AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Choose an item.

Assignment Cover Sheet

Assignment Title:	Mid Term Project		
Assignment No:	01	Date of Submission:	18 March 2024
Course Title:	Introduction to Data Science		
Course Code:	CSC4180	Section:	C
Semester:	Spring 2023-24	Course Teacher:	TOHEDUL ISLAM

Declaration and Statement of Authorship:

1. I/we hold a copy of this Assignment/Case-Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case-Study is my/our original work and no part of it has been copied from any other student's work or from any other source except where due acknowledgement is made.
3. No part of this Assignment/Case-Study has been written for me/us by any other person except where such collaborationhas been authorized by the concerned teacher and is clearly acknowledged in the assignment.
4. I/we have not previously submitted or currently submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared and archived for the purpose of detecting plagiarism.
6. I/we give permission for a copy of my/our marked work to be retained by the Faculty for review and comparison, including review by external examiners.
7. I/we understand thatPlagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a formofcheatingandisaveryseriousacademicoffencethatmayleadtoexpulsionfromtheUniversity. Plagiarized material can be drawn from, and presented in, written, graphic and visual form, including electronic data, and oral presentations. Plagiarism occurs when the origin of them arterial used is not appropriately cited.
8. I/we also understand that enabling plagiarism is the act of assisting or allowing another person to plagiarize or to copy my/our work.

* *Student(s) must complete all details except the faculty use part.*
** Please submit all assignments to your course teacher or the office of the concerned teacher.

Group Name/No.: 10

No	Name	ID	Program	Signature
1	Tarikul Islam Nishat	21-44632-1	CSE	
2	Mohammad Rafiul Haque	21-44631-1	CSE	
3			Choose an item.	
4			Choose an item.	
5			Choose an item.	
6			Choose an item.	
7			Choose an item.	
8			Choose an item.	
9			Choose an item.	
10			Choose an item.	

Faculty use only		
FACULTYCOMMENTS	**Marks Obtained**	
	Total Marks	

Description about the dataset:

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

The dataset contains the following attributes:

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

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

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

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

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

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

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

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

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

Importing the dataset

Code:

```
dataset <- read.csv("D:/10th Semester/DS/pj/Dataset_midterm_section(C).csv",
header=TRUE, sep=",")
dataset
```

Output:

```
> dataset <- read.csv("D:/10th Semester/DS/pj/Dataset_midterm_section(C).csv", header=TRUE, sep=",")
> dataset
   Age Infection Smoking SystolicBP DiastolicBP    BS BodyTemp HeartRate RiskLevel  X X.1         X.2
1   25       yes       1        130          80 15.00       98        86 high risk NA  NA
2   35       yes       1        140          90 13.00       98        70 high risk NA  NA     Smoking
3   29       yes       1         90          70  8.00      100        80 high risk NA  NA       1=yes
4   30       yes       1        140          85  7.00       98        70 high risk NA  NA 2=sometimes
5   35        no       3        120          60  6.10       98        76  low risk NA  NA        3=no
6   23       yes       1        140          80  7.01       98        70 high risk NA  NA
7   23                 2        130          70  7.01       98        78  mid risk NA  NA
8   NA       yes       1         85          60 11.00      102        86 high risk NA  NA
9   32  marginal       2        120          90  6.90       98        70  mid risk NA  NA
10  42       yes       1        130          80 18.00       98        70 high risk NA  NA
11  23        no       3         90          60  7.01       98        76  low risk NA  NA
12  19  marginal       2        120          80  7.00       98        70  mid risk NA  NA
13  25        no       3        110          89  7.01       98        77  low risk NA  NA
14  20  marginal      NA        120          75  7.01      100        70  mid risk NA  NA
15  48  marginal       2        120          80 11.00       98        88  mid risk NA  NA
16  15        no       3        120          NA  7.01       98        70  low risk NA  NA
17  50       yes       1        140          90 15.00       98        90 high risk NA  NA
18  25       yes       1        140         100  7.01       98        80 high risk NA  NA
19  30  marginal       2        120          80  6.90      101        76  mid risk NA  NA
20  10        no       3         70          50  6.90       98        70  low risk NA  NA
```

Explanation:

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

Column Names:

Code:

```
names(dataset)
```

Output:

```
> names(dataset)
[1] "Age"         "Infection"   "Smoking"     "SystolicBP"  "DiastolicBP" "BS"          "BodyTemp"    "HeartRate"   "RiskLevel"
> |
```

Explanation:

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

Data Annotation:

Code:

```
dataset$Infection<-factor(dataset$Infection,levels = c("yes","no","marginal"),labels = c(1,2,3))

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

dataset
```

Output:

```
> dataset$RiskLevel<-factor(dataset$RiskLevel,levels = c("high risk","low risk","mid risk"),labels = c(1,2,3))
> dataset
   Age Infection Smoking SystolicBP DiastolicBP    BS BodyTemp HeartRate RiskLevel
1   25         1       1        130          80 15.00       98        86         1
2   35         1       1        140          90 13.00       98        70         1
3   29         1       1         90          70  8.00      100        80         1
4   30         1       1        140          85  7.00       98        70         1
5   35         2       3        120          60  6.10       98        76         2
6   23         1       1        140          80  7.01       98        70         1
7   23      <NA>       2        130          70  7.01       98        78         3
8   NA         1       1         85          60 11.00      102        86         1
9   32         3       2        120          90  6.90       98        70         3
10  42         1       1        130          80 18.00       98        70         1
11  23         2       3         90          60  7.01       98        76         2
12  19         3       2        120          80  7.00       98        70         3
13  25         2       3        110          89  7.01       98        77         2
14  20         3      NA        120          75  7.01      100        70         3
15  48         3       2        120          80 11.00       98        88         3
16  15         2       3        120          NA  7.01       98        70         2
17  50         1       1        140          90 15.00       98        90         1
18  25         1       1        140         100  7.01       98        80         1
19  30         3       2        120          80  6.90      101        76         3
20  10         2       3         70          50  6.90       98        70         2
21  40         1       1        140         100 18.00       98        90         1
22  50         3       2        140          80  6.70       98        70         3
23  21         2       3         90          65  7.50       98        76         2
24  18         2       3         90          60  7.50       98        70         2
25  NA         2       3        120          80  7.50       98        76         2
26  16         2       3        100          70  7.20       98        80         2
```

Explanation:

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

- `dataset$RiskLevel <- factor(dataset$RiskLevel, levels = c("high risk", "low risk", "mid risk"), labels = c(1, 2, 3))`: Similarly, this line converts the column RiskLevel in the dataset into a factor variable. The levels "high risk", "low risk", and "mid risk" are specified, and corresponding labels 1, 2, and 3 are assigned to them.

