

American International University-Bangladesh (AIUB)  
**Department of Computer Science  
Faculty of Science & Technology (FST)**

Introduction to Data Science

SEC: B

Supervised By

TOHEDUL ISLAM

A Introduction to Data Science Project Submitted

By

|  |  |  |
| --- | --- | --- |
| **Semester: Summer\_23\_24** | | **Section: B** |
| SN | Student Name | Student ID |
| 1 | MAHMUD AL ALVI | 22-46128-1 |
| 2 | MD MAHADI HASAN NAYEEM | 22-46137-1 |

**Dataset Description:**

**T**his project focuses on the comprehensive preparation and analysis of a dataset comprising records of 105 passengers from the Titanic disaster. These records contain 10 features per passenger, including demographic and survival information. The primary objective of this project is to properly moderate the dataset by addressing issues such as missing, noisy, and invalid values. In addition, for advanced analysis, continuous or numeric attributes may be converted into categorical attributes or vice versa, filtering, mismatched value correction and normalization methods can be applied.

The project begins with a thorough investigation of visualization techniques to identify patterns and develop informed approaches for handling missing data. Methods for identifying and correcting invalid and incorrect data are also explored to enhance the dataset's quality and reliability. Using various visualization techniques, such as box plots and bar charts, essential data points can be analyzed and visualized more efficiently, leading to more intelligent analysis.

By establishing a solid foundation for dataset preparation and univariate data exploration, the project aims to facilitate sophisticated analyses that yield valuable insights into the factors influencing survival on the Titanic. These insights have the potential to inform safety measures and enhance our understanding of historical events.

This report outlines the methodologies utilized, challenges encountered, and insights gained throughout the project, providing a comprehensive overview of the data preparation and analysis processes undertaken.

**1. Importing Dataset into Rstudio**

**Code:**

mydata<-read.csv("E:/8th Sem/MIDTERM/DATA SCIENCE/project/Midterm Project 2/Midterm Project/Mid\_Dataset.csv",header=TRUE,sep=",")

mydata

**Output:**

A screenshot of a computer

Description automatically generated

**Description:**

Here is the code of import the dataset as csv file. It is the output of the dataset which is imported in RStudio.

**2. Handling Missing Values in Dataset**

**Code:**

mydata$Gender <- ifelse(mydata$Gender == "", NA, mydata$Gender)

mydata$fare <- ifelse(mydata$fare == "", NA, mydata$fare)

mydata$embarked <- ifelse(mydata$embarked == "", NA, mydata$embarked)

mydata$class <- ifelse(mydata$class == "", NA, mydata$class)

mydata$who <- ifelse(mydata$who == "", NA, mydata$who)

mydata

**Output:**

A screenshot of a computer screen

Description automatically generated

**Description:**

This code replaces empty string entries in the Gender, fare, embarked, class, and who columns of mydata with NA to mark them as missing values.

**3. Graph of Missing Values of the Dataset**

**Code:**

missing\_values\_plot <- ggplot(missing\_values\_df, aes(x = reorder(Variable, -Missing), y = Missing)) +

geom\_bar(stat = "identity", fill = "lightblue") +

labs(title = "Missing Values in Dataset",

x = "Variables",

y = "Number of Missing Values") +

theme(axis.text.x = element\_text(angle = 45, hjust = 1))

print(missing\_values\_plot)

**Output:A graph with blue squares

Description automatically generated**

**Description:**

This code creates a bar plot (missing\_values\_plot) using ggplot2 to visualize the number of missing values for each variable in the dataset .The plot displays variables reordered by the count of missing values on the x-axis and the corresponding number of missing values on the y-axis. Axes labels and plot title are specified to enhance clarity, with x-axis text rotated for better readability.

**4. Summarizing Central Tendency and Spread**

**Code:**

summary(mydata)

**Output:**

A screenshot of a computer code

Description automatically generated

**Description:**

This code provides a summary of mydata, displaying key measures of central tendency (mean, median) and spread (min, max, quartiles) for each column.

**5. Exploring Data Structure**

**Code:**

str(mydata)

**Output:**

A screenshot of a computer code

Description automatically generated

**Description:**

This code displays the structure of mydata, summarizing its dimensions and variable types for quick reference.

**6. Detecting Null Values**

**Code:**

mydata[mydata == ""] <- NA

is.na(mydata)

**Output:**

A white background with black text

Description automatically generated

**Description:**

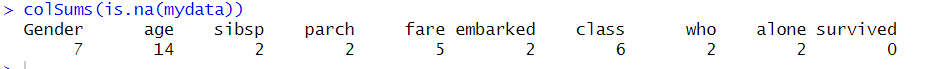
This code replaces all empty string entries in mydata with NA to mark them as null values. It then checks for the presence of NA values in the dataset, outputting TRUE for null values and FALSE otherwise, identifying the location of missing data points.

**7. Counting Null Values in Each Column**

**Code:**

colSums(is.na(mydata))

**Output:**



**Description:**

This code counts the number of NA (null) values in each column of mydata and returns the result. This helps to understand the extent of missing data in each column.

**8. Identifying Rows of Null Values**

**Code:**

which(is.na(mydata$Gender))

which(is.na(mydata$age))

which(is.na(mydata$sibsp))

which(is.na(mydata$parch))

which(is.na(mydata$fare))

which(is.na(mydata$embarked))

which(is.na(mydata$class))

which(is.na(mydata$who))

which(is.na(mydata$alone))

**Output:**

**A screenshot of a computer code

Description automatically generated**

**Description:**

This code identifies the specific rows in mydata that contain NA (null) values for each specified column (Gender, age, sibsp, parch, fare, embarked, class, who, and alone). It returns the indices of these rows, helping to locate where the missing data points are in the dataset.

**9. Removing Rows with Missing Values (Discard Instance)**

**Code:**

remove<-na.omit(mydata)

remove

**Output:**

A screenshot of a computer screen

Description automatically generated

**Description:**

This code removes all rows with NA values from mydata, resulting in a new dataset called remove that contains only complete cases.

