**Description about the dataset**:

The "Maternal Health Risk" dataset encompasses various attributes related to maternal health risk assessment.

**The dataset contains the following attributes**:

1. **Age**: This attribute represents the age of the individuals under assessment. Age is likely to be a crucial factor in determining maternal health risks.

2. **Infection**: This attribute indicates the presence of infection, categorized as 1 for yes, 2 for no, and 3 for marginal. Infections during pregnancy can pose significant risks to both the mother and the unborn child.

3. **Smoking**: This attribute denotes the smoking status of the individuals, categorized as 1 for yes, 2 for sometimes, and 3 for no. Smoking during pregnancy can increase the risk of various complications.

4. **SystolicBcp**: This attribute represents the systolic blood pressure of the individuals. High blood pressure during pregnancy can indicate conditions like preeclampsia, posing serious risks to maternal health.

5. **DiastolicBcp**: This attribute represents the diastolic blood pressure of the individuals. Like systolic blood pressure, diastolic blood pressure is an important indicator of maternal health, especially concerning conditions like preeclampsia.

6. **Bs**: This attribute likely represents blood sugar levels. Monitoring blood sugar levels during pregnancy is crucial, especially for individuals with gestational diabetes, as it affects both maternal and fetal health.

7. **Body Temperature**: This attribute represents the body temperature of the individuals. Abnormal body temperatures can indicate infections or other underlying health issues.

8. **Heart Rate**: This attribute represents the heart rate of the individuals. Changes in heart rate can signify various cardiovascular or metabolic issues, which can impact maternal health.

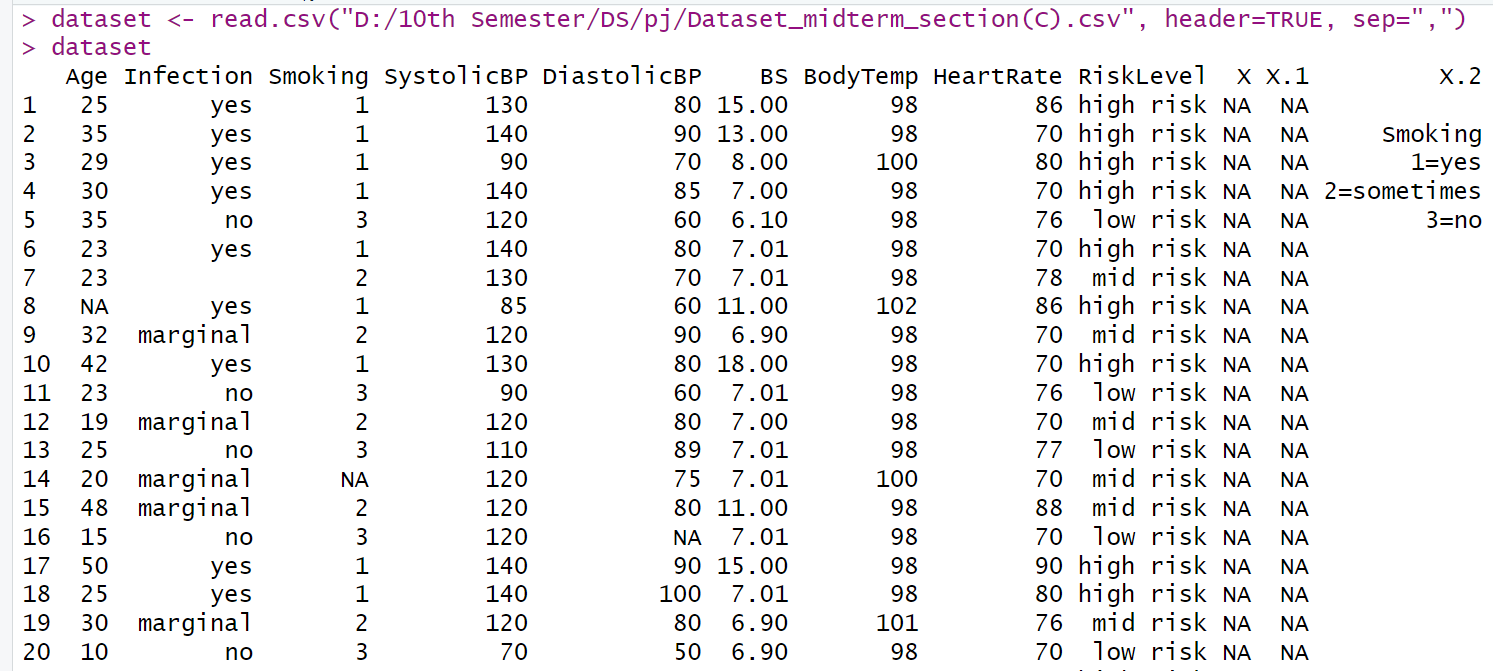
9**. Risk Level**: This attribute categorizes the overall risk level into three categories: high risk (1), low risk (2), and mid-risk (3). This categorization likely considers the collective impact of the other attributes on maternal health.

**Importing the dataset**

Code:

|  |
| --- |
| dataset <- read.csv("D:/10th Semester/DS/pj/Dataset\_midterm\_section(C).csv", header=TRUE, sep=",")  dataset |

Output:



Explanation:

The dataset was initially converted from XLSX to CSV format and subsequently imported into RStudio using the read.csv function. Two arguments were specified: the file path of the CSV file and na. strings = c(""), which was utilized to substitute empty strings with NAs during the import process.

**Column Names:**

Code:

|  |
| --- |
| names(dataset) |

Output:



Explanation:

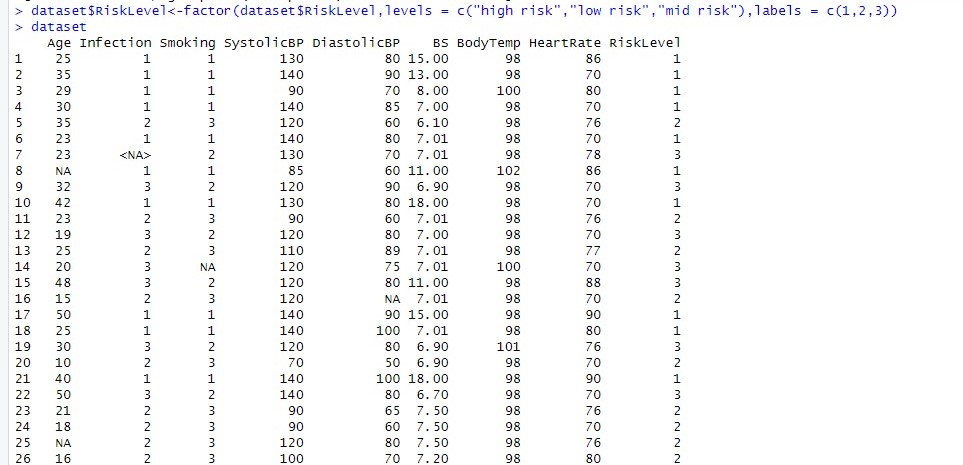
This function is used to retrieve the column names of a dataset. It returns a list or array containing the names of all the columns in the dataset, This allows access and manipulate specific columns by their names.

