4/26/2025

Introduction To Data Science Project

Section A

Group7

Group 7

Name	ID
1) Md Hasib Sarkar	22-46449-1
2) Mimjamam Ul Haque Monmoy	22-46476-1
3) Shraboni Biswas Naboni	22-47701-2
4) Ujjoyeni Dey	22-49001-3

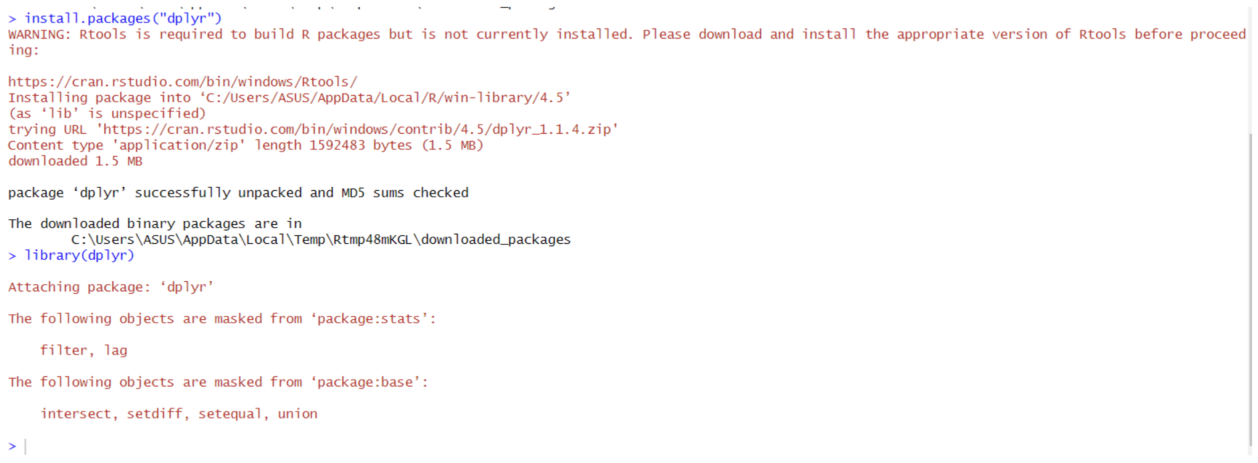
Project Dataset Description

Our project has used the Cleveland Heart Disease dataset, which contains information about 216 patients. The Cleveland dataset contains 303 original instances, which use 76 attributes, but researchers typically analysed a focused subset of 14 important characteristics. The attributes needed for heart disease assessment include demographic statistics, clinical measurement data, and diagnostic assessment outcomes. The essential attributes measure patient age in years and gender status represented as male by 1 and female by 0. The classification of chest pain differentiates between various types of symptoms faced by patients. Patients receive blood pressure tests in mm Hg during their first hospital check-up and cholesterol screenings in mg/dl before the test begins. The variable represents blood sugar status before a meal and takes two values to indicate if the reading exceeds 120 mg/dl. The maximum heart rate measurement occurs during exercise, while the ECG test provides results about resting electrocardiographic findings. When physical activities cause angina attacks, it is categorized as exercise-induced angina. The measurements of ST depression occur in relation to resting levels to show exercise-caused alterations. The ST segment slope receives evaluation when exercise reaches its peak. Fluoroscopy determines the number of major vessels, while thallium stress tests deliver the thalassemia results. Notes the final data category known as the target or 'num' to represent heart disease detection, which returns results as zero for no heart disease but one for a diagnosis. This prepared dataset creates the basis for heart disease diagnosis through predictive models and statistical analysis techniques.

Installing Necessary Packages:

```
install.packages("dplyr")
library(dplyr)
```

dplyr was used for data manipulation.



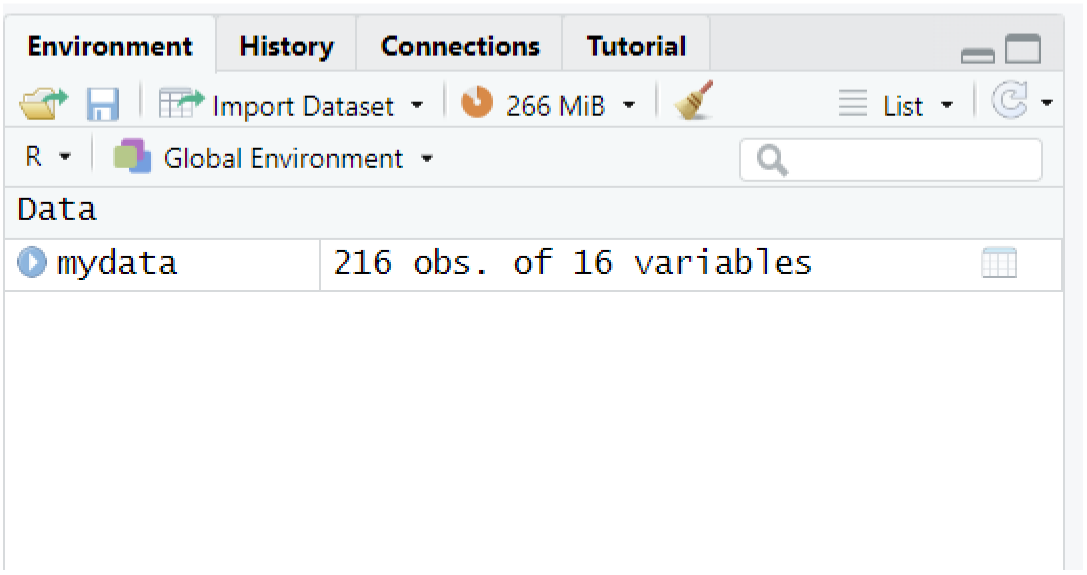
```
> install.packages("dplyr")
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropriate version of Rtools before proceed
ing:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into ‘C:/Users/ASUS/AppData/Local/R/win-library/4.5’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.5/dplyr_1.1.4.zip'
Content type 'application/zip' length 1592483 bytes (1.5 MB)
downloaded 1.5 MB

package ‘dplyr’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\ASUS\AppData\Local\Temp\Rtmp48mKGL\downloaded_packages
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

> 
```

dplyr package was successfully loaded.

Loading The Dataset :

```
mydata<-read.csv("E:/AIUB ( Study )/8th Semester/Introduction To Data Science/
                     Mid/Project/heart_disease_uci - modified.csv",header = TRUE, sep = ",", na.strings = c("?", "", "NA"))
```

This line reads the heart disease dataset from a specified file path using the read.csv() function. It sets the first row as column headers (header = TRUE), uses commas to separate values (sep = ","), and treats "?", empty strings, and "NA" as missing values (na.strings).

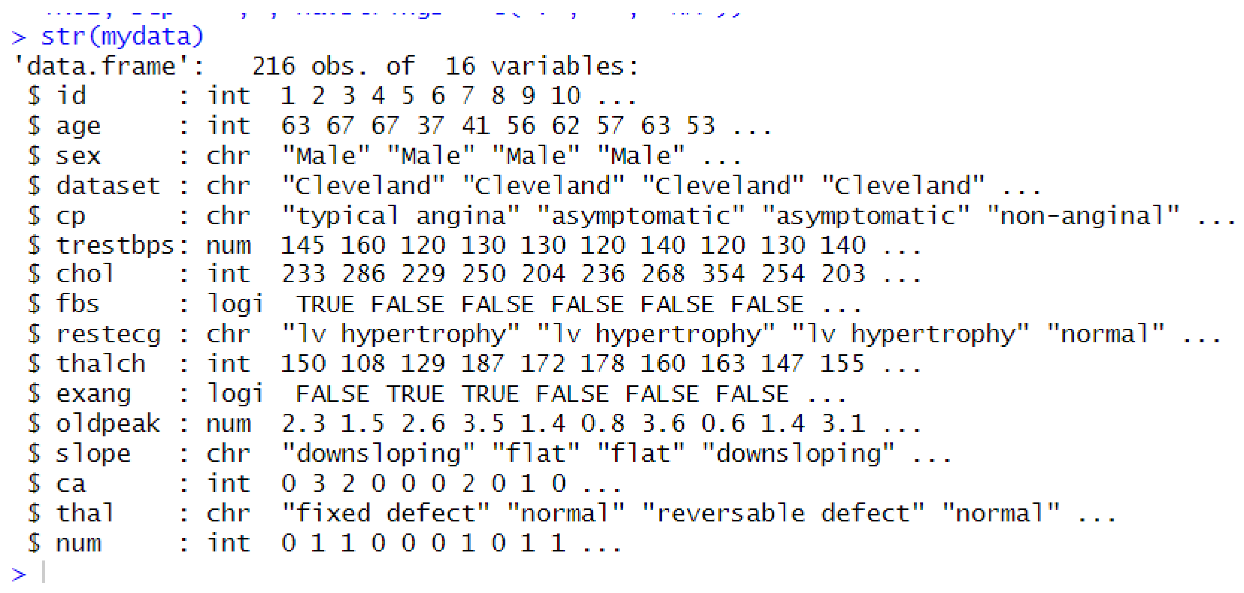


```
Data
mydata          216 obs. of 16 variables
```

No visible output is shown in the console because the dataset is assigned to the variable mydata. However, the data is successfully loaded into RStudio and ready for further analysis.

```
str(mydata)
```

It displays the structure of the dataset. It shows the number of observations and variables, along with each variable's name, data type, and a preview of its values.

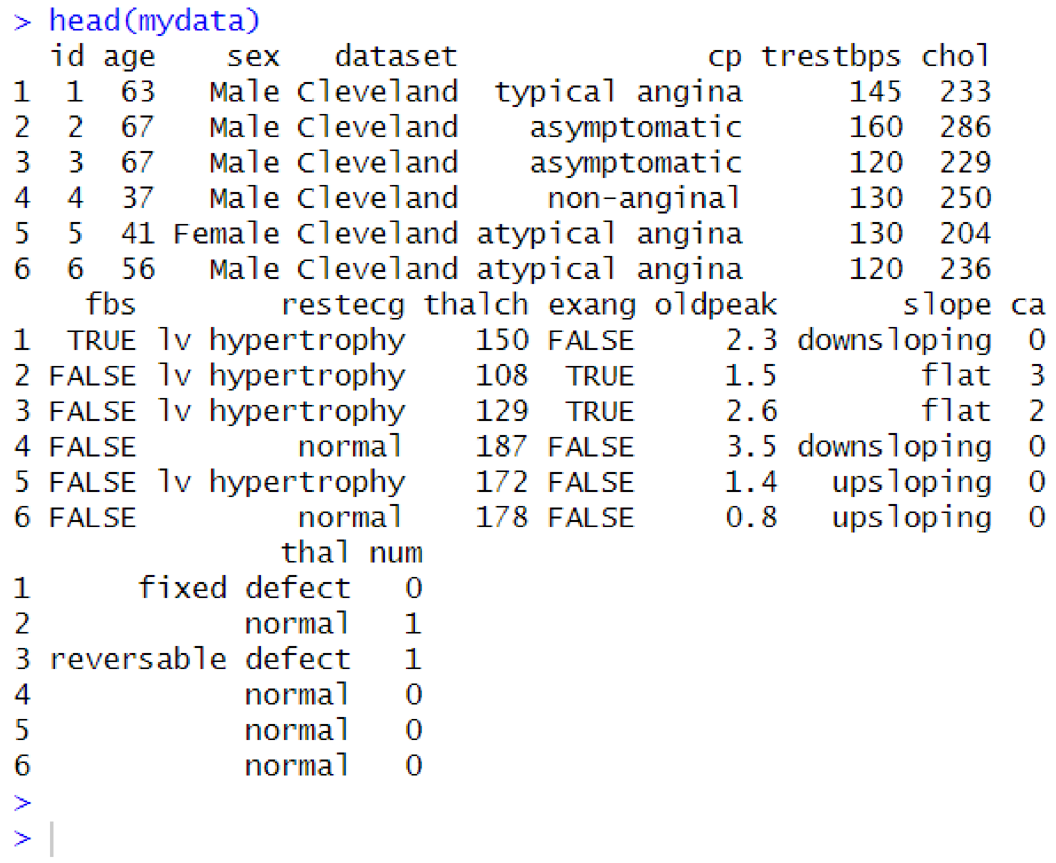


```
> str(mydata)
'data.frame':	216 obs. of  16 variables:
 $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ age     : int  63 67 67 37 41 56 62 57 63 53 ...
 $ sex     : chr  "Male" "Male" "Male" "Male" ...
 $ dataset : chr  "Cleveland" "Cleveland" "Cleveland" "Cleveland" ...
 $ cp      : chr  "typical angina" "asymptomatic" "asymptomatic" "non-anginal" ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : int  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : logi  TRUE FALSE FALSE FALSE FALSE FALSE ...
 $ restecg : chr  "lv hypertrophy" "lv hypertrophy" "lv hypertrophy" "normal" ...
 $ thalch  : int  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : logi  FALSE TRUE TRUE FALSE FALSE FALSE ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ slope   : chr  "downsloping" "flat" "flat" "downsloping" ...
 $ ca      : int  0 3 2 0 0 0 2 0 1 0 ...
 $ thal    : chr  "fixed defect" "normal" "reversable defect" "normal" ...
 $ num     : int  0 1 1 0 0 0 1 0 1 1 ...
> |
```

The output of str(mydata) displayed that the dataset contains 216 observations and 16 variables such as age, sex, chol, and thal etc. It showed the data types (e.g., numeric, character, logical) and sample values for each column, confirming that the data was loaded and structured correctly for further processing.

```
head(mydata)
```

This function is used to display the first six rows of the dataset. This helps verify that the data was loaded correctly.

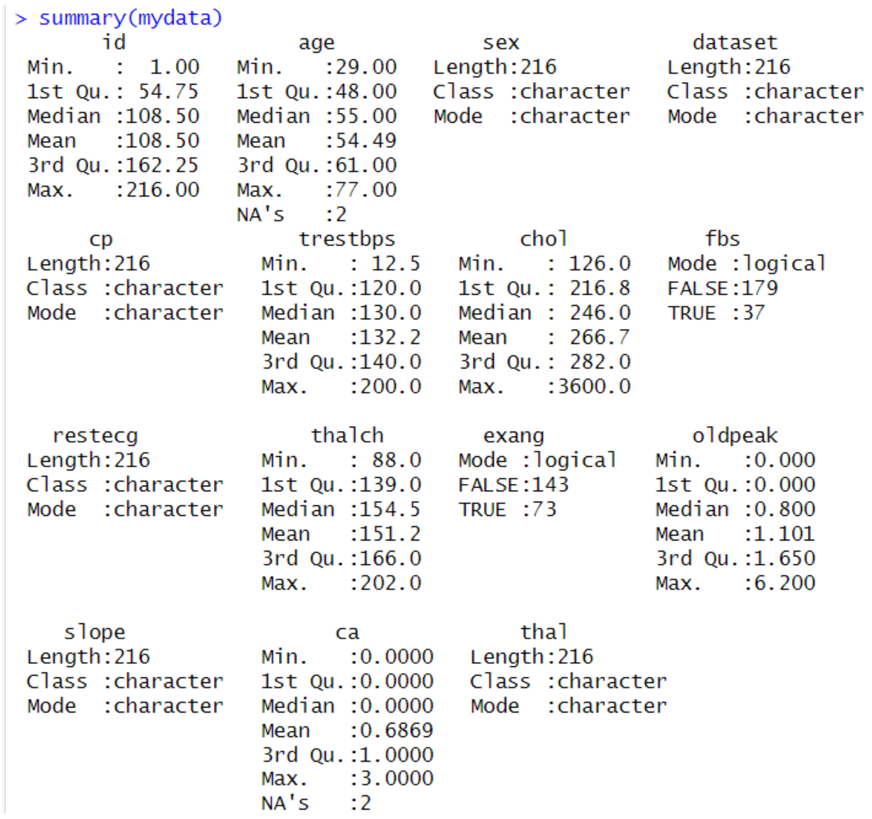


```
> head(mydata)
  id age    sex   dataset              cp trestbps chol
1  1  63   Male Cleveland  typical angina      145  233
2  2  67   Male Cleveland    asymptomatic      160  286
3  3  67   Male Cleveland    asymptomatic      120  229
4  4  37   Male Cleveland     non-anginal      130  250
5  5  41 Female Cleveland atypical angina      130  204
6  6  56   Male Cleveland atypical angina      120  236
    fbs        restecg thalch exang oldpeak       slope ca
1  TRUE lv hypertrophy    150 FALSE     2.3 downsloping  0
2 FALSE lv hypertrophy    108  TRUE     1.5        flat  3
3 FALSE lv hypertrophy    129  TRUE     2.6        flat  2
4 FALSE         normal    187 FALSE     3.5 downsloping  0
5 FALSE lv hypertrophy    172 FALSE     1.4   upsloping  0
6 FALSE         normal    178 FALSE     0.8   upsloping  0
               thal num
1      fixed defect   0
2            normal   1
3 reversable defect   1
4            normal   0
5            normal   0
6            normal   0
>
> 
```

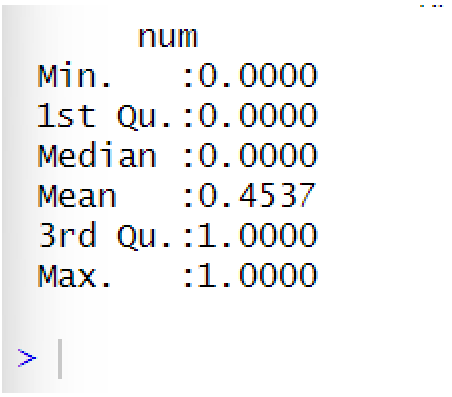
The output shows the top 6 rows of the dataset, including column names and sample values for each attribute. This confirms that the dataset was successfully imported and that the data types and structure are as expected.

```
summary(mydata)
```

This function is used to get a quick statistical overview of the entire dataset. It provides key summary statistics for each column — including minimum, maximum, mean, median, and quartiles for numeric variables, and frequencies for categorical variables. This helps identify missing values, detect outliers, and understand the basic structure of the data.



```
> summary(mydata)
       id              age            sex              dataset         
 Min.   :  1.00   Min.   :29.00   Length:216         Length:216        
 1st Qu.: 54.75   1st Qu.:48.00   Class :character   Class :character  
 Median :108.50   Median :55.00   Mode  :character   Mode  :character  
 Mean   :108.50   Mean   :54.49                                        
 3rd Qu.:162.25   3rd Qu.:61.00                                        
 Max.   :216.00   Max.   :77.00                                        
                  NA's   :2                                            
      cp               trestbps          chol           fbs         
 Length:216         Min.   : 12.5   Min.   : 126.0   Mode :logical  
 Class :character   1st Qu.:120.0   1st Qu.: 216.8   FALSE:179      
 Mode  :character   Median :130.0   Median : 246.0   TRUE :37       
                    Mean   :132.2   Mean   : 266.7                  
                    3rd Qu.:140.0   3rd Qu.: 282.0                  
                    Max.   :200.0   Max.   :3600.0                  
                                                                    
   restecg              thalch          exang           oldpeak     
 Length:216         Min.   : 88.0   Mode :logical   Min.   :0.000  
 Class :character   1st Qu.:139.0   FALSE:143       1st Qu.:0.000  
 Mode  :character   Median :154.5   TRUE :73        Median :0.800  
                    Mean   :151.2                   Mean   :1.101  
                    3rd Qu.:166.0                   3rd Qu.:1.650  
                    Max.   :202.0                   Max.   :6.200  
                                                                   
    slope                 ca              thal          
 Length:216         Min.   :0.0000   Length:216        
 Class :character   1st Qu.:0.0000   Class :character  
 Mode  :character   Median :0.0000   Mode  :character  
                    Mean   :0.6869                     
                    3rd Qu.:1.0000                     
                    Max.   :3.0000                     
                    NA's   :2                          
```

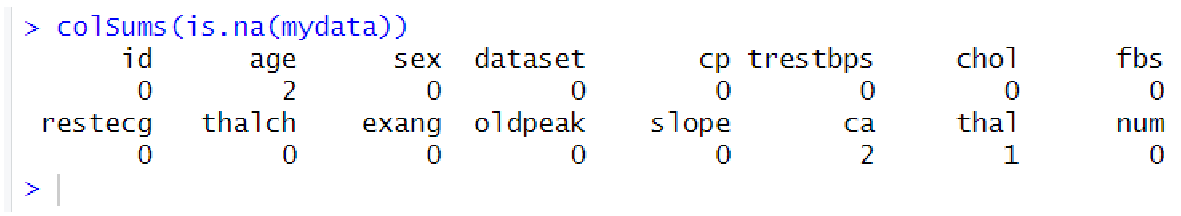


```
      num        
 Min.   :0.0000  
 1st Qu.:0.0000  
 Median :0.0000  
 Mean   :0.4537  
 3rd Qu.:1.0000  
 Max.   :1.0000  

> 
```

The output displays summary statistics for each column in the dataset. For numeric columns, it shows values like Min, 1st Qu., Median, Mean, 3rd Qu., and Max. For categorical or logical columns, it shows the count of each category and how many NA values are present.

```
colSums(is.na(mydata))
```

It is used to check for missing (NA) values in a data frame or matrix (mydata).

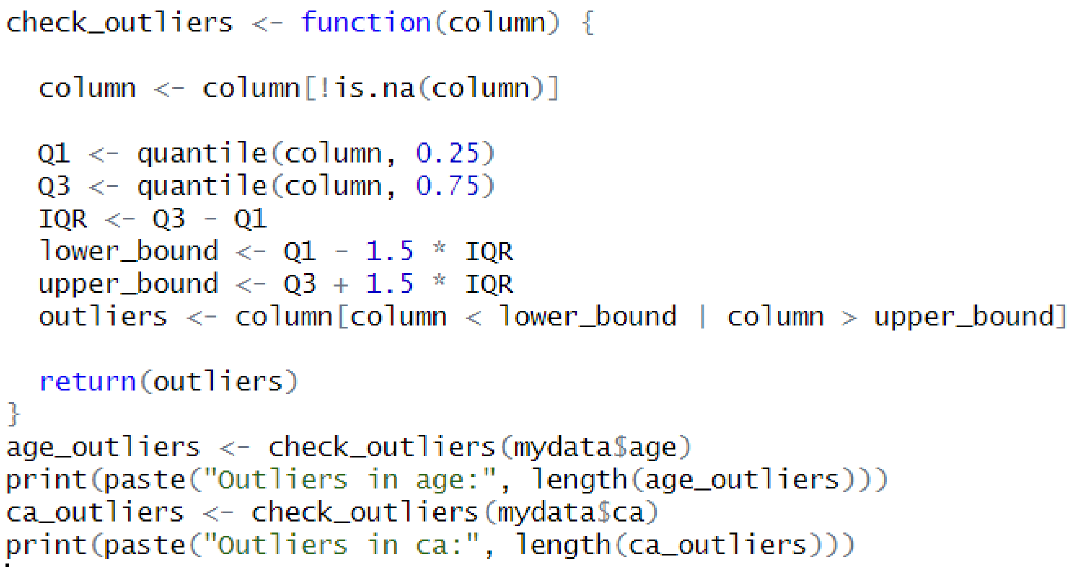


```
> colSums(is.na(mydata))
      id      age      sex  dataset       cp trestbps     chol      fbs 
       0        2        0        0        0        0        0        0 
 restecg   thalch    exang  oldpeak    slope       ca     thal      num 
       0        0        0        0        0        2        1        0 
> 
```

The output is a vector where each element corresponds to a column in the data frame or matrix mydata. The values represent the total count of missing (NA) values in each respective column. Here, age has 2, ca has 2, and thal has 1 missing value.