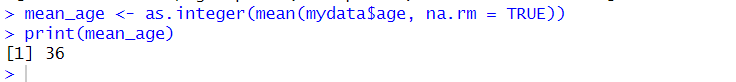
**10. Calculating Mean Age**

**Code:**

mean\_age <- as.integer(mean(mydata$age, na.rm = TRUE))

print(mean\_age)

**Output:**

**Description:**

This code calculates the mean age from the age column in mydata, ignoring NA values (na.rm = TRUE). The result, stored in mean\_age, is then printed as an integer value.

**11. Imputing Missing Values with Mean(Age)**

**Code:**

mydata <- mydata %>%

mutate(age = ifelse(is.na(age), mean\_age, age))

mydata

**Output:**

A screenshot of a computer

Description automatically generated

**Description:**

This code replaces missing values in the age column of mydata with the previously calculated mean age (mean\_age). It updates mydata with these imputed values, ensuring more complete data for further analysis.

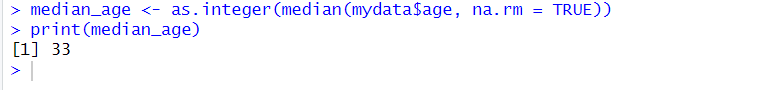
**12. Calculating Median Age**

**Code:**

median\_age <- as.integer(median(mydata$age, na.rm = TRUE))

print(median\_age)

**Output:**

**Description:**

This code calculates the median age from the age column in mydata, ignoring NA values (na.rm = TRUE). The result, stored in median\_age, is then printed as an integer value.

**13. Imputing Missing Values with Median(Age)**

**Code:**

mydata <- mydata %>%

mutate(age = ifelse(is.na(age), median\_age, age))

mydata

**Output:**

A screenshot of a computer

Description automatically generated

**Description:**

This code replaces missing values in mydata$age with the median age (median\_age), ensuring all data points are accounted for before analysis.

**14. Imputing Missing Values with Mode(Sibsp)**

**Code:**

column\_name <- "sibsp"

mode\_value <- as.numeric(names(which.max(table(mydata[[column\_name]]))))

print(mode\_value)

**Output:**



**Code:**

column\_name <- "sibsp"

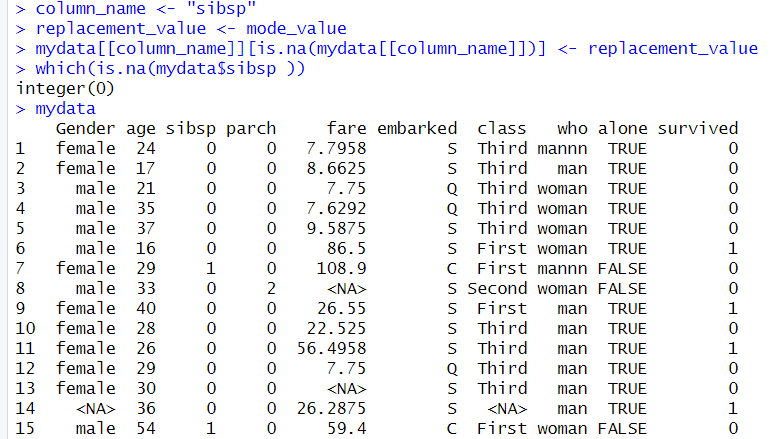
replacement\_value <- mode\_value

mydata[[column\_name]][is.na(mydata[[column\_name]])] <- replacement\_value

which(is.na(mydata$sibsp ))

mydata

**Output:**



A screenshot of a computer

Description automatically generated

**Description:**

This code calculates the mode value for the sibsp column in mydata. It then replaces any NA values in the sibsp column with this mode value, ensuring all missing data is filled with the most frequent value in that column.

**15. Imputing Missing Values with Mode(Parch)**

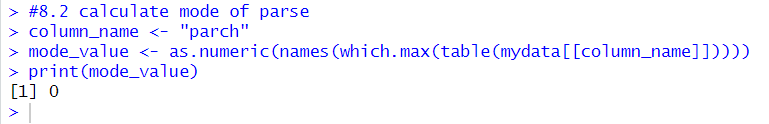
**Code:**

column\_name <- "parch"

mode\_value <- as.numeric(names(which.max(table(mydata[[column\_name]]))))

print(mode\_value)

**Output:**



**Code:**

column\_name <- "parch"

replacement\_value <- mode\_value

mydata[[column\_name]][is.na(mydata[[column\_name]])] <- replacement\_value

which(is.na(mydata$parch ))

mydata

**Output:**

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description:**

This code calculates the mode value for the parch column in mydata. It then replaces any NA values in the parch column with this mode value, ensuring all missing data is filled with the most frequent value in that column.

**16. Imputing Missing Values with Mode(Alone)**

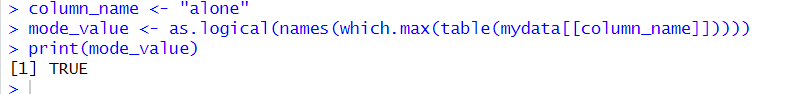
**Code:**

column\_name <- "alone"

mode\_value <- as.logical(names(which.max(table(mydata[[column\_name]]))))

print(mode\_value)

**Output:**

****

**Code:**

column\_name <- "alone"

replacement\_value <- mode\_value

mydata[[column\_name]][is.na(mydata[[column\_name]])] <- replacement\_value

which(is.na(mydata$alone ))

mydata

**Output:**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description:**

This code calculates the mode value for the alone column in mydata. It then replaces any NA values in the alone column with this mode value, ensuring all missing data is filled with the most frequent value in that column.

**17. Imputing Missing Values with Mode(Gender)**

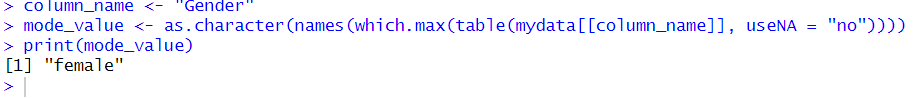
**Code:**

column\_name <- "Gender"

mode\_value <- as.character(names(which.max(table(mydata[[column\_name]], useNA = "no"))))

print(mode\_value)

**Output:**

****

**Code:**

column\_name <- "Gender"

replacement\_value <- mode\_value

mydata[[column\_name]][is.na(mydata[[column\_name]])] <- replacement\_value

which(is.na(mydata$Gender ))

mydata

**Output:**

A screenshot of a computer

Description automatically generated

**Description:**

This code calculates the mode value for the Gender column in mydata, excluding NA values. It then replaces any NA values in the Gender column with this mode value, ensuring all missing data is filled with the most frequent value in that column.

