste("Total missing values:", sum(is.na(df[5]))))
```

➤ Output:

```
> most_frequent_heart_rate <- names(sort(table(df$Heart_Rate), decreasing = TRUE)[1])
> df$Heart_Rate[is.na(df$Heart_Rate)] <- most_frequent_heart_rate
> print(paste("Total missing values:", sum(is.na(df[5]))))
[1] "Total missing values: 0"
```

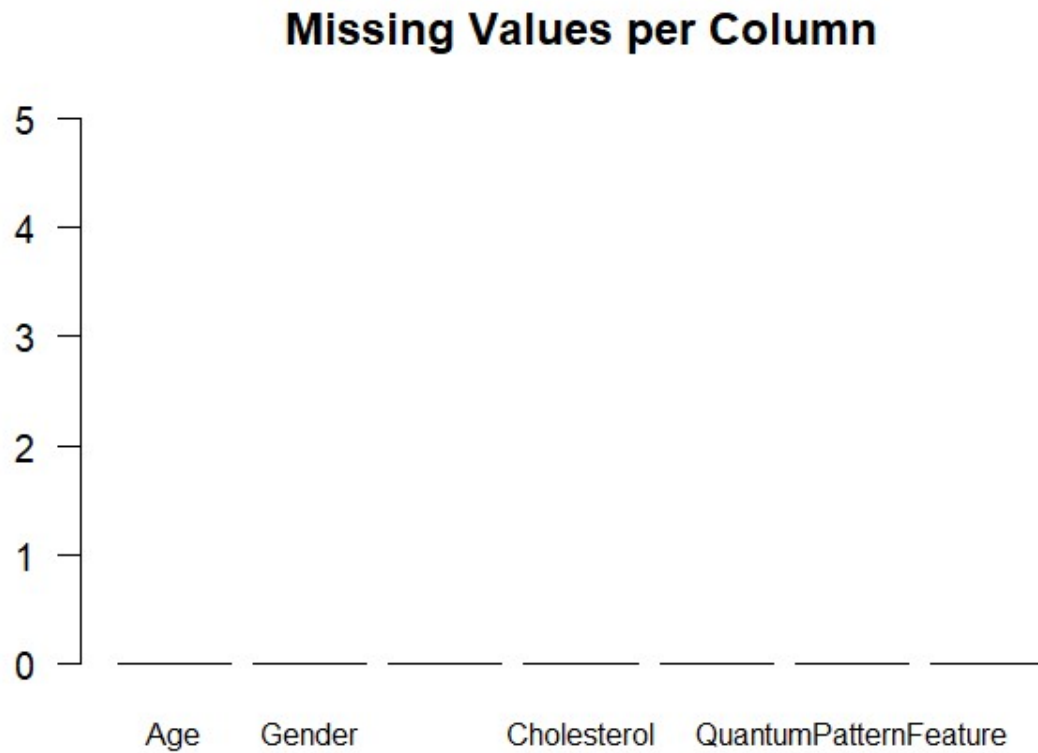
Here the table function was used to calculate the frequency of each value in the Heart_Rate column. The sort function, with decreasing = TRUE, ranked these frequencies in descending order to identify the most frequent value. The names function extracted this most common value, and missing values in the Heart_Rate column were replaced with it. Finally, the sum and is.na function confirmed there were no remaining missing values in the column, as the output indicated zero missing values.

5. Graph After Removing Missing Values:

➤ Code:

```
num_of_missing_value_per_column <- sapply(df, function(x) sum(is.na(x)))
missingVal<-colSums(is.na(df))
barplot(num_of_missing_value_per_column,
        main = "Missing Values per Column",
        col = "cyan",
        las = 1,
        cex.names = 0.8,
        ylim = c(0, 5))
```

➤ Output:



The bar plot shows us that after performing mode and mean in the missing value cells there are no more missing values in the dataset.

Converting Categorical Values into Numerical Values: The blood pressure column was already converted into numerical values during the process of handling missing values. Now we need to encode the heart rate column.

➤ Code:

```
df$Heart_Rate <- ifelse(df$Heart_Rate == "Low",0,ifelse(df$Heart_Rate == "High", 1, NA))
str(df)
```

➤ Output:

```

> df$Heart_Rate <- ifelse(df$Heart_Rate == "Low",0,ifelse(df$Heart_Rate == "High", 1, NA))
> str(df)
'data.frame': 151 obs. of 7 variables:
 $ Age      : num  68 58 44 72 37 50 68 55 52 40 ...
 $ Gender   : num  1 0 0 1 0 1 0 0 0 1 ...
 $ BloodPressure : num  105 97 93 93 145 114 156 156 133 121 ...
 $ Cholesterol : int  191 249 190 183 166 271 225 236 266 255 ...
 $ Heart_Rate : num  1 0 0 1 1 0 0 0 1 0 ...
 $ QuantumPatternFeature: num  8.36 9.25 7.94 6.5 7.65 ...
 $ HeartDisease : int  1 0 1 1 1 0 1 0 0 0 ...

```

Here the values in the Heart_Rate column were converted from categorical to numeric using the ifelse function. This function assigned 0 for "Low," 1 for "High," and NA for any other values. The output of str function confirmed that the conversion worked successfully, showing the Heart_Rate column as num (numeric type) along with the updated dataset structure.

Removing Outliers: For removing outliers we need to find the indices where outliers are in each column and then removing after removing all the duplicate rows.

1. Finding Outliers Indices:

➤ Code:

```

find_outliers <- function(column) {
  if (!is.numeric(column)) {
    stop("The input column must be numeric.")
  }
  Q1 <- quantile(column, 0.25, na.rm = TRUE)
  Q3 <- quantile(column, 0.75, na.rm = TRUE)
  IQR_value <- Q3 - Q1
  lower_bound <- Q1 - 1.5 * IQR_value
  upper_bound <- Q3 + 1.5 * IQR_value
  outlier_indices <- which(column < lower_bound | column > upper_bound)
  return(outlier_indices)
}
outlier_indices_age <- find_outliers(df$Age)
outlier_indices_blood_pressure <- find_outliers(df$BloodPressure)
outlier_indices_cholesterol <- find_outliers(df$Cholesterol)
outlier_indices_quantum_pattern_feature <- find_outliers(df$QuantumPatternFeature)
print(outlier_indices_age)
print(outlier_indices_blood_pressure)
print(outlier_indices_cholesterol)
print(outlier_indices_quantum_pattern_feature)

```

➤ Output:

```

> print(outlier_indices_age)
[1] 76 78 114
> print(outlier_indices_blood_pressure)
integer(0)
> print(outlier_indices_cholesterol)
integer(0)
> print(outlier_indices_quantum_pattern_feature)
[1] 139

```

Outliers were detected in numeric columns such as Age, BloodPressure, Cholesterol, and QuantumPatternFeature and Interquartile Range (IQR) method was used. This method identifies values below $Q1 - 1.5 * IQR$ or above $Q3 + 1.5 * IQR$. Outliers were found in Age at indices [76, 78, 114], and in QuantumPatternFeature at index [139]. No outliers were detected in BloodPressure and Cholesterol, as

indicated by the output 0. Columns with binary values like Gender, Heart_Rate, and HeartDisease were excluded from this process, as IQR-based detection is not applicable to categorical or binary data types.

2. Removing Outliers: After detecting the outliers outliers were removed from the data frame.

➤ **Code:**

```
all_outlier_indices <- unique(c(outlier_indices_age,
                                outlier_indices_blood_pressure,
                                outlier_indices_cholesterol,
                                outlier_indices_quantum_pattern_feature))
df <- df[-all_outlier_indices, ]
```

➤ **Output:**

```
> all_outlier_indices <- unique(c(outlier_indices_age,
+                                outlier_indices_blood_pressure,
+                                outlier_indices_cholesterol,
+                                outlier_indices_quantum_pattern_feature))
> df <- df[-all_outlier_indices, ]
```

A combined list of all outlier indices from the Age, BloodPressure, Cholesterol, and QuantumPatternFeature columns was created using the c function. The unique function was applied to ensure that duplicate indices were removed, preventing rows from being unnecessarily removed multiple times. The dataset was then updated to exclude all rows corresponding to the outlier indices. This effectively removed the identified outliers from the dataset.

Data Filtering:

➤ **Code:**