Summary of the structure of data set:

Code:

```
str(dataset)
```

Output:

```
> str(dataset)
'data.frame':   200 obs. of  9 variables:
 $ Age        : int  25 35 29 30 35 23 23 NA 32 42 ...
 $ Infection  : Factor w/ 3 levels "1","2","3": 1 1 1 1 2 1 NA 1 3 1 ...
 $ Smoking    : int  1 1 1 1 3 1 2 1 2 1 ...
 $ SystolicBP : int  130 140 90 140 120 140 130 85 120 130 ...
 $ DiastolicBP: int  80 90 70 85 60 80 70 60 90 80 ...
 $ BS         : num  15 13 8 7 6.1 7.01 7.01 11 6.9 18 ...
 $ BodyTemp   : int  98 98 100 98 98 98 98 102 98 98 ...
 $ HeartRate  : int  86 70 80 70 76 70 78 86 70 70 ...
 $ RiskLevel  : Factor w/ 3 levels "1","2","3": 1 1 1 1 2 1 3 1 3 1 ...
> 
```

Explanation:

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

Descriptive Statistics Using summary() Function:

Code:

```
summary(dataset)
```

Output:

```
> summary(dataset)
      Age          Infection    Smoking          SystolicBP      DiastolicBP          BS            BodyTemp          HeartRate      RiskLevel
 Min.   : 10.00   1    :61    Min.   :1.000   Min.   : 70.0   Min.   : 49.00   Min.   : 6.000   Min.   :-160.00   Min.   :60.00   1:65
 1st Qu.: 21.00   2    :77    1st Qu.:1.000   1st Qu.:100.0   1st Qu.: 65.00   1st Qu.: 6.875   1st Qu.:  98.00   1st Qu.:70.00   2:81
 Median : 25.00   3    :52    Median :2.000   Median :120.0   Median : 80.00   Median : 7.150   Median :  98.00   Median :76.00   3:54
 Mean   : 31.97   NA's:10     Mean   :2.077   Mean   :114.8   Mean   : 78.32   Mean   : 8.831   Mean   :  95.94   Mean   :74.89
 3rd Qu.: 40.00               3rd Qu.:3.000   3rd Qu.:130.0   3rd Qu.: 90.00   3rd Qu.: 8.000   3rd Qu.:  98.00   3rd Qu.:80.00
 Max.   :170.00               Max.   :3.000   Max.   :160.0   Max.   :100.00   Max.   :19.000   Max.   : 103.00   Max.   :90.00
 NA's   :5                    NA's   :4                        NA's   :4
> 
```

Explanation:

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

Standard deviation:

Code:

```
dataset %>% summarise_if(is.numeric, sd)
```

Output:

```
> dataset %>% summarise_if(is.numeric, sd)
  Age Smoking SystolicBP DiastolicBP       BS BodyTemp HeartRate
1  NA      NA   19.54269          NA 3.662789 25.31591   7.85815
> |
```

Explanation:

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

Null values in each column:

Code:

```
colSums(is.na(dataset))
```

Output:

```
> colSums(is.na(dataset))
        Age   Infection     Smoking  SystolicBP DiastolicBP          BS    BodyTemp   HeartRate   RiskLevel
          5          10           4           0           4           0           0           0           0
> |
```

Explanation:

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

Specific row number of Null values:

Code:

```
which(is.na(dataset$Age))
which(is.na(dataset$Infection))
```

```
which(is.na(dataset$Smoking))
which(is.na(dataset$SystolicBP))
which(is.na(dataset$DiastolicBP ))
which(is.na(dataset$BS))
which(is.na(dataset$BodyTemp))
which(is.na(dataset$HeartRate))
which(is.na(dataset$RiskLevel))
```

Output:

```
> which(is.na(dataset$Age))
[1]   8  25  40  65 101
> which(is.na(dataset$Infection))
 [1]   7  27  59  69 107 117 151 153 158 180
> which(is.na(dataset$Smoking))
[1]  14  34  61 103
> which(is.na(dataset$SystolicBP))
integer(0)
> which(is.na(dataset$DiastolicBP ))
[1]  16  39  79 103
> which(is.na(dataset$BS))
integer(0)
> which(is.na(dataset$BodyTemp))
integer(0)
> which(is.na(dataset$HeartRate))
integer(0)
> which(is.na(dataset$RiskLevel))
integer(0)
> 
```

Explanation:

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

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

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

Remove null values from data set:

Code:

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

Output:

```
> dataset <- na.omit(dataset)
> dataset
   Age Infection Smoking SystolicBP DiastolicBP    BS BodyTemp HeartRate RiskLevel
1   25         1       1        130          80 15.00       98        86         1
2   35         1       1        140          90 13.00       98        70         1
3   29         1       1         90          70  8.00      100        80         1
4   30         1       1        140          85  7.00       98        70         1
5   35         2       3        120          60  6.10       98        76         2
6   23         1       1        140          80  7.01       98        70         1
9   32         3       2        120          90  6.90       98        70         3
10  42         1       1        130          80 18.00       98        70         1
11  23         2       3         90          60  7.01       98        76         2
12  19         3       2        120          80  7.00       98        70         3
13  25         2       3        110          89  7.01       98        77         2
15  48         3       2        120          80 11.00       98        88         3
17  50         1       1        140          90 15.00       98        90         1
18  25         1       1        140         100  7.01       98        80         1
19  30         3       2        120          80  6.90      101        76         3
20  10         2       3         70          50  6.90       98        70         2
21  40         1       1        140         100 18.00       98        90         1
22  50         3       2        140          80  6.70       98        70         3
23  21         2       3         90          65  7.50       98        76         2
24  18         2       3         90          60  7.50       98        70         2
26  16         2       3        100          70  7.20       98        80         2
28  22         2       3        100          65  7.20       98        70         2
29  49         2       3        120          90  7.20       98        77         2
30  28         2       3         90          60  7.20     -150        82         2
31  20         2       3        100          90  7.10       98        88         2
32  23         2       3        100          85  7.10       98        66         2
```