**18.Imputing Missing Values with Mode(Class)**

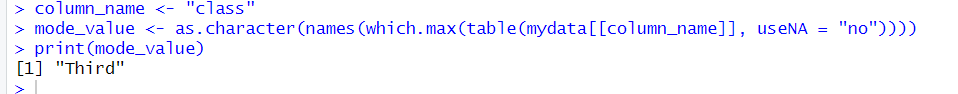
**Code:**

column\_name <- "class"

mode\_value <- as.character(names(which.max(table(mydata[[column\_name]], useNA = "no"))))

print(mode\_value)

**Output:**



**Code:**

column\_name <- "class"

replacement\_value <- mode\_value

mydata[[column\_name]][is.na(mydata[[column\_name]])] <- replacement\_value

which(is.na(mydata$class ))

mydata

**Output:**

A screenshot of a computer

Description automatically generated

**Description:**

This code calculates the mode value for the class column in mydata, excluding NA values. It then replaces any NA values in the class column with this mode value, ensuring all missing data is filled with the most frequent value in that column.

**19.Imputing Missing Values with Mode (Embarked):**

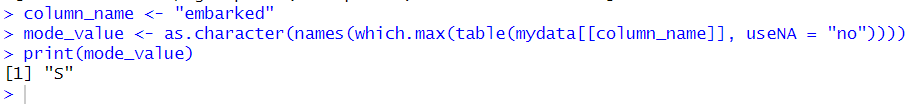
**Code:**

column\_name <- "embarked"

mode\_value <- as.character(names(which.max(table(mydata[[column\_name]], useNA = "no"))))

print(mode\_value)

**Output:**



**Code:**

column\_name <- "embarked"

replacement\_value <- mode\_value

mydata[[column\_name]][is.na(mydata[[column\_name]])] <- replacement\_value

which(is.na(mydata$embarked ))

mydata

**Output:A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description:**

This code calculates the mode value for the embarked column in mydata, excluding NA values. It then replaces any NA values in the embarked column with this mode value, ensuring all missing data is filled with the most frequent value in that column. The updated mydata is displayed after imputation

**20.Imputing Missing Values with Mode(Who):**

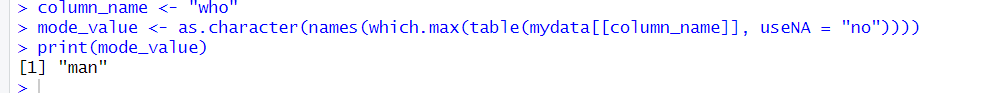
**Code:**

column\_name <- "who"

mode\_value <- as.character(names(which.max(table(mydata[[column\_name]], useNA = "no"))))

print(mode\_value)

**Output:**

****

**Code:**

column\_name <- "who"

replacement\_value <- mode\_value

mydata[[column\_name]][is.na(mydata[[column\_name]])] <- replacement\_value

which(is.na(mydata$who ))

mydata

**Output: A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Description:**

This code calculates the mode value for the who column in mydata, excluding NA values. It then replaces any NA values in the who column with this mode value, ensuring all missing data is filled with the most frequent value in that column.

**21. Outlier Detection and Removal (Age):**

**Code:**

Q1 <- quantile(mydata$age[mydata$age >= 40 & mydata$age <= 95], 0.25)

Q3 <- quantile(mydata$age[mydata$age >= 40 & mydata$age <= 95], 0.75)

IQR\_value <- Q3 - Q1

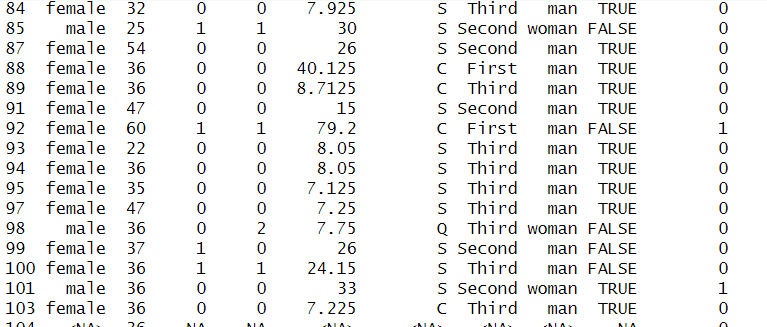
threshold <- 1.5

outlier\_condition <- (mydata$age < (Q1 - threshold \* IQR\_value)) | (mydata$age > (Q3 + threshold \* IQR\_value))

mydata <- mydata[!outlier\_condition, ]

mydata

**Output:**

****

**Description:**

This code identifies and removes outliers from the `age` variable in `mydata` for ages between 40 and 95 using the Interquartile Range (IQR) method. It calculates the IQR, defines an outlier condition based on a threshold, and then filters out the rows where the age values are considered outliers.

**22.Invalid Value Detection (Fare)**

**Code:**

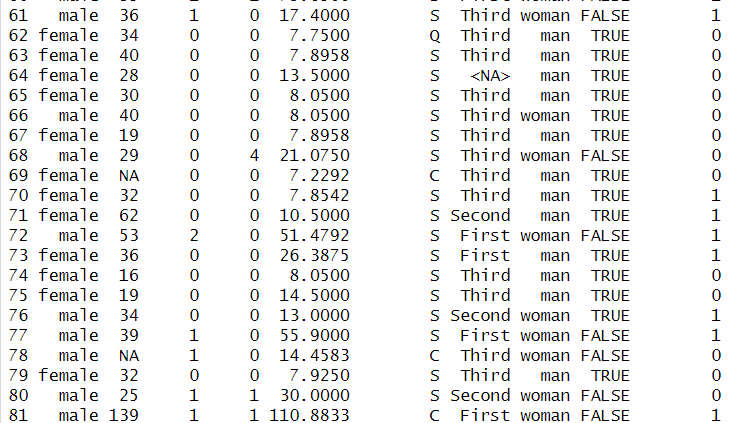
mydata <- mydata %>%

mutate(fare = as.numeric(as.character(fare))) %>%

filter(!is.na(fare))

mydata

**Output:**



**Description:**

This code converts the fare column in mydata to numeric format, ensuring any invalid values are converted to NA. It then filters out rows where fare is NA, effectively removing invalid entries from the dataset and retaining only numeric fare values for further analysis.