```
filtered_data <- df[df$Age >= 20 & df$Age <= 80 & df$BloodPressure >= 80 & df$BloodPressure <= 120, ]
View(head(filtered_data))
```

➤ **Output:**

	Age	Gender	BloodPressure	Cholesterol	Heart_Rate	QuantumPatternFeature	HeartDisease
1	68	1	105	191	1	8.362241	1
2	58	0	97	249	0	9.249002	0
3	44	0	93	190	0	7.942542	1
4	72	1	93	183	1	6.495155	1
6	50	1	114	271	0	8.631604	0
14	69	1	108	180	0	6.799794	1

A filter was applied to the dataset df to include only rows where Age is between 20 and 80, and BloodPressure is between 80 and 120. The View command displays the first few rows of the filtered dataset in RStudio's data viewer, allowing inspection of the filtered results. This step ensures the data meets the defined criteria for analysis.

Normalizing The Dataset:

➤ **Code:**

```

normalize_dataset <- df
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
normalize_dataset$Age <- normalize(df$Age)
normalize_dataset$BloodPressure <- normalize(df$BloodPressure)
normalize_dataset$Cholesterol <- normalize(df$Cholesterol)
normalize_dataset$QuantumPatternFeature <- normalize(df$QuantumPatternFeature)
View(normalize_dataset)

```

➤ Output:

	Age	Gender	BloodPressure	Cholesterol	Heart_Rate	QuantumPatternFeature	HeartDisease
1	0.77551020	1	0.16853933	0.265306122	1	0.474452510	1
2	0.57142857	0	0.07865169	0.659863946	0	0.691012485	0
3	0.28571429	0	0.03370787	0.258503401	0	0.371955962	1
4	0.85714286	1	0.03370787	0.210884354	1	0.018482887	1
5	0.14285714	0	0.61797753	0.095238095	1	0.301465377	1
6	0.40816327	1	0.26966292	0.809523810	0	0.540234977	0
7	0.77551020	0	0.74157303	0.496598639	0	0.278422394	1
8	0.51020408	0	0.74157303	0.571428571	0	0.667348358	0
9	0.44897959	0	0.48314607	0.775510204	1	0.666085609	0
10	0.20408163	1	0.34831461	0.700680272	0	0.797196678	0
11	0.20408163	1	0.55056180	0.564625850	0	0.564829610	0
12	0.46938776	1	0.67415730	0.163265306	0	0.626499056	0
13	0.71428571	0	0.56179775	0.367346939	1	0.275001665	1
14	0.79591837	1	0.20224719	0.190476190	0	0.092880079	1
15	0.46938776	1	0.22471910	0.891156463	1	0.676636421	0
16	0.04081633	1	0.04494382	0.646258503	1	0.708982893	0

The dataset df was copied into normalize_dataset to preserve the original data. A normalization function was applied to the Age, BloodPressure, Cholesterol, and QuantumPatternFeature columns, scaling their values to a range between 0 and 1 by calculating $(x - \min(x)) / (\max(x) - \min(x))$. The View command displayed the normalized dataset, where it could be observed that the column values had been successfully transformed to fall within the range of 0 and 1.

Balancing The Dataset:

1. Finding The Class Distribution Using Graph:

➤ Code:

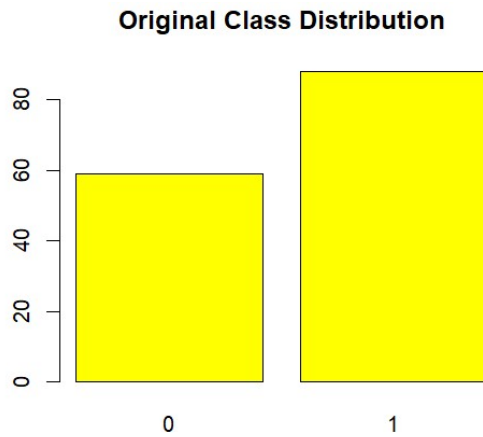
```