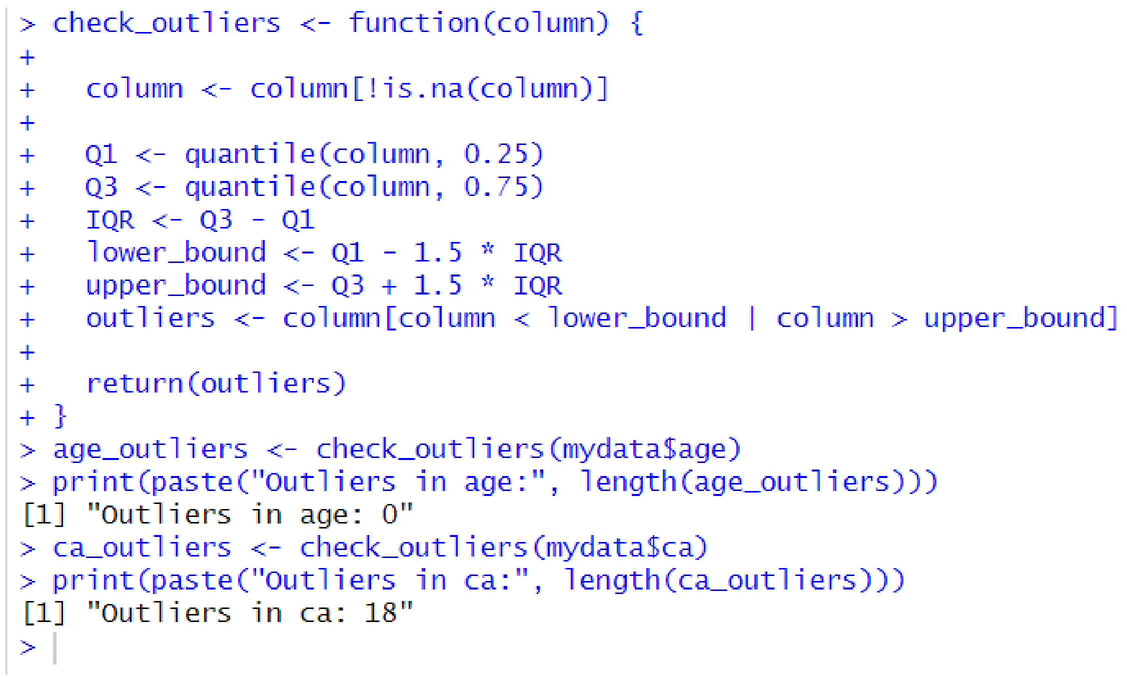
Handling Missing Values:

Checking Outliers Using the IQR Method :



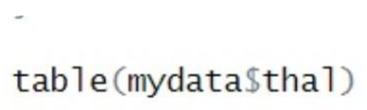
```
check_outliers <- function(column) {

  column <- column[!is.na(column)]

  Q1 <- quantile(column, 0.25)
  Q3 <- quantile(column, 0.75)
  IQR <- Q3 - Q1
  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR
  outliers <- column[column < lower_bound | column > upper_bound]

  return(outliers)
}
age_outliers <- check_outliers(mydata$age)
print(paste("Outliers in age:", length(age_outliers)))
ca_outliers <- check_outliers(mydata$ca)
print(paste("Outliers in ca:", length(ca_outliers)))
```

The check_outliers function detects outliers in a given numeric column by first removing missing values (NA). It then calculates the first (Q1) and third (Q3) quartiles, the interquartile range (IQR), and the lower and upper bounds for outliers (1.5 times the IQR below Q1 or above Q3). The function returns the values that fall outside these bounds. The outliers for the age and ca columns of mydata are then calculated and printed, showing the count of outliers in each.



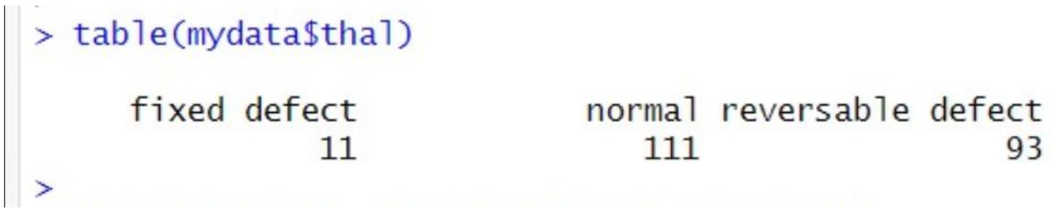
```
> check_outliers <- function(column) {
+ 
+   column <- column[!is.na(column)]
+ 
+   Q1 <- quantile(column, 0.25)
+   Q3 <- quantile(column, 0.75)
+   IQR <- Q3 - Q1
+   lower_bound <- Q1 - 1.5 * IQR
+   upper_bound <- Q3 + 1.5 * IQR
+   outliers <- column[column < lower_bound | column > upper_bound]
+ 
+   return(outliers)
+ }
> age_outliers <- check_outliers(mydata$age)
> print(paste("Outliers in age:", length(age_outliers)))
[1] "Outliers in age: 0"
> ca_outliers <- check_outliers(mydata$ca)
> print(paste("Outliers in ca:", length(ca_outliers)))
[1] "Outliers in ca: 18"
> 
```

The output displays the number of outliers in the age and ca columns, which are 0 and 18 respectively. It shows the total count of outlier values detected based on the IQR method for each respective column.



```
table(mydata$thal)
```

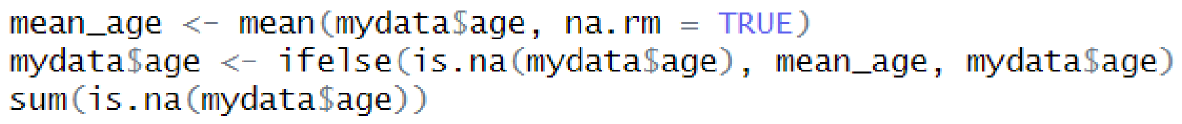
The table(mydata$thal) command was executed to count the occurrences of each unique value in the thal column. By using the table() function, the dataset was automatically scanned, and the frequency of each category was calculated.



```
> table(mydata$thal)

   fixed defect            normal reversable defect 
             11               111                93 
> 
```

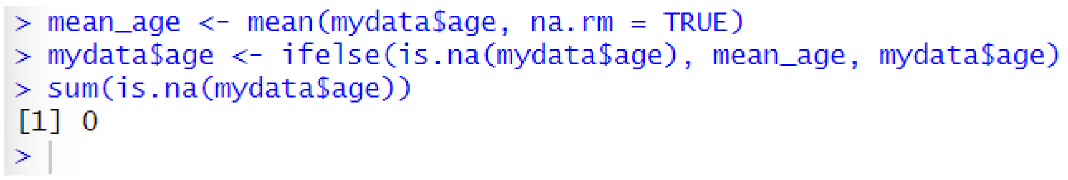
Here "normal" appeared 111 times, making it the most frequent category. "Reversable defect" was recorded 93 times, while "fixed defect" occurred only 11 times. Based on this frequency distribution, it was decided that missing values in the thal column should be replaced with "normal," since it was the most common and representative category in the data.

Replace Missing Values :



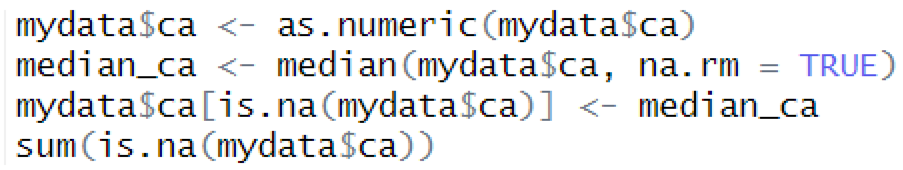
```
mean_age <- mean(mydata$age, na.rm = TRUE)
mydata$age <- ifelse(is.na(mydata$age), mean_age, mydata$age)
sum(is.na(mydata$age))
```

The code calculates the mean of the age column, excluding any missing values (NA), using mean(mydata$age, na.rm = TRUE). Then, it uses the ifelse function to replace any missing values (NA) in the age column with the calculated mean age.



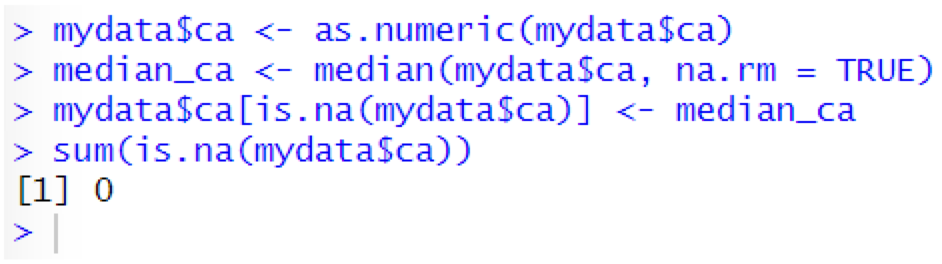
```
> mean_age <- mean(mydata$age, na.rm = TRUE)
> mydata$age <- ifelse(is.na(mydata$age), mean_age, mydata$age)
> sum(is.na(mydata$age))
[1] 0
> |
```

The output does not produce a direct display but modifies the age column of mydata, replacing any missing values (NA) with the computed mean age, ensuring that the age column no longer contains NA values.



```
mydata$ca <- as.numeric(mydata$ca)
median_ca <- median(mydata$ca, na.rm = TRUE)
mydata$ca[is.na(mydata$ca)] <- median_ca
sum(is.na(mydata$ca))
```

The code first converts the ca column in the mydata data frame to numeric format using as.numeric(mydata$ca). It then calculates the median of the ca column, excluding missing values (NA), using median(mydata$ca, na.rm = TRUE). The is.na(mydata$ca) function is used to identify and replace missing values in the ca column with the calculated median. Finally, the code calculates the total number of remaining missing values in the ca column using sum(is.na(mydata$ca)).

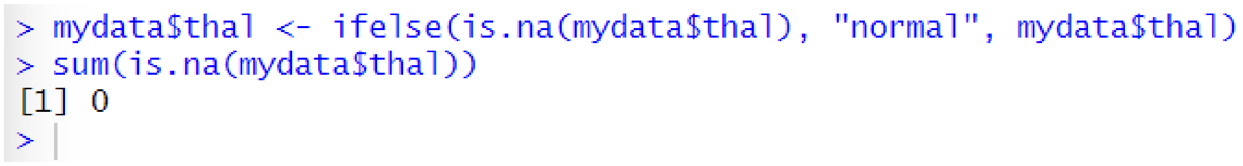


```
> mydata$ca <- as.numeric(mydata$ca)
> median_ca <- median(mydata$ca, na.rm = TRUE)
> mydata$ca[is.na(mydata$ca)] <- median_ca
> sum(is.na(mydata$ca))
[1] 0
> |
```

The output returns the total count of missing values (NA) left in the ca column after replacing the existing NAs with the median value. The output shows 0 since all missing values were successfully replaced.

```
mydata$thal <- ifelse(is.na(mydata$thal), "normal", mydata$thal)
sum(is.na(mydata$thal))
```

The code replaces missing values (NA) in the thal column of the mydata data frame with the string "normal" using the ifelse function. It checks for NA values in thal and substitutes them with "normal". The sum(is.na(mydata$thal)) function then calculates and returns the total number of remaining missing values in the thal column.



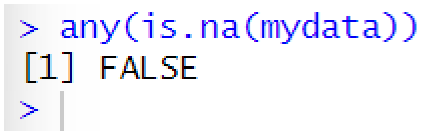
```
> mydata$thal <- ifelse(is.na(mydata$thal), "normal", mydata$thal)
> sum(is.na(mydata$thal))
[1] 0
> 
```

The output shows the total number of missing values (NA) left in the thal column after replacing the NAs with the string "normal". The output is 0 since all missing values were successfully replaced.

<u>Any Missing Values Or Not:</u>

```
any(is.na(mydata))
```

The code any(is.na(mydata)) checks if there are any missing values (NA) in the entire mydata data frame. It returns TRUE if at least one missing value is found, and FALSE if there are no missing values in the dataset.



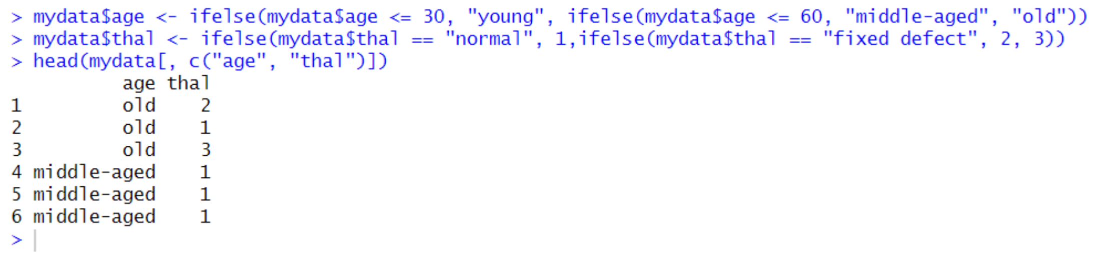
```
> any(is.na(mydata))
[1] FALSE
> 
```

FALSE means there are no missing values.

<u>Convert age and thal To Categorical Attributes:</u>

```
mydata$age <- ifelse(mydata$age <= 30, "young",
                     ifelse(mydata$age <= 60, "middle-aged", "old"))

mydata$thal <- ifelse(mydata$thal == "normal", 1,
                           ifelse(mydata$thal == "fixed defect", 2, 3))

head(mydata[, c("age", "thal")])
```

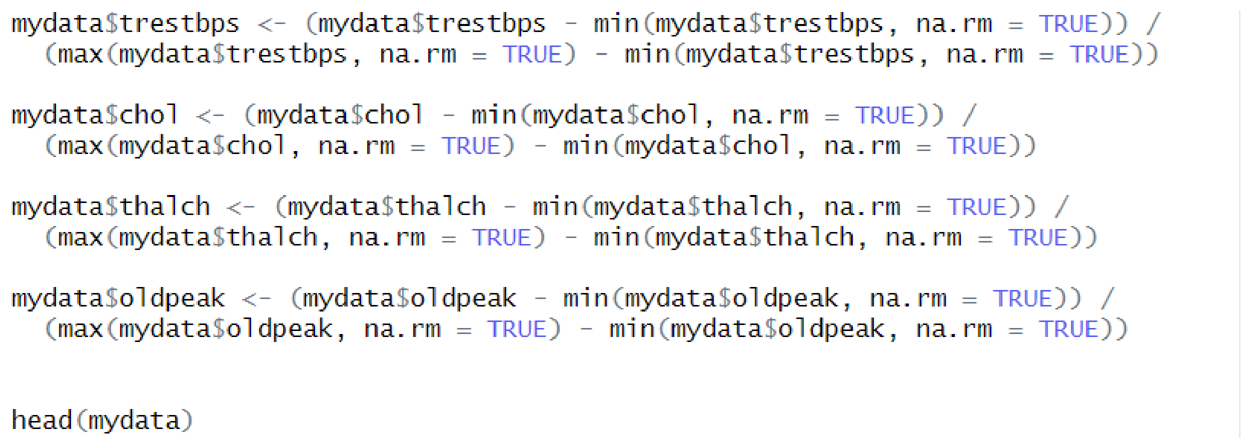
The code first categorizes the age column into three groups: "young" for ages less than or equal to 30, "middle-aged" for ages between 31 and 60, and "old" for ages above 60, using nested ifelse statements. Next, it converts the thal column from categorical values ("normal", "fixed defect", and others) into numeric values: 1 for "normal", 2 for "fixed defect", and 3 for other values. The head(mydata[, c("age", "thal")]) displays the first six rows of the age and thal columns after the transformations.



```
> mydata$age <- ifelse(mydata$age <= 30, "young", ifelse(mydata$age <= 60, "middle-aged", "old"))
> mydata$thal <- ifelse(mydata$thal == "normal", 1,ifelse(mydata$thal == "fixed defect", 2, 3))
> head(mydata[, c("age", "thal")])
          age thal
1         old    2
2         old    1
3         old    3
4 middle-aged    1
5 middle-aged    1
6 middle-aged    1
> 
```

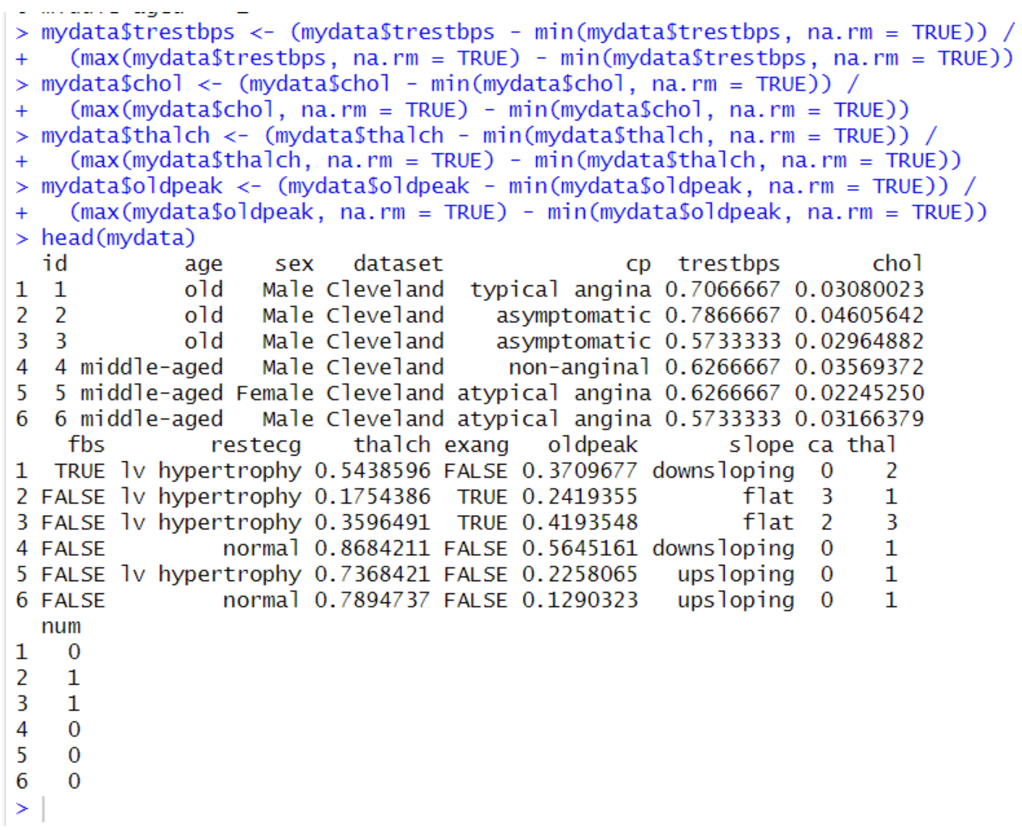
The output shows the first six rows of the age and thal columns, with age categorized as "young", "middle-aged", or "old", and thal converted into numeric values (1, 2, or 3) based on the original categories.

<u>Normalization for trestbps, chol,thalch :</u>



```
mydata$trestbps <- (mydata$trestbps - min(mydata$trestbps, na.rm = TRUE)) /
  (max(mydata$trestbps, na.rm = TRUE) - min(mydata$trestbps, na.rm = TRUE))

mydata$chol <- (mydata$chol - min(mydata$chol, na.rm = TRUE)) /
  (max(mydata$chol, na.rm = TRUE) - min(mydata$chol, na.rm = TRUE))

mydata$thalch <- (mydata$thalch - min(mydata$thalch, na.rm = TRUE)) /
  (max(mydata$thalch, na.rm = TRUE) - min(mydata$thalch, na.rm = TRUE))

mydata$oldpeak <- (mydata$oldpeak - min(mydata$oldpeak, na.rm = TRUE)) /
  (max(mydata$oldpeak, na.rm = TRUE) - min(mydata$oldpeak, na.rm = TRUE))


head(mydata)
```

The code normalizes the trestbps, chol, thalch, and oldpeak columns in the mydata data frame using Min-Max normalization. Each column is scaled so that its values range from 0 to 1 by subtracting the minimum value and dividing by the range (max - min). The na.rm = TRUE argument ensures that missing values are ignored during the calculations. Finally, head(mydata) displays the first six rows of the data frame after the normalization.



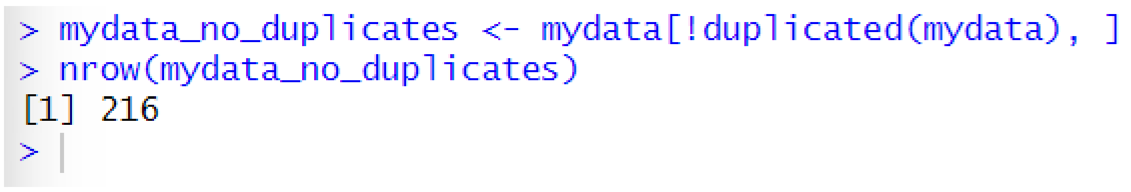
```
> mydata$trestbps <- (mydata$trestbps - min(mydata$trestbps, na.rm = TRUE)) /
+   (max(mydata$trestbps, na.rm = TRUE) - min(mydata$trestbps, na.rm = TRUE))
> mydata$chol <- (mydata$chol - min(mydata$chol, na.rm = TRUE)) /
+   (max(mydata$chol, na.rm = TRUE) - min(mydata$chol, na.rm = TRUE))
> mydata$thalch <- (mydata$thalch - min(mydata$thalch, na.rm = TRUE)) /
+   (max(mydata$thalch, na.rm = TRUE) - min(mydata$thalch, na.rm = TRUE))
> mydata$oldpeak <- (mydata$oldpeak - min(mydata$oldpeak, na.rm = TRUE)) /
+   (max(mydata$oldpeak, na.rm = TRUE) - min(mydata$oldpeak, na.rm = TRUE))
> head(mydata)
  id         age    sex  dataset              cp  trestbps       chol
1  1         old   Male Cleveland typical angina 0.7066667 0.03080023
2  2         old   Male Cleveland   asymptomatic 0.7866667 0.04605642
3  3         old   Male Cleveland   asymptomatic 0.5733333 0.02964882
4  4 middle-aged   Male Cleveland    non-anginal 0.6266667 0.03569372
5  5 middle-aged Female Cleveland atypical angina 0.6266667 0.02245250
6  6 middle-aged   Male Cleveland atypical angina 0.5733333 0.03166379
    fbs        restecg    thalch exang   oldpeak       slope ca thal
1  TRUE lv hypertrophy 0.5438596 FALSE 0.3709677 downsloping  0    2
2 FALSE lv hypertrophy 0.1754386  TRUE 0.2419355        flat  3    1
3 FALSE lv hypertrophy 0.3596491  TRUE 0.4193548        flat  2    3
4 FALSE         normal 0.8684211 FALSE 0.5645161 downsloping  0    1
5 FALSE lv hypertrophy 0.7368421 FALSE 0.2258065   upsloping  0    1
6 FALSE         normal 0.7894737 FALSE 0.1290323   upsloping  0    1
  num
1   0
2   1
3   1
4   0
5   0
6   0
> 
```

The output shows the first six rows of the mydata data frame, with the trestbps, chol, thalch, and oldpeak columns normalized to a 0-1 range. The values in these columns are now scaled according to their respective minimum and maximum values.