**23. Invalid Value Detection(Who):**

**Code:**

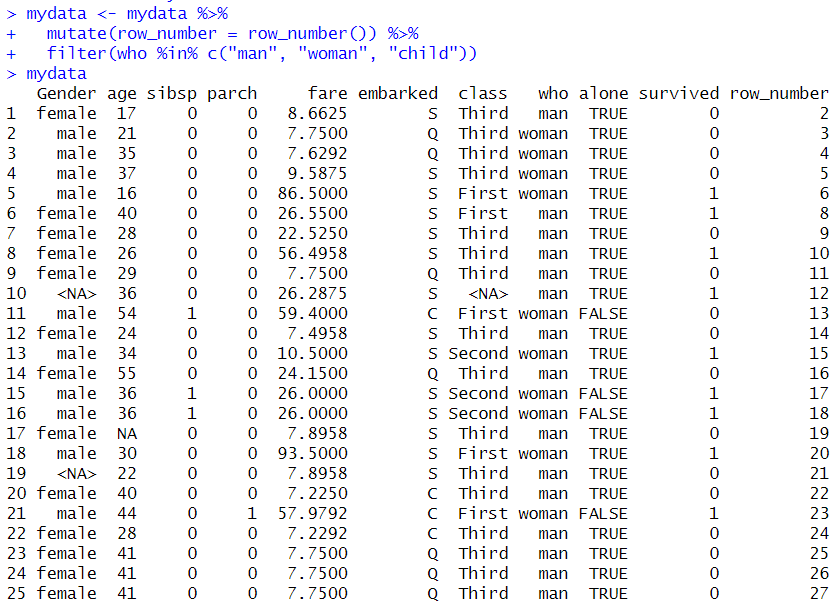
mydata <- mydata %>%

mutate(row\_number = row\_number()) %>%

filter(who %in% c("man", "woman", "child"))

mydata

**Output:**

****

**Description:**

This code identifies and removes rows from mydata where the who column contains invalid values. It retains only those rows where who is either "man", "woman", or "child", ensuring the dataset is cleansed of any rows with incorrect 'who' values.

**24.Converting Categorical Variables to Numeric Factors:**

**Code:**

mydata <- mydata %>%

mutate(

Gender = factor(Gender, levels = c("male", "female"), labels = c(1, 2)),

embarked = factor(embarked, levels = c("C","Q","S"), labels = c(1,2,3)),

class = factor(class , levels = c("First", "Second", "Third"), labels = c(1, 2, 3)),

who = factor(who, levels = c("man", "woman","child"), labels = c(1, 2,3)),

alone = factor(alone, levels = c("TRUE", "FALSE"), labels = c(1, 2))

)

mydata

**Output:**

**A screenshot of a computer program

Description automatically generated**

**Description:**

This code converts categorical variables (Gender, embarked, class, who, alone) in mydata into numeric factors for further analysis. Each categorical variable is transformed using the factor() function with specified levels and labels to map categorical values to corresponding numeric codes.

**25.Converting Numerical Variables to Categorical Factors:**

**Code:**

mydata <- mydata %>%

mutate(

survived = factor(survived, levels = c(0, 1), labels = c("Dead", "Alive")),

)

mydata

**Output:A screenshot of a computer

Description automatically generated**

**Description:**

This code converts the survived column in mydata from a numeric representation (0 for dead, 1 for alive) to a factor with descriptive labels ("Dead" and "Alive"). This transformation helps in interpreting and analyzing survival data more intuitively.

**26.Categorizing Age into Sets:**

**Code:**

mydata <- mydata %>%

mutate(age\_category = case\_when(

age < 18 ~ "child",

age >= 18 & age <= 30 ~ "young",

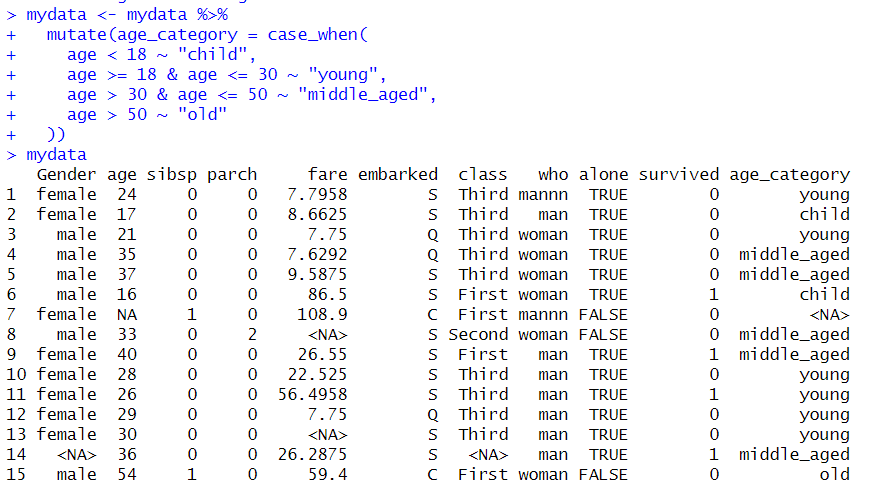
age > 30 & age <= 50 ~ "middle\_aged",

age > 50 ~ "old"

))

mydata

**Output:**

****

**Description:**

This code categorizes the age attribute in mydata into different age groups (age\_category). Age values are classified as "child" for ages under 18, "young" for ages 18 to 30, "middle\_aged" for ages 31 to 50, and "old" for ages over 50. The resulting dataset mydata includes the new age\_category column for further analysis based on age groups.

**27.Normalizing the Age Attribute:**

**Code:**

column\_name <- "age"

column <- mydata[[column\_name]]

min\_value <- min(column)

max\_value <- max(column)

normalized\_column <- (column - min\_value) / (max\_value - min\_value)

mydata[[column\_name]] <- normalized\_column

mydata$age

mydata

**Output:**

**A screenshot of a computer screen

Description automatically generated**

**Description:**

This code normalizes the age attribute in mydata to a range between 0 and 1. It calculates the minimum and maximum values of age, then performs min-max normalization on the column. The normalized values replace the original age column in mydata, ensuring that age values are scaled uniformly for analysis or modeling purposes.