Explanation:

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

Replacing Na value with mean :

Code

```
filled_data_mean <- dataset
filled_data_mean$Age              <-              ifelse(is.na(filled_data_mean$Age),
mean(filled_data_mean$Age, na.rm = TRUE), filled_data_mean$Age)
filled_data_mean$Infection<-             ifelse(is.na(filled_data_mean$Infection),
mean(filled_data_mean$Infection, na.rm = TRUE), filled_data_mean$Infection)
filled_data_mean$Smoking<-                 ifelse(is.na(filled_data_mean$Smoking),
mean(filled_data_mean$Smoking, na.rm = TRUE), filled_data_mean$Smoking)
filled_data_mean$DiastolicBP<-         ifelse(is.na(filled_data_mean$DiastolicBP),
mean(filled_data_mean$DiastolicBP, na.rm = TRUE), filled_data_mean$DiastolicBP)
```

```
filled_data_mean$BS<- ifelse(is.na(filled_data_mean$BS), mean(filled_data_mean$BS,
na.rm = TRUE), filled_data_mean$BS)
print(filled_data_mean)
```

Output:

```
> filled_data_mean$Smoking<- ifelse(is.na(filled_data_mean$Smoking), mean(filled_data_mean$Smoking, na.rm = TRUE), filled_da
ta_mean$Smoking)
> filled_data_mean$DiastolicBP<- ifelse(is.na(filled_data_mean$DiastolicBP), mean(filled_data_mean$DiastolicBP, na.rm = TRU
E), filled_data_mean$DiastolicBP)
> print(filled_data_mean)
        Age Infection  Smoking SystolicBP DiastolicBP    BS BodyTemp HeartRate RiskLevel
1  25.00000         1 1.000000        130    80.00000 15.00       98        86         1
2  35.00000         1 1.000000        140    90.00000 13.00       98        70         1
3  29.00000         1 1.000000         90    70.00000  8.00      100        80         1
4  30.00000         1 1.000000        140    85.00000  7.00       98        70         1
5  35.00000         2 3.000000        120    60.00000  6.10       98        76         2
6  23.00000         1 1.000000        140    80.00000  7.01       98        70         1
7  23.00000        NA 2.000000        130    70.00000  7.01       98        78         3
8  31.96923         1 1.000000         85    60.00000 11.00      102        86         1
9  32.00000         3 2.000000        120    90.00000  6.90       98        70         3
10 42.00000         1 1.000000        130    80.00000 18.00       98        70         1
11 23.00000         2 3.000000         90    60.00000  7.01       98        76         2
12 19.00000         3 2.000000        120    80.00000  7.00       98        70         3
13 25.00000         2 3.000000        110    89.00000  7.01       98        77         2
14 20.00000         3 2.076531        120    75.00000  7.01      100        70         3
15 48.00000         3 2.000000        120    80.00000 11.00       98        88         3
16 15.00000         2 3.000000        120    78.31633  7.01       98        70         2
17 50.00000         1 1.000000        140    90.00000 15.00       98        90         1
18 25.00000         1 1.000000        140   100.00000  7.01       98        80         1
19 30.00000         3 2.000000        120    80.00000  6.90      101        76         3
20 10.00000         2 3.000000         70    50.00000  6.90       98        70         2
```

Explanation:

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

- `filled_data_mean <- dataset`: This line creates a new dataset called filled_data_mean, which is a copy of the original dataset.
- `filled_data_mean$Age <- ifelse(is.na(filled_data_mean$Age) mean(filled_data_mean$Age, na.rm = TRUE) filled_data_mean$Age)`: These lines checks if there are missing values (NA) in the "`Age`" column of the `filled_data_mean` dataset. If there are missing values, it replaces them with the mean of the "`Age`" column (calculated using `mean()` function) while ignoring NA values (`na.rm = TRUE`). If there are no missing values, it leaves the original values unchanged.
- Similarly, the next lines perform the same operation for other columns in the dataset:
  `filled_data_mean$Infection`
  `filled_data_mean$Smoking`
  `filled_data_mean$DiastolicBP`
  `filled_data_mean$BS`

- `print(filled_data_mean):`

  Finally, this line prints the modified dataset filled_data_mean with missing values replaced by their respective column means.

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

Replacing Na value with median:

Code:

```
filled_data_median <- dataset
filled_data_median$Age <- ifelse(is.na(filled_data_median$Age), median(filled_data_median$Age, na.rm = TRUE), filled_data_median$Age)
filled_data_median$Infection <- ifelse(is.na(filled_data_median$Infection), median(filled_data_median$Infection, na.rm = TRUE), filled_data_median$Infection)
filled_data_median$Smoking <- ifelse(is.na(filled_data_median$Smoking), median(filled_data_median$Smoking, na.rm = TRUE), filled_data_median$Smoking)
filled_data_median$DiastolicBP <- ifelse(is.na(filled_data_median$DiastolicBP), median(filled_data_median$DiastolicBP, na.rm = TRUE), filled_data_median$DiastolicBP)
print(filled_data_median)
```