**Data Annotation:**

Code:

|  |
| --- |
| dataset$Infection<-factor(dataset$Infection,levels = c("yes","no","marginal"),labels = c(1,2,3))  dataset$RiskLevel<-factor(dataset$RiskLevel,levels = c("high risk","low risk","mid risk"),labels = c(1,2,3))  dataset |

Output:



Explanation:

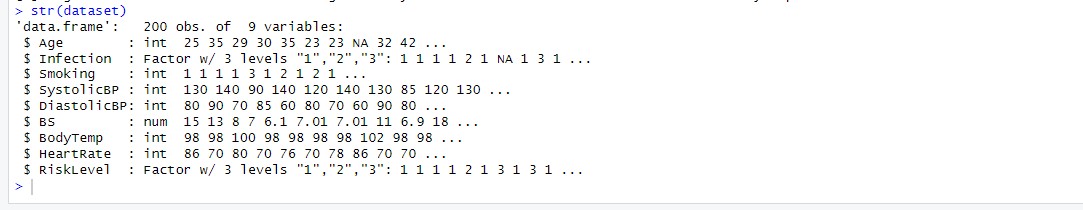
* dataset$Infection <- factor(dataset$Infection, levels = c("yes", "no", "marginal"), labels = c(1, 2, 3)): This line converts the column Infection in the dataset into a factor variable. The levels "yes", "no", and "marginal" are specified, and corresponding labels 1, 2, and 3 are assigned to them.
* dataset$RiskLevel <- factor(dataset$RiskLevel, levels = c("high risk", "low risk", "mid risk"), labels = c(1, 2, 3)): Similarly, this line converts the column RiskLevel in the dataset into a factor variable. The levels "high risk", "low risk", and "mid risk" are specified, and corresponding labels 1, 2, and 3 are assigned to them.

**Summary of the structure of data set:**

Code:

|  |
| --- |
| str(dataset) |

Output:



Explanation:

Here the str() function used to display the structure of a dataset. Specifically, it provides a compact display of the internal structure of an R object. When applied to a dataset, it outputs information about the data type of each column, the number of observations (rows), and potentially additional details about the dataset's structure. This function is useful for quickly understanding the composition of a dataset, such as the types of variables it contains and their dimensions.

**Descriptive Statistics Using summary() Function**:

Code:

|  |
| --- |
| summary(dataset) |

Output:

A black and white text

Description automatically generated

Explanation:

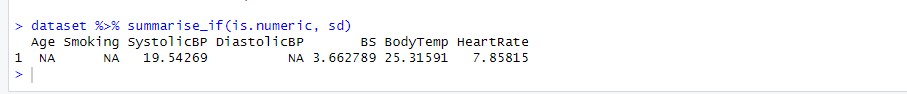
Here, the summary(dataset) is a function used to summarize the data contained in a dataset. When applied to a dataset, it provides statistical summaries for each variable in the dataset. These summaries typically include measures such as minimum, maximum, median, mean, and quartiles for numeric variables, and frequency counts for categorical variables. This function is helpful for getting a quick overview of the distribution and characteristics of the data in the dataset.

**Standard deviation:**

Code:

|  |
| --- |
| dataset %>% summarise\_if(is.numeric, sd) |

Output:



Explanation:

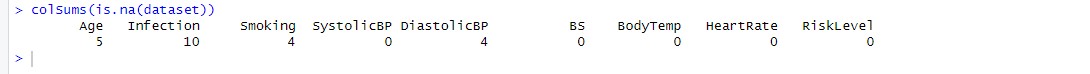
This code utilizes the summarise\_if() function from the dplyr package to calculate the standard deviation (sd) of numeric variables within the dataset. The %>% operator (also known as the pipe operator) is used to pass the dataset to the summarise\_if() function. The is.numeric condition specifies that the summary function (sd) should be applied only to numeric columns.

**Null values in each column:**

Code:

|  |
| --- |
| colSums(is.na(dataset)) |

Output:



Explanation:  
Here, the colSums(is.na(dataset)) command is used to count the number of missing values (NA) in each column of a dataset. This command returns a vector where each element represents the number of missing values in the corresponding column of the dataset. This is useful for quickly assessing the extent of missing data in each column.

**Specific row number of Null values:**

Code:

|  |
| --- |
| which(is.na(dataset$Age))  which(is.na(dataset$Infection))  which(is.na(dataset$Smoking))  which(is.na(dataset$SystolicBP))  which(is.na(dataset$DiastolicBP ))  which(is.na(dataset$BS))  which(is.na(dataset$BodyTemp))  which(is.na(dataset$HeartRate))  which(is.na(dataset$RiskLevel)) |

Output:

A screenshot of a computer

Description automatically generated

Explanation:

Here, the which() function is used to determine the indices of elements that satisfy a certain condition. It takes a logical expression as its argument and returns the indices of the elements for which the expression is true.

For example, which(is.na(dataset$Age)) returns the indices of the elements in the "Age" column of the dataset where the value is NA (missing).

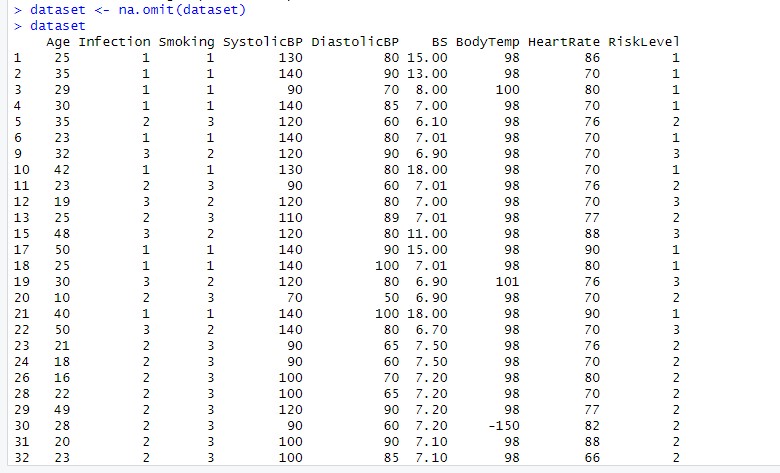
Here, each line of code returns a vector of indices where, missing values are present in the respective columns of the dataset.

**Remove null values from data set:**

Code:

|  |
| --- |
| dataset <- na.omit(dataset)  dataset |

Output:



Explanation:

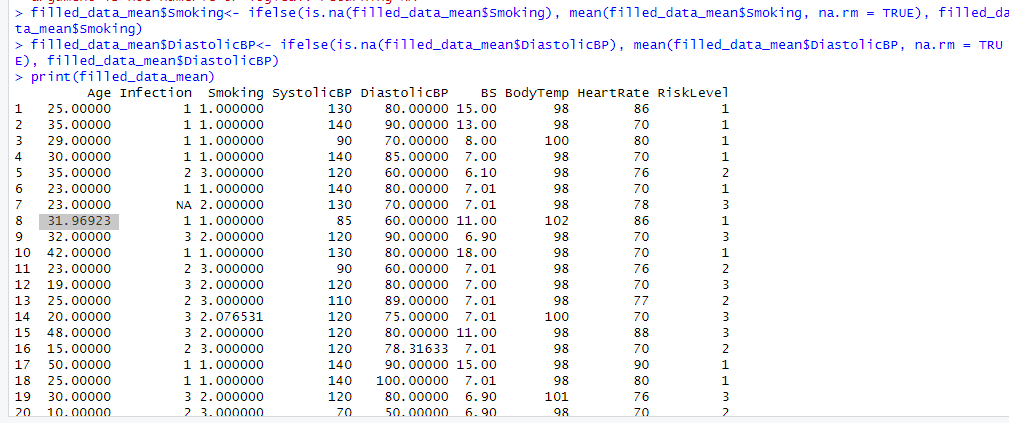
The line dataset <- na.omit(dataset) in R overwrites the existing dataset with a modified version where any rows containing missing values (NA) are removed. After executing this line, the dataset variable will contain all the rows from the original dataset that have complete information (no missing values).