**42.Resolving 'who' Attribute Mismatches:**

**Code:**

mydata <- mydata %>%

mutate(who = case\_when(

Gender == "male" & who != "child" ~ "man",

Gender == "female" & who != "child" ~ "woman",

TRUE ~ who

))

Mydata

**Output:**

**A screenshot of a computer code

Description automatically generated**

**Description:**

* Updates who to "man" if Gender is "male" and who is not "child".
* Updates who to "woman" if Gender is "female" and who is not "child".
* Leaves who unchanged (TRUE ~ who) if no conditions are met.

**29.Finding And Removing Duplicate Value(Age)**

**Code:**

duplicates\_age<-distinct(mydata,age,.keep\_all = TRUE)

duplicates\_age

**Output:**

**A screenshot of a computer

Description automatically generated**

**Description:**

This code uses the distinct() function to find unique rows in mydata based on the age column. The result, stored in duplicates\_age, displays each unique row where the age value is distinct, preserving all columns (keep\_all = TRUE).

**30.Finding And Removing Duplicate Value(Fare)**

**Code:**

duplicates\_fare<-distinct(mydata,fare,.keep\_all = TRUE)

duplicates\_fare

**Output:**

**A screenshot of a computer

Description automatically generated**

**Description:**

This code uses the distinct() function to find unique rows in mydata based on the fare column. The result, stored in duplicates\_fare, displays each unique row where the fare value is distinct, preserving all columns (keep\_all = TRUE). This helps in identifying unique fare values in the dataset.

**31.Filtering Data by Criteria**

**Code:**

data<-filter(mydata,Gender=="female")

data

data<-filter(mydata,age > 15 & age < 45)

data

data<-filter(mydata,sibsp >1 )

data

data<-filter(mydata,parch >1 )

data

data<-filter(mydata,embarked=="Q")

data

data<-filter(mydata,class=="Third")

data

**Output:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Description:**

This code demonstrates filtering operations on mydata using different criteria:

* Gender == "female": Filters rows where Gender is "female".
* age > 15 & age < 45: Filters rows where age is between 15 and 45.
* sibsp > 1: Filters rows where sibsp (number of siblings/spouses aboard) is greater than 1.
* parch > 1: Filters rows where parch (number of parents/children aboard) is greater than 1.
* embarked == "Q": Filters rows where embarked (port of embarkation) is "Q" (Queenstown).
* class == "Third": Filters rows where class is "Third".

**Bar Plot:**

**32.Age Distribution:**

**Code:**

mydata$age\_group <- cut(mydata$age, breaks = c(0, 18, 30,50, Inf),

labels = c("1-18", "19-30","31-50", "50+"), include.lowest = TRUE)

age\_plot <- ggplot(mydata, aes(x = age\_group, fill = age\_group)) +

geom\_bar() +

labs(title = "Age Distribution",

x = "Age Group",

y = "Count") +

scale\_fill\_manual(values = c("1-18" = "yellow", "19-30" = "lightblue","31-50" = "green", "50+" = "red")) +

scale\_x\_discrete(labels = c("1-18" = "1-18", "19-30" = "19-30","31-50" = "31-50", "50+"= "50+"))

print(age\_plot)

**Output:**

**A graph of a number of people

Description automatically generated**

**Description:**

This code uses ggplot2 to create a bar plot (age\_plot) displaying the distribution of passenger ages categorized into groups (age\_group) in mydata, with manual color assignments for each age group.

**33.Survival Distribution**

**Code:**

mydata$survived\_group <- cut(mydata$survived, breaks = c(-Inf, 0.5, Inf),

labels = c("0", "1"))

survived\_plot <- ggplot(mydata, aes(x = survived\_group, fill = survived\_group)) +

geom\_bar() +

labs(title = "Survived Distribution",

x = "Survived Group",

y = "Count") +

scale\_fill\_manual(values = c("0" = "green", "1" = "red")) +

scale\_x\_discrete(labels = c("0" = "0", "1" = "1"))

print(survived\_plot)

**Output:**

**A graph with red and green squares

Description automatically generated**

**Description:**

This code creates a bar plot (survived\_plot) using ggplot2 to visualize the distribution of survival status in mydata, categorized into groups ("0" for not survived, "1" for survived). Colors are manually assigned to each group, with labels and titles for clarity.

**34.Gender Distribution:**

**Code:**

sex\_plot <- ggplot(mydata, aes(x = factor(Gender), fill = factor(Gender))) +

geom\_bar() +

labs(title = "Distribution of Gender",

x = "Gender",

y = "Count") +

scale\_fill\_manual(values = c("male" = "blue", "female" = "pink")) +

scale\_x\_discrete(labels = c("male" = "male", "female" = "female"))

print(sex\_plot)

**Output:** **A graph with a blue and pink squares

Description automatically generated with medium confidenceDescription:**

This code utilizes ggplot2 to create a bar plot (sex\_plot) illustrating the distribution of gender (Gender) in mydata. Each bar represents the count of passengers categorized as "male" or "female", with manual color assignments and axis labels (x for Gender and y for Count) for enhanced visualization.

**35.Embarked Distribution:**

**Code:**

embarked\_plot <- ggplot(mydata, aes(x = factor(embarked), fill = factor(embarked))) +

geom\_bar() +

labs(title = "Distribution by Embarked",

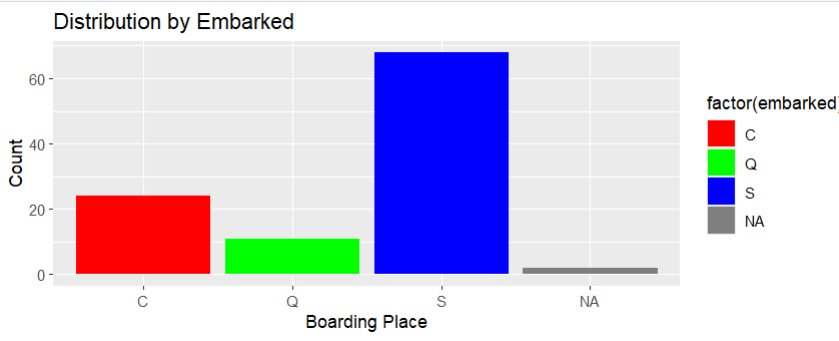
x = "Boarding Place",

y = "Count") +

scale\_fill\_manual(values = c("C" = "red", "Q" = "green","S" = "blue")) +

scale\_x\_discrete(labels = c("C" = "C", "Q" = "Q","S" = "S"))

print(embarked\_plot)

**Output:** **Description:**

This code creates a bar plot (embarked\_plot) using ggplot2 to visualize the distribution of passengers based on their boarding place (embarked) in mydata. Each bar represents the count of passengers who embarked from "C" (Cherbourg), "Q" (Queenstown), and "S" (Southampton), with manually assigned colors and axis labels (x for Boarding Place and y for Count) for clarity and visual appeal.