Output:

```
> filled_data_median$SystolicBP <- ifelse(is.na(filled_data_median$SystolicBP), median(filled_data_median$SystolicBP, na.rm = TRUE), filled_data_median$SystolicBP)
> filled_data_median$DiastolicBP <- ifelse(is.na(filled_data_median$DiastolicBP), median(filled_data_median$DiastolicBP, na.rm = TRUE), filled_data_median$DiastolicBP)
> filled_data_median$BS <- ifelse(is.na(filled_data_median$BS), median(filled_data_median$BS, na.rm = TRUE), filled_data_median$BS)
> filled_data_median$BodyTemp <- ifelse(is.na(filled_data_median$BodyTemp), median(filled_data_median$BodyTemp, na.rm = TRUE), filled_data_median$BodyTemp)
> filled_data_median$HeartRate <- ifelse(is.na(filled_data_median$HeartRate), median(filled_data_median$HeartRate, na.rm = TRUE), filled_data_median$HeartRate)
> filled_data_median$RiskLevel <- ifelse(is.na(filled_data_median$RiskLevel), median(filled_data_median$RiskLevel, na.rm = TRUE), filled_data_median$RiskLevel)
> print(filled_data_median)
   Age Infection Smoking SystolicBP DiastolicBP    BS BodyTemp HeartRate RiskLevel
1   25         1       1        130          80 15.00       98        86         1
2   35         1       1        140          90 13.00       98        70         1
3   29         1       1         90          70  8.00      100        80         1
4   30         1       1        140          85  7.00       98        70         1
5   35         2       3        120          60  6.10       98        76         2
6   23         1       1        140          80  7.01       98        70         1
7   23      <NA>       2        130          70  7.01       98        78         3
8   25         1       1         85          60 11.00      102        86         1
9   32         3       2        120          90  6.90       98        70         3
10  42         1       1        130          80 18.00       98        70         1
11  23         2       3         90          60  7.01       98        76         2
12  19         3       2        120          80  7.00       98        70         3
13  25         2       3        110          89  7.01       98        77         2
14  20         3       2        120          75  7.01      100        70         3
15  48         3       2        120          80 11.00       98        88         3
16  15         2       3        120          80  7.01       98        70         2
17  50         1       1        140          90 15.00       98        90         1
18  25         1       1        140         100  7.01       98        80         1
19  30         3       2        120          80  6.90      101        76         3
20  10         2       3         70          50  6.90       98        70         2
```

Explanation:

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

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

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

Replacing Na value with mode :

Code:

```
filled_data_mode <- dataset

filled_data_mode$Age <- ifelse(is.na(filled_data_mode$Age),
                               names(sort(table(filled_data_mode$Age), decreasing = TRUE))[1],
                               filled_data_mode$Age)
filled_data_mode$Infection  <- ifelse(is.na(filled_data_mode$Infection ),
                               names(sort(table(filled_data_mode$Infection ), decreasing = TRUE))[1],
                               filled_data_mode$Infection )
filled_data_mode$Smoking   <- ifelse(is.na(filled_data_mode$Smoking  ),
                                      names(sort(table(filled_data_mode$Smoking      ),   decreasing   =
TRUE))[1],
                                      filled_data_mode$Smoking  )

filled_data_mode$SystolicBP <- ifelse(is.na(filled_data_mode$SystolicBP),
                                      names(sort(table(filled_data_mode$SystolicBP),    decreasing    =
TRUE))[1],
```

```
                                      filled_data_mode$SystolicBP)

filled_data_mode$DiastolicBP <- ifelse(is.na(filled_data_mode$DiastolicBP),
                                       names(sort(table(filled_data_mode$DiastolicBP),    decreasing    =
TRUE))[1],
                                       filled_data_mode$DiastolicBP)

filled_data_mode$BS <- ifelse(is.na(filled_data_mode$BS),
                              names(sort(table(filled_data_mode$BS), decreasing = TRUE))[1],
                              filled_data_mode$BS)

filled_data_mode$BodyTemp <- ifelse(is.na(filled_data_mode$BodyTemp),
                                    names(sort(table(filled_data_mode$BodyTemp), decreasing = TRUE))[1],
                                    filled_data_mode$BodyTemp)

filled_data_mode$HeartRate <- ifelse(is.na(filled_data_mode$HeartRate),
                                     names(sort(table(filled_data_mode$HeartRate),     decreasing     =
TRUE))[1],
                                     filled_data_mode$HeartRate)

filled_data_mode$RiskLevel <- ifelse(is.na(filled_data_mode$RiskLevel),
                                     names(sort(table(filled_data_mode$RiskLevel),     decreasing     =
TRUE))[1],
                                     filled_data_mode$RiskLevel)

print(filled_data_mode)
```

Output:

```
   Age Infection Smoking SystolicBP DiastolicBP    BS BodyTemp HeartRate RiskLevel
1   25         1       1        130          80 15.00       98        86         1
2   35         1       1        140          90 13.00       98        70         1
3   29         1       1         90          70  8.00      100        80         1
4   30         1       1        140          85  7.00       98        70         1
5   35         2       3        120          60  6.10       98        76         2
6   23         1       1        140          80  7.01       98        70         1
7   23         2       2        130          70  7.01       98        78         3
8   23         1       1         85          60 11.00      102        86         1
9   32         3       2        120          90  6.90       98        70         3
10  42         1       1        130          80 18.00       98        70         1
11  23         2       3         90          60  7.01       98        76         2
12  19         3       2        120          80  7.00       98        70         3
13  25         2       3        110          89  7.01       98        77         2
14  20         3       3        120          75  7.01      100        70         3
15  48         3       2        120          80 11.00       98        88         3
16  15         2       3        120          90  7.01       98        70         2
17  50         1       1        140          90 15.00       98        90         1
18  25         1       1        140         100  7.01       98        80         1
19  30         3       2        120          80  6.90      101        76         3
20  10         2       3         70          50  6.90       98        70         2
21  40         1       1        140         100 18.00       98        90         1
22  50         3       2        140          80  6.70       98        70         3
23  21         2       3         90          65  7.50       98        76         2
24  18         2       3         90          60  7.50       98        70         2
```

Explanation:

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

- `filled_data_mode <- dataset`: This line creates a new dataset named `filled_data_mode` as a copy of the original dataset.