Remove Duplicate Rows :

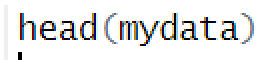
```
mydata_no_duplicates <- mydata[!duplicated(mydata), ]
nrow(mydata_no_duplicates)
```

The code removes duplicate rows from the mydata data frame using the duplicated() function, which identifies duplicated rows, and the negation (!) ensures that only non-duplicated rows are kept. The result is stored in mydata_no_duplicates. The function nrow(mydata_no_duplicates) is then used to check the number of rows in the data frame after duplicates have been removed.



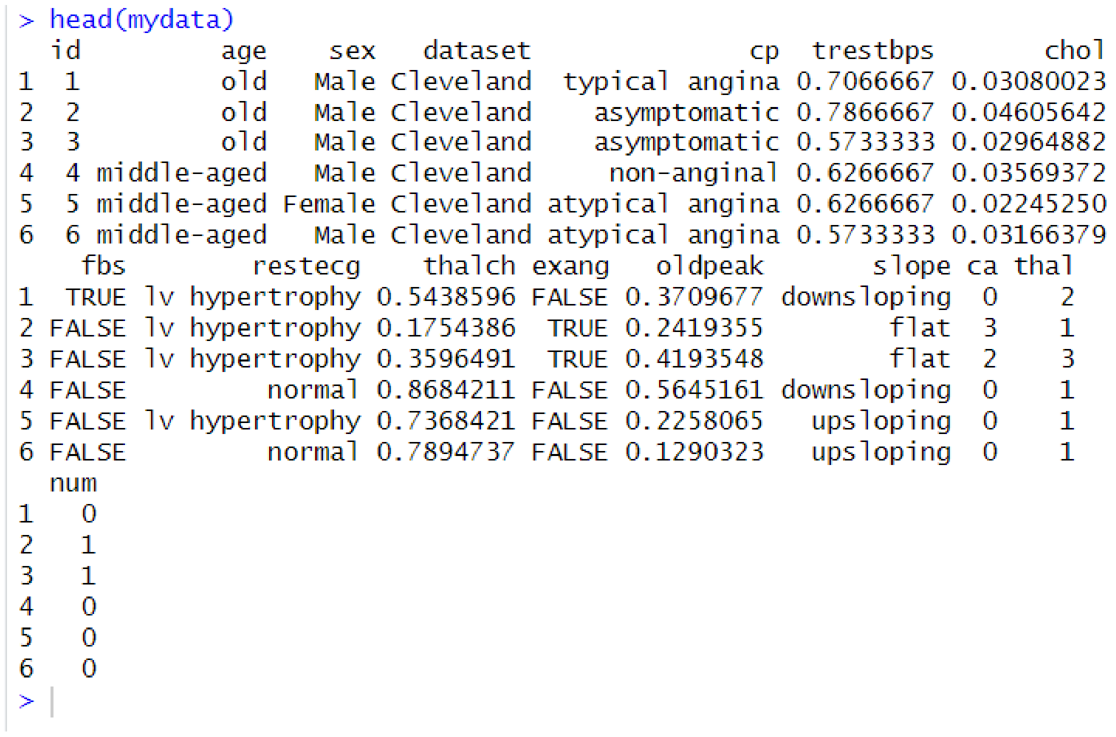
```
> mydata_no_duplicates <- mydata[!duplicated(mydata), ]
> nrow(mydata_no_duplicates)
[1] 216
> 
```

The output returns the number of rows in the mydata_no_duplicates data frame, indicating how many unique rows remain after removing duplicates from the original dataset.



```r
head(mydata)
```

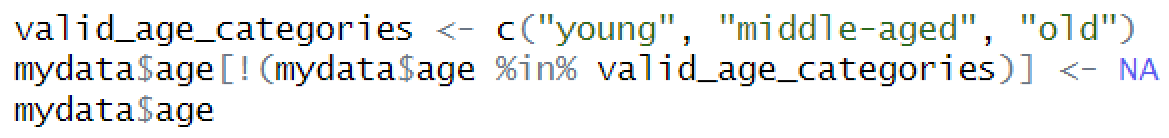
The code head(mydata) displays the first six rows of the mydata data frame. This function is commonly used to quickly inspect the structure and content of the dataset, showing the initial entries for all columns.



```
> head(mydata)
  id         age    sex   dataset              cp  trestbps       chol
1  1         old   Male Cleveland  typical angina 0.7066667 0.03080023
2  2         old   Male Cleveland    asymptomatic 0.7866667 0.04605642
3  3         old   Male Cleveland    asymptomatic 0.5733333 0.02964882
4  4 middle-aged   Male Cleveland     non-anginal 0.6266667 0.03569372
5  5 middle-aged Female Cleveland atypical angina 0.6266667 0.02245250
6  6 middle-aged   Male Cleveland atypical angina 0.5733333 0.03166379
    fbs        restecg    thalch exang   oldpeak       slope ca thal
1  TRUE lv hypertrophy 0.5438596 FALSE 0.3709677 downsloping  0    2
2 FALSE lv hypertrophy 0.1754386  TRUE 0.2419355        flat  3    1
3 FALSE lv hypertrophy 0.3596491  TRUE 0.4193548        flat  2    3
4 FALSE         normal 0.8684211 FALSE 0.5645161 downsloping  0    1
5 FALSE lv hypertrophy 0.7368421 FALSE 0.2258065   upsloping  0    1
6 FALSE         normal 0.7894737 FALSE 0.1290323   upsloping  0    1
  num
1   0
2   1
3   1
4   0
5   0
6   0
> 
```

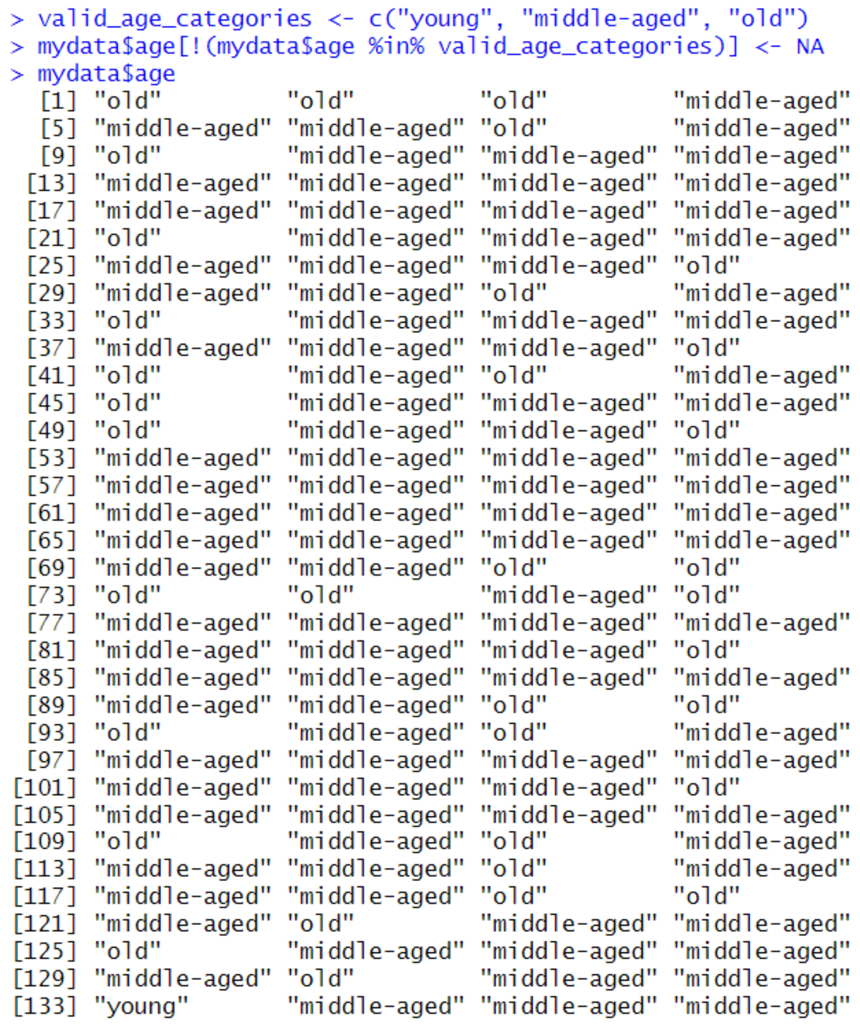
The output shows the first six rows of the mydata data frame, providing a preview of the data and the values in each column.

<u>Detecting Invalid Data :</u>

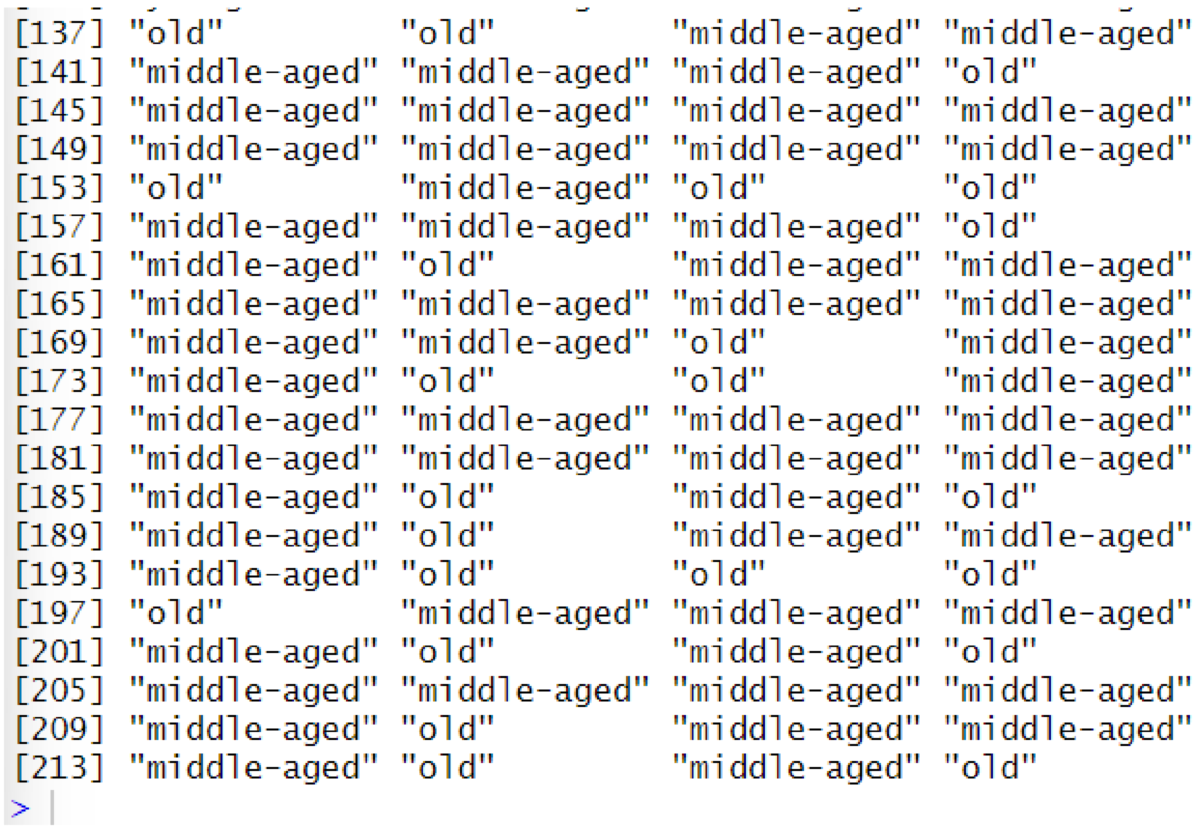


```r
valid_age_categories <- c("young", "middle-aged", "old")
mydata$age[!(mydata$age %in% valid_age_categories)] <- NA
mydata$age
```

The code defines a vector valid_age_categories containing the valid age categories: "young", "middle-aged", and "old". It then sets any value in the age column of mydata that is not one of these categories to NA using the %in% operator and negation (!). Finally, it displays the updated age column.

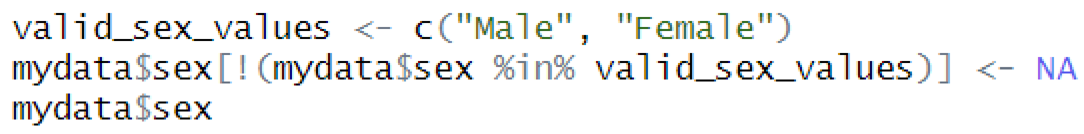


```
> valid_age_categories <- c("young", "middle-aged", "old")
> mydata$age[!(mydata$age %in% valid_age_categories)] <- NA
> mydata$age
  [1] "old"         "old"         "old"         "middle-aged"
  [5] "middle-aged" "middle-aged" "old"         "middle-aged"
  [9] "old"         "middle-aged" "middle-aged" "middle-aged"
 [13] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
 [17] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
 [21] "old"         "middle-aged" "middle-aged" "middle-aged"
 [25] "middle-aged" "middle-aged" "middle-aged" "old"
 [29] "middle-aged" "middle-aged" "old"         "middle-aged"
 [33] "old"         "middle-aged" "middle-aged" "middle-aged"
 [37] "middle-aged" "middle-aged" "middle-aged" "old"
 [41] "old"         "middle-aged" "old"         "middle-aged"
 [45] "old"         "middle-aged" "middle-aged" "middle-aged"
 [49] "old"         "middle-aged" "middle-aged" "old"
 [53] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
 [57] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
 [61] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
 [65] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
 [69] "middle-aged" "middle-aged" "old"         "old"
 [73] "old"         "old"         "middle-aged" "old"
 [77] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
 [81] "middle-aged" "middle-aged" "middle-aged" "old"
 [85] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
 [89] "middle-aged" "middle-aged" "old"         "old"
 [93] "old"         "middle-aged" "old"         "middle-aged"
 [97] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
[101] "middle-aged" "middle-aged" "middle-aged" "old"
[105] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
[109] "old"         "middle-aged" "old"         "middle-aged"
[113] "middle-aged" "middle-aged" "old"         "middle-aged"
[117] "middle-aged" "middle-aged" "old"         "old"
[121] "middle-aged" "old"         "middle-aged" "middle-aged"
[125] "old"         "middle-aged" "middle-aged" "middle-aged"
[129] "middle-aged" "old"         "middle-aged" "middle-aged"
[133] "young"       "middle-aged" "middle-aged" "middle-aged"
```



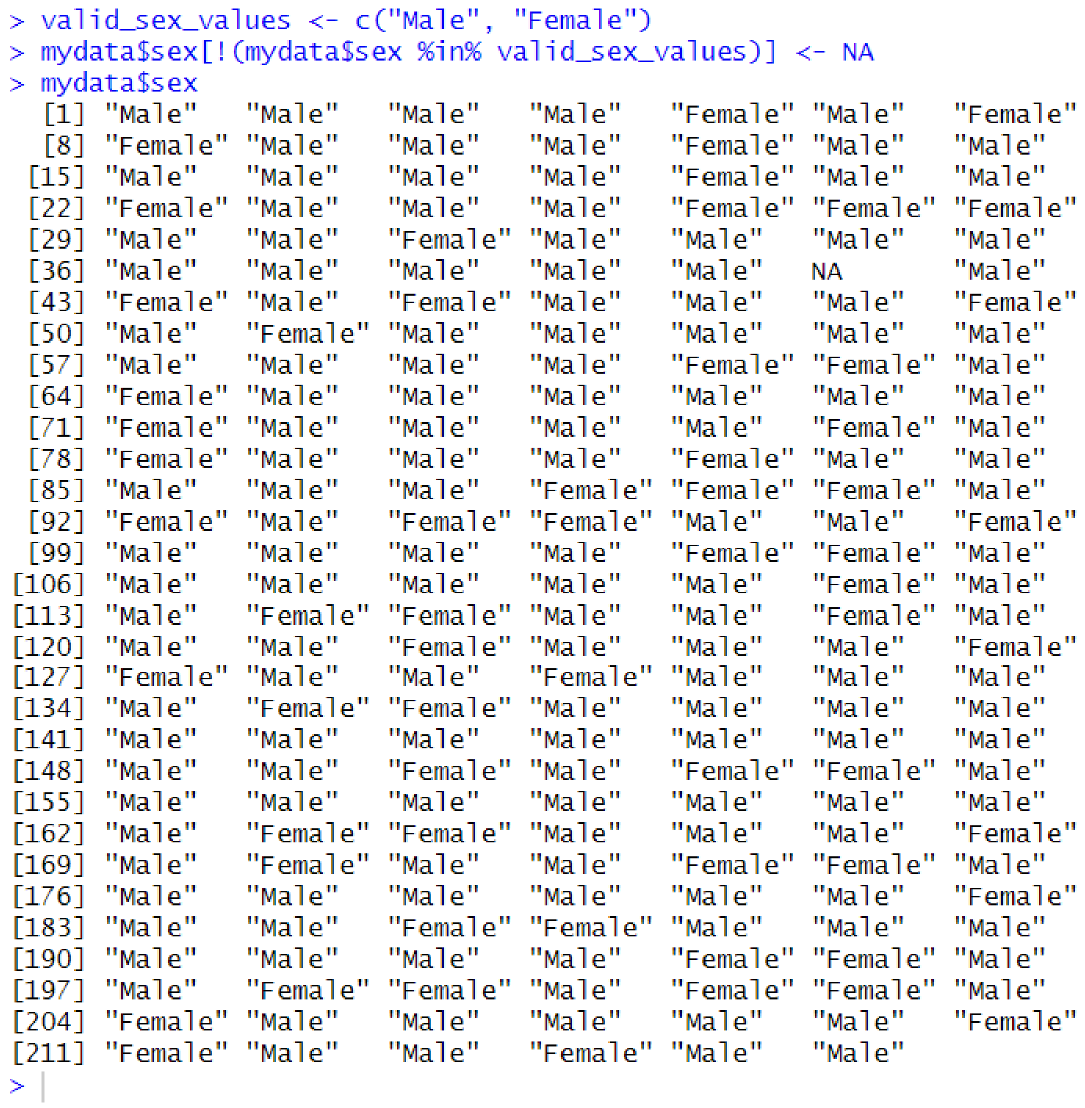
```
[137] "old"         "old"         "middle-aged" "middle-aged"
[141] "middle-aged" "middle-aged" "middle-aged" "old"
[145] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
[149] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
[153] "old"         "middle-aged" "old"         "old"
[157] "middle-aged" "middle-aged" "middle-aged" "old"
[161] "middle-aged" "old"         "middle-aged" "middle-aged"
[165] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
[169] "middle-aged" "middle-aged" "old"         "middle-aged"
[173] "middle-aged" "old"         "old"         "middle-aged"
[177] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
[181] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
[185] "middle-aged" "old"         "middle-aged" "old"
[189] "middle-aged" "old"         "middle-aged" "middle-aged"
[193] "middle-aged" "old"         "old"         "old"
[197] "old"         "middle-aged" "middle-aged" "middle-aged"
[201] "middle-aged" "old"         "middle-aged" "old"
[205] "middle-aged" "middle-aged" "middle-aged" "middle-aged"
[209] "middle-aged" "old"         "middle-aged" "middle-aged"
[213] "middle-aged" "old"         "middle-aged" "old"
>
```

The output shows the age column of the mydata data frame, where any values not belonging to the categories "young", "middle-aged", or "old" have been replaced with NA.



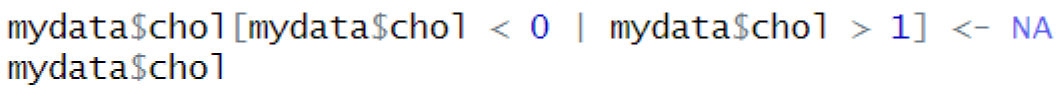
```
valid_sex_values <- c("Male", "Female")
mydata$sex[!(mydata$sex %in% valid_sex_values)] <- NA
mydata$sex
```

The code defines a vector valid_sex_values with the valid sex categories "Male" and "Female". It then sets any value in the sex column of mydata that is not one of these categories to NA. Finally, it displays the updated sex column.



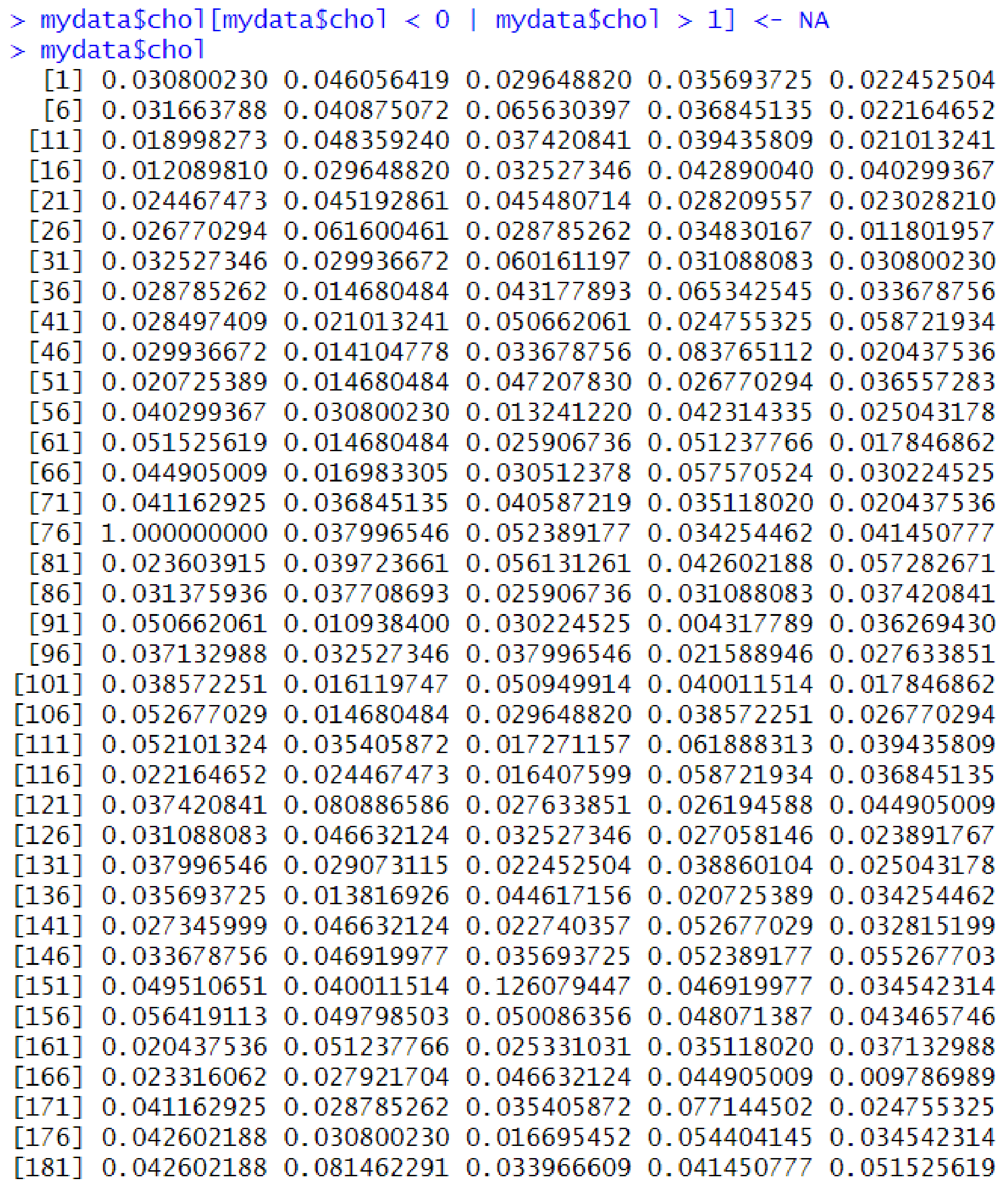
```
> valid_sex_values <- c("Male", "Female")
> mydata$sex[!(mydata$sex %in% valid_sex_values)] <- NA
> mydata$sex
  [1] "Male"   "Male"   "Male"   "Male"   "Female" "Male"   "Female"
  [8] "Female" "Male"   "Male"   "Male"   "Female" "Male"   "Male"  
 [15] "Male"   "Male"   "Male"   "Male"   "Female" "Male"   "Male"  
 [22] "Female" "Male"   "Male"   "Male"   "Female" "Female" "Female"
 [29] "Male"   "Male"   "Female" "Male"   "Male"   "Male"   "Male"  
 [36] "Male"   "Male"   "Male"   "Male"   "Male"   NA       "Male"  
 [43] "Female" "Male"   "Female" "Male"   "Male"   "Male"   "Female"
 [50] "Male"   "Female" "Male"   "Male"   "Male"   "Male"   "Male"  
 [57] "Male"   "Male"   "Male"   "Male"   "Female" "Female" "Male"  
 [64] "Female" "Male"   "Male"   "Male"   "Male"   "Male"   "Male"  
 [71] "Female" "Male"   "Male"   "Male"   "Male"   "Female" "Male"  
 [78] "Female" "Male"   "Male"   "Male"   "Female" "Male"   "Male"  
 [85] "Male"   "Male"   "Male"   "Female" "Female" "Female" "Male"  
 [92] "Female" "Male"   "Female" "Female" "Male"   "Male"   "Female"
 [99] "Male"   "Male"   "Male"   "Male"   "Female" "Female" "Male"  
[106] "Male"   "Male"   "Male"   "Male"   "Male"   "Female" "Male"  
[113] "Male"   "Female" "Female" "Male"   "Male"   "Female" "Male"  
[120] "Male"   "Male"   "Female" "Male"   "Male"   "Male"   "Female"
[127] "Female" "Male"   "Male"   "Female" "Male"   "Male"   "Male"  
[134] "Male"   "Female" "Female" "Male"   "Male"   "Male"   "Male"  
[141] "Male"   "Male"   "Male"   "Male"   "Male"   "Male"   "Male"  
[148] "Male"   "Male"   "Female" "Male"   "Female" "Female" "Male"  
[155] "Male"   "Male"   "Male"   "Male"   "Male"   "Male"   "Male"  
[162] "Male"   "Female" "Female" "Male"   "Male"   "Male"   "Female"
[169] "Male"   "Female" "Male"   "Male"   "Female" "Female" "Male"  
[176] "Male"   "Male"   "Male"   "Male"   "Male"   "Male"   "Female"
[183] "Male"   "Male"   "Female" "Female" "Male"   "Male"   "Male"  
[190] "Male"   "Male"   "Male"   "Male"   "Female" "Female" "Male"  
[197] "Male"   "Female" "Female" "Male"   "Female" "Female" "Male"  
[204] "Female" "Male"   "Male"   "Male"   "Male"   "Male"   "Female"
[211] "Female" "Male"   "Male"   "Female" "Male"   "Male"  
> 
```

The output shows the sex column of the mydata data frame, where any invalid values have been replaced with NA.