**Replacing Na value with mean :**

Code

|  |
| --- |
| filled\_data\_mean <- dataset  filled\_data\_mean$Age <- ifelse(is.na(filled\_data\_mean$Age), mean(filled\_data\_mean$Age, na.rm = TRUE), filled\_data\_mean$Age)  filled\_data\_mean$Infection<- ifelse(is.na(filled\_data\_mean$Infection), mean(filled\_data\_mean$Infection, na.rm = TRUE), filled\_data\_mean$Infection)  filled\_data\_mean$Smoking<- ifelse(is.na(filled\_data\_mean$Smoking), mean(filled\_data\_mean$Smoking, na.rm = TRUE), filled\_data\_mean$Smoking)  filled\_data\_mean$DiastolicBP<- ifelse(is.na(filled\_data\_mean$DiastolicBP), mean(filled\_data\_mean$DiastolicBP, na.rm = TRUE), filled\_data\_mean$DiastolicBP)  filled\_data\_mean$BS<- ifelse(is.na(filled\_data\_mean$BS), mean(filled\_data\_mean$BS, na.rm = TRUE), filled\_data\_mean$BS)  print(filled\_data\_mean) |

Output:



Explanation:

The code above is used to fill missing values in the dataset with the mean of each respective column. Here's a brief explanation of each line:

* filled\_data\_mean <- dataset:  
  This line creates a new dataset called filled\_data\_mean, which is a copy of the original dataset.
* filled\_data\_mean$Age <- ifelse(is.na(filled\_data\_mean$Age) mean(filled\_data\_mean$Age, na.rm = TRUE)  
  filled\_data\_mean$Age) :  
  These lines checks if there are missing values (NA) in the "Age" column of the filled\_data\_mean dataset. If there are missing values, it replaces them with the mean of the "Age" column (calculated using mean() function) while ignoring NA values (na.rm = TRUE). If there are no missing values, it leaves the original values unchanged.
* Similarly, the next lines perform the same operation for other columns in the dataset:  
  filled\_data\_mean$Infection  
  filled\_data\_mean$Smoking  
  filled\_data\_mean$DiastolicBP  
  filled\_data\_mean$BS
* print(filled\_data\_mean):  
  Finally, this line prints the modified dataset filled\_data\_mean with missing values replaced by their respective column means.

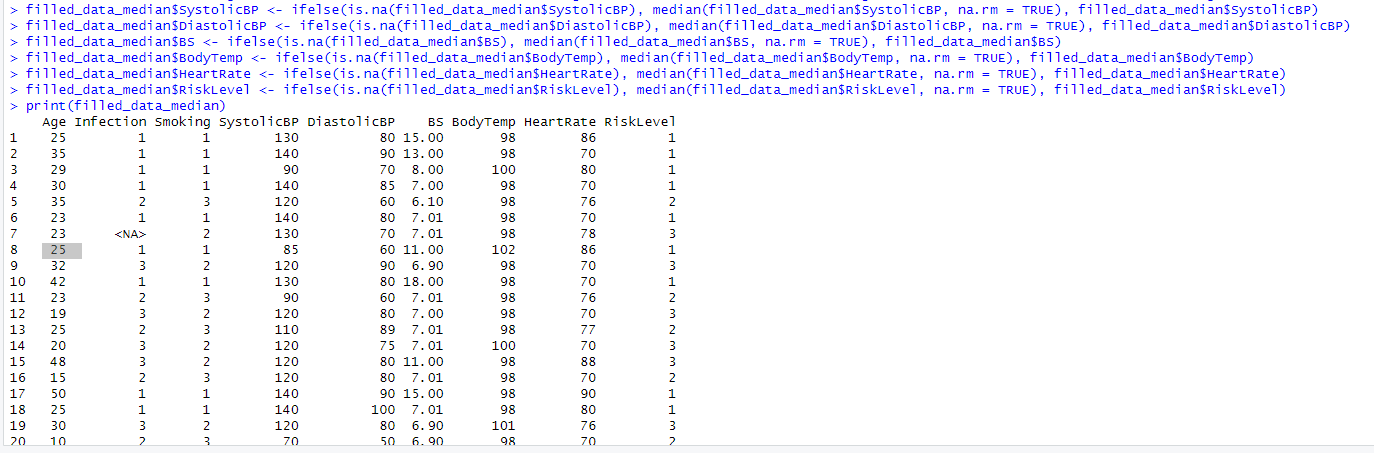
Overall, this code fills missing values in the dataset with the mean of each respective column.

**Replacing Na value with median:**

Code:

|  |
| --- |
| filled\_data\_median <- dataset  filled\_data\_median$Age <- ifelse(is.na(filled\_data\_median$Age), median(filled\_data\_median$Age, na.rm = TRUE), filled\_data\_median$Age)  filled\_data\_median$Infection <- ifelse(is.na(filled\_data\_median$Infection), median(filled\_data\_median$Infection, na.rm = TRUE), filled\_data\_median$Infection)  filled\_data\_median$Smoking <- ifelse(is.na(filled\_data\_median$Smoking), median(filled\_data\_median$Smoking, na.rm = TRUE), filled\_data\_median$Smoking)  filled\_data\_median$DiastolicBP <- ifelse(is.na(filled\_data\_median$DiastolicBP), median(filled\_data\_median$DiastolicBP, na.rm = TRUE), filled\_data\_median$DiastolicBP)  print(filled\_data\_median) |

Output:



Explanation:

The code above fills missing values in the dataset with the median of each respective column. Here's a brief explanation of each line:

* filled\_data\_median <- dataset:  
  This line creates a new dataset named filled\_data\_median, which is a copy of the original dataset.
* filled\_data\_median$Age <- ifelse(is.na(filled\_data\_median$Age) median(filled\_data\_median$Age, na.rm = TRUE)  
  filled\_data\_median$Age):   
  This line checks if there are missing values (NA) in the "Age" column of the filled\_data\_median dataset. If there are missing values, it replaces them with the median of the "Age" column (calculated using median() function) while ignoring NA values (na.rm = TRUE). If there are no missing values, it leaves the original values unchanged.
* Similarly, the next lines perform the same operation for other columns in the dataset:  
  filled\_data\_median$Infection  
  filled\_data\_median$Smoking  
  filled\_data\_median$DiastolicBP
* print(filled\_data\_median):   
  Finally, this line prints the modified dataset filled\_data\_median with missing values replaced by their respective column medians.

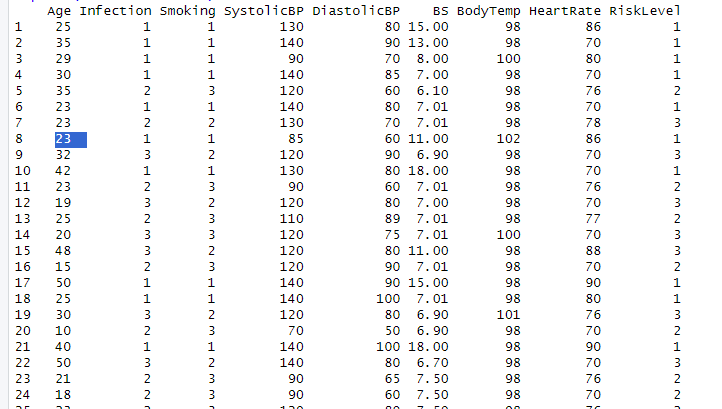
Overall, this code replaces missing values in the dataset with the median of each respective column, which is another common approach for handling missing data, particularly when the data is skewed or contains outliers.