    - For each column in the dataset (Age, Infection, Smoking, SystolicBP, DiastolicBP, BS, BodyTemp, HeartRate, and RiskLevel), the following steps are performed:

    - `ifelse(is.na(filled_data_mode$Column))`: Checks if there are missing values in the column.

    - `names(sort(table(filled_data_mode$Column), decreasing = TRUE))[1]`: Calculates the mode of the column by first creating a frequency table using `table()`, sorting it in descending order using `sort()`, and then extracting the name of the most frequent value (mode) using `names()`.

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

- The `print(filled_data_mode)` statement prints the modified dataset `filled_data_mode` with missing values replaced by their respective column modes.

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

Missing values visualization:

Age:

Code:

```
missing_data <- data.frame(
  Missing = c("Available", "Missing"),
  Age = c(sum(!is.na(dataset$Age)), sum(is.na(dataset$Age)))
)

ggplot(missing_data, aes(x = Missing, y = Age, fill = Missing)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("#619CFF", "#F8766D")) +
  labs(
    title = "Missing Instances of Age",
    x = "Data Availability",
    y = "Age"
  ) +
  theme_minimal()
```

Output:

Infection:

Code:

```
missing_data <- data.frame(
  Missing = c("Available", "Missing"),
  Infection            =            c(sum(!is.na(dataset$Infection)),
sum(is.na(dataset$Infection)))
)

ggplot(missing_data, aes(x = Missing, y = Infection, fill = Missing))
+
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("red", "green")) +
  labs(
    title = "Missing Instances of Infection",
    x = "Data Availability",
    y = "Infection"
  ) +
  theme_minimal()
```

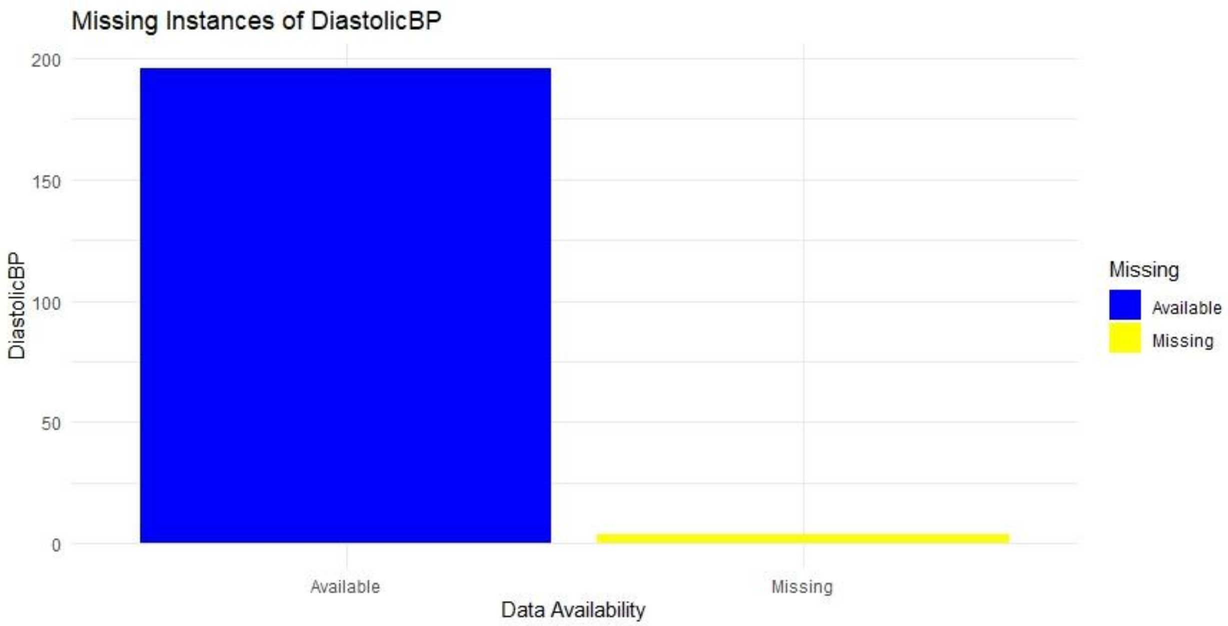
Output:

DiastolicBP:

Code:

```
missing_data <- data.frame(

  Missing = c("Available", "Missing"),

  DiastolicBP          =          c(sum(!is.na(dataset$DiastolicBP)),
sum(is.na(dataset$DiastolicBP)))

)

ggplot(missing_data, aes(x = Missing, y = DiastolicBP, fill = Missing))
+

  geom_bar(stat = "identity") +

  scale_fill_manual(values = c("blue", "yellow")) +

  labs(

    title = "Missing Instances of DiastolicBP",

    x = "Data Availability",

    y = "DiastolicBP"

  ) +

  theme_minimal()
```

Output:



Explanation:

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

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

- Within each block of code:
  - The named `missing_data` is created with two columns: "Missing" indicating the availability of data ("`Available`" or "`Missing`"), and the variable of interest (e.g., "`Age`", "`Infection`", or "`DiastolicBP`").
  - The counts of missing and available data for the variable of interest are calculated using `` sum(!is.na())` and `sum(is.na()) ``, respectively.

- A bar plot is generated using `ggplot2`, with "`Missing`" on the x-axis, the count of data on the y-axis, and the fill color indicating data availability.