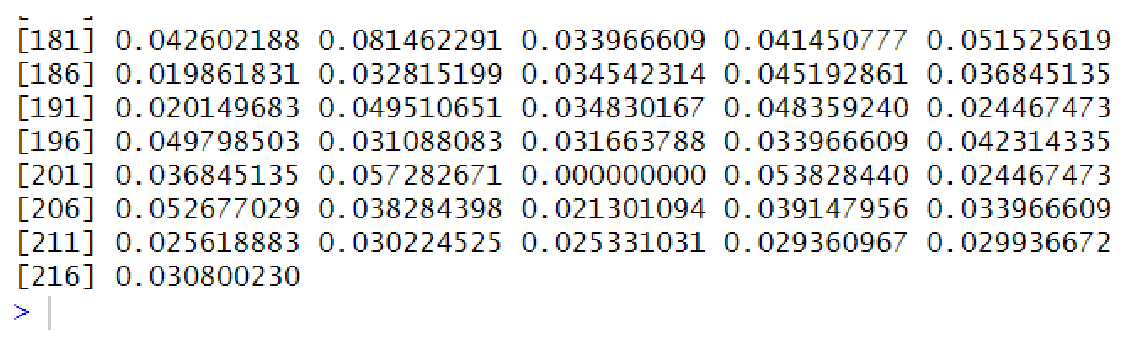


```
mydata$chol[mydata$chol < 0 | mydata$chol > 1] <- NA
mydata$chol
```

The code sets any value in the chol column of mydata that is either less than 0 or greater than 1 to NA. This filters out invalid values, assuming the column should only contain values within the 0 to 1 range.



```
> mydata$chol[mydata$chol < 0 | mydata$chol > 1] <- NA
> mydata$chol
  [1] 0.030800230 0.046056419 0.029648820 0.035693725 0.022452504
  [6] 0.031663788 0.040875072 0.065630397 0.036845135 0.022164652
 [11] 0.018998273 0.048359240 0.037420841 0.039435809 0.021013241
 [16] 0.012089810 0.029648820 0.032527346 0.042890040 0.040299367
 [21] 0.024467473 0.045192861 0.045480714 0.028209557 0.023028210
 [26] 0.026770294 0.061600461 0.028785262 0.034830167 0.011801957
 [31] 0.032527346 0.029936672 0.060161197 0.031088083 0.030800230
 [36] 0.028785262 0.014680484 0.043177893 0.065342545 0.033678756
 [41] 0.028497409 0.021013241 0.050662061 0.024755325 0.058721934
 [46] 0.029936672 0.014104778 0.033678756 0.083765112 0.020437536
 [51] 0.020725389 0.014680484 0.047207830 0.026770294 0.036557283
 [56] 0.040299367 0.030800230 0.013241220 0.042314335 0.025043178
 [61] 0.051525619 0.014680484 0.025906736 0.051237766 0.017846862
 [66] 0.044905009 0.016983305 0.030512378 0.057570524 0.030224525
 [71] 0.041162925 0.036845135 0.040587219 0.035118020 0.020437536
 [76] 1.000000000 0.037996546 0.052389177 0.034254462 0.041450777
 [81] 0.023603915 0.039723661 0.056131261 0.042602188 0.057282671
 [86] 0.031375936 0.037708693 0.025906736 0.031088083 0.037420841
 [91] 0.050662061 0.010938400 0.030224525 0.004317789 0.036269430
 [96] 0.037132988 0.032527346 0.037996546 0.021588946 0.027633851
[101] 0.038572251 0.016119747 0.050949914 0.040011514 0.017846862
[106] 0.052677029 0.014680484 0.029648820 0.038572251 0.026770294
[111] 0.052101324 0.035405872 0.017271157 0.061888313 0.039435809
[116] 0.022164652 0.024467473 0.016407599 0.058721934 0.036845135
[121] 0.037420841 0.080886586 0.027633851 0.026194588 0.044905009
[126] 0.031088083 0.046632124 0.032527346 0.027058146 0.023891767
[131] 0.037996546 0.029073115 0.022452504 0.038860104 0.025043178
[136] 0.035693725 0.013816926 0.044617156 0.020725389 0.034254462
[141] 0.027345999 0.046632124 0.022740357 0.052677029 0.032815199
[146] 0.033678756 0.046919977 0.035693725 0.052389177 0.055267703
[151] 0.049510651 0.040011514 0.126079447 0.046919977 0.034542314
[156] 0.056419113 0.049798503 0.050086356 0.048071387 0.043465746
[161] 0.020437536 0.051237766 0.025331031 0.035118020 0.037132988
[166] 0.023316062 0.027921704 0.046632124 0.044905009 0.009786989
[171] 0.041162925 0.028785262 0.035405872 0.077144502 0.024755325
[176] 0.042602188 0.030800230 0.016695452 0.054404145 0.034542314
[181] 0.042602188 0.081462291 0.033966609 0.041450777 0.051525619
```

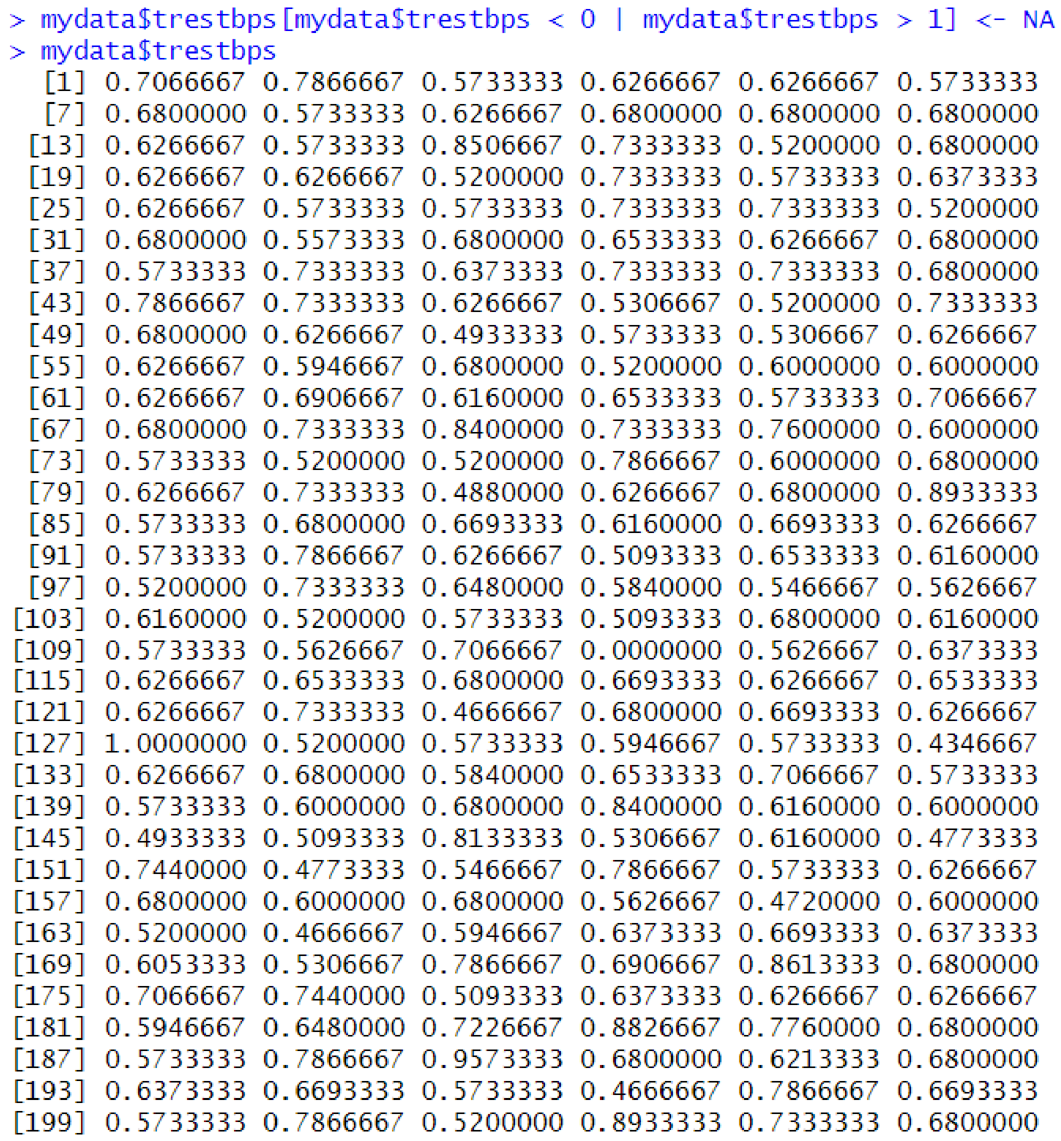


```
[181] 0.042602188 0.081462291 0.033966609 0.041450777 0.051525619
[186] 0.019861831 0.032815199 0.034542314 0.045192861 0.036845135
[191] 0.020149683 0.049510651 0.034830167 0.048359240 0.024467473
[196] 0.049798503 0.031088083 0.031663788 0.033966609 0.042314335
[201] 0.036845135 0.057282671 0.000000000 0.053828440 0.024467473
[206] 0.052677029 0.038284398 0.021301094 0.039147956 0.033966609
[211] 0.025618883 0.030224525 0.025331031 0.029360967 0.029936672
[216] 0.030800230
> 
```

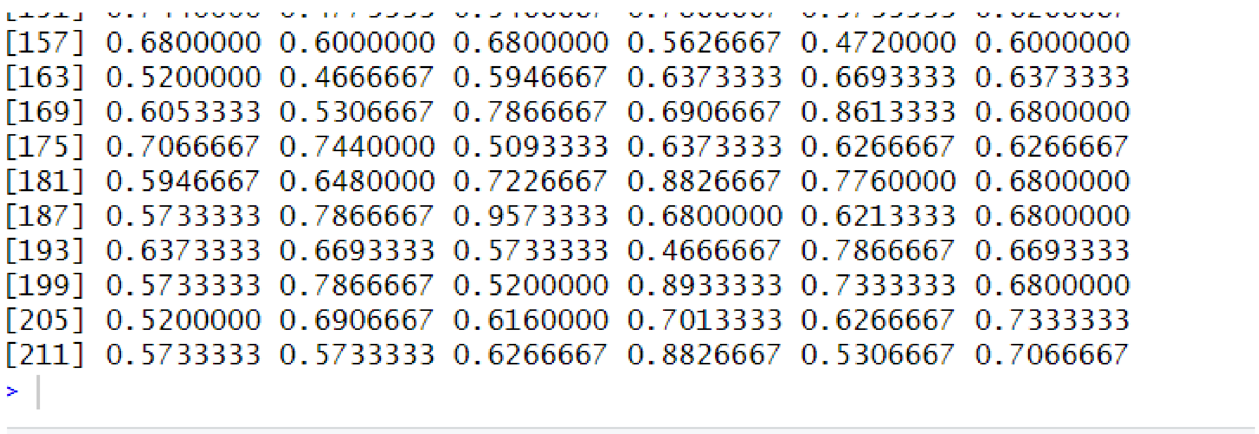
The output shows the chol column of the mydata data frame, where any values outside the range of 0 to 1 have been replaced with NA.

```
mydata$trestbps[mydata$trestbps < 0 | mydata$trestbps > 1] <- NA
mydata$trestbps
```

The code sets any value in the trestbps column of mydata that is less than 0 or greater than 1 to NA, ensuring only valid values within the range of 0 to 1 remain.

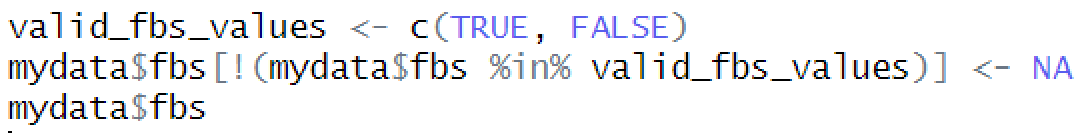


```
> mydata$trestbps[mydata$trestbps < 0 | mydata$trestbps > 1] <- NA
> mydata$trestbps
  [1] 0.7066667 0.7866667 0.5733333 0.6266667 0.6266667 0.5733333
  [7] 0.6800000 0.5733333 0.6266667 0.6800000 0.6800000 0.6800000
 [13] 0.6266667 0.5733333 0.8506667 0.7333333 0.5200000 0.6800000
 [19] 0.6266667 0.6266667 0.5200000 0.7333333 0.5733333 0.6373333
 [25] 0.6266667 0.5733333 0.5733333 0.7333333 0.7333333 0.5200000
 [31] 0.6800000 0.5573333 0.6800000 0.6533333 0.6266667 0.6800000
 [37] 0.5733333 0.7333333 0.6373333 0.7333333 0.7333333 0.6800000
 [43] 0.7866667 0.7333333 0.6266667 0.5306667 0.5200000 0.7333333
 [49] 0.6800000 0.6266667 0.4933333 0.5733333 0.5306667 0.6266667
 [55] 0.6266667 0.5946667 0.6800000 0.5200000 0.6000000 0.6000000
 [61] 0.6266667 0.6906667 0.6160000 0.6533333 0.5733333 0.7066667
 [67] 0.6800000 0.7333333 0.8400000 0.7333333 0.7600000 0.6000000
 [73] 0.5733333 0.5200000 0.5200000 0.7866667 0.6000000 0.6800000
 [79] 0.6266667 0.7333333 0.4880000 0.6266667 0.6800000 0.8933333
 [85] 0.5733333 0.6800000 0.6693333 0.6160000 0.6693333 0.6266667
 [91] 0.5733333 0.7866667 0.6266667 0.5093333 0.6533333 0.6160000
 [97] 0.5200000 0.7333333 0.6480000 0.5840000 0.5466667 0.5626667
[103] 0.6160000 0.5200000 0.5733333 0.5093333 0.6800000 0.6160000
[109] 0.5733333 0.5626667 0.7066667 0.0000000 0.5626667 0.6373333
[115] 0.6266667 0.6533333 0.6800000 0.6693333 0.6266667 0.6533333
[121] 0.6266667 0.7333333 0.4666667 0.6800000 0.6693333 0.6266667
[127] 1.0000000 0.5200000 0.5733333 0.5946667 0.5733333 0.4346667
[133] 0.6266667 0.6800000 0.5840000 0.6533333 0.7066667 0.5733333
[139] 0.5733333 0.6000000 0.6800000 0.8400000 0.6160000 0.6000000
[145] 0.4933333 0.5093333 0.8133333 0.5306667 0.6160000 0.4773333
[151] 0.7440000 0.4773333 0.5466667 0.7866667 0.5733333 0.6266667
[157] 0.6800000 0.6000000 0.6800000 0.5626667 0.4720000 0.6000000
[163] 0.5200000 0.4666667 0.5946667 0.6373333 0.6693333 0.6373333
[169] 0.6053333 0.5306667 0.7866667 0.6906667 0.8613333 0.6800000
[175] 0.7066667 0.7440000 0.5093333 0.6373333 0.6266667 0.6266667
[181] 0.5946667 0.6480000 0.7226667 0.8826667 0.7760000 0.6800000
[187] 0.5733333 0.7866667 0.9573333 0.6800000 0.6213333 0.6800000
[193] 0.6373333 0.6693333 0.5733333 0.4666667 0.7866667 0.6693333
[199] 0.5733333 0.7866667 0.5200000 0.8933333 0.7333333 0.6800000
```



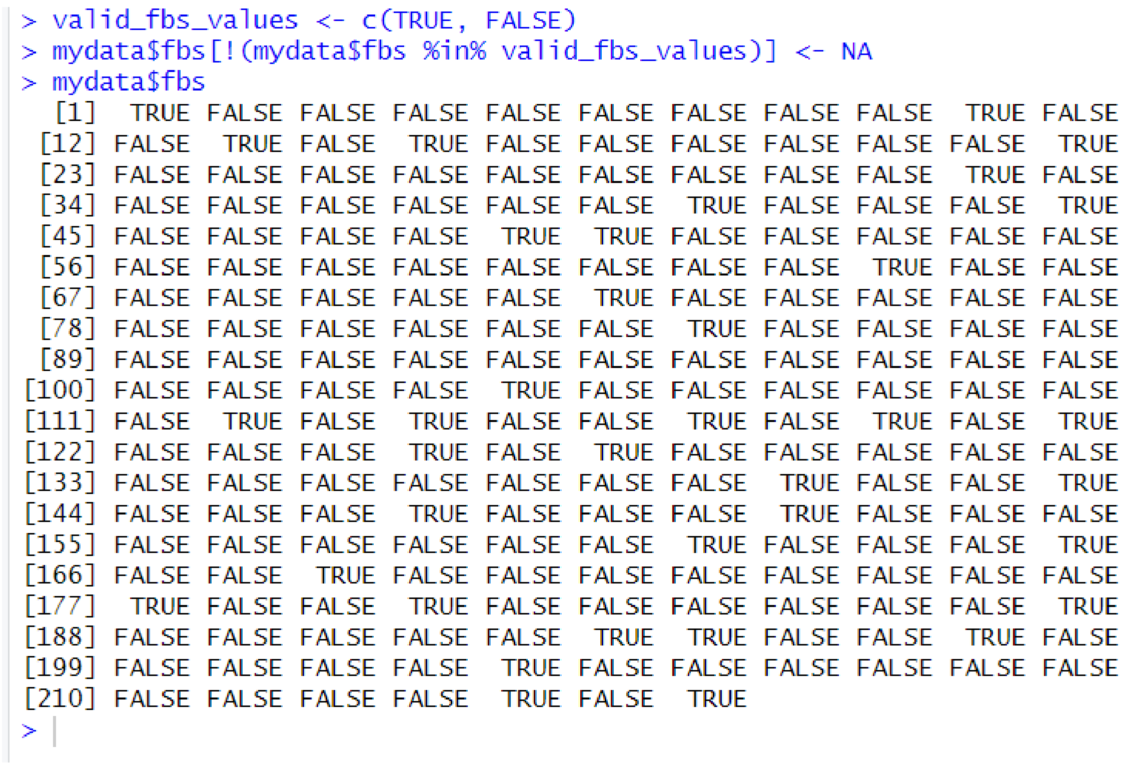
```
[157] 0.6800000 0.6000000 0.6800000 0.5626667 0.4720000 0.6000000
[163] 0.5200000 0.4666667 0.5946667 0.6373333 0.6693333 0.6373333
[169] 0.6053333 0.5306667 0.7866667 0.6906667 0.8613333 0.6800000
[175] 0.7066667 0.7440000 0.5093333 0.6373333 0.6266667 0.6266667
[181] 0.5946667 0.6480000 0.7226667 0.8826667 0.7760000 0.6800000
[187] 0.5733333 0.7866667 0.9573333 0.6800000 0.6213333 0.6800000
[193] 0.6373333 0.6693333 0.5733333 0.4666667 0.7866667 0.6693333
[199] 0.5733333 0.7866667 0.5200000 0.8933333 0.7333333 0.6800000
[205] 0.5200000 0.6906667 0.6160000 0.7013333 0.6266667 0.7333333
[211] 0.5733333 0.5733333 0.6266667 0.8826667 0.5306667 0.7066667
> 
```

The output shows the trestbps column of the mydata data frame, with any values outside the 0 to 1 range replaced by NA.



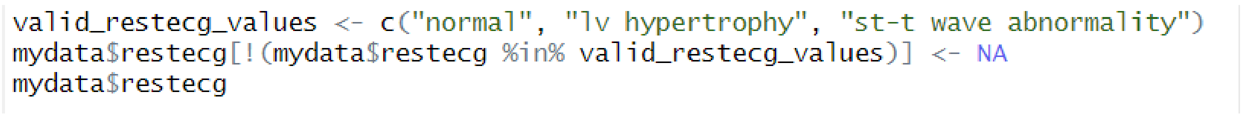
```
valid_fbs_values <- c(TRUE, FALSE)
mydata$fbs[!(mydata$fbs %in% valid_fbs_values)] <- NA
mydata$fbs
```

The code defines a vector valid_fbs_values with valid values TRUE and FALSE for the fbs column. It then sets any value in the fbs column of mydata that is not TRUE or FALSE to NA.