**36.Class Distribution:**

**Code:**

class\_plot <- ggplot(mydata, aes(x = factor(class), fill = factor(class))) +

geom\_bar() +

labs(title = "Distribution by class",

x = "Class",

y = "Count") +

scale\_fill\_manual(values = c("First" = "purple", "Second" = "lightblue","Third" = "pink3")) +

scale\_x\_discrete(labels = c("First" = "First", "Second" = "Second","Third" = "Third"))

print(class\_plot)

**Output:** **A graph with a number of squares

Description automatically generated with medium confidenceDescription:**

This code creates a bar plot (class\_plot) using ggplot2 to visualize the distribution of passengers based on their class (class) in mydata. Each bar represents the count of passengers in "First", "Second", and "Third" classes, with manually assigned colors and axis labels (x for Class and y for Count) for clarity and visual appeal.

**37.Alone Status Distribution:**

**Code:**

alone\_plot <- ggplot(mydata, aes(x = factor(alone), fill = factor(alone))) +

geom\_bar() +

labs(title = "Distribution of alone",

x = "alone or not alone",

y = "Count") +

scale\_fill\_manual(values = c("TRUE" = "green4", "FALSE" = "red4")) +

scale\_x\_discrete(labels = c("TRUE" = "ALONE", "FALSE" = "NOT ALONE"))

print(alone\_plot)

**Output:** **A green and red squares

Description automatically generated**

**Description:**

This code utilizes ggplot2 to create a bar plot (alone\_plot) illustrating the distribution of passengers based on whether they were traveling alone (alone) in mydata. Each bar represents the count of passengers categorized as "ALONE" (True) or "NOT ALONE" (False), with manual color assignments and axis labels (x for Alone or Not Alone and y for Count) for clarity and visual appeal.

**38.Statistical Summary Visualization**

**Code:**

selected\_columns <- c("age", "sibsp", "parch", "survived")

selected\_data <- mydata[selected\_columns]

means <- sapply(selected\_data, mean, na.rm = TRUE)

medians <- sapply(selected\_data, median, na.rm = TRUE)

mode\_func <- function(x) {

ux <- unique(x)

ux[which.max(tabulate(match(x, ux)))]

}

modes <- sapply(selected\_data, mode\_func)

summary\_data <- data.frame(Columns = names(selected\_data),Mean = means,Median = medians,Mode = modes)

print(summary\_data)

barplot(t(summary\_data[, -1]), beside = TRUE, main = "Mean, Median, and Mode for numeric columns",xlab = "Columns", ylab = "Values",col = c("red", "green", "blue"),legend.text = TRUE)

legend("topright", legend = colnames(summary\_data)[-1], fill = c("red", "green", "blue"))

**Output:**

A black text on a white background

Description automatically generated

**A white background with black text

Description automatically generated**

**Description:**

This code computes and visualizes the mean, median, and mode for numeric columns ("age", "sibsp", "parch", "survived") in mydata. It first calculates these measures using sapply() and custom functions for mode calculation (mode\_func). The summary data (summary\_data) is then displayed, showing columns along with their respective mean, median, and mode values. Finally, a bar plot visualizes these statistics side by side, providing a comparative view across columns with legends for clarity.

**39.Processing Fare Attribute:**

**Code:**

str(mydata$fare)

mydata$fare <- as.numeric(mydata$fare)

mean\_fare <- as.integer(mean(mydata$fare, na.rm = TRUE))

print(mean\_fare)

**Output:**





**Description:**

This code snippet first inspects the structure of the fare attribute in mydata using str() to understand its current type and structure. Next, it converts fare to numeric format using as.numeric() to facilitate numerical operations. Lastly, it computes the mean fare (mean\_fare) from mydata, ignoring missing values (na.rm = TRUE), and prints the result as an integer.

**40.Replace Missing Values with Mean(Fare)**

**Code:**

mydata <- mydata %>%

mutate(fare = ifelse(is.na(fare), mean\_fare, fare))

mydata

**Output:** A screenshot of a computer screen

Description automatically generated

**Description:**

This code snippet utilizes the mutate() function from dplyr to replace missing values in the fare column of mydata with the previously computed mean fare (mean\_fare). The updated dataset (mydata) is then displayed to show the changes made.

**41.Rounding Operations on Fare Data**

**Code:**

round(mydata$fare)

ceiling(mydata$fare)

floor(mydata$fare)

**Output:** **A number grid with numbers

Description automatically generated with medium confidence**

**Description:**

This code snippet demonstrates the application of rounding operations (round(), ceiling(), and floor()) on the fare column of mydata.

* round(mydata$fare) rounds each fare value to the nearest integer.
* ceiling(mydata$fare) rounds each fare value up to the nearest integer.
* floor(mydata$fare) rounds each fare value down to the nearest integer.