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

Analysis and Visualization:

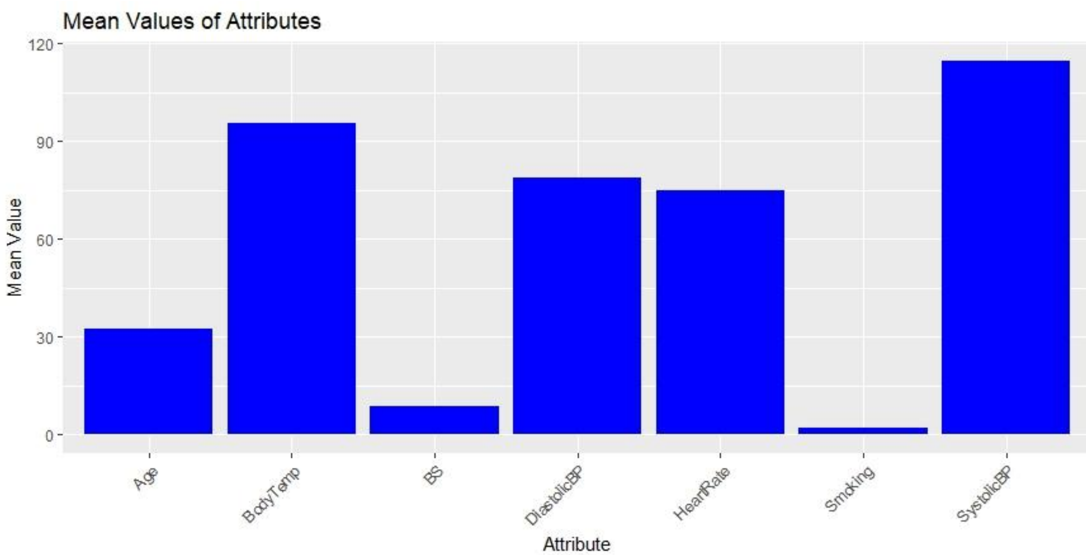
Mean:

Code:

```
mean_age <- mean(mydata$Age, na.rm = TRUE)
mean_smoking <- mean(mydata$Smoking, na.rm = TRUE)
mean_systolicBP <- mean(mydata$SystolicBP, na.rm = TRUE)
mean_diastolicBP <- mean(mydata$DiastolicBP, na.rm = TRUE)
mean_bs <- mean(mydata$BS, na.rm = TRUE)
mean_bodytemp <- mean(mydata$BodyTemp, na.rm = TRUE)
mean_heartrate <- mean(mydata$HeartRate, na.rm = TRUE)
mean_age
mean_smoking
mean_systolicBP
mean_diastolicBP
mean_bs
mean_bodytemp
mean_heartrate
mean_data <- data.frame(attribute = c("Age", "Smoking", "SystolicBP",
"DiastolicBP", "BS", "BodyTemp", "HeartRate"),
  mean_value    =    c(mean_age,    mean_smoking,    mean_systolicBP,
mean_diastolicBP, mean_bs, mean_bodytemp, mean_heartrate))

ggplot(mean_data, aes(x = attribute, y = mean_value)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(title = "Mean Values of Attributes",
       x = "Attribute", y = "Mean Value") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Output:

```
R 4.3.3 · ~/
> mean_smoking <- mean(mydata$Smoking, na.rm = TRUE)
> mean_systolicBP <- mean(mydata$SystolicBP, na.rm = TRUE)
> mean_diastolicBP <- mean(mydata$DiastolicBP, na.rm = TRUE)
> mean_bs <- mean(mydata$BS, na.rm = TRUE)
> mean_bodytemp <- mean(mydata$BodyTemp, na.rm = TRUE)
> mean_heartrate <- mean(mydata$HeartRate, na.rm = TRUE)
> mean_age
[1] 32.38674
> mean_smoking
[1] 2.060773
> mean_systolicBP
[1] 114.7735
> mean_diastolicBP
[1] 78.8674
> mean_bs
[1] 8.904088
> mean_bodytemp
[1] 95.65746
> mean_heartrate
[1] 74.91713
> mean_data <- data.frame(attribute = c("Age", "Smoking", "SystolicBP", "DiastolicBP", "BS", "BodyTemp", "HeartRate"),
+   mean_value = c(mean_age, mean_smoking, mean_systolicBP, mean_diastolicBP, mean_bs, mean_bodytemp, mean_heartrate))
>
> # Plot mean values using a bar plot
```



Explanation:

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

- The lines `mean_age <- mean(mydata$Age, na.rm = TRUE)`, `mean_smoking <- mean(mydata$Smoking, na.rm = TRUE)`, and so on, calculate the mean value for each attribute in the dataset, ignoring any missing values (`` `na.rm = TRUE` ``).
- The mean values for each attribute are printed using `mean_age`, `mean_smoking`, `mean_systolicBP`, and so on.

- The `data.frame()` function is used to create a data frame named `mean_data`, which contains two columns: "`attribute`" (the names of the attributes) and "`mean_value`" (the corresponding mean values).
- The `ggplot2` package is used to create a bar plot (`geom_bar()`) with "attribute" on the x-axis and "`mean_value`" on the y-axis. The bars are filled with blue color. Axis labels and title are added using `labs()`, and the x-axis labels are rotated by 45 degrees for better readability using `theme(axis.text.x = element_text(angle = 45, hjust = 1))`.

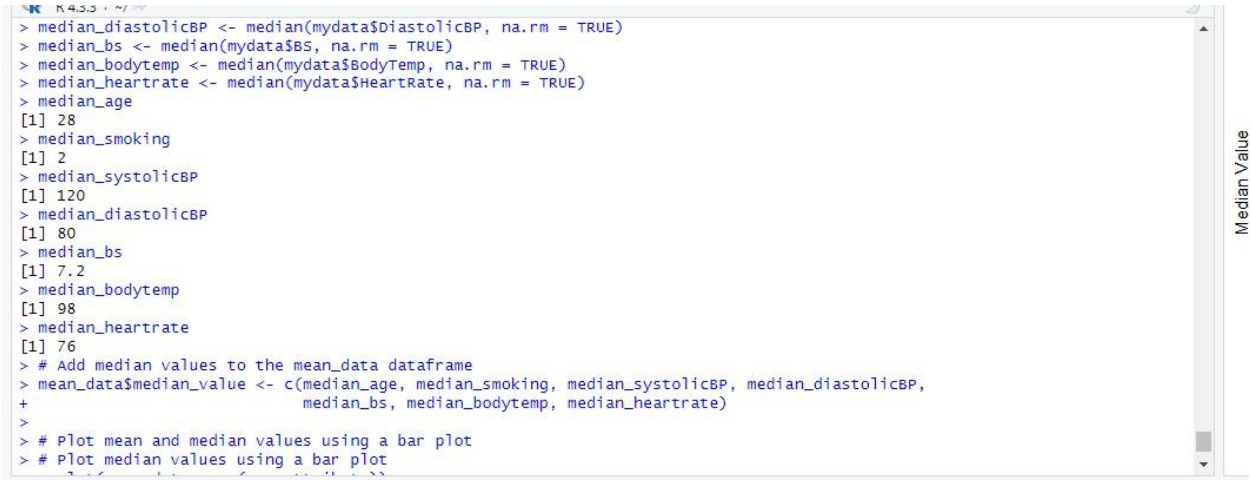
In summary, this code computes the mean values of various attributes from a dataset and visualizes them using a bar plot, providing an overview of the average values for each attribute.

Median:

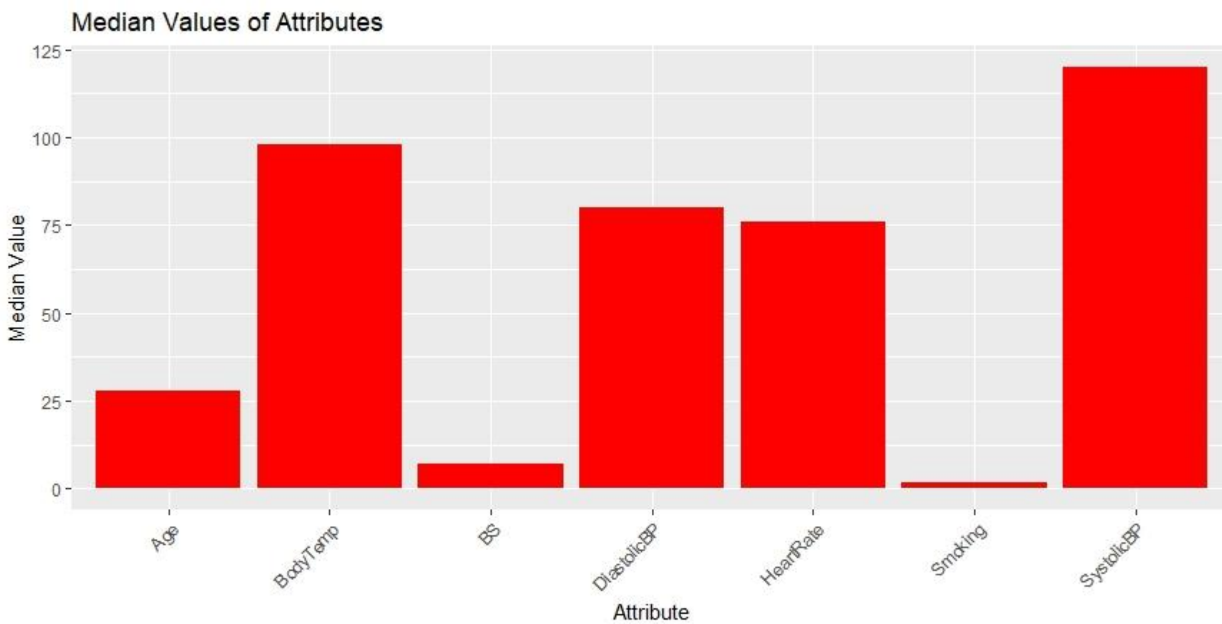
Code:

```
median_age <- median(mydata$Age, na.rm = TRUE)
median_smoking <- median(mydata$Smoking, na.rm = TRUE)
median_systolicBP <- median(mydata$SystolicBP, na.rm = TRUE)
median_diastolicBP <- median(mydata$DiastolicBP, na.rm = TRUE)
median_bs <- median(mydata$BS, na.rm = TRUE)
median_bodytemp <- median(mydata$BodyTemp, na.rm = TRUE)
median_heartrate <- median(mydata$HeartRate, na.rm = TRUE)
median_age
median_smoking
median_systolicBP
median_diastolicBP
median_bs
median_bodytemp
median_heartrate
mean_data$median_value      <-      c(median_age,      median_smoking,
median_systolicBP, median_diastolicBP,
                            median_bs,               median_bodytemp,
median_heartrate)
ggplot(mean_data, aes(x = attribute)) +
  geom_bar(aes(y = median_value), stat = "identity", fill = "red") +
  labs(title = "Median Values of Attributes",
       x = "Attribute", y = "Median Value") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Output:



```
> median_diastolicBP <- median(mydata$DiastolicBP, na.rm = TRUE)
> median_bs <- median(mydata$BS, na.rm = TRUE)
> median_bodytemp <- median(mydata$BodyTemp, na.rm = TRUE)
> median_heartrate <- median(mydata$HeartRate, na.rm = TRUE)
> median_age
[1] 28
> median_smoking
[1] 2
> median_systolicBP
[1] 120
> median_diastolicBP
[1] 80
> median_bs
[1] 7.2
> median_bodytemp
[1] 98
> median_heartrate
[1] 76
> # Add median values to the mean_data dataframe
> mean_data$median_value <- c(median_age, median_smoking, median_systolicBP, median_diastolicBP,
+                             median_bs, median_bodytemp, median_heartrate)
>
> # Plot mean and median values using a bar plot
> # Plot median values using a bar plot
```



Explanation

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

- For each attribute (Age, Smoking, SystolicBP, DiastolicBP, BS, BodyTemp, HeartRate), the code calculates the median value using the `median()` function, while ignoring any missing values (`na.rm = TRUE`).

- The median values for each attribute are stored in separate variables (`median_age`, `median_smoking`, etc.).
- The median values are added to the existing `mean_data` data frame as a new column named `median_value`.
- The `ggplot()` function is used to create a bar plot of the median values. Each bar represents an attribute, and the height of the bar represents its median value.