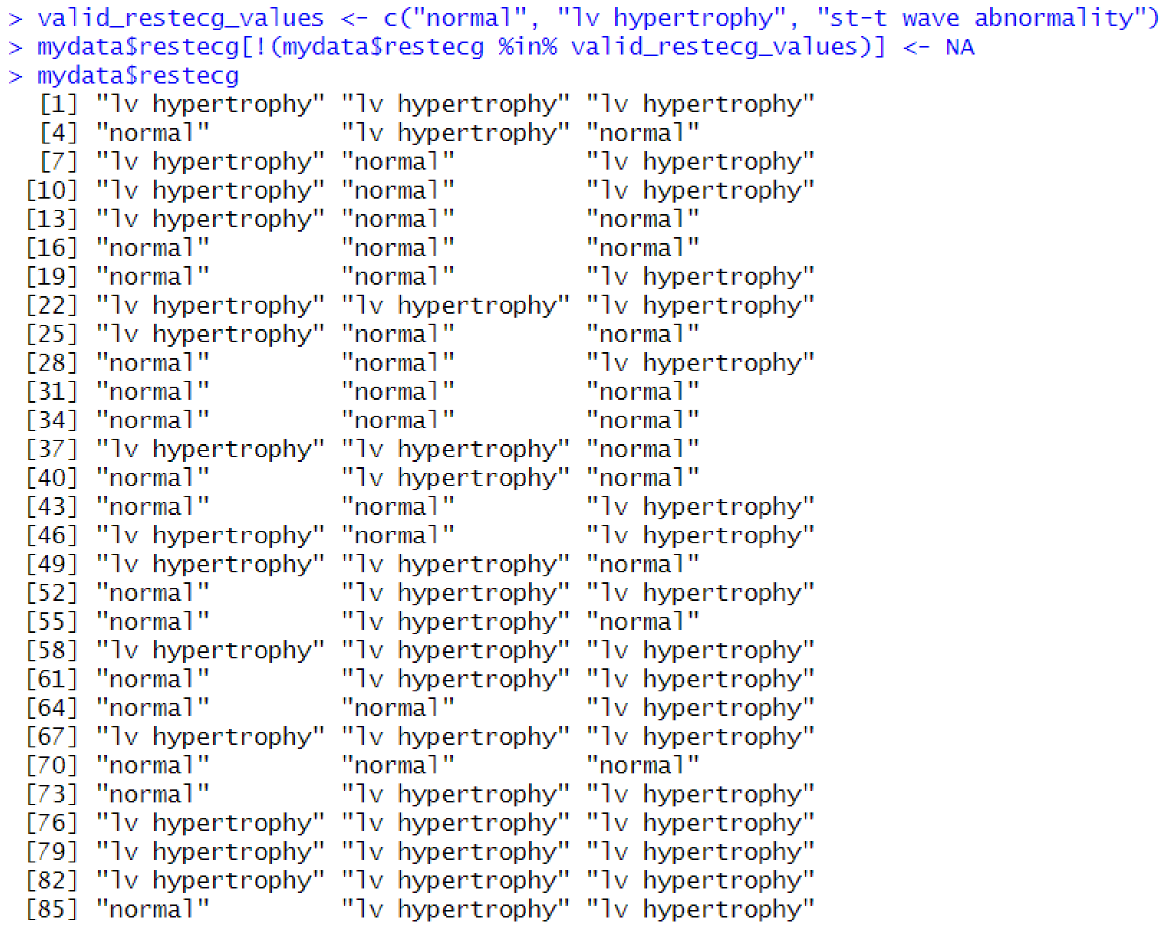
```
> valid_fbs_values <- c(TRUE, FALSE)
> mydata$fbs[!(mydata$fbs %in% valid_fbs_values)] <- NA
> mydata$fbs
  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
 [12] FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
 [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
 [34] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
 [45] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
 [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
 [67] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
 [78] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
 [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[100] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
[111] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE
[122] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
[133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
[144] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
[155] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
[166] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[177]  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[188] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE
[199] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
[210] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
> 
```

The output shows the fbs column of the mydata data frame, with any invalid values replaced by NA.

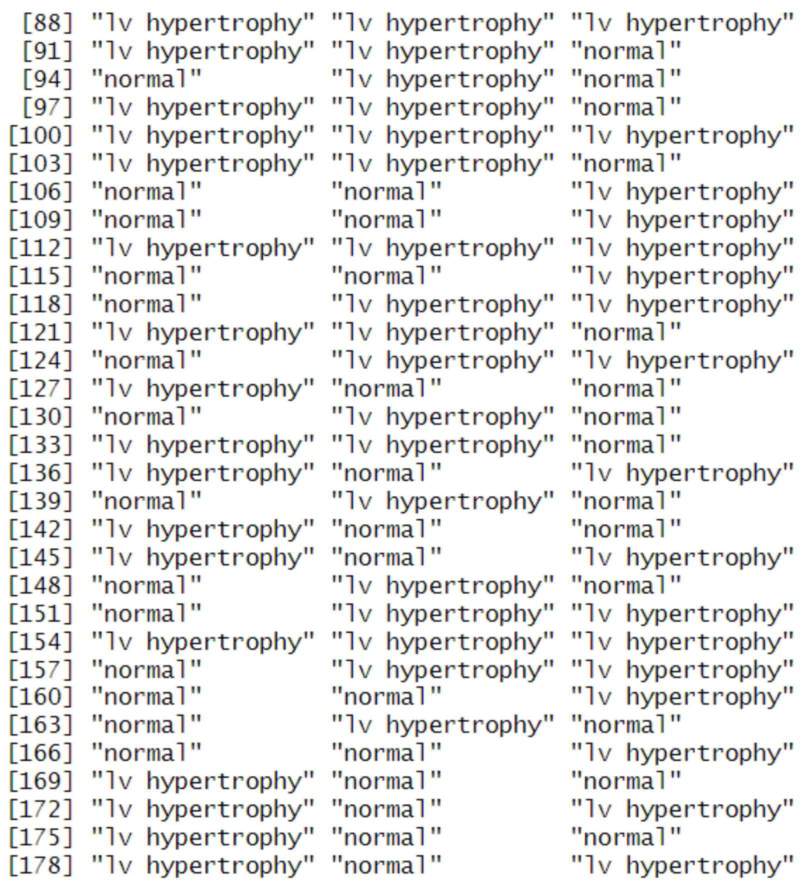


```
valid_restecg_values <- c("normal", "lv hypertrophy", "st-t wave abnormality")
mydata$restecg[!(mydata$restecg %in% valid_restecg_values)] <- NA
mydata$restecg
```

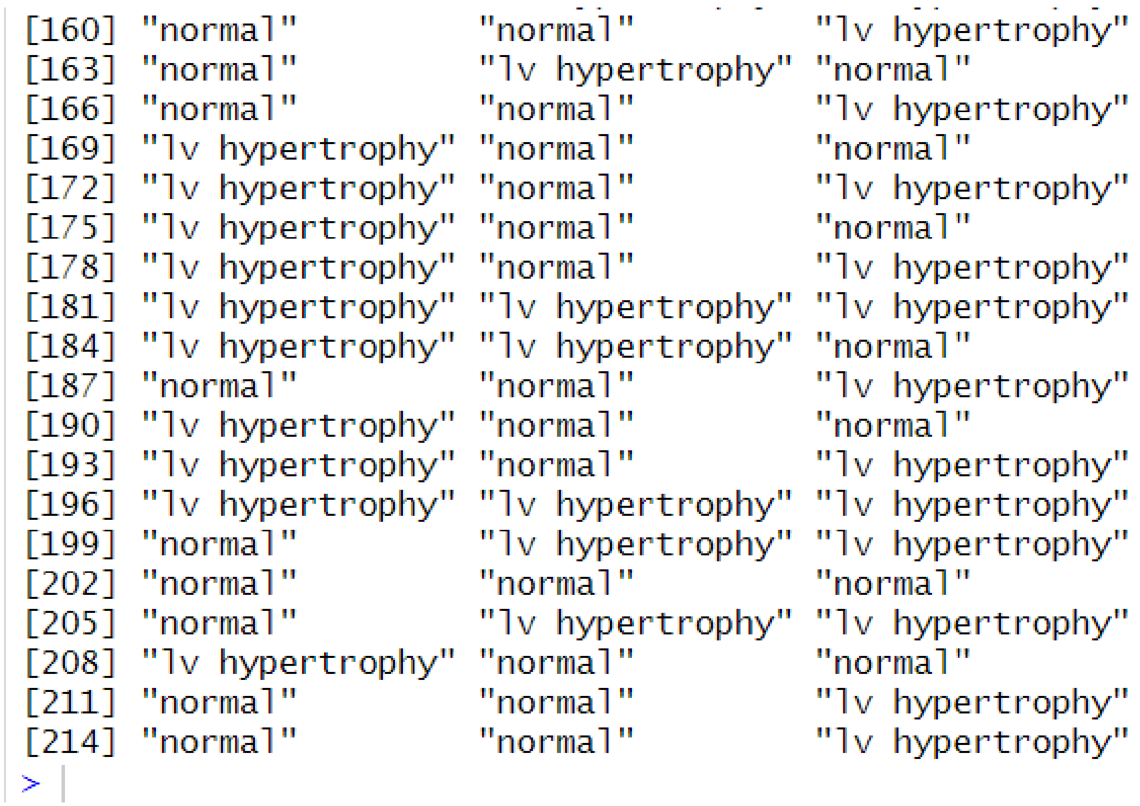
The code defines a vector valid_restecg_values containing the valid values for the restecg column: "normal", "lv hypertrophy", and "st-t wave abnormality". It then sets any value in the restecg column of mydata that is not one of these valid values to NA.



```
> valid_restecg_values <- c("normal", "lv hypertrophy", "st-t wave abnormality")
> mydata$restecg[!(mydata$restecg %in% valid_restecg_values)] <- NA
> mydata$restecg
  [1] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
  [4] "normal"         "lv hypertrophy" "normal"
  [7] "lv hypertrophy" "normal"         "lv hypertrophy"
 [10] "lv hypertrophy" "normal"         "lv hypertrophy"
 [13] "lv hypertrophy" "normal"         "normal"
 [16] "normal"         "normal"         "normal"
 [19] "normal"         "normal"         "lv hypertrophy"
 [22] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
 [25] "lv hypertrophy" "normal"         "normal"
 [28] "normal"         "normal"         "lv hypertrophy"
 [31] "normal"         "normal"         "normal"
 [34] "normal"         "normal"         "normal"
 [37] "lv hypertrophy" "lv hypertrophy" "normal"
 [40] "normal"         "lv hypertrophy" "normal"
 [43] "normal"         "normal"         "lv hypertrophy"
 [46] "lv hypertrophy" "normal"         "lv hypertrophy"
 [49] "lv hypertrophy" "lv hypertrophy" "normal"
 [52] "normal"         "lv hypertrophy" "lv hypertrophy"
 [55] "normal"         "lv hypertrophy" "normal"
 [58] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
 [61] "normal"         "lv hypertrophy" "lv hypertrophy"
 [64] "normal"         "normal"         "lv hypertrophy"
 [67] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
 [70] "normal"         "normal"         "normal"
 [73] "normal"         "lv hypertrophy" "lv hypertrophy"
 [76] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
 [79] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
 [82] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
 [85] "normal"         "lv hypertrophy" "lv hypertrophy"
```



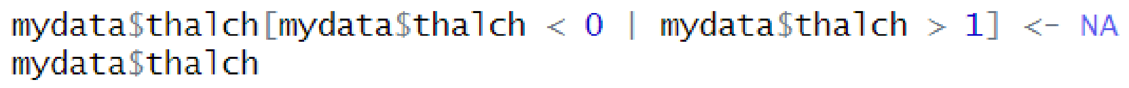
```
 [88] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
 [91] "lv hypertrophy" "lv hypertrophy" "normal"
 [94] "normal"         "lv hypertrophy" "normal"
 [97] "lv hypertrophy" "lv hypertrophy" "normal"
[100] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
[103] "lv hypertrophy" "lv hypertrophy" "normal"
[106] "normal"         "normal"         "lv hypertrophy"
[109] "normal"         "normal"         "lv hypertrophy"
[112] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
[115] "normal"         "normal"         "lv hypertrophy"
[118] "normal"         "lv hypertrophy" "lv hypertrophy"
[121] "lv hypertrophy" "lv hypertrophy" "normal"
[124] "normal"         "lv hypertrophy" "lv hypertrophy"
[127] "lv hypertrophy" "normal"         "normal"
[130] "normal"         "lv hypertrophy" "normal"
[133] "lv hypertrophy" "lv hypertrophy" "normal"
[136] "lv hypertrophy" "normal"         "lv hypertrophy"
[139] "normal"         "lv hypertrophy" "normal"
[142] "lv hypertrophy" "normal"         "normal"
[145] "lv hypertrophy" "normal"         "lv hypertrophy"
[148] "normal"         "lv hypertrophy" "normal"
[151] "normal"         "lv hypertrophy" "lv hypertrophy"
[154] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
[157] "normal"         "lv hypertrophy" "lv hypertrophy"
[160] "normal"         "normal"         "lv hypertrophy"
[163] "normal"         "lv hypertrophy" "normal"
[166] "normal"         "normal"         "lv hypertrophy"
[169] "lv hypertrophy" "normal"         "normal"
[172] "lv hypertrophy" "normal"         "lv hypertrophy"
[175] "lv hypertrophy" "normal"         "normal"
[178] "lv hypertrophy" "normal"         "lv hypertrophy"
```



```
[160] "normal"         "normal"         "lv hypertrophy"
[163] "normal"         "lv hypertrophy" "normal"
[166] "normal"         "normal"         "lv hypertrophy"
[169] "lv hypertrophy" "normal"         "normal"
[172] "lv hypertrophy" "normal"         "lv hypertrophy"
[175] "lv hypertrophy" "normal"         "normal"
[178] "lv hypertrophy" "normal"         "lv hypertrophy"
```

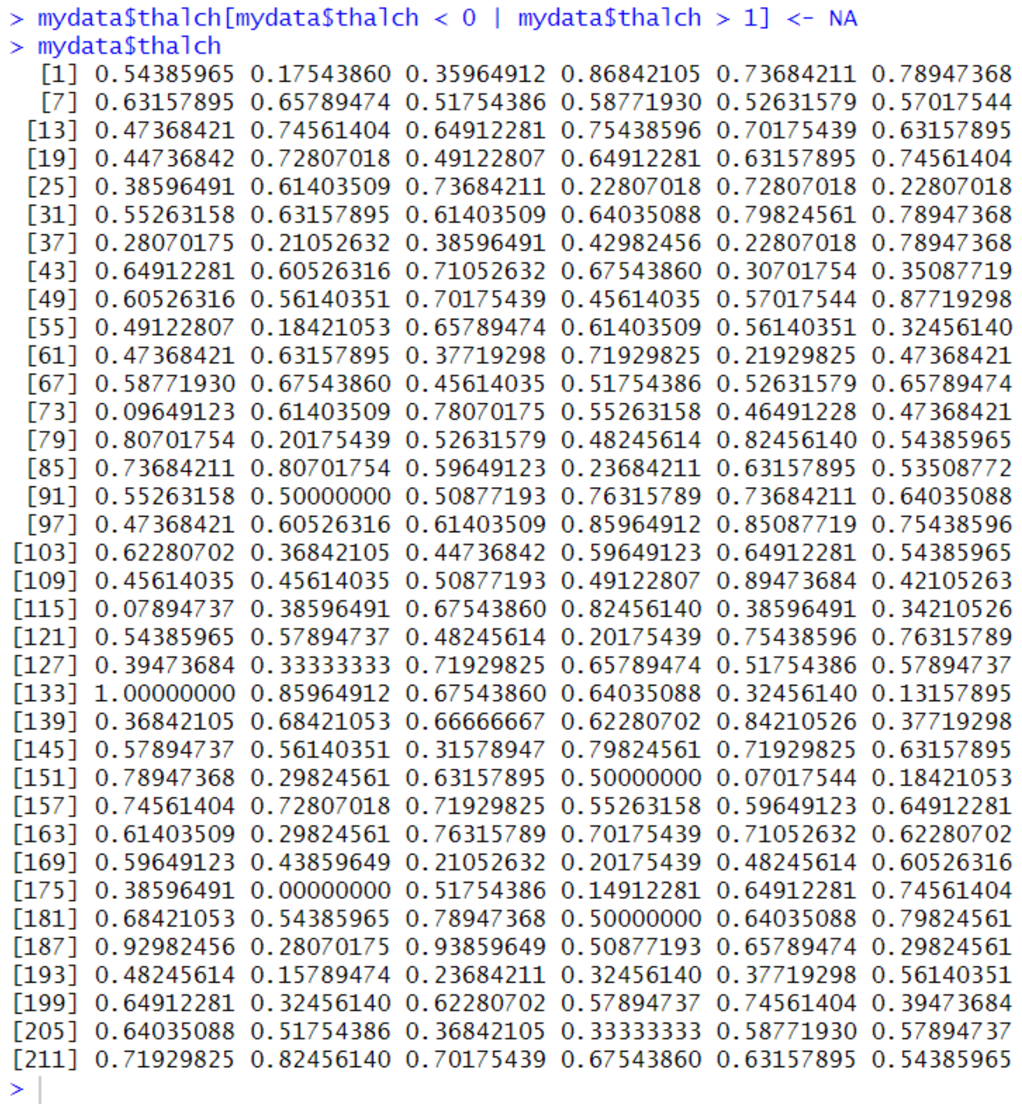
```
[181] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
[184] "lv hypertrophy" "lv hypertrophy" "normal"
[187] "normal"         "normal"         "lv hypertrophy"
[190] "lv hypertrophy" "normal"         "normal"
[193] "lv hypertrophy" "normal"         "lv hypertrophy"
[196] "lv hypertrophy" "lv hypertrophy" "lv hypertrophy"
[199] "normal"         "lv hypertrophy" "lv hypertrophy"
[202] "normal"         "normal"         "normal"
[205] "normal"         "lv hypertrophy" "lv hypertrophy"
[208] "lv hypertrophy" "normal"         "normal"
[211] "normal"         "normal"         "lv hypertrophy"
[214] "normal"         "normal"         "lv hypertrophy"
> 
```

The output shows the restecg column of the mydata data frame, with any invalid values replaced by NA.



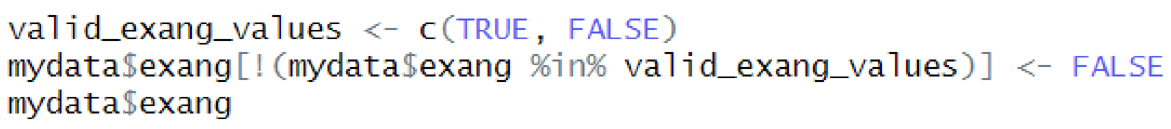
```
mydata$thalch[mydata$thalch < 0 | mydata$thalch > 1] <- NA
mydata$thalch
```

The code sets any value in the thalch column of mydata that is less than 0 or greater than 1 to NA, ensuring only valid values within the 0 to 1 range remain.



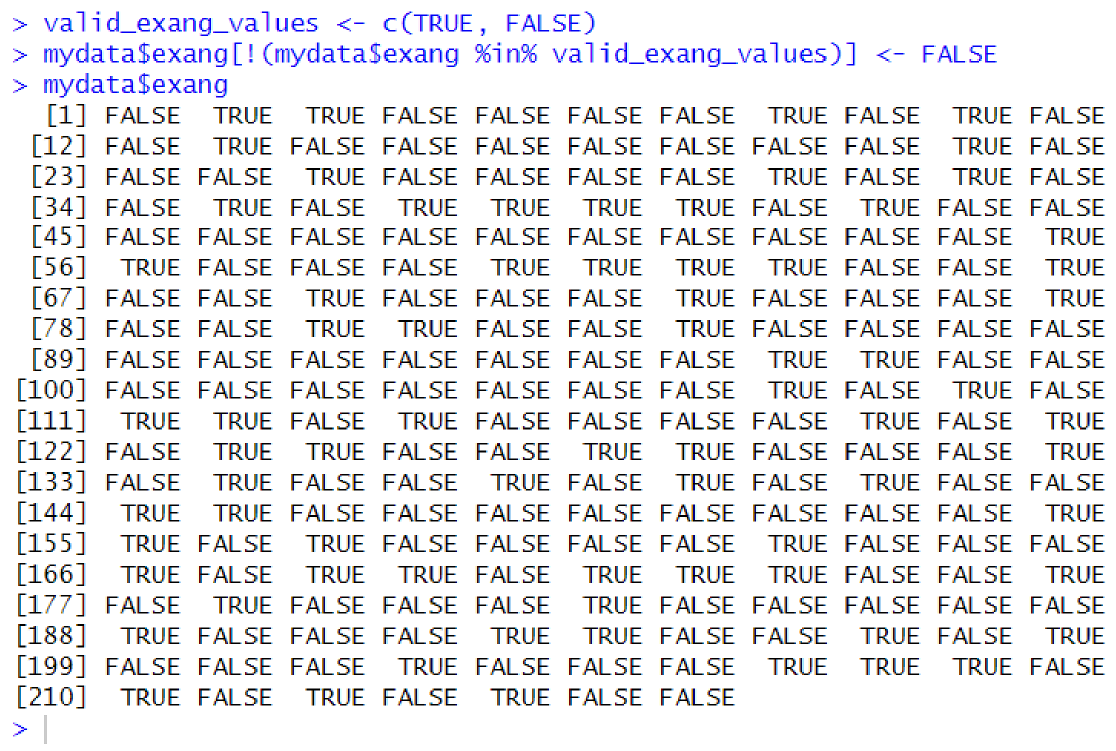
```
> mydata$thalch[mydata$thalch < 0 | mydata$thalch > 1] <- NA
> mydata$thalch
  [1] 0.54385965 0.17543860 0.35964912 0.86842105 0.73684211 0.78947368
  [7] 0.63157895 0.65789474 0.51754386 0.58771930 0.52631579 0.57017544
 [13] 0.47368421 0.74561404 0.64912281 0.75438596 0.70175439 0.63157895
 [19] 0.44736842 0.72807018 0.49122807 0.64912281 0.63157895 0.74561404
 [25] 0.38596491 0.61403509 0.73684211 0.22807018 0.72807018 0.22807018
 [31] 0.55263158 0.63157895 0.61403509 0.64035088 0.79824561 0.78947368
 [37] 0.28070175 0.21052632 0.38596491 0.42982456 0.22807018 0.78947368
 [43] 0.64912281 0.60526316 0.71052632 0.67543860 0.30701754 0.35087719
 [49] 0.60526316 0.56140351 0.70175439 0.45614035 0.57017544 0.87719298
 [55] 0.49122807 0.18421053 0.65789474 0.61403509 0.56140351 0.32456140
 [61] 0.47368421 0.63157895 0.37719298 0.71929825 0.21929825 0.47368421
 [67] 0.58771930 0.67543860 0.45614035 0.51754386 0.52631579 0.65789474
 [73] 0.09649123 0.61403509 0.78070175 0.55263158 0.46491228 0.47368421
 [79] 0.80701754 0.20175439 0.52631579 0.48245614 0.82456140 0.54385965
 [85] 0.73684211 0.80701754 0.59649123 0.23684211 0.63157895 0.53508772
 [91] 0.55263158 0.50000000 0.50877193 0.76315789 0.73684211 0.64035088
 [97] 0.47368421 0.60526316 0.61403509 0.85964912 0.85087719 0.75438596
[103] 0.62280702 0.36842105 0.44736842 0.59649123 0.64912281 0.54385965
[109] 0.45614035 0.45614035 0.50877193 0.49122807 0.89473684 0.42105263
[115] 0.07894737 0.38596491 0.67543860 0.82456140 0.38596491 0.34210526
[121] 0.54385965 0.57894737 0.48245614 0.20175439 0.75438596 0.76315789
[127] 0.39473684 0.33333333 0.71929825 0.65789474 0.51754386 0.57894737
[133] 1.00000000 0.85964912 0.67543860 0.64035088 0.32456140 0.13157895
[139] 0.36842105 0.68421053 0.66666667 0.62280702 0.84210526 0.37719298
[145] 0.57894737 0.56140351 0.31578947 0.79824561 0.71929825 0.63157895
[151] 0.78947368 0.29824561 0.63157895 0.50000000 0.07017544 0.18421053
[157] 0.74561404 0.72807018 0.71929825 0.55263158 0.59649123 0.64912281
[163] 0.61403509 0.29824561 0.76315789 0.70175439 0.71052632 0.62280702
[169] 0.59649123 0.43859649 0.21052632 0.20175439 0.48245614 0.60526316
[175] 0.38596491 0.00000000 0.51754386 0.14912281 0.64912281 0.74561404
[181] 0.68421053 0.54385965 0.78947368 0.50000000 0.64035088 0.79824561
[187] 0.92982456 0.28070175 0.93859649 0.50877193 0.65789474 0.29824561
[193] 0.48245614 0.15789474 0.23684211 0.32456140 0.37719298 0.56140351
[199] 0.64912281 0.32456140 0.62280702 0.57894737 0.74561404 0.39473684
[205] 0.64035088 0.51754386 0.36842105 0.33333333 0.58771930 0.57894737
[211] 0.71929825 0.82456140 0.70175439 0.67543860 0.63157895 0.54385965
> 
```

The output shows the thalch column of the mydata data frame, with any values outside the 0 to 1 range replaced by NA.