idata <- normalize_dataset
class_dist <- table(idata$HeartDisease)
class_dist

barplot(table(idata$HeartDisease), main = "Original Class Distribution", col = "yellow")

```

➤ Output:



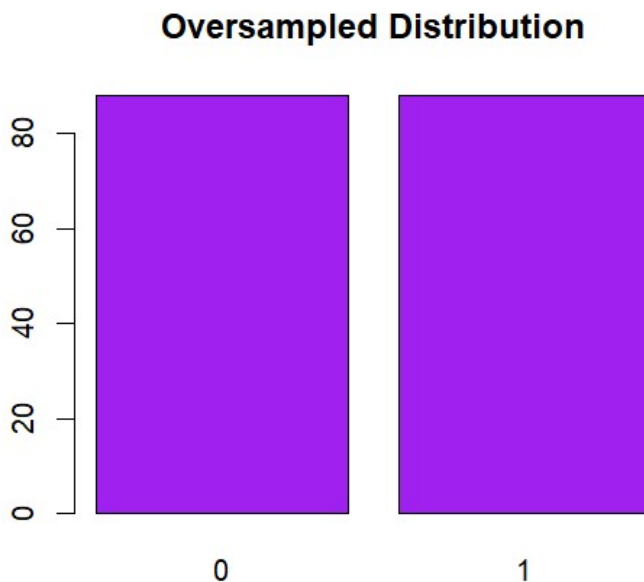
Here we start by assigning the normalized dataset to the variable `idata`. The `table` function is then used to calculate the frequency distribution of the `HeartDisease` variable, and the result is stored in `class_dist`. A bar plot is generated using `barplot` to visualize the frequency distribution, with the title "Original Class Distribution" and yellow-colored bars. The results indicate that the dataset is not balanced, as the frequency of one class is significantly higher than the other, which could lead to biases in predictive modeling.

2. Balancing Using Over Sampling:

➤ Code:

```
if (class_dist[1] > class_dist[2]) {  
  majority <- filter(idata, HeartDisease == 0)  
  minority <- filter(idata, HeartDisease == 1)  
} else {  
  majority <- filter(idata, HeartDisease == 1)  
  minority <- filter(idata, HeartDisease == 0)  
}  
set.seed(100)  
os_minority <- minority %>% sample_n(nrow(majority), replace = TRUE)  
os_balanced_data <- bind_rows(majority, os_minority)  
table(os_balanced_data$HeartDisease)  
barplot(table(os_balanced_data$HeartDisease), main = "Oversampled Distribution", col = "purple")
```

➤ Output:



We first identify the majority and minority classes in the target variable `HeartDisease` using the `table`

function. Based on the distribution, it separates the data into majority and minority groups. The `set.seed(100)` ensures reproducibility, and the `sample_n` function is used to oversample the minority class to match the size of the majority class, with replacement. The two groups are then combined into a new dataset, `os_balanced_data`, using the `bind_rows` function. A bar plot is generated to visualize the distribution, showing that the classes in the `HeartDisease` target variable are now balanced. This step addresses the initial class imbalance effectively.

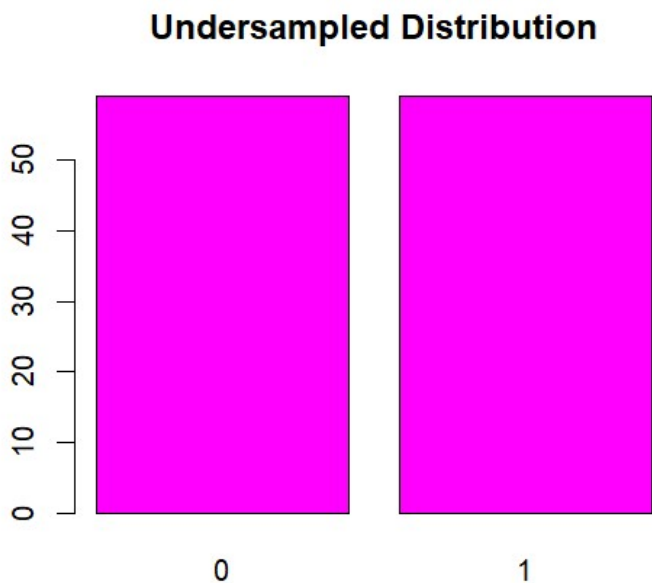
3. Balancing Using Under Sampling:

➤ Code:

```
us_majority <- majority %>% sample_n(nrow(minority))
us_balanced_data <- bind_rows(minority, us_majority)
table(us_balanced_data$HeartDisease)

barplot(table(us_balanced_data$HeartDisease), main = "Undersampled Distribution", col = "magenta")
```

➤ Output:



The code begins by visualizing the distribution of the oversampled data using a bar plot with purple-colored bars. Next, it performs undersampling by taking a random sample from the majority class using `sample_n`, ensuring its size matches the minority class. The two balanced groups are then combined into a new dataset, `us_balanced_data`, using `bind_rows`. A new bar plot is created with magenta-colored bars to visualize the distribution after undersampling. The output of `table` confirms that both classes in the target variable `HeartDisease` are now balanced, with 59 observations each.

Splitting The Dataset:

➤ Code:


```
n <- nrow(final_data)
random_index <- sample(1:n, size = 0.8 * n)
train_data <- final_data[random_index, ]
test_data <- final_data[-random_index, ]
View(train_data)
View(test_data)
```

➤ Output:

Gender	BloodPressure	Cholesterol	Heart_Rate	QuantumPatternFeature
1	0.04494382	0.646258503	1	0.70898289
1	0.93258427	0.149659864	0	0.51907184
0	0.00000000	0.414965986	0	0.17174796
1	0.50561798	0.326530612	1	0.78746094
0	0.28089888	0.074829932	0	0.12350461
0	0.82022472	0.353741497	0	0.39425764
1	0.21348315	0.945578231	0	0.83729072
1	0.61797753	0.673469388	0	0.27019970
0	1.00000000	0.707482993	0	0.71371244
0	0.65168539	0.714285714	0	0.68515836
0	0.87640449	0.530612245	0	0.15457352
0	0.05617978	0.612244898	1	0.53725731
0	0.43820225	0.081632653	1	0.47516352
0	0.12359551	0.346938776	0	0.52455456
0	0.02247191	0.061224490	1	0.03148548
1	0.49438202	0.993197279	0	0.51161269

Gender	BloodPressure	Cholesterol	Heart_Rate	QuantumPatternFeature
0	0.56179775	0.36734694	1	0.275001665
1	0.22471910	0.80952381	0	0.579464126
0	0.29213483	0.05442177	0	0.355745727
1	0.69662921	0.64625850	0	0.367897581
1	0.52808989	0.35374150	1	0.429020976
1	0.28089888	0.44897959	0	0.179298872
0	0.39325843	0.36054422	0	0.437285470
0	1.00000000	0.85714286	0	0.606815711
1	0.30337079	0.39455782	0	0.355226026
1	0.97752809	0.14965986	0	0.271871384
1	0.30337079	0.36734694	0	0.568529957
1	0.75280899	0.06122449	1	0.349860786
1	0.44943820	0.15646259	0	0.009148975
1	0.03370787	0.09523810	1	0.118941475
0	0.83146067	0.07482993	1	0.289160997
0	0.08988764	0.23129252	0	0.107940862

The dataset `final_data` was split into training and testing sets. The rows specified by `random_index` were assigned to `train_data`, while the remaining rows were assigned to `test_data`. The `View` function was used to display the contents of both datasets in RStudio's data viewer for inspection. This step ensures that the data is divided properly for model training and evaluation.

Comparing Central Tendencies Across Age:

➤ Code:

```
age_summary <- final_data %>%
  group_by(Gender) %>%
  summarise(
    Mean_Age = mean(Age, na.rm = TRUE),
    Median_Age = median(Age, na.rm = TRUE),
    Mode_Age = as.numeric(names(which.max(table(final_data$Age)))),
    .groups = "drop"
  )
print(age_summary)
```

➤ Output:

```
> print(age_summary)
# A tibble: 2 × 4
  Gender Mean_Age Median_Age Mode_Age
  <dbl>   <dbl>   <dbl>   <dbl>
1     0     0.487     0.480     0.469
2     1     0.484     0.469     0.469
```

Here we calculate the central tendencies of the `Age` variable grouped by `Gender`. Using the `group_by`

function, the dataset is split into groups based on the Gender column. The summarise function computes the mean (Mean_Age), median (Median_Age), and mode (Mode_Age) for each group, while na.rm = TRUE ensures missing values are excluded from calculations. The .groups = "drop" argument removes grouping after summarization.

The output is a tibble showing that for Gender group 0, the mean, median, and mode of Age are approximately 0.487, 0.480, and 0.469, respectively, while for Gender group 1, these values are approximately 0.484, 0.469, and 0.469, respectively. This indicates slight differences in the central tendencies of Age between the two gender groups.

Comparing Central Tendencies Across Heart Rate:

➤ Code:

```
age_summary <- final_data %>%
  group_by(Heart_Rate) %>%
  summarise(
    Mean_Age = mean(Age, na.rm = TRUE),
    Median_Age = median(Age, na.rm = TRUE),
    Mode_Age = as.numeric(names(which.max(table(final_data$Age)))),
    .groups = "drop"
  )
print(age_summary)
```

➤ Output:

```
> print(age_summary)
# A tibble: 2 × 4
  Gender Mean_Age Median_Age Mode_Age
  <dbl>   <dbl>     <dbl>   <dbl>
1     0     0.487     0.480     0.469
2     1     0.484     0.469     0.469
```

Here we group the final_data dataset by Heart_Rate and calculates the central tendencies of Age, including mean, median, and mode for each group. The group_by function organizes the dataset into groups based on the Heart_Rate column. The summarise function calculates the mean, median, and mode for the Age column within each group, while na.rm = TRUE ensures missing values are excluded from calculations. The output is a tibble showing that for Heart_Rate group 0, the mean, median, and mode of Age are approximately 0.488, 0.480, and 0.469, respectively, while for Heart_Rate group 1, these values are approximately 0.481, 0.469, and 0.469, respectively. This reflects slight differences in central tendencies of Age between the two Heart_Rate groups. The use of .groups = "drop" removes grouping after summarization.

Comparing Spread Across Age:

➤ Code:

```
age_spread_summary <- final_data %>%
  group_by(Gender) %>%
  summarise(
    Range_Age = max(Age, na.rm = TRUE) - min(Age, na.rm = TRUE),
    IQR_Age = IQR(Age, na.rm = TRUE),
    Variance_Age = var(Age, na.rm = TRUE),
    SD_Age = sd(Age, na.rm = TRUE),
    .groups = "drop"
  )

print(age_spread_summary)
```

➤ Output:

```
> print(age_spread_summary)
# A tibble: 2 × 5
  Gender Range_Age IQR_Age Variance_Age SD_Age
  <dbl>   <dbl>   <dbl>      <dbl> <dbl>
1     0         1     0.413     0.0752 0.274
2     1         1     0.434     0.0804 0.284
```

Here we calculate the spread measures for the Age variable grouped by Gender. Using the `group_by` function, the dataset is separated into groups based on the Gender column. Then, the `summarise()` function computes the following metrics for each group: `Range_Age` (difference between max and min Age), `IQR_Age` (interquartile range), `Variance_Age` (variance of Age), and `SD_Age` (standard deviation of Age). The `.groups = "drop"` argument ensures grouping is removed after summarization. The output is a tibble showing that for Gender group 0, the range is 1, the IQR is 0.413, the variance is 0.0752, and the standard deviation is 0.274. For Gender group 1, the range is also 1, but the IQR is 0.434, the variance is 0.0804, and the standard deviation is 0.284. These results highlight slight differences in the spread of Age between genders.

Conclusion: The final dataset was meticulously prepared by handling missing values, detecting outliers, balancing the target variable, and applying normalization. These data preparation steps ensure a clean and structured dataset, suitable for reliable analysis and predictive modeling.