**Replacing Na value with mode :**

Code:

|  |
| --- |
| filled\_data\_mode <- dataset  filled\_data\_mode$Age <- ifelse(is.na(filled\_data\_mode$Age),  names(sort(table(filled\_data\_mode$Age), decreasing = TRUE))[1],  filled\_data\_mode$Age)  filled\_data\_mode$Infection <- ifelse(is.na(filled\_data\_mode$Infection ),  names(sort(table(filled\_data\_mode$Infection ), decreasing = TRUE))[1],  filled\_data\_mode$Infection )  filled\_data\_mode$Smoking <- ifelse(is.na(filled\_data\_mode$Smoking ),  names(sort(table(filled\_data\_mode$Smoking ), decreasing = TRUE))[1],  filled\_data\_mode$Smoking )  filled\_data\_mode$SystolicBP <- ifelse(is.na(filled\_data\_mode$SystolicBP),  names(sort(table(filled\_data\_mode$SystolicBP), decreasing = TRUE))[1],  filled\_data\_mode$SystolicBP)  filled\_data\_mode$DiastolicBP <- ifelse(is.na(filled\_data\_mode$DiastolicBP),  names(sort(table(filled\_data\_mode$DiastolicBP), decreasing = TRUE))[1],  filled\_data\_mode$DiastolicBP)  filled\_data\_mode$BS <- ifelse(is.na(filled\_data\_mode$BS),  names(sort(table(filled\_data\_mode$BS), decreasing = TRUE))[1],  filled\_data\_mode$BS)  filled\_data\_mode$BodyTemp <- ifelse(is.na(filled\_data\_mode$BodyTemp),  names(sort(table(filled\_data\_mode$BodyTemp), decreasing = TRUE))[1],  filled\_data\_mode$BodyTemp)  filled\_data\_mode$HeartRate <- ifelse(is.na(filled\_data\_mode$HeartRate),  names(sort(table(filled\_data\_mode$HeartRate), decreasing = TRUE))[1],  filled\_data\_mode$HeartRate)  filled\_data\_mode$RiskLevel <- ifelse(is.na(filled\_data\_mode$RiskLevel),  names(sort(table(filled\_data\_mode$RiskLevel), decreasing = TRUE))[1],  filled\_data\_mode$RiskLevel)  print(filled\_data\_mode) |

Output:



Explanation:

The code above is used to replace missing values in each column of the dataset with the mode (most frequently occurring value) of that respective column. Let's break down what each part of the code does:

* filled\_data\_mode <- dataset : This line creates a new dataset named filled\_data\_mode as a copy of the original dataset.

* For each column in the dataset (Age, Infection, Smoking, SystolicBP, DiastolicBP, BS, BodyTemp, HeartRate, and RiskLevel), the following steps are performed:
* ifelse(is.na(filled\_data\_mode$Column)): Checks if there are missing values in the column.
* names(sort(table(filled\_data\_mode$Column), decreasing = TRUE))[1]: Calculates the mode of the column by first creating a frequency table using table(), sorting it in descending order using sort(), and then extracting the name of the most frequent value (mode) using names().

* If there are missing values in the column, the mode is used to replace those missing values. Otherwise, the original values are retained.

* The print(filled\_data\_mode) statement prints the modified dataset filled\_data\_mode with missing values replaced by their respective column modes.

Overall, this code block is a method for imputing missing values in a dataset by replacing them with the mode of each respective column, which is the value that appears most frequently in that column. This approach is often used when dealing with categorical or discrete variables.

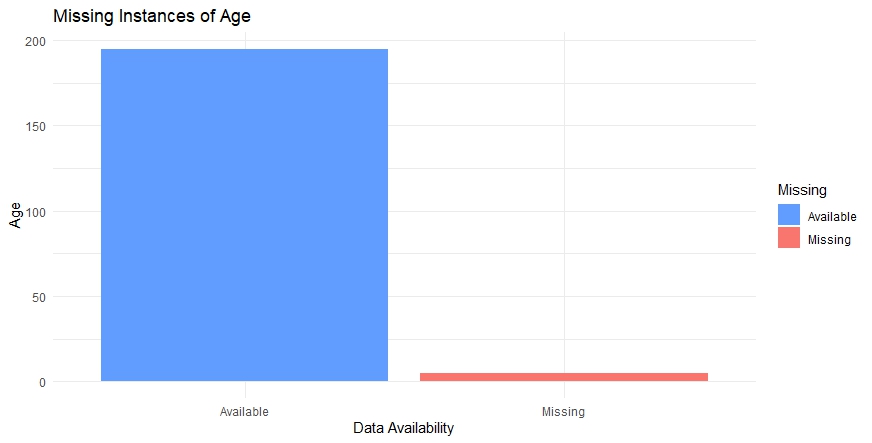
**Missing values visualization:**

**Age:**

Code:

|  |
| --- |
| missing\_data <- data.frame(  Missing = c("Available", "Missing"),  Age = c(sum(!is.na(dataset$Age)), sum(is.na(dataset$Age)))  )  ggplot(missing\_data, aes(x = Missing, y = Age, fill = Missing)) +  geom\_bar(stat = "identity") +  scale\_fill\_manual(values = c("#619CFF", "#F8766D")) +  labs(  title = "Missing Instances of Age",  x = "Data Availability",  y = "Age"  ) +  theme\_minimal() |

Output:



**Infection:**

Code:

|  |
| --- |
| missing\_data <- data.frame(  Missing = c("Available", "Missing"),  Infection = c(sum(!is.na(dataset$Infection)), sum(is.na(dataset$Infection)))  )  ggplot(missing\_data, aes(x = Missing, y = Infection, fill = Missing)) +  geom\_bar(stat = "identity") +  scale\_fill\_manual(values = c("red", "green")) +  labs(  title = "Missing Instances of Infection",  x = "Data Availability",  y = "Infection"  ) +  theme\_minimal() |

Output:

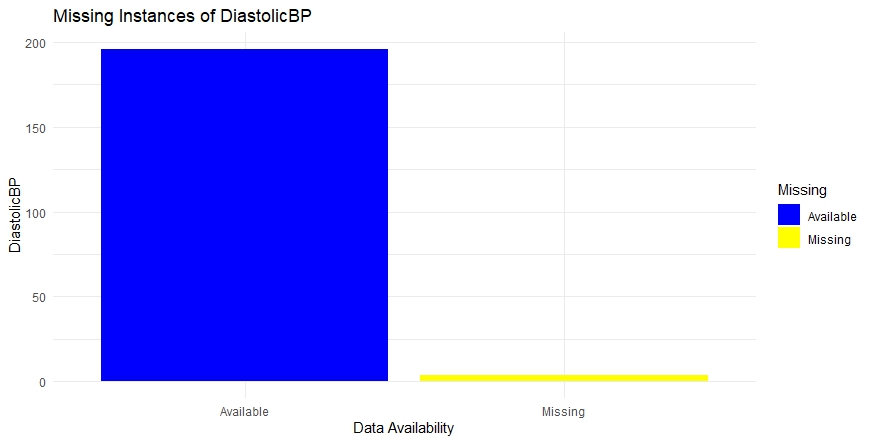


**DiastolicBP:**

Code:

|  |
| --- |
| missing\_data <- data.frame(  Missing = c("Available", "Missing"),  DiastolicBP = c(sum(!is.na(dataset$DiastolicBP)), sum(is.na(dataset$DiastolicBP)))  )  ggplot(missing\_data, aes(x = Missing, y = DiastolicBP, fill = Missing)) +  geom\_bar(stat = "identity") +  scale\_fill\_manual(values = c("blue", "yellow")) +  labs(  title = "Missing Instances of DiastolicBP",  x = "Data Availability",  y = "DiastolicBP"  ) +  theme\_minimal() |

Output:



Explanation:

The code above generates bar plots showing the distribution of missing and available data for different variables in the dataset. Here's a brief explanation:

* Three separate blocks of code are used to generate bar plots for three variables: "Age", "Infection", and "DiastolicBP". Each block of code follows a similar structure.