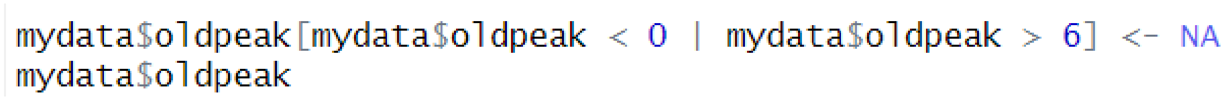
```
valid_exang_values <- c(TRUE, FALSE)
mydata$exang[!(mydata$exang %in% valid_exang_values)] <- FALSE
mydata$exang
```

The code defines a vector valid_exang_values containing valid values TRUE and FALSE for the exang column. It then sets any value in the exang column of mydata that is not TRUE or FALSE to FALSE.



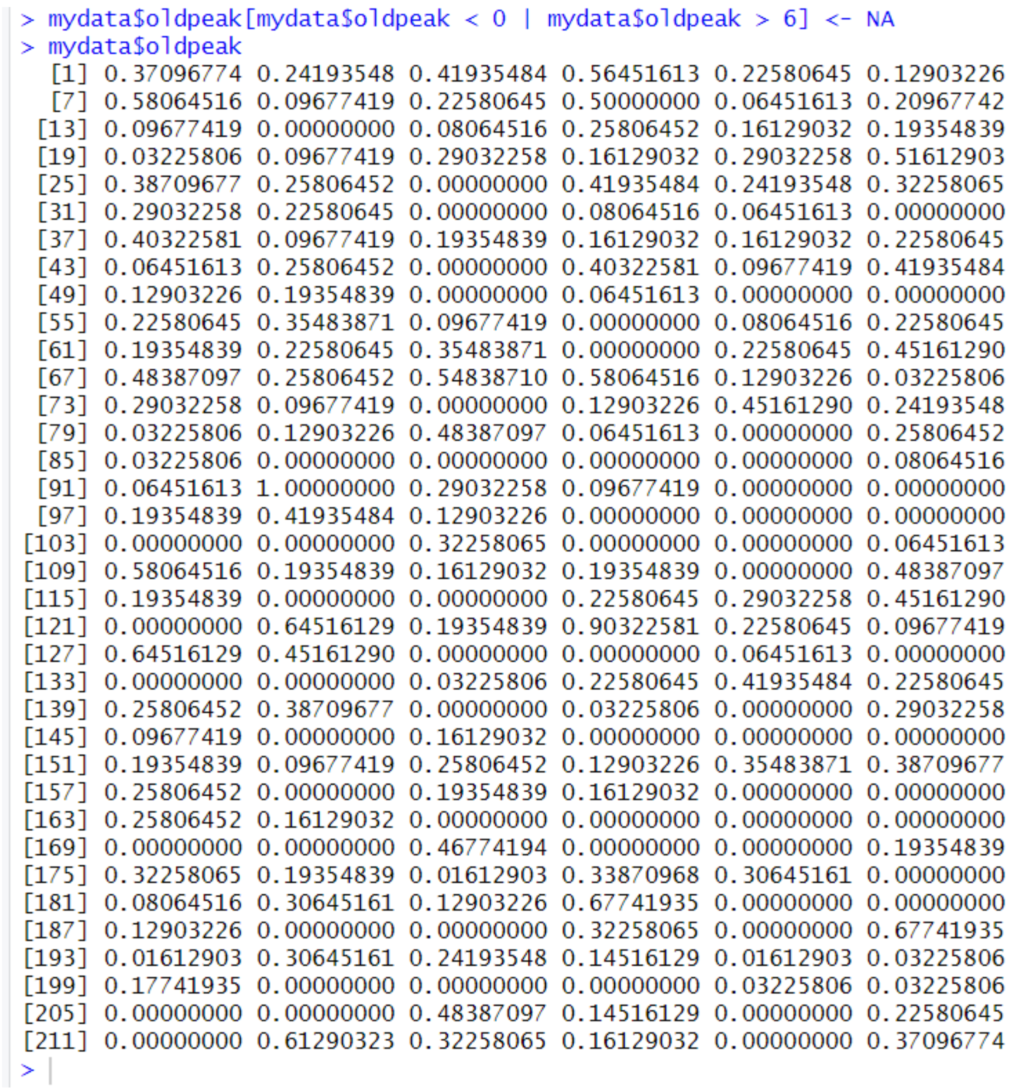
```
> valid_exang_values <- c(TRUE, FALSE)
> mydata$exang[!(mydata$exang %in% valid_exang_values)] <- FALSE
> mydata$exang
  [1] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
 [12] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
 [23] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
 [34] FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
 [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
 [56]  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
 [67] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
 [78] FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
 [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
[111]  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
[122] FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE
[133] FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE
[144]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[155]  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
[166]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
[177] FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
[188]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE
[199] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE
[210]  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE
> 
```

The output shows the exang column of the mydata data frame, with any invalid values replaced by FALSE.



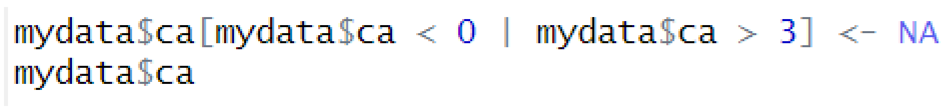
```
mydata$oldpeak[mydata$oldpeak < 0 | mydata$oldpeak > 6] <- NA
mydata$oldpeak
```

The code sets any value in the oldpeak column of mydata that is less than 0 or greater than 6 to NA, ensuring only valid values within the 0 to 6 range remain.



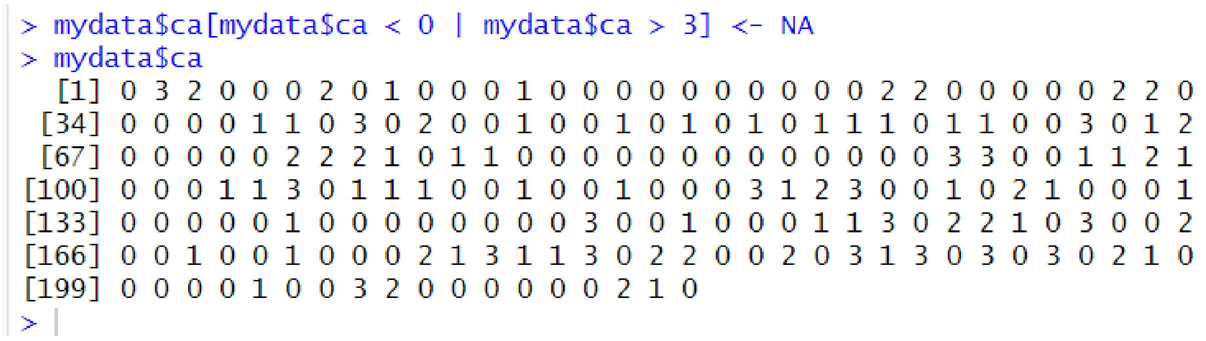
```
> mydata$oldpeak[mydata$oldpeak < 0 | mydata$oldpeak > 6] <- NA
> mydata$oldpeak
  [1] 0.37096774 0.24193548 0.41935484 0.56451613 0.22580645 0.12903226
  [7] 0.58064516 0.09677419 0.22580645 0.50000000 0.06451613 0.20967742
 [13] 0.09677419 0.00000000 0.08064516 0.25806452 0.16129032 0.19354839
 [19] 0.03225806 0.09677419 0.29032258 0.16129032 0.29032258 0.51612903
 [25] 0.38709677 0.25806452 0.00000000 0.41935484 0.24193548 0.32258065
 [31] 0.29032258 0.22580645 0.00000000 0.08064516 0.06451613 0.00000000
 [37] 0.40322581 0.09677419 0.19354839 0.16129032 0.16129032 0.22580645
 [43] 0.06451613 0.25806452 0.00000000 0.40322581 0.09677419 0.41935484
 [49] 0.12903226 0.19354839 0.00000000 0.06451613 0.00000000 0.00000000
 [55] 0.22580645 0.35483871 0.09677419 0.00000000 0.08064516 0.22580645
 [61] 0.19354839 0.22580645 0.35483871 0.00000000 0.22580645 0.45161290
 [67] 0.48387097 0.25806452 0.54838710 0.58064516 0.12903226 0.03225806
 [73] 0.29032258 0.09677419 0.00000000 0.12903226 0.45161290 0.24193548
 [79] 0.03225806 0.12903226 0.48387097 0.06451613 0.00000000 0.25806452
 [85] 0.03225806 0.00000000 0.00000000 0.00000000 0.00000000 0.08064516
 [91] 0.06451613 1.00000000 0.29032258 0.09677419 0.00000000 0.00000000
 [97] 0.19354839 0.41935484 0.12903226 0.00000000 0.00000000 0.00000000
[103] 0.00000000 0.00000000 0.32258065 0.00000000 0.00000000 0.06451613
[109] 0.58064516 0.19354839 0.16129032 0.19354839 0.00000000 0.48387097
[115] 0.19354839 0.00000000 0.00000000 0.22580645 0.29032258 0.45161290
[121] 0.00000000 0.64516129 0.19354839 0.90322581 0.22580645 0.09677419
[127] 0.64516129 0.45161290 0.00000000 0.00000000 0.06451613 0.00000000
[133] 0.00000000 0.00000000 0.03225806 0.22580645 0.41935484 0.22580645
[139] 0.25806452 0.38709677 0.00000000 0.03225806 0.00000000 0.29032258
[145] 0.09677419 0.00000000 0.16129032 0.00000000 0.00000000 0.00000000
[151] 0.19354839 0.09677419 0.25806452 0.12903226 0.35483871 0.38709677
[157] 0.25806452 0.00000000 0.19354839 0.16129032 0.00000000 0.00000000
[163] 0.25806452 0.16129032 0.00000000 0.00000000 0.00000000 0.00000000
[169] 0.00000000 0.00000000 0.46774194 0.00000000 0.00000000 0.19354839
[175] 0.32258065 0.19354839 0.01612903 0.33870968 0.30645161 0.00000000
[181] 0.08064516 0.30645161 0.12903226 0.67741935 0.00000000 0.00000000
[187] 0.12903226 0.00000000 0.00000000 0.32258065 0.00000000 0.67741935
[193] 0.01612903 0.30645161 0.24193548 0.14516129 0.01612903 0.03225806
[199] 0.17741935 0.00000000 0.00000000 0.00000000 0.03225806 0.03225806
[205] 0.00000000 0.00000000 0.48387097 0.14516129 0.00000000 0.22580645
[211] 0.00000000 0.61290323 0.32258065 0.16129032 0.00000000 0.37096774
> 
```

The output shows the oldpeak column of the mydata data frame, with any values outside the 0 to 6 range replaced by NA.



```
mydata$ca[mydata$ca < 0 | mydata$ca > 3] <- NA
mydata$ca
```

The code sets any value in the ca column of mydata that is less than 0 or greater than 3 to NA, ensuring only valid values within the 0 to 3 range remain.

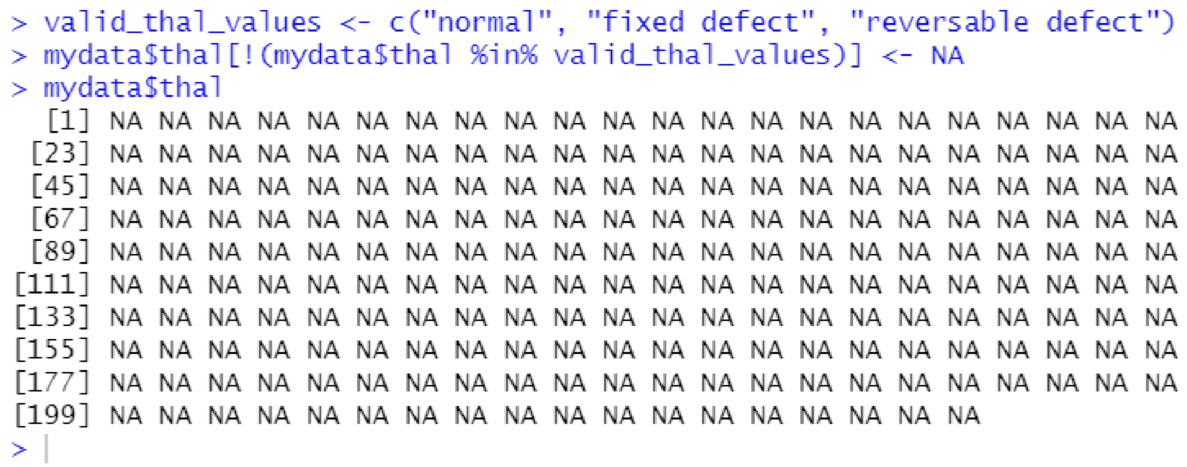


```
> mydata$ca[mydata$ca < 0 | mydata$ca > 3] <- NA
> mydata$ca
  [1] 0 3 2 0 0 0 2 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 2 2 0
 [34] 0 0 0 0 1 1 0 3 0 2 0 0 1 0 0 1 0 1 0 1 0 1 1 1 0 1 1 0 0 3 0 1 2
 [67] 0 0 0 0 0 2 2 2 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 3 3 0 0 1 1 2 1
[100] 0 0 0 1 1 3 0 1 1 1 0 0 1 0 0 1 0 0 0 3 1 2 3 0 0 1 0 2 1 0 0 0 1
[133] 0 0 0 0 0 1 0 0 0 0 0 0 0 0 3 0 0 1 0 0 0 1 1 3 0 2 2 1 0 3 0 0 2
[166] 0 0 1 0 0 1 0 0 0 2 1 3 1 1 3 0 2 2 0 0 2 0 3 1 3 0 3 0 3 0 2 1 0
[199] 0 0 0 0 1 0 0 3 2 0 0 0 0 0 0 2 1 0
> 
```

The output shows the ca column of the mydata data frame, with any values outside the 0 to 3 range replaced by NA.

```
valid_thal_values <- c("normal", "fixed defect", "reversable defect")
mydata$thal[!(mydata$thal %in% valid_thal_values)] <- NA
mydata$thal
```

The code defines a vector valid_thal_values containing valid values for the thal column: "normal", "fixed defect", and "reversable defect". It then sets any value in the thal column of mydata that is not one of these valid values to NA.

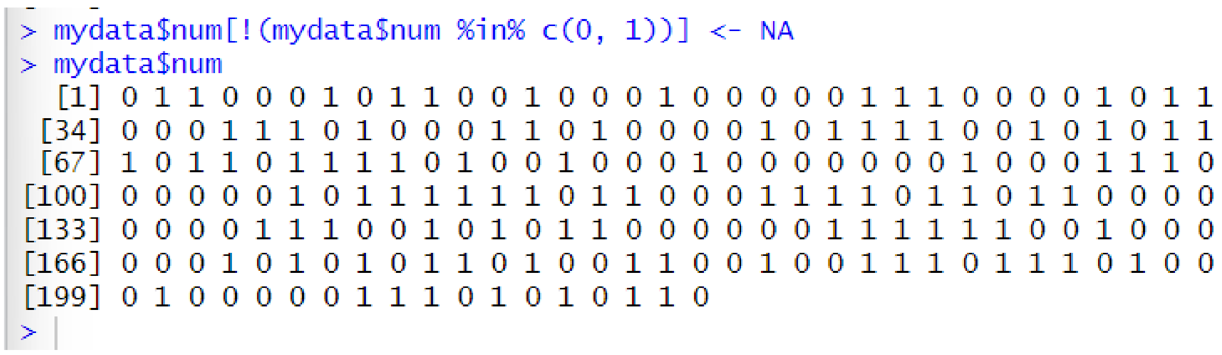


```
> valid_thal_values <- c("normal", "fixed defect", "reversable defect")
> mydata$thal[!(mydata$thal %in% valid_thal_values)] <- NA
> mydata$thal
  [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [23] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [45] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [67] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [89] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[111] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[133] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[155] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[177] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[199] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
> 
```

The output shows the thal column of the mydata data frame, with any invalid values replaced by NA.

```
mydata$num[!(mydata$num %in% c(0, 1))] <- NA
mydata$num
```

The code checks the num column of mydata and sets any value that is not 0 or 1 to NA, ensuring only valid values (0 or 1) remain in the column.

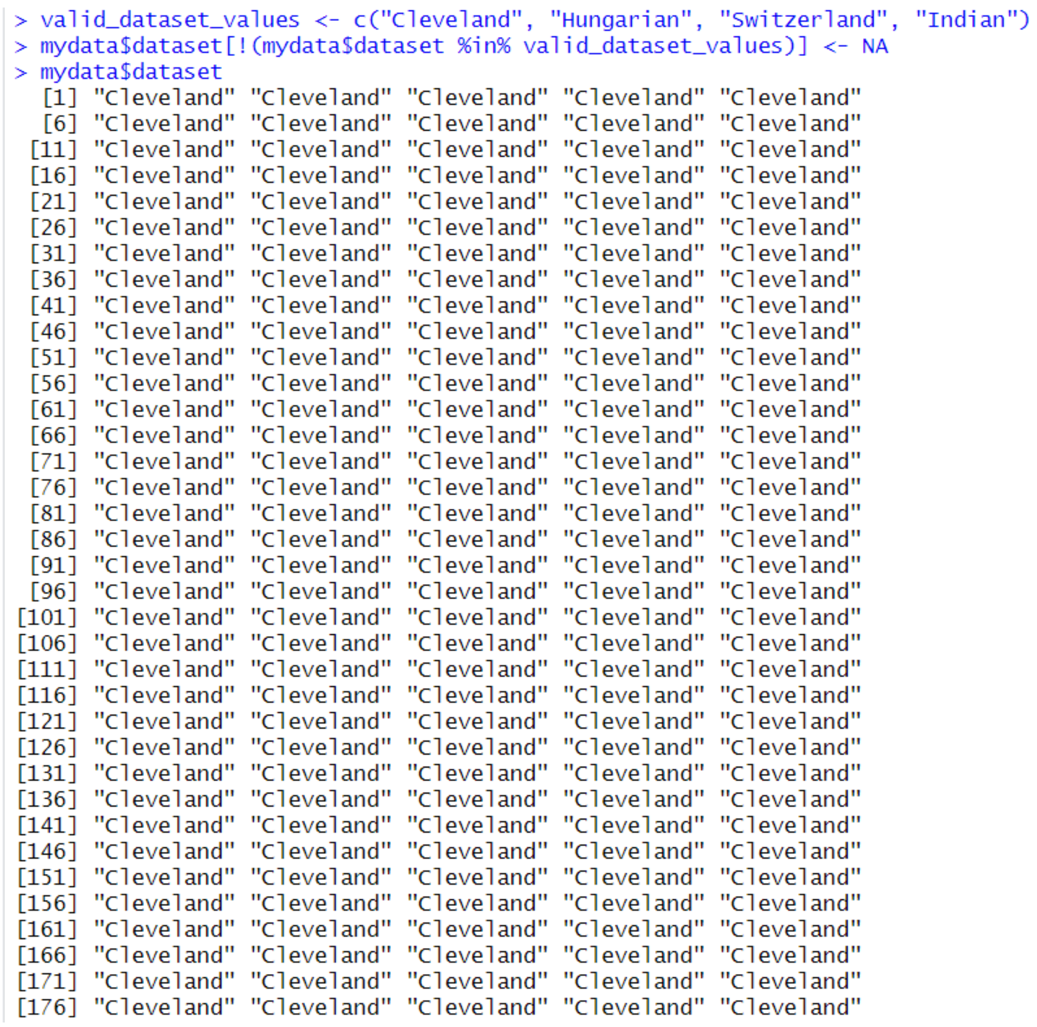


```
> mydata$num[!(mydata$num %in% c(0, 1))] <- NA
> mydata$num
  [1] 0 1 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 1 1
 [34] 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 1 1
 [67] 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 0
[100] 0 0 0 0 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 0 0 0
[133] 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 0 0 0
[166] 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 1 0 0 1 0 0 1 1 1 0 1 1 1 0 1 0 0
[199] 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 1 0
> 
```

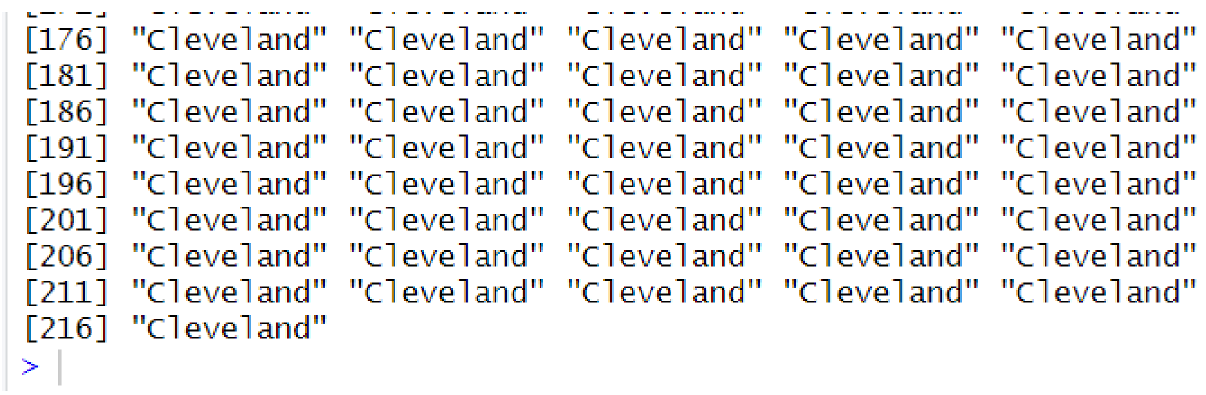
The output shows the num column of the mydata data frame, with any values other than 0 or 1 replaced by NA.

```
valid_dataset_values <- c("Cleveland", "Hungarian", "Switzerland", "Indian")
mydata$dataset[!(mydata$dataset %in% valid_dataset_values)] <- NA
mydata$dataset
```

The code defines a vector valid_dataset_values containing the valid values for the dataset column: "Cleveland", "Hungarian", "Switzerland", and "Indian". It then sets any value in the dataset column of mydata that is not one of these valid values to NA.



```
> valid_dataset_values <- c("Cleveland", "Hungarian", "Switzerland", "Indian")
> mydata$dataset[!(mydata$dataset %in% valid_dataset_values)] <- NA
> mydata$dataset
  [1] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
  [6] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [11] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [16] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [21] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [26] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [31] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [36] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [41] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [46] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [51] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [56] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [61] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [66] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [71] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [76] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [81] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [86] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [91] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
 [96] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[101] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[106] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[111] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[116] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[121] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[126] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[131] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[136] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[141] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[146] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[151] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[156] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[161] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[166] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[171] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[176] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
```

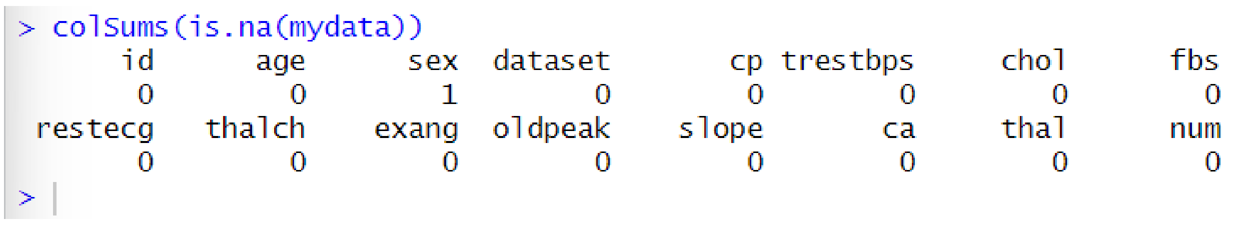


```
[176] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[181] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[186] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[191] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[196] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[201] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[206] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[211] "Cleveland" "Cleveland" "Cleveland" "Cleveland" "Cleveland"
[216] "Cleveland"
> 
```

The output shows the dataset column of the mydata data frame, with any invalid values replaced by NA.

```
colSums(is.na(mydata))
```

The code colSums(is.na(mydata)) calculates the total number of missing values (NA) in each column of the mydata data frame by first identifying NAs with is.na() and then summing them column-wise using colSums().