**42.Generating boxplot for numerical column**

**Code:**

numerical\_columns <- sapply(mydata, is.numeric)

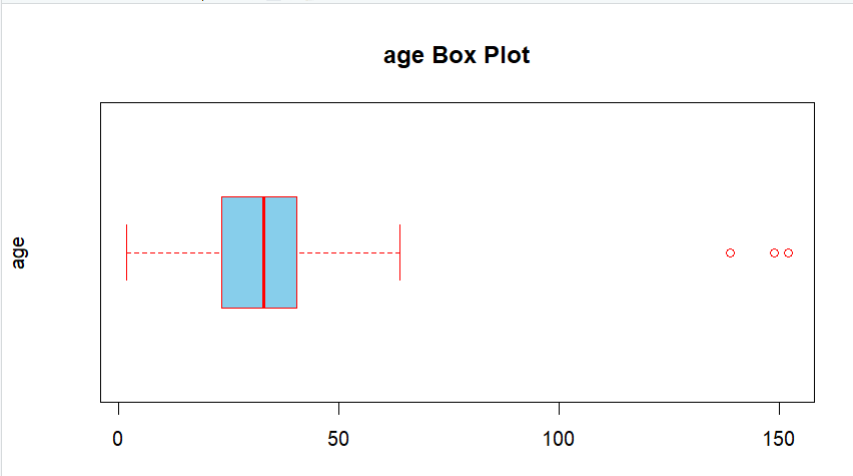
for (col in names(mydata)[numerical\_columns]) {

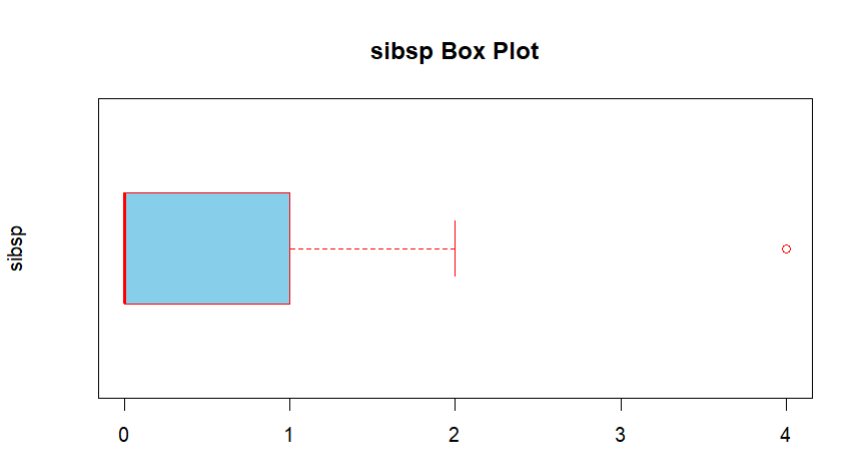
boxplot(mydata[[col]], main = paste(col, "Box Plot"), ylab = col, col = "skyblue",

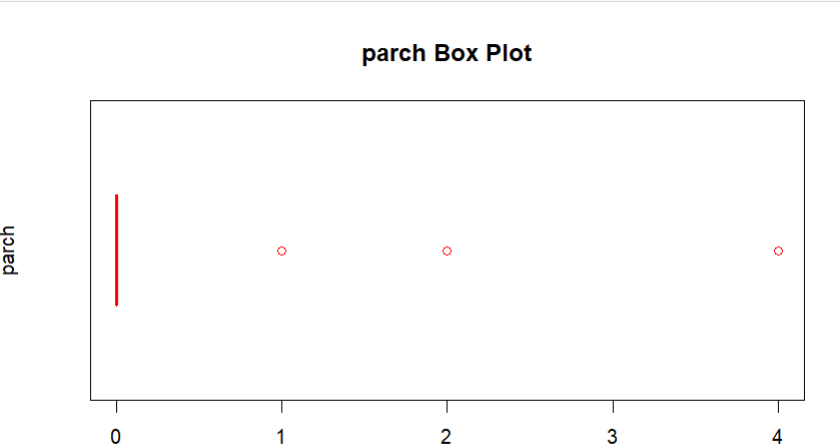
border = "red", notch = FALSE, horizontal = TRUE)

}

**Output:**

****

****

****

**A graph with a blue line

Description automatically generated**

**Description:**

This code generates a horizontal boxplot for each numeric column in the mydata data frame. It first identifies which columns are numeric using sapply. Then, it iterates over these columns, creating a boxplot with specific styling options such as sky blue color and red borders. Each boxplot is titled with the column name followed by "Box Plot".

**43.Balancing the “Survived ” column in the dataset**

**Code:**

survived\_distribution <- mydata %>% count(survived)

print(survived\_distribution)

majority\_survived\_label <- survived\_distribution %>% filter(n == max(n)) %>% pull(survived)

minority\_survived\_label <- survived\_distribution %>% filter(n == min(n)) %>% pull(survived)

print(paste("Majority class:", majority\_survived\_label))

print(paste("Minority class:", minority\_survived\_label))

majority\_survived <- mydata %>% filter(survived == majority\_survived\_label)

minority\_survived <- mydata %>% filter(survived == minority\_survived\_label)

set.seed(123)

undersampled\_majority <- majority\_survived %>% sample\_n(nrow(minority\_survived))

balanced\_dataset <- bind\_rows(undersampled\_majority, minority\_survived)

balanced\_dataset

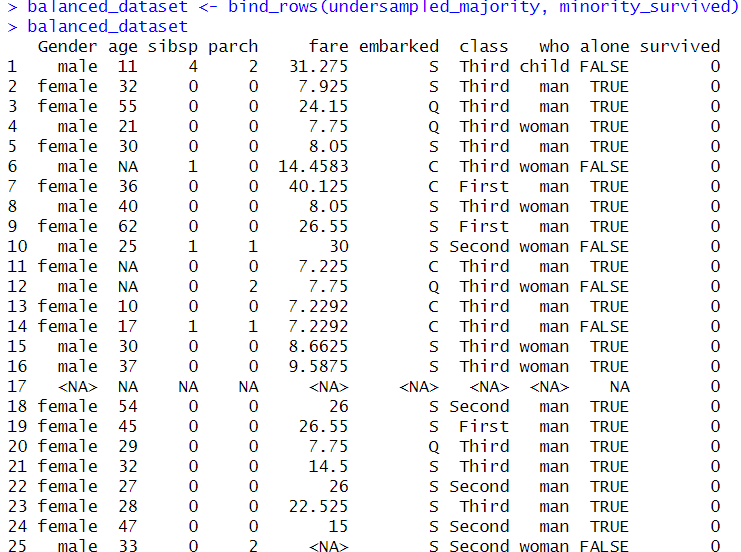
balanced\_distribution <- balanced\_dataset %>%

count(survived) %>%

mutate(percentage = n / sum(n) \* 100)

print(balanced\_distribution)

**Output:**

****

**A screenshot of a computer

Description automatically generated**

**Description:**

This code balances the 'survived' column in a dataset by undersampling the majority class. It first determines the distribution of the 'survived' column and identifies the majority and minority classes. Then, it undersamples the majority class to match the size of the minority class and combines them to create a balanced dataset. Finally, it checks and prints the new distribution to confirm the balance.