* Within each block of code:
  + The named missing\_data is created with two columns: "Missing" indicating the availability of data ("Available" or "Missing"), and the variable of interest (e.g., "Age", "Infection", or "DiastolicBP").
  + The counts of missing and available data for the variable of interest are calculated using sum(!is.na())` and `sum(is.na()), respectively.
  + A bar plot is generated using ggplot2, with "Missing" on the x-axis, the count of data on the y-axis, and the fill color indicating data availability.

In general, this code provides a visual representation of missing data for each variable in the dataset, making it easier to identify which variables have missing values and the extent of missingness.

**Analysis and Visualization:**

**Mean:**

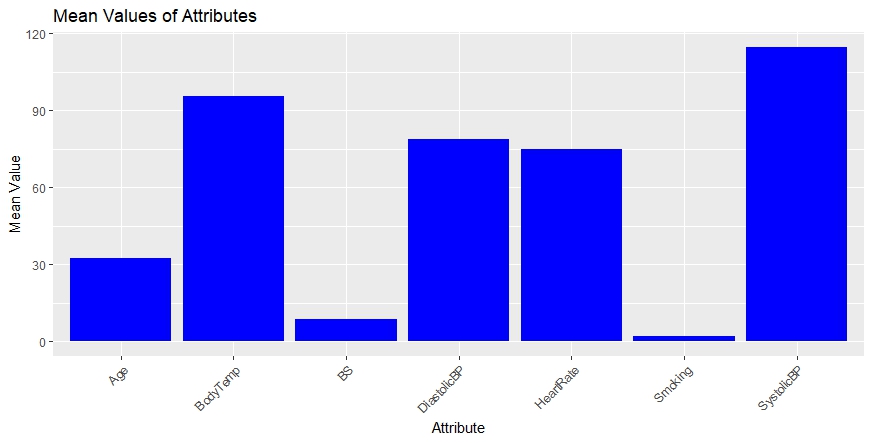
Code:

|  |
| --- |
| mean\_age <- mean(mydata$Age, na.rm = TRUE)  mean\_smoking <- mean(mydata$Smoking, na.rm = TRUE)  mean\_systolicBP <- mean(mydata$SystolicBP, na.rm = TRUE)  mean\_diastolicBP <- mean(mydata$DiastolicBP, na.rm = TRUE)  mean\_bs <- mean(mydata$BS, na.rm = TRUE)  mean\_bodytemp <- mean(mydata$BodyTemp, na.rm = TRUE)  mean\_heartrate <- mean(mydata$HeartRate, na.rm = TRUE)  mean\_age  mean\_smoking  mean\_systolicBP  mean\_diastolicBP  mean\_bs  mean\_bodytemp  mean\_heartrate  mean\_data <- data.frame(attribute = c("Age", "Smoking", "SystolicBP", "DiastolicBP", "BS", "BodyTemp", "HeartRate"),  mean\_value = c(mean\_age, mean\_smoking, mean\_systolicBP, mean\_diastolicBP, mean\_bs, mean\_bodytemp, mean\_heartrate))  ggplot(mean\_data, aes(x = attribute, y = mean\_value)) +  geom\_bar(stat = "identity", fill = "blue") +  labs(title = "Mean Values of Attributes",  x = "Attribute", y = "Mean Value") +  theme(axis.text.x = element\_text(angle = 45, hjust = 1)) |

Output:

A computer screen shot of a computer

Description automatically generated



Explanation:

The code above calculates the mean values of different attributes from a dataset named mydata, and then visualizes these mean values using a bar plot. Here's a brief explanation of what each part of the code does:

* The lines mean\_age <- mean(mydata$Age, na.rm = TRUE), mean\_smoking <- mean(mydata$Smoking, na.rm = TRUE), and so on, calculate the mean value for each attribute in the dataset, ignoring any missing values (`na.rm = TRUE`).
* The mean values for each attribute are printed using mean\_age, mean\_smoking, mean\_systolicBP, and so on.
* The data.frame() function is used to create a data frame named mean\_data, which contains two columns: "attribute" (the names of the attributes) and "mean\_value" (the corresponding mean values).
* The ggplot2 package is used to create a bar plot (geom\_bar()) with "attribute" on the x-axis and "mean\_value" on the y-axis. The bars are filled with blue color. Axis labels and title are added using labs(), and the x-axis labels are rotated by 45 degrees for better readability using theme(axis.text.x = element\_text(angle = 45, hjust = 1)).

In summary, this code computes the mean values of various attributes from a dataset and visualizes them using a bar plot, providing an overview of the average values for each attribute.

**Median:**

Code:

|  |
| --- |
| median\_age <- median(mydata$Age, na.rm = TRUE)  median\_smoking <- median(mydata$Smoking, na.rm = TRUE)  median\_systolicBP <- median(mydata$SystolicBP, na.rm = TRUE)  median\_diastolicBP <- median(mydata$DiastolicBP, na.rm = TRUE)  median\_bs <- median(mydata$BS, na.rm = TRUE)  median\_bodytemp <- median(mydata$BodyTemp, na.rm = TRUE)  median\_heartrate <- median(mydata$HeartRate, na.rm = TRUE)  median\_age  median\_smoking  median\_systolicBP  median\_diastolicBP  median\_bs  median\_bodytemp  median\_heartrate  mean\_data$median\_value <- c(median\_age, median\_smoking, median\_systolicBP, median\_diastolicBP,  median\_bs, median\_bodytemp, median\_heartrate)  ggplot(mean\_data, aes(x = attribute)) +  geom\_bar(aes(y = median\_value), stat = "identity", fill = "red") +  labs(title = "Median Values of Attributes",  x = "Attribute", y = "Median Value") +  theme(axis.text.x = element\_text(angle = 45, hjust = 1)) |

Output:

A computer screen shot of a computer

Description automatically generated

A graph showing a number of individuals

Description automatically generated

Explanation

The code above calculates the median value for several attributes in a dataset and visualizes the results using a bar plot. Here's a brief explanation:

* For each attribute (Age, Smoking, SystolicBP, DiastolicBP, BS, BodyTemp, HeartRate), the code calculates the median value using the median() function, while ignoring any missing values (na.rm = TRUE).
* The median values for each attribute are stored in separate variables (median\_age, median\_smoking, etc.).
* The median values are added to the existing mean\_data data frame as a new column named median\_value.
* The `ggplot()` function is used to create a bar plot of the median values. Each bar represents an attribute, and the height of the bar represents its median value.

In summary, this code calculates and visualizes the median values of different attributes in a dataset, providing insights into the central tendencies of these attributes.