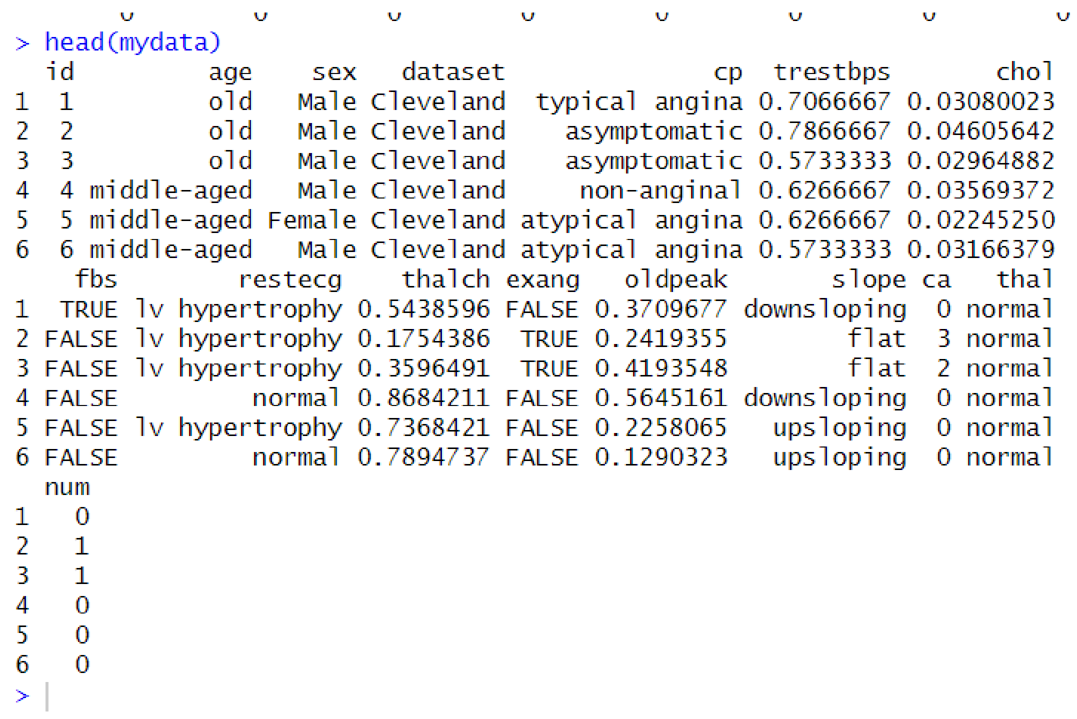


```
> colSums(is.na(mydata))
      id      age      sex  dataset       cp trestbps     chol      fbs 
       0        0        1        0        0        0        0        0 
 restecg   thalch    exang  oldpeak    slope       ca     thal      num 
       0        0        0        0        0        0        0        0 
> 
```

The output shows a list of all columns in the mydata data frame along with the count of missing (NA) values in each column.

```
head(mydata)
```

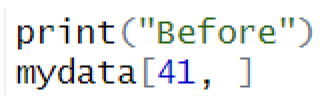
The code head(mydata) displays the first six rows of the mydata data frame, allowing a quick view of the dataset's structure and sample values.



```
> head(mydata)
  id         age    sex  dataset              cp  trestbps       chol
1  1         old   Male Cleveland typical angina 0.7066667 0.03080023
2  2         old   Male Cleveland    asymptomatic 0.7866667 0.04605642
3  3         old   Male Cleveland    asymptomatic 0.5733333 0.02964882
4  4 middle-aged   Male Cleveland     non-anginal 0.6266667 0.03569372
5  5 middle-aged Female Cleveland atypical angina 0.6266667 0.02245250
6  6 middle-aged   Male Cleveland atypical angina 0.5733333 0.03166379
    fbs          restecg    thalch exang   oldpeak       slope ca   thal
1  TRUE lv hypertrophy 0.5438596 FALSE 0.3709677 downsloping  0 normal
2 FALSE lv hypertrophy 0.1754386  TRUE 0.2419355        flat  3 normal
3 FALSE lv hypertrophy 0.3596491  TRUE 0.4193548        flat  2 normal
4 FALSE           normal 0.8684211 FALSE 0.5645161 downsloping  0 normal
5 FALSE lv hypertrophy 0.7368421 FALSE 0.2258065   upsloping  0 normal
6 FALSE           normal 0.7894737 FALSE 0.1290323   upsloping  0 normal
  num
1   0
2   1
3   1
4   0
5   0
6   0
> 
```

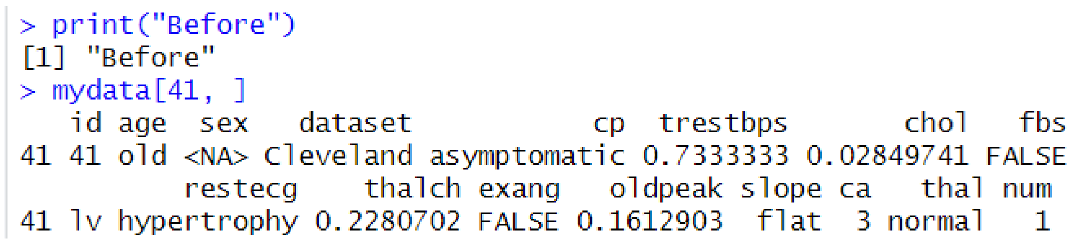
The output shows the top six rows of mydata, giving a snapshot of the current state of the dataset after all transformations and cleaning.

<u>Handle Value :</u>



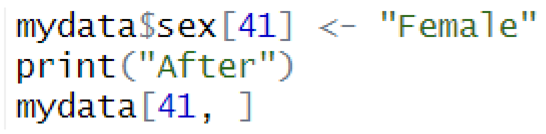
```
print("Before")
mydata[41, ]
```

The code prints the label "Before" and then displays the 41st row of the mydata data frame using mydata[41,], showing all column values for that specific row.



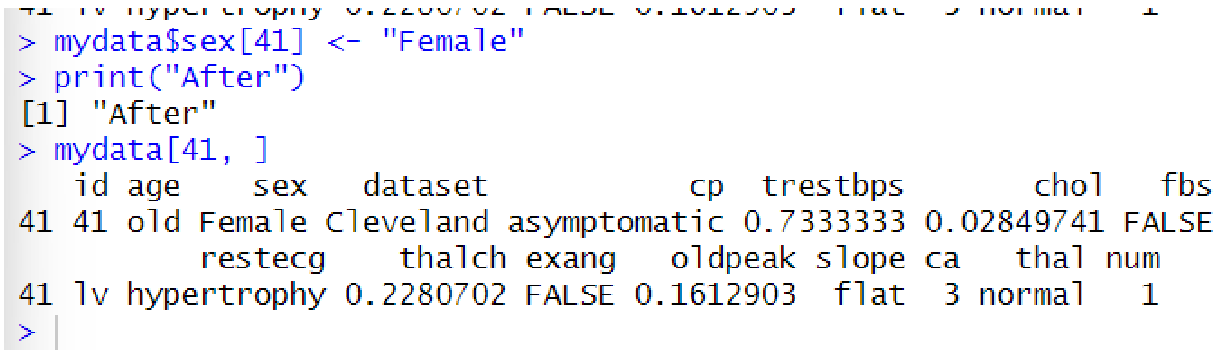
```
> print("Before")
[1] "Before"
> mydata[41, ]
   id age  sex  dataset           cp  trestbps       chol   fbs
41 41 old <NA> Cleveland asymptomatic 0.7333333 0.02849741 FALSE
          restecg   thalch exang   oldpeak slope ca   thal num
41 lv hypertrophy 0.2280702 FALSE 0.1612903  flat  3 normal   1
```

The output shows the text "Before" followed by all the data in row 41 of mydata, providing a view of that row's contents prior to any changes.



```
mydata$sex[41] <- "Female"
print("After")
mydata[41, ]
```

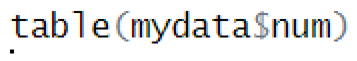
The code updates the sex value in row 41 of the mydata data frame to "Female". It then prints the label "After" and displays the updated 41st row using mydata[41,].



```
> mydata$sex[41] <- "Female"
> print("After")
[1] "After"
> mydata[41, ]
   id age    sex  dataset           cp  trestbps       chol   fbs
41 41 old Female Cleveland asymptomatic 0.7333333 0.02849741 FALSE
          restecg   thalch exang   oldpeak slope ca   thal num
41 lv hypertrophy 0.2280702 FALSE 0.1612903  flat  3 normal   1
> 
```

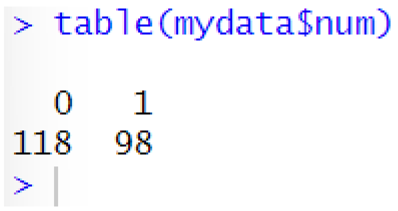
The output shows the text "After" followed by the contents of row 41 in mydata, reflecting the updated value "Female" in the sex column.

<u>Data Imbalance :</u>



```
table(mydata$num)
```

The code table(mydata$num) generates a frequency table of the values in the num column of the mydata data frame, counting how many times each unique value appears.



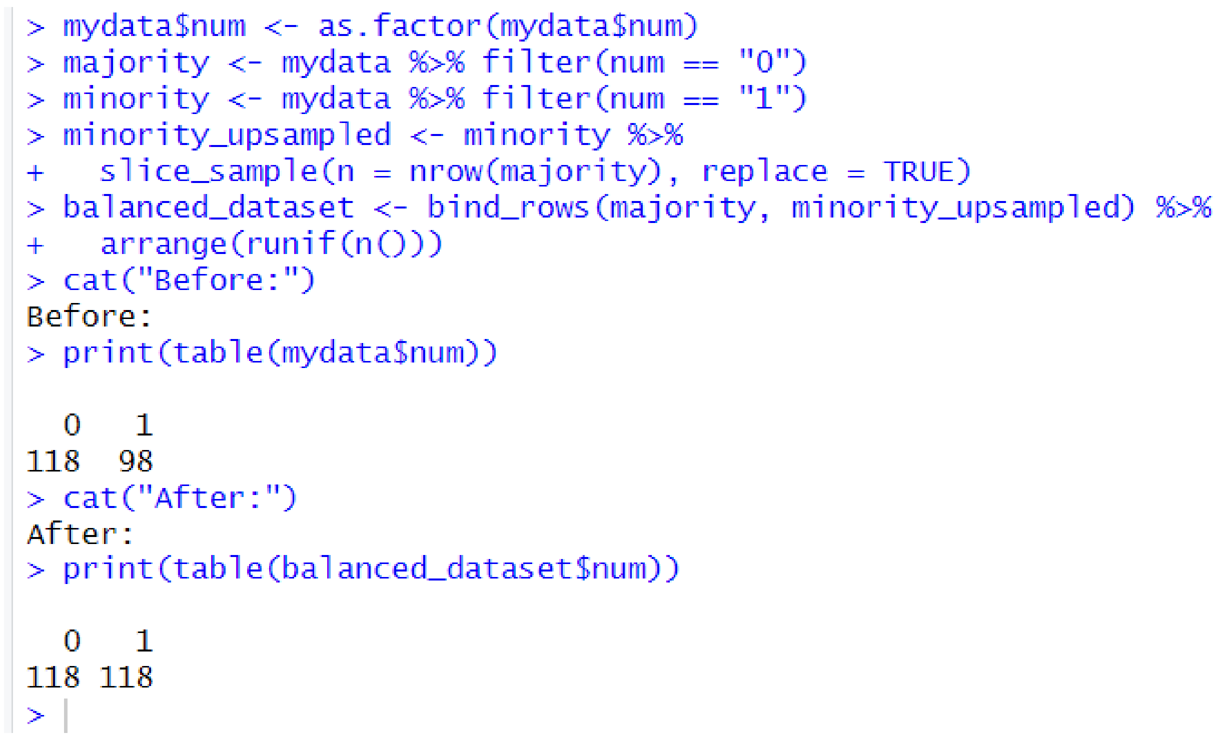
```
> table(mydata$num)

  0   1 
118  98 
> 
```

The output displays the number of occurrences of each unique value in the num column, typically showing how many entries are labeled 0, 1, or NA if missing values are present.

Upsample The Minority Class :

```
mydata$num <- as.factor(mydata$num)

majority <- mydata %>% filter(num == "0")
minority <- mydata %>% filter(num == "1")

minority_upsampled <- minority %>%
  slice_sample(n = nrow(majority), replace = TRUE)


balanced_dataset <- bind_rows(majority, minority_upsampled) %>%
  arrange(runif(n()))


cat("Before:")
print(table(mydata$num))

cat("After:")
print(table(balanced_dataset$num))
```

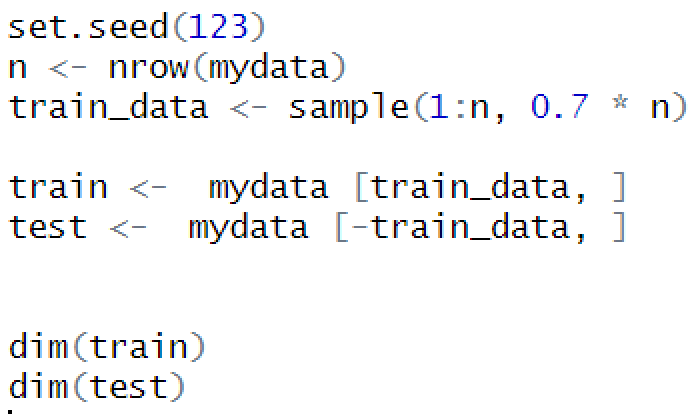
The code first converts the num column in mydata to a factor. It then separates the data into two groups: majority (rows with num == "0") and minority (rows with num == "1"). The minority group is upsampled with replacement to match the size of the majority group using slice_sample(). The balanced dataset is created by combining both groups and shuffling the rows. Finally, it prints the class distribution before and after balancing using table().



```
> mydata$num <- as.factor(mydata$num)
> majority <- mydata %>% filter(num == "0")
> minority <- mydata %>% filter(num == "1")
> minority_upsampled <- minority %>%
+   slice_sample(n = nrow(majority), replace = TRUE)
> balanced_dataset <- bind_rows(majority, minority_upsampled) %>%
+   arrange(runif(n()))
> cat("Before:")
Before:
> print(table(mydata$num))

  0   1 
118  98 
> cat("After:")
After:
> print(table(balanced_dataset$num))

  0   1 
118 118 
> 
```

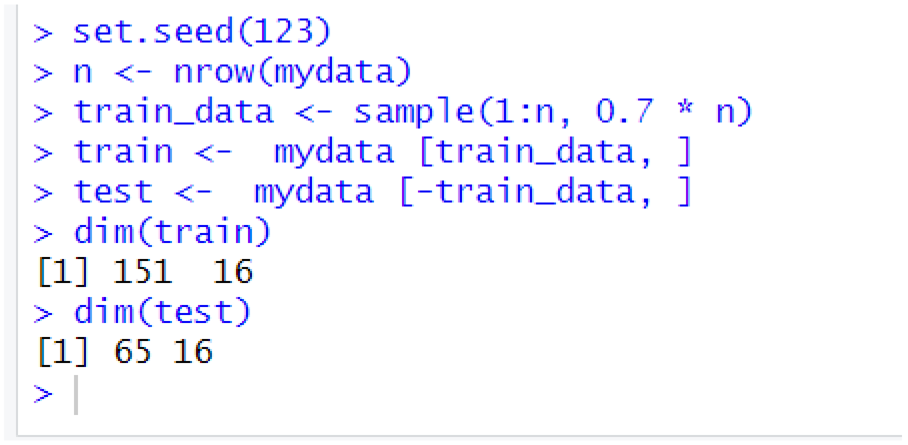
The output first displays the original class distribution in the num column, showing how many entries belong to each class ("0" and "1"). Then, it shows the new distribution in balanced_dataset, where both classes have equal counts, confirming successful upsampling.

Split Dataset :



```
set.seed(123)
n <- nrow(mydata)
train_data <- sample(1:n, 0.7 * n)

train <-  mydata [train_data, ]
test <-  mydata [-train_data, ]


dim(train)
dim(test)
```

The code sets a random seed for reproducibility, calculates the number of rows in mydata, and randomly samples 70% of the indices for training data. It then splits the dataset into train (70%) and test (30%) subsets based on these indices. Finally, it uses dim() to display the dimensions (rows and columns) of both subsets.

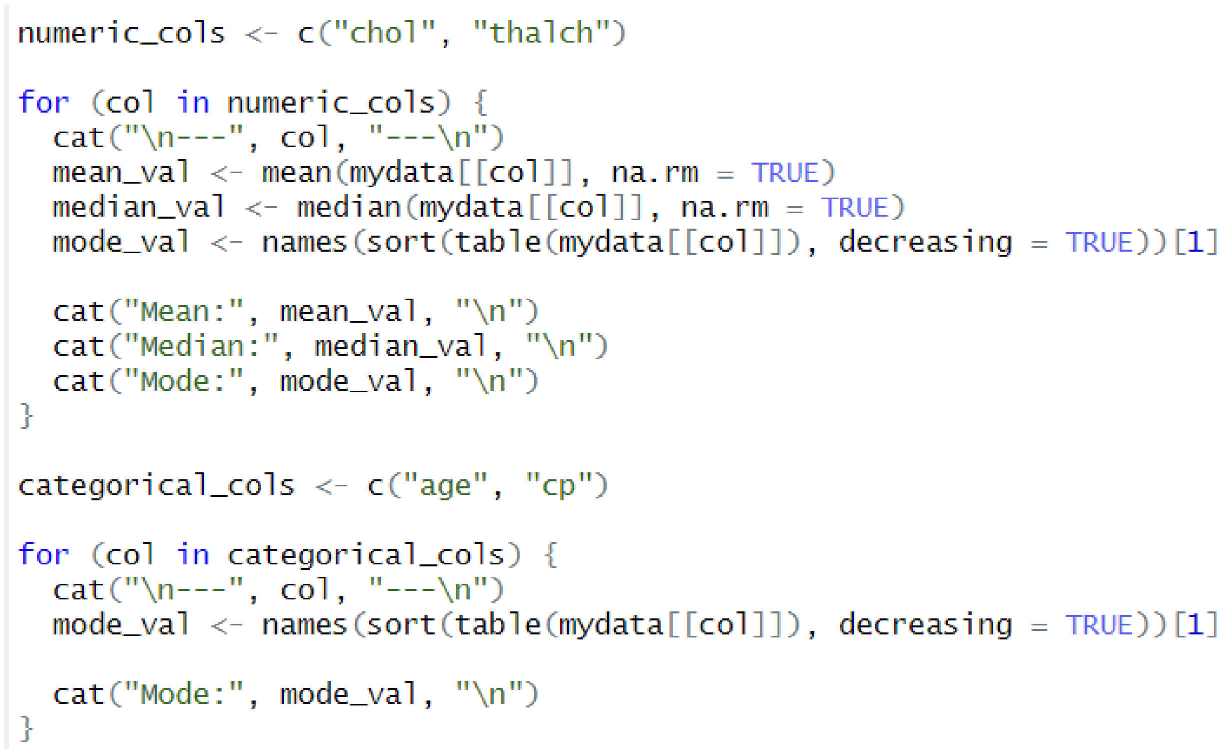


```
> set.seed(123)
> n <- nrow(mydata)
> train_data <- sample(1:n, 0.7 * n)
> train <-  mydata [train_data, ]
> test <-  mydata [-train_data, ]
> dim(train)
[1] 151  16
> dim(test)
[1] 65 16
> 
```

The output shows the dimensions of the train and test datasets, indicating how many rows and columns are present in each after the 70-30 split.