**44.Balancing the “Gender ” column in the dataset**

**Code:**

Gender\_distribution <- mydata %>% count(Gender)

print(Gender\_distribution)

majority\_Gender\_label <- Gender\_distribution %>% filter(n == max(n)) %>% pull(Gender)

minority\_Gender\_label <- Gender\_distribution %>% filter(n == min(n)) %>% pull(Gender)

print(paste("Majority class:", majority\_Gender\_label))

print(paste("Minority class:", minority\_Gender\_label))

majority\_Gender <- mydata %>% filter(Gender == majority\_Gender\_label)

minority\_Gender <- mydata %>% filter(Gender == minority\_Gender\_label)

set.seed(123)

undersampled\_majority <- majority\_Gender %>% sample\_n(nrow(minority\_Gender))

balanced\_dataset <- bind\_rows(undersampled\_majority, minority\_Gender)

balanced\_dataset

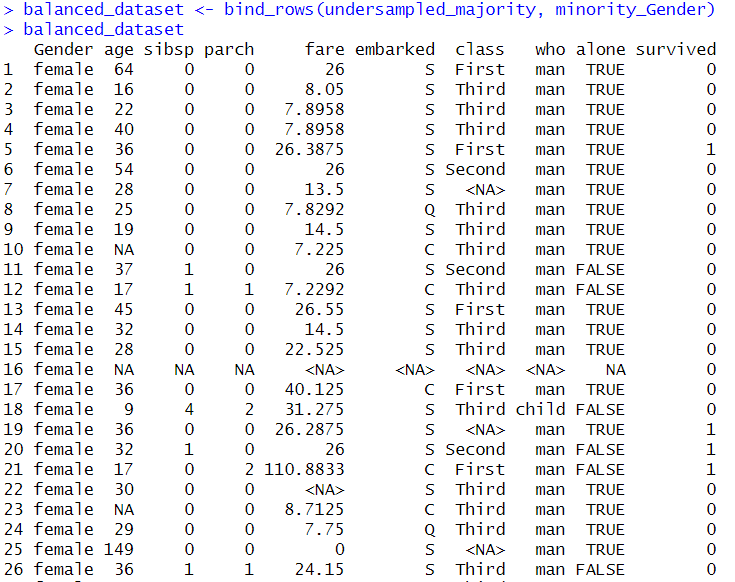
balanced\_distribution <- balanced\_dataset %>%

count(Gender) %>%

mutate(percentage = n / sum(n) \* 100)

print(balanced\_distribution)

**Output:**

****

**A screen shot of a computer

Description automatically generated**

**Description:**

This code balances the 'Gender' column in a dataset by undersampling the majority gender class. It identifies the majority and minority gender classes and then undersamples the majority class to match the size of the minority class. The undersampled majority class is combined with the minority class to create a balanced dataset. Finally, it checks and prints the new distribution to ensure balance.

**45.Balancing the “Alone ” column in the dataset**

**Code:**

alone\_distribution <- mydata %>% count(alone)

print(alone\_distribution)

majority\_alone\_label <- alone\_distribution %>% filter(n == max(n)) %>% pull(alone)

minority\_alone\_label <- alone\_distribution %>% filter(n == min(n)) %>% pull(alone)

print(paste("Majority class:", majority\_alone\_label))

print(paste("Minority class:", minority\_alone\_label))

majority\_alone <- mydata %>% filter(alone == majority\_alone\_label)

minority\_alone <- mydata %>% filter(alone == minority\_alone\_label)

set.seed(123)

undersampled\_majority <- majority\_alone %>% sample\_n(nrow(minority\_alone))

balanced\_dataset <- bind\_rows(undersampled\_majority, minority\_alone)

balanced\_dataset

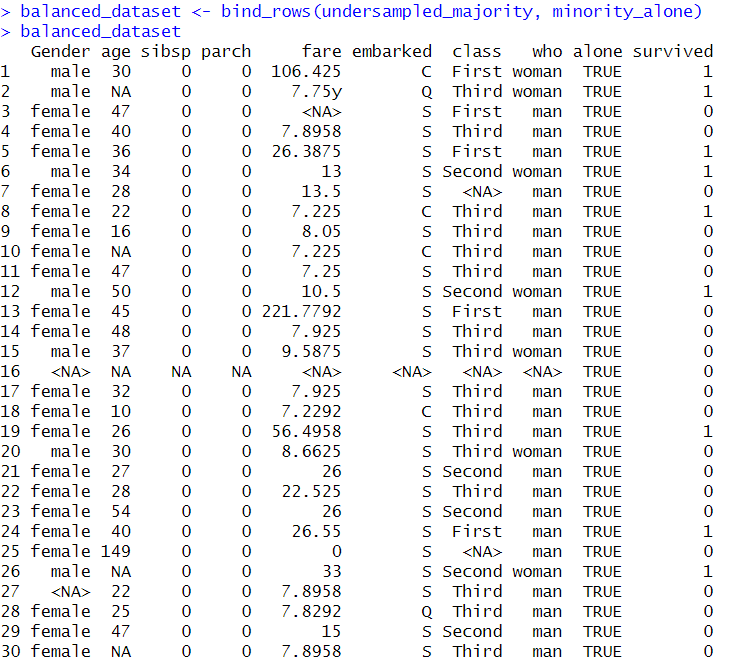
balanced\_distribution <- balanced\_dataset %>%

count(alone) %>%

mutate(percentage = n / sum(n) \* 100)

print(balanced\_distribution)

**Output:**



A white background with black text

Description automatically generated

**Description:**

This code balances the 'alone' column in a dataset by undersampling the majority class. It first determines the distribution of the 'alone' column and identifies the majority and minority classes. The majority class is then undersampled to match the size of the minority class, and both classes are combined to create a balanced dataset. Finally, it checks and prints the new distribution to confirm the balance.