**Mode:**

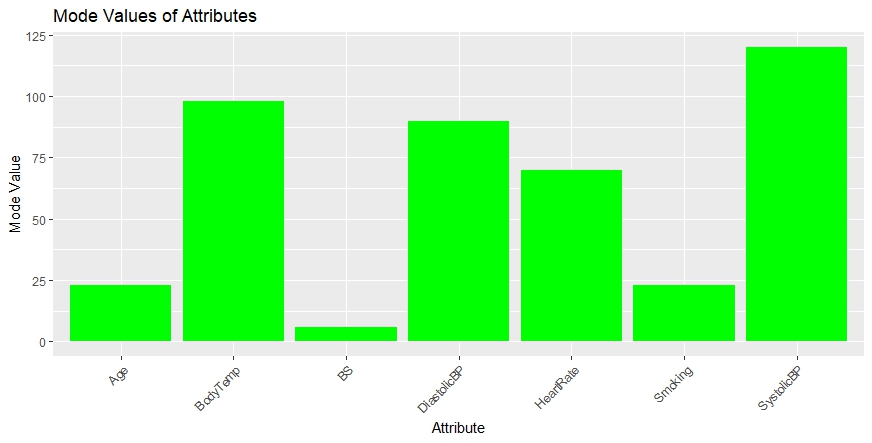
Code:

|  |
| --- |
| calculate\_mode <- function(x) {  unique\_x <- unique(x)  freq <- tabulate(match(x, unique\_x))  mode\_value <- unique\_x[which.max(freq)]  return(mode\_value)  }  mode\_age <- calculate\_mode(mydata$Age)  mode\_smoking <-calculate\_mode(mydata$Smoking)  mode\_systolicBP <- calculate\_mode(mydata$SystolicBP)  mode\_diastolicBP <- calculate\_mode(mydata$DiastolicBP)  mode\_bs <- calculate\_mode(mydata$BS)  mode\_bodytemp <- calculate\_mode(mydata$BodyTemp)  mode\_heartrate <- calculate\_mode(mydata$HeartRate)  mode\_age  mode\_smoking  mode\_systolicBP  mode\_diastolicBP  mode\_bs  mode\_bodytemp  mode\_heartrate  mean\_data$mode\_value <- c(mode\_age, mode\_age, mode\_systolicBP, mode\_diastolicBP,  mode\_bs, mode\_bodytemp, mode\_heartrate)  ggplot(mean\_data, aes(x = attribute)) +  geom\_bar(aes(y = mode\_value), stat = "identity", fill = "green") +  labs(title = "Mode Values of Attributes",  x = "Attribute", y = "Mode Value") +  theme(axis.text.x = element\_text(angle = 45, hjust = 1)) |

Output:

A computer screen shot of a computer screen

Description automatically generated



Explanation:

The code above defines a custom function to calculate the mode (most frequent value) of a vector and then uses this function to find the mode of several attributes in a dataset. It also visualizes the mode values using a bar plot. Here's a brief explanation:

* The calculate\_mode() function takes a vector `x` as input and calculates its mode using the following steps:
* unique\_x <- unique(x): Finds the unique values in the vector.
* freq <- tabulate(match(x, unique\_x)): Calculates the frequency of each unique value.
* mode\_value <- unique\_x[which.max(freq)]: Finds the value with the highest frequency, which represents the mode.
* return(mode\_value): Returns the mode value.
* For each attribute (Age, Smoking, SystolicBP, DiastolicBP, BS, BodyTemp, HeartRate) in the dataset:
* The mode value is calculated using the calculate\_mode() function.
* Each mode value is stored in separate variables (mode\_age, mode\_smoking, etc.).
* The mode values are added to the existing `mean\_data` data frame as a new column named mode\_value.
* The ggplot() function is used to create a bar plot of the mode values. Each bar represents an attribute, and the height of the bar represents its mode value.

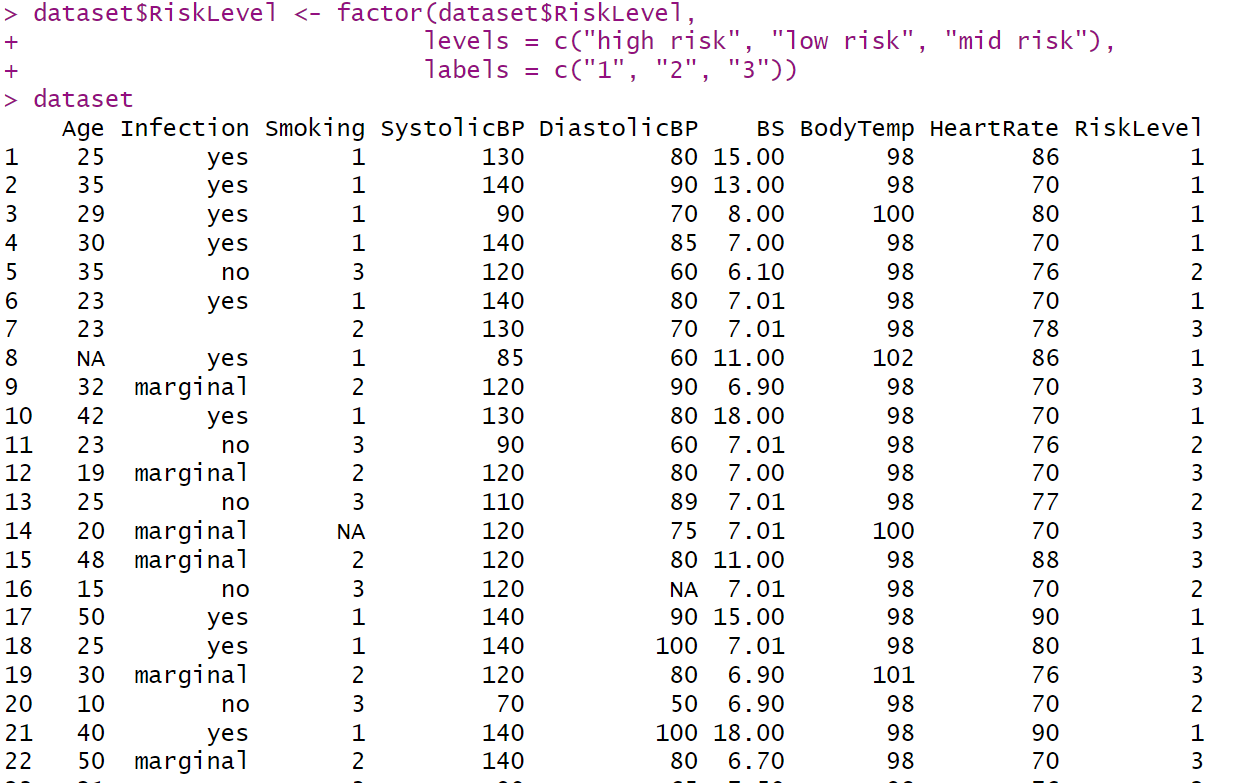
In general, this code calculates and visualizes the mode values of different attributes in a dataset, providing insights into the most frequently occurring values.