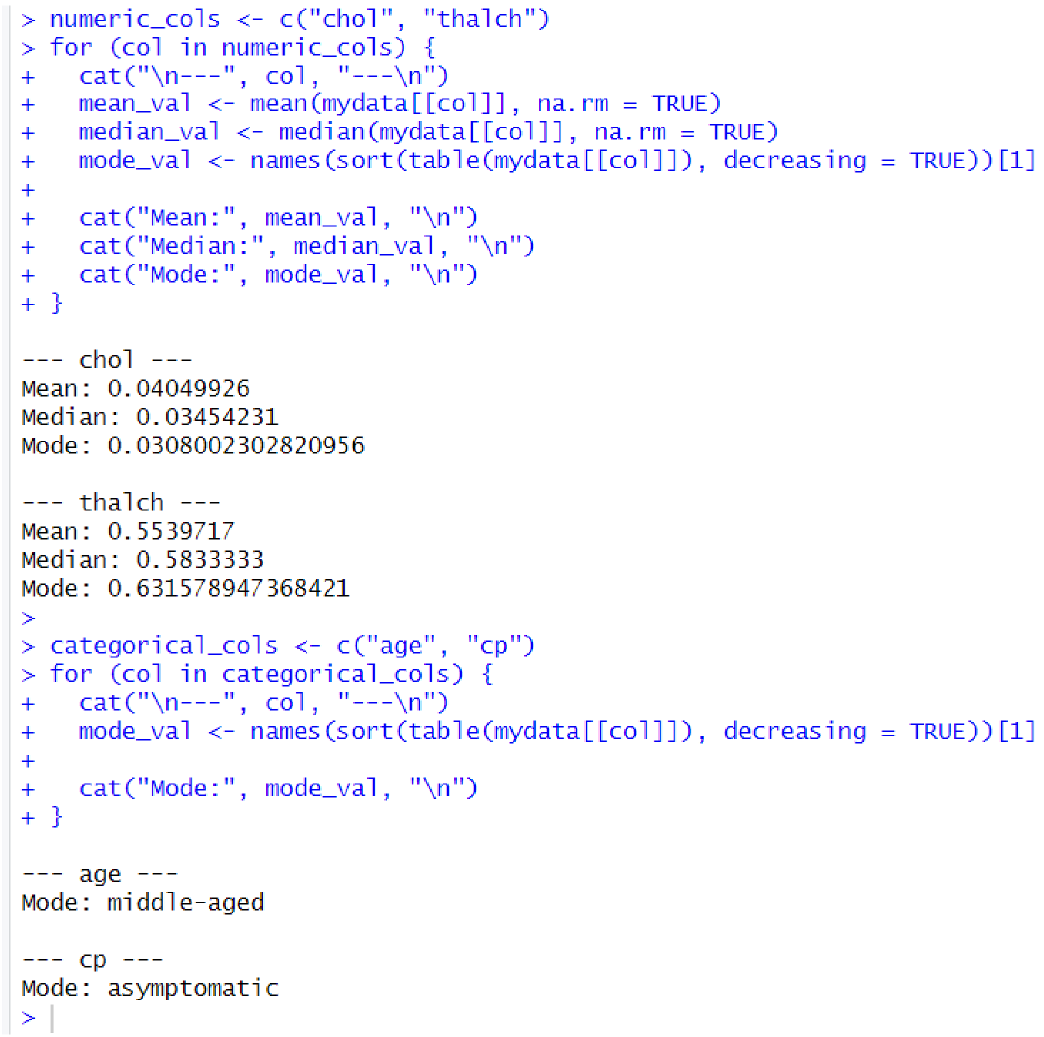
Compute The Central Tendencies (Mean, Median, Mode):

Two numeric columns: 'chol', 'thalch' and Two categorical columns: 'age', 'cp'



```
numeric_cols <- c("chol", "thalch")

for (col in numeric_cols) {
  cat("\n---", col, "---\n")
  mean_val <- mean(mydata[[col]], na.rm = TRUE)
  median_val <- median(mydata[[col]], na.rm = TRUE)
  mode_val <- names(sort(table(mydata[[col]]), decreasing = TRUE))[1]

  cat("Mean:", mean_val, "\n")
  cat("Median:", median_val, "\n")
  cat("Mode:", mode_val, "\n")
}

categorical_cols <- c("age", "cp")

for (col in categorical_cols) {
  cat("\n---", col, "---\n")
  mode_val <- names(sort(table(mydata[[col]]), decreasing = TRUE))[1]

  cat("Mode:", mode_val, "\n")
}
```

The code calculates summary statistics for selected numeric and categorical columns in the mydata data frame. For the numeric columns "chol" and "thalch", it computes and prints the mean, median, and mode. For the categorical columns "age" and "cp", it identifies and prints only the mode (most frequent value). These operations are done using loops for efficiency.



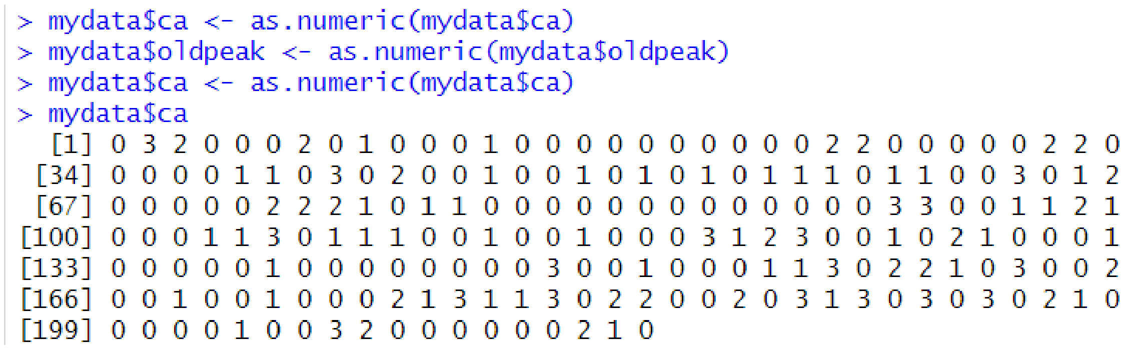
```
> numeric_cols <- c("chol", "thalch")
> for (col in numeric_cols) {
+   cat("\n---", col, "---\n")
+   mean_val <- mean(mydata[[col]], na.rm = TRUE)
+   median_val <- median(mydata[[col]], na.rm = TRUE)
+   mode_val <- names(sort(table(mydata[[col]]), decreasing = TRUE))[1]
+
+   cat("Mean:", mean_val, "\n")
+   cat("Median:", median_val, "\n")
+   cat("Mode:", mode_val, "\n")
+ }

--- chol ---
Mean: 0.04049926
Median: 0.03454231
Mode: 0.0308002302820956

--- thalch ---
Mean: 0.5539717
Median: 0.5833333
Mode: 0.631578947368421
>
> categorical_cols <- c("age", "cp")
> for (col in categorical_cols) {
+   cat("\n---", col, "---\n")
+   mode_val <- names(sort(table(mydata[[col]]), decreasing = TRUE))[1]
+
+   cat("Mode:", mode_val, "\n")
+ }

--- age ---
Mode: middle-aged

--- cp ---
Mode: asymptomatic
>
```

The output displays the mean, median, and mode for the numeric columns "chol" and "thalch", followed by the mode for the categorical columns "age" and "cp", each section clearly labeled by column name.

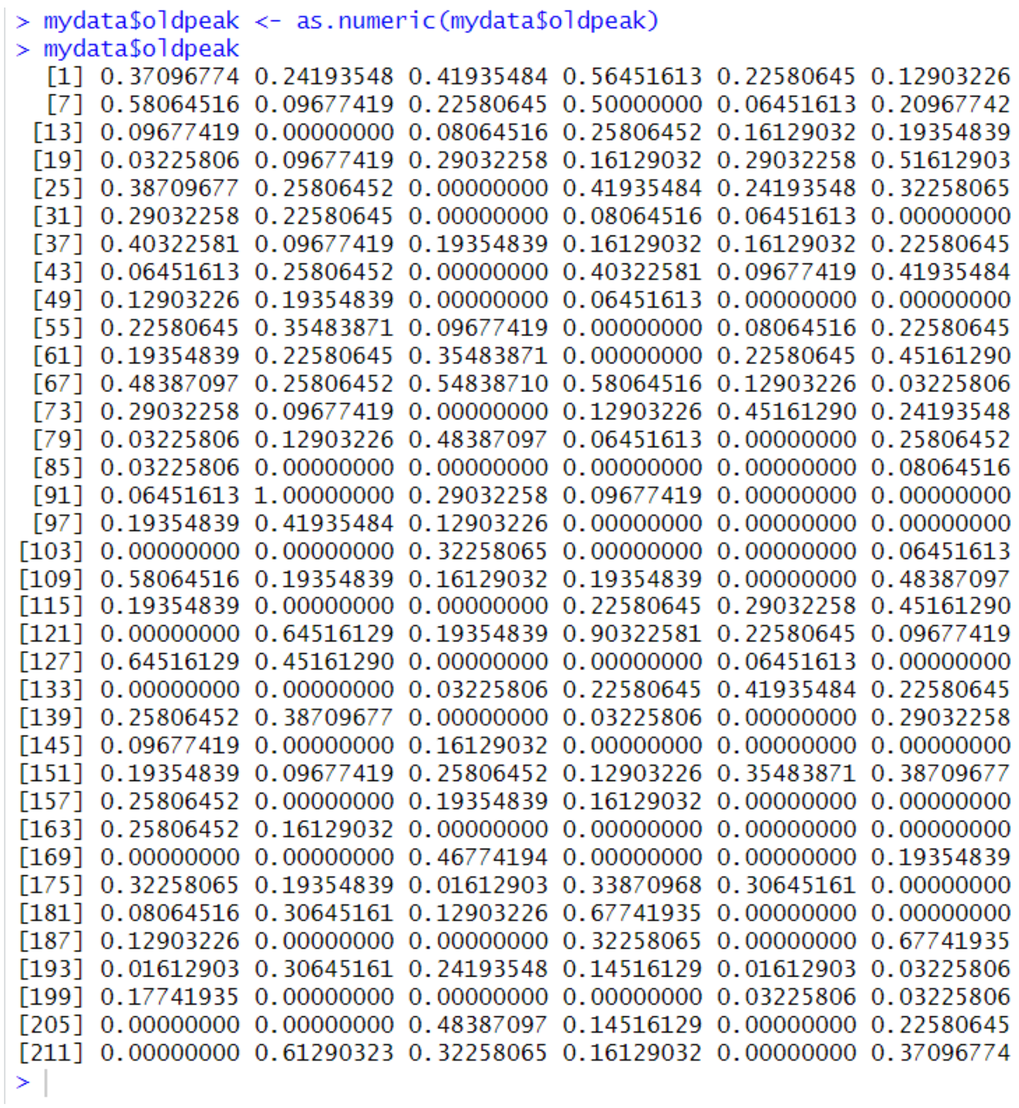
Ensure Columns Are Numeric :

```
mydata$ca <- as.numeric(mydata$ca)
mydata$ca

mydata$oldpeak <- as.numeric(mydata$oldpeak)
mydata$oldpeak
```

The code converts the ca and oldpeak columns in the mydata data frame to numeric data types using as.numeric(), ensuring they can be used in numerical analyses or modeling.



```
> mydata$ca <- as.numeric(mydata$ca)
> mydata$oldpeak <- as.numeric(mydata$oldpeak)
> mydata$ca <- as.numeric(mydata$ca)
> mydata$ca
  [1] 0 3 2 0 0 0 2 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 2 2 0
 [34] 0 0 0 0 1 1 0 3 0 2 0 0 1 0 0 1 0 1 0 1 0 1 1 1 0 1 1 0 0 3 0 1 2
 [67] 0 0 0 0 0 2 2 2 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 0 0 1 1 2 1
[100] 0 0 0 1 1 3 0 1 1 1 0 0 1 0 0 1 0 0 0 3 1 2 3 0 0 1 0 2 1 0 0 0 1
[133] 0 0 0 0 0 1 0 0 0 0 0 0 0 0 3 0 0 1 0 0 0 1 1 3 0 2 2 1 0 3 0 0 2
[166] 0 0 1 0 0 1 0 0 0 2 1 3 1 1 3 0 2 2 0 0 2 0 3 1 3 0 3 0 3 0 2 1 0
[199] 0 0 0 0 1 0 0 3 2 0 0 0 0 0 0 2 1 0
```

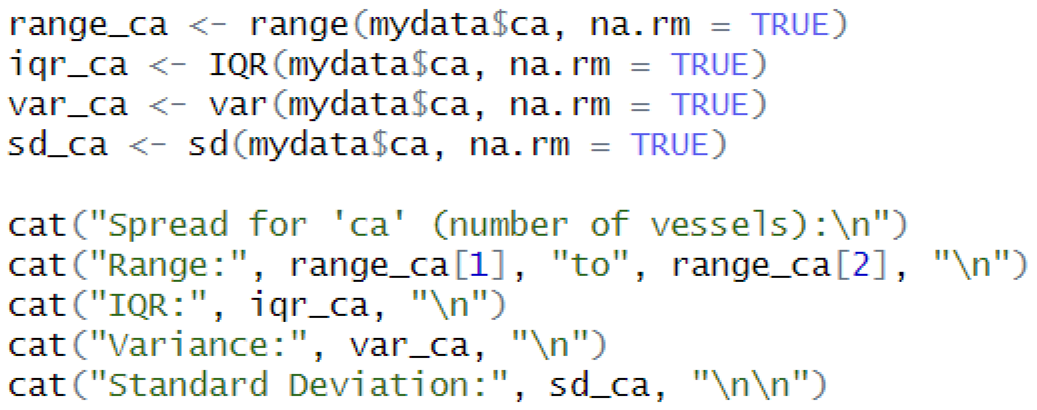


```
> mydata$oldpeak <- as.numeric(mydata$oldpeak)
> mydata$oldpeak
  [1] 0.37096774 0.24193548 0.41935484 0.56451613 0.22580645 0.12903226
  [7] 0.58064516 0.09677419 0.22580645 0.50000000 0.06451613 0.20967742
 [13] 0.09677419 0.00000000 0.08064516 0.25806452 0.16129032 0.19354839
 [19] 0.03225806 0.09677419 0.29032258 0.16129032 0.29032258 0.51612903
 [25] 0.38709677 0.25806452 0.00000000 0.41935484 0.24193548 0.32258065
 [31] 0.29032258 0.22580645 0.00000000 0.08064516 0.06451613 0.00000000
 [37] 0.40322581 0.09677419 0.19354839 0.16129032 0.16129032 0.22580645
 [43] 0.06451613 0.25806452 0.00000000 0.40322581 0.09677419 0.41935484
 [49] 0.12903226 0.19354839 0.00000000 0.06451613 0.00000000 0.00000000
 [55] 0.22580645 0.35483871 0.09677419 0.00000000 0.08064516 0.22580645
 [61] 0.19354839 0.22580645 0.35483871 0.00000000 0.22580645 0.45161290
 [67] 0.48387097 0.25806452 0.54838710 0.58064516 0.12903226 0.03225806
 [73] 0.29032258 0.09677419 0.00000000 0.12903226 0.45161290 0.24193548
 [79] 0.03225806 0.12903226 0.48387097 0.06451613 0.00000000 0.25806452
 [85] 0.03225806 0.00000000 0.00000000 0.00000000 0.00000000 0.08064516
 [91] 0.06451613 1.00000000 0.29032258 0.09677419 0.00000000 0.00000000
 [97] 0.19354839 0.41935484 0.12903226 0.00000000 0.00000000 0.00000000
[103] 0.00000000 0.00000000 0.32258065 0.00000000 0.00000000 0.06451613
[109] 0.58064516 0.19354839 0.16129032 0.19354839 0.00000000 0.48387097
[115] 0.19354839 0.00000000 0.00000000 0.22580645 0.29032258 0.45161290
[121] 0.00000000 0.64516129 0.19354839 0.90322581 0.22580645 0.09677419
[127] 0.64516129 0.45161290 0.00000000 0.00000000 0.06451613 0.00000000
[133] 0.00000000 0.00000000 0.03225806 0.22580645 0.41935484 0.22580645
[139] 0.25806452 0.38709677 0.00000000 0.03225806 0.00000000 0.29032258
[145] 0.09677419 0.00000000 0.16129032 0.00000000 0.00000000 0.00000000
[151] 0.19354839 0.09677419 0.25806452 0.12903226 0.35483871 0.38709677
[157] 0.25806452 0.00000000 0.19354839 0.16129032 0.00000000 0.00000000
[163] 0.25806452 0.16129032 0.00000000 0.00000000 0.00000000 0.00000000
[169] 0.00000000 0.00000000 0.46774194 0.00000000 0.00000000 0.19354839
[175] 0.32258065 0.19354839 0.01612903 0.33870968 0.30645161 0.00000000
[181] 0.08064516 0.30645161 0.12903226 0.67741935 0.00000000 0.00000000
[187] 0.12903226 0.00000000 0.00000000 0.32258065 0.00000000 0.67741935
[193] 0.01612903 0.30645161 0.24193548 0.14516129 0.01612903 0.03225806
[199] 0.17741935 0.00000000 0.00000000 0.00000000 0.03225806 0.03225806
[205] 0.00000000 0.00000000 0.48387097 0.14516129 0.00000000 0.22580645
[211] 0.00000000 0.61290323 0.32258065 0.16129032 0.00000000 0.37096774
> |
```

Both ca and oldpeak columns are now stored as numeric variables in mydata.

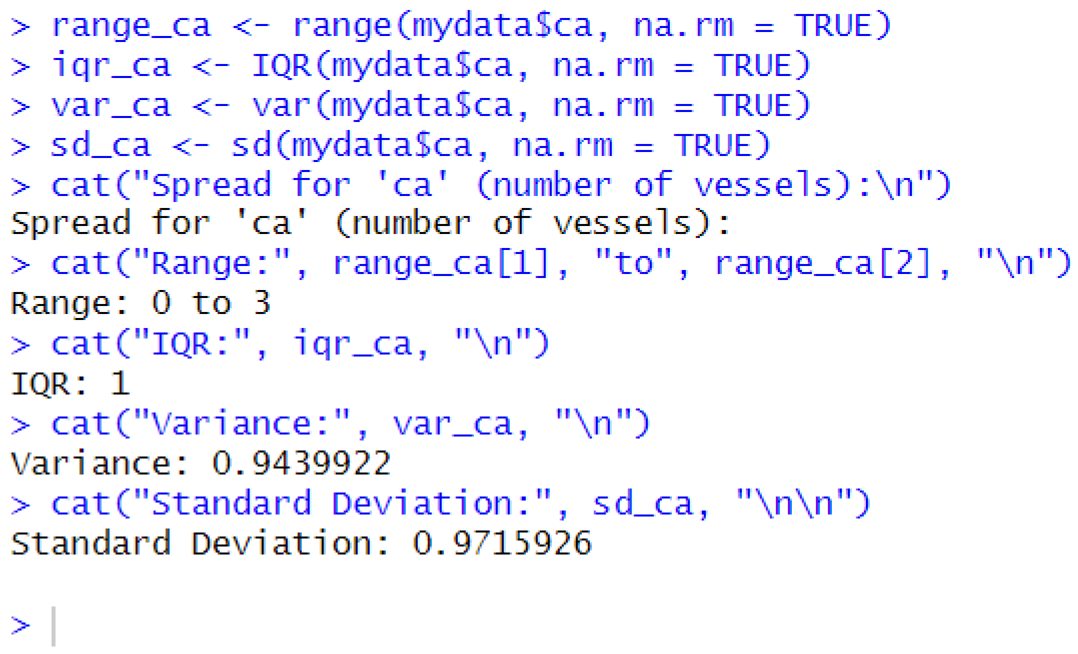
<u>Compute The Spread (Range, IQR, Variance, Standard Deviation) For Two Attributes:</u>

<u>Attribute: ca</u>



```
range_ca <- range(mydata$ca, na.rm = TRUE)
iqr_ca <- IQR(mydata$ca, na.rm = TRUE)
var_ca <- var(mydata$ca, na.rm = TRUE)
sd_ca <- sd(mydata$ca, na.rm = TRUE)

cat("Spread for 'ca' (number of vessels):\n")
cat("Range:", range_ca[1], "to", range_ca[2], "\n")
cat("IQR:", iqr_ca, "\n")
cat("Variance:", var_ca, "\n")
cat("Standard Deviation:", sd_ca, "\n\n")
```

This code calculates the spread of the variable ca (number of vessels) from the dataset mydata. It computes the range, interquartile range (IQR), variance, and standard deviation, while excluding any missing values (na.rm = TRUE). The results are then printed to the console.



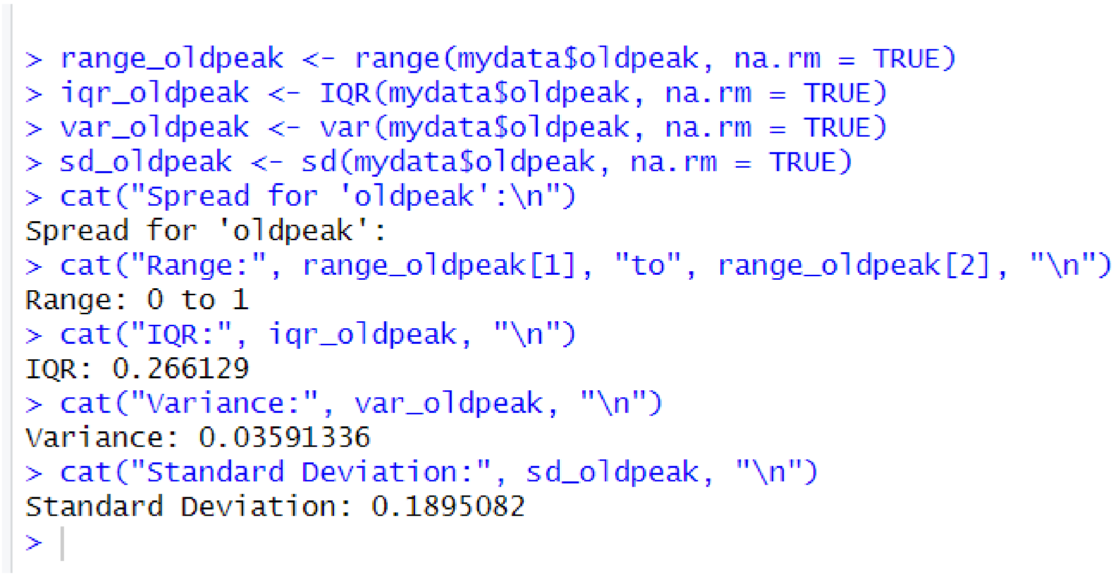
```
> range_ca <- range(mydata$ca, na.rm = TRUE)
> iqr_ca <- IQR(mydata$ca, na.rm = TRUE)
> var_ca <- var(mydata$ca, na.rm = TRUE)
> sd_ca <- sd(mydata$ca, na.rm = TRUE)
> cat("Spread for 'ca' (number of vessels):\n")
Spread for 'ca' (number of vessels):
> cat("Range:", range_ca[1], "to", range_ca[2], "\n")
Range: 0 to 3
> cat("IQR:", iqr_ca, "\n")
IQR: 1
> cat("Variance:", var_ca, "\n")
Variance: 0.9439922
> cat("Standard Deviation:", sd_ca, "\n\n")
Standard Deviation: 0.9715926

> |
```

The output displays the range (minimum and maximum values), IQR, variance, and standard deviation of the ca variable, which provides insights into its distribution and variability.

Attribute: oldpeak

```r
range_oldpeak <- range(mydata$oldpeak, na.rm = TRUE)
iqr_oldpeak <- IQR(mydata$oldpeak, na.rm = TRUE)
var_oldpeak <- var(mydata$oldpeak, na.rm = TRUE)
sd_oldpeak <- sd(mydata$oldpeak, na.rm = TRUE)

cat("Spread for 'oldpeak':\n")
cat("Range:", range_oldpeak[1], "to", range_oldpeak[2], "\n")
cat("IQR:", iqr_oldpeak, "\n")
cat("Variance:", var_oldpeak, "\n")
cat("Standard Deviation:", sd_oldpeak, "\n")
```

This code calculates the spread of the variable oldpeak from the dataset mydata. It computes the range, interquartile range (IQR), variance, and standard deviation, excluding missing values (na.rm = TRUE). The results are then printed for easy interpretation.



```
> range_oldpeak <- range(mydata$oldpeak, na.rm = TRUE)
> iqr_oldpeak <- IQR(mydata$oldpeak, na.rm = TRUE)
> var_oldpeak <- var(mydata$oldpeak, na.rm = TRUE)
> sd_oldpeak <- sd(mydata$oldpeak, na.rm = TRUE)
> cat("Spread for 'oldpeak':\n")
Spread for 'oldpeak':
> cat("Range:", range_oldpeak[1], "to", range_oldpeak[2], "\n")
Range: 0 to 1
> cat("IQR:", iqr_oldpeak, "\n")
IQR: 0.266129
> cat("Variance:", var_oldpeak, "\n")
Variance: 0.03591336
> cat("Standard Deviation:", sd_oldpeak, "\n")
Standard Deviation: 0.1895082
> 
```

The output shows the range (minimum and maximum values), IQR, variance, and standard deviation of the oldpeak variable, offering a detailed view of its distribution and variability.