**Numeric attributes to categorical attributes:**

Code:

|  |
| --- |
| dataset$RiskLevel <- factor(dataset$RiskLevel,  levels = c("high risk","low risk","mid risk"),  labels = c("1", "2", "3"))  dataset |

Output:



Explanation:

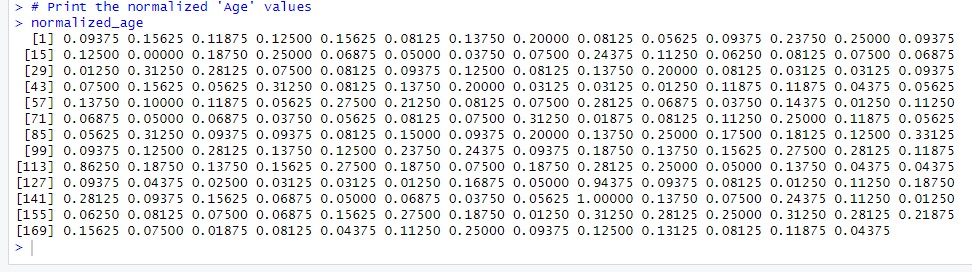
The code above converts the "RiskLevel" column in the dataset to a factor variable with three levels: "high risk", "low risk", and "mid risk". Additionally, it assigns corresponding numerical labels "1", "2", and "3" to these levels. This conversion is useful for categorical data analysis and ensures that the "RiskLevel" variable is treated appropriately as a factor with specific ordered levels in subsequent analyses.

**Normalization method:**

Code:

|  |
| --- |
| age <- mydata$Age  min\_max\_normalization <- function(x) {  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))  }  normalized\_age <- min\_max\_normalization(age)  normalized\_age |

Output:



Explanation :

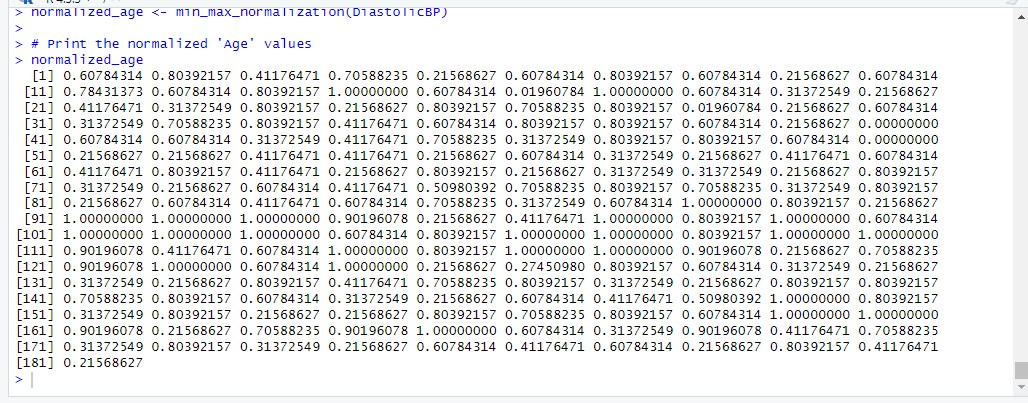
The code above defines a function called `min\_max\_normalization` that performs min-max normalization on a given vector `x`. Min-max normalization rescales the values of the vector to a range between 0 and 1 based on the minimum and maximum values of the vector. Then, it applies this function to the "Age" column of the dataset `mydata`, storing the normalized values in a new variable called `normalized\_age`. This normalization technique is commonly used to scale numerical data to a common range for better comparison and analysis.

**DiastolicBP:**

Code:

|  |
| --- |
| DiastolicBP <- mydata$DiastolicBP  min\_max\_normalization <- function(x) {  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))  }  normalized\_age <- min\_max\_normalization(DiastolicBP)  normalized\_age |

Output:



Explanation:

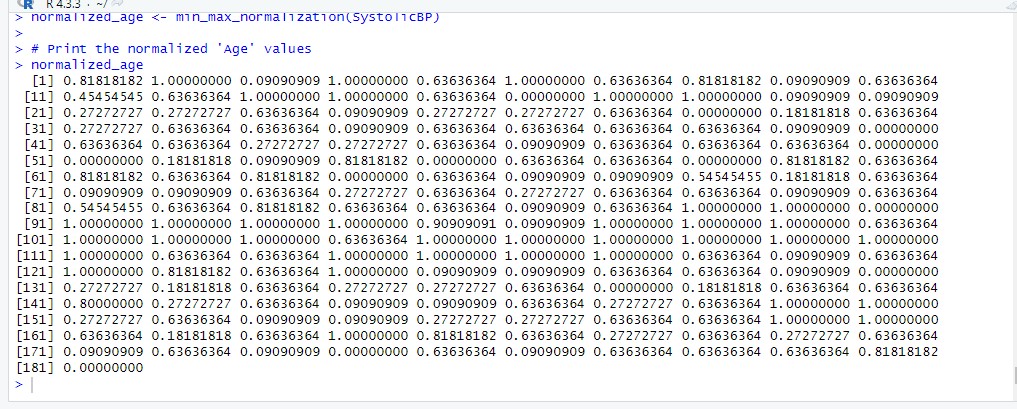
The code above extracts the "DiastolicBP" column from the dataset mydata and then applies min-max normalization to this column using a custom function called min\_max\_normalization. The resulting normalized values are stored in a new variable named normalized\_age. This normalization process scales the values of the "DiastolicBP" column to a range between 0 and 1 based on their minimum and maximum values, facilitating comparison and analysis across different datasets or variables.

**SystolicBP:**

Code:

|  |
| --- |
| SystolicBP<- mydata$SystolicBP  min\_max\_normalization <- function(x) {  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))  }  normalized\_age <- min\_max\_normalization(SystolicBP)  normalized\_age |

Output:



Explanation:

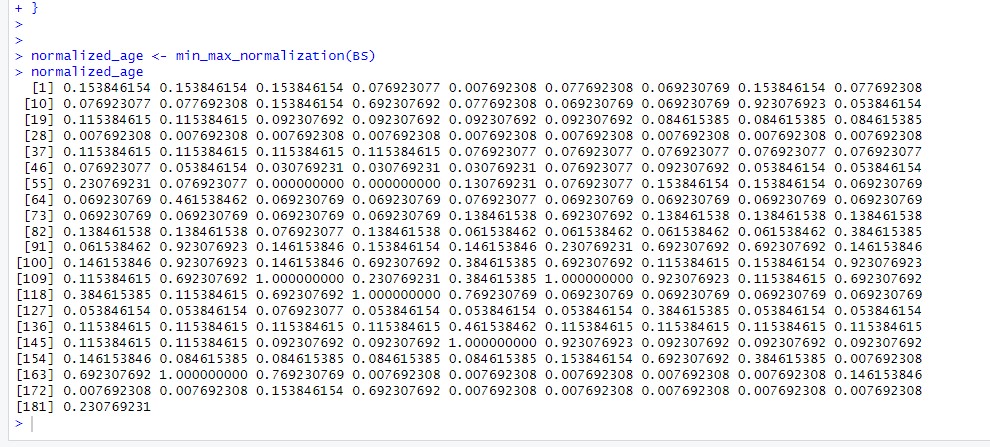
The code above extracts the "SystolicBP" column from the dataset mydata and then applies min-max normalization to this column using a custom function called min\_max\_normalization. The resulting normalized values are stored in a new variable named normalized\_age. This normalization process scales the values of the "SystolicBP" column to a range between 0 and 1 based on their minimum and maximum values, facilitating comparison and analysis across different datasets or variables. Finally, it prints the normalized values of the "SystolicBP" column.

**BS:**

Code:

|  |
| --- |
| BS<- mydata$BS  min\_max\_normalization <- function(x) {  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))  }  normalized\_age <- min\_max\_normalization(BS)  normalized\_age |

Output:



Explanation:

The code above extracts the "BS" column from the dataset mydata and then applies min-max normalization to this column using a custom function called min\_max\_normalization. The resulting normalized values are stored in a new variable named normalized\_age. This normalization process scales the values of the "BS" column to a range between 0 and 1 based on their minimum and maximum values, facilitating comparison and analysis across different datasets or variables.

**BodyTemp:**

Code:

|  |
| --- |
| BodyTemp<- mydata$BodyTemp  min\_max\_normalization <- function(x) {  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))  }  normalized\_age <- min\_max\_normalization(BodyTemp)  normalized\_age |

Output:

A screenshot of a computer

Description automatically generated

Explanation:

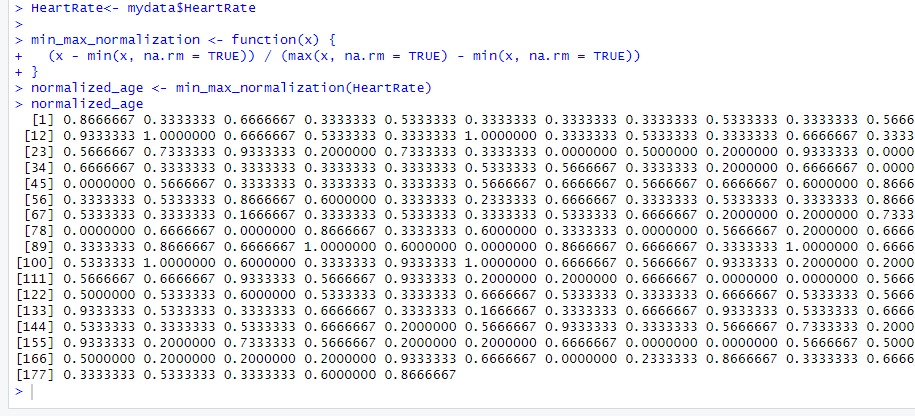
The code above extracts the "BodyTemp" column from the dataset mydata and then applies min-max normalization to this column using a custom function called min\_max\_normalization. The resulting normalized values are stored in a new variable named normalized\_age. This normalization process scales the values of the "BodyTemp" column to a range between 0 and 1 based on their minimum and maximum values, facilitating comparison and analysis across different datasets or variables.

**HeartRate**

Code:

|  |
| --- |
| HeartRate<- mydata$HeartRate  min\_max\_normalization <- function(x) {  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))  }  normalized\_age <- min\_max\_normalization(HeartRate)  normalized\_age |

Output:



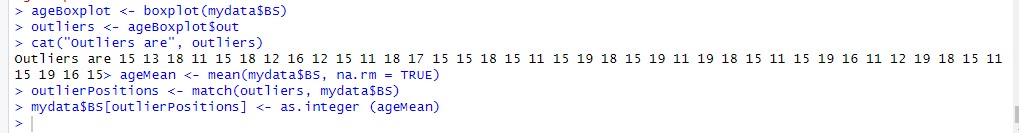
Explanation:  
The code above extracts the "HeartRate" column from the dataset mydata and then applies min-max normalization to this column using a custom function called min\_max\_normalization. The resulting normalized values are stored in a new variable named normalized\_age. This normalization process scales the values of the "HeartRate" column to a range between 0 and 1 based on their minimum and maximum values, facilitating comparison and analysis across different datasets or variables. Finally, it prints the normalized values of the "HeartRate" column.

**Replacing BS outliers with mean value:**

Code:

|  |
| --- |
| ageBoxplot <- boxplot(mydata$BS)  outliers <- ageBoxplot$out  cat("Outliers are", outliers)  ageMean <- mean(mydata$BS, na.rm = TRUE)  outlierPositions <- match(outliers, mydata$BS)  mydata$BS[outlierPositions] <- as.integer (ageMean) |

Output:



A line drawing of a line

Description automatically generated with medium confidence

Explanation:

The code above creates a boxplot of the "BS" column in the dataset mydata and identifies outliers using the boxplot statistics. It then calculates the mean of the "BS" column, finds the positions of the outliers within the dataset, and replaces these outlier values with the integer value of the mean. This process is aimed at addressing outliers in the "BS" column by replacing them with the mean value.

**Replacing BodyTemp outliers with mean value:**

Code:

|  |
| --- |
| ageBoxplot <- boxplot(mydata$BodyTemp)  outliers <- ageBoxplot$out  cat("Outliers are", outliers)  ageMean <- mean(mydata$BodyTemp, na.rm = TRUE)  outlierPositions <- match(outliers, mydata$BodyTemp)  mydata$BodyTemp[outlierPositions] <- as.integer (ageMean) |

Output:

A screenshot of a computer

Description automatically generated

A black line with a white background

Description automatically generated

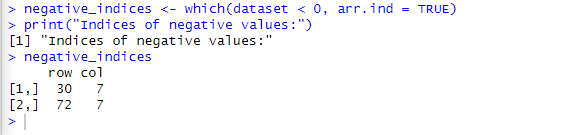
Explanation:  
This code creates a boxplot of the "BodyTemp" column in the dataset mydata and identifies outliers using the boxplot statistics. It then calculates the mean of the "BodyTemp" column, finds the positions of the outliers within the dataset, and replaces these outlier values with the integer value of the mean. This process aims to handle outliers in the "BodyTemp" column by replacing them with the mean value.

**Find Invalid values:**

Code:

|  |
| --- |
| negative\_indices <- which(dataset < 0, arr.ind = TRUE)  print("Indices of negative values:")  negative\_indices |

Output:



Explanation:

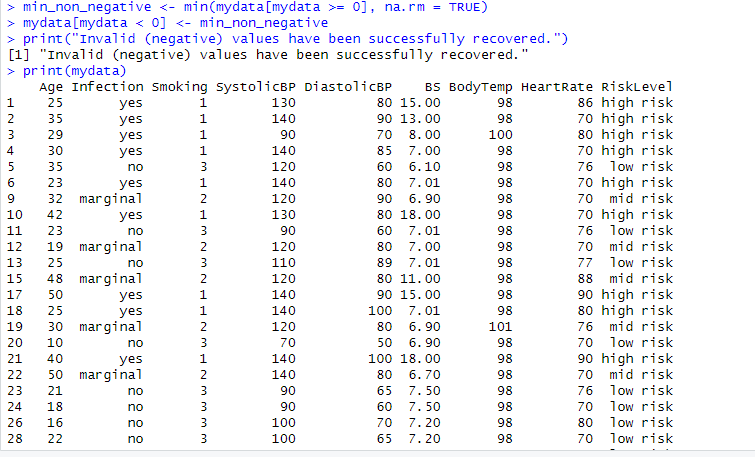
This code finds the indices of negative values in the dataset and prints them. The `which()` function with the condition `< 0` identifies the indices where the dataset has negative values. Setting `arr.ind = TRUE` ensures that the indices are returned as an array. Finally, it prints the indices of the negative values in the dataset.

**Remove Invalid Value**

Code:

|  |
| --- |
| min\_non\_negative <- min(mydata[mydata >= 0], na.rm = TRUE)  mydata[mydata < 0] <- min\_non\_negative  print("Invalid (negative) values have been successfully recovered.")  print(mydata) |

Output:



Explanation:  
This code finds the minimum non-negative value (min\_non\_negative) in the dataset mydata, replaces all negative values in mydata with this minimum non-negative value, prints a message indicating successful recovery of negative values, and finally prints the updated dataset mydata.