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

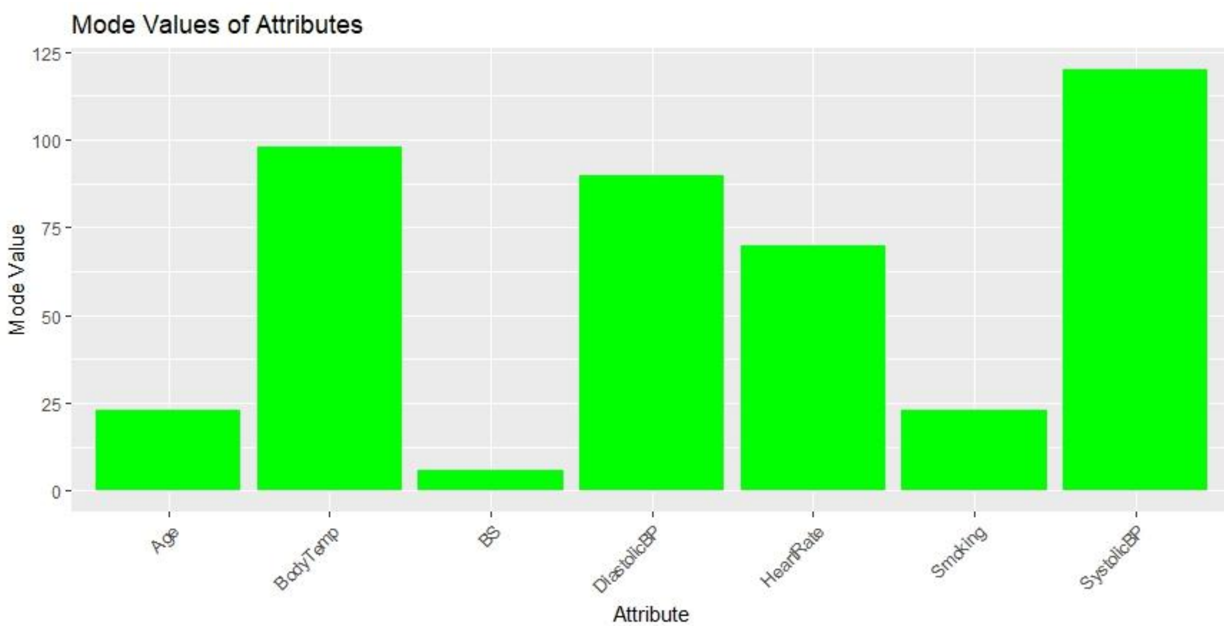
Mode:

Code:

```
calculate_mode <- function(x) {
  unique_x <- unique(x)
  freq <- tabulate(match(x, unique_x))
  mode_value <- unique_x[which.max(freq)]
  return(mode_value)
}

mode_age <- calculate_mode(mydata$Age)
mode_smoking <-calculate_mode(mydata$Smoking)
mode_systolicBP <- calculate_mode(mydata$SystolicBP)
mode_diastolicBP <- calculate_mode(mydata$DiastolicBP)
mode_bs <- calculate_mode(mydata$BS)
mode_bodytemp <- calculate_mode(mydata$BodyTemp)
mode_heartrate <- calculate_mode(mydata$HeartRate)
mode_age
mode_smoking
mode_systolicBP
mode_diastolicBP
mode_bs
mode_bodytemp
mode_heartrate

mean_data$mode_value <- c(mode_age, mode_age, mode_systolicBP, mode_diastolicBP,
                          mode_bs, mode_bodytemp, mode_heartrate)

ggplot(mean_data, aes(x = attribute)) +
  geom_bar(aes(y = mode_value), stat = "identity", fill = "green") +
  labs(title = "Mode Values of Attributes",
       x = "Attribute", y = "Mode Value") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Output:

```
R 4.3.3 · ~/
>
> # Calculate mode values for each attribute
> mode_age <- calculate_mode(mydata$Age)
> mode_smoking <-calculate_mode(mydata$Smoking)
> mode_systolicBP <- calculate_mode(mydata$SystolicBP)
> mode_diastolicBP <- calculate_mode(mydata$DiastolicBP)
> mode_bs <- calculate_mode(mydata$BS)
> mode_bodytemp <- calculate_mode(mydata$BodyTemp)
> mode_heartrate <- calculate_mode(mydata$HeartRate)
> mode_age
[1] 23
> mode_smoking
[1] 3
> mode_systolicBP
[1] 120
> mode_diastolicBP
[1] 90
> mode_bs
[1] 6.1
> mode_bodytemp
[1] 98
> mode_heartrate
[1] 70
>
```



Explanation:

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

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

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

Numeric attributes to categorical attributes:

Code:

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

Output:

```
> dataset$RiskLevel <- factor(dataset$RiskLevel,
+                             levels = c("high risk", "low risk", "mid risk"),
+                             labels = c("1", "2", "3"))
> dataset
   Age Infection Smoking SystolicBP DiastolicBP    BS BodyTemp HeartRate RiskLevel
1   25       yes       1        130          80 15.00       98        86         1
2   35       yes       1        140          90 13.00       98        70         1
3   29       yes       1         90          70  8.00      100        80         1
4   30       yes       1        140          85  7.00       98        70         1
5   35        no       3        120          60  6.10       98        76         2
6   23       yes       1        140          80  7.01       98        70         1
7   23                 2        130          70  7.01       98        78         3
8   NA       yes       1         85          60 11.00      102        86         1
9   32  marginal       2        120          90  6.90       98        70         3
10  42       yes       1        130          80 18.00       98        70         1
11  23        no       3         90          60  7.01       98        76         2
12  19  marginal       2        120          80  7.00       98        70         3
13  25        no       3        110          89  7.01       98        77         2
14  20  marginal      NA        120          75  7.01      100        70         3
15  48  marginal       2        120          80 11.00       98        88         3
16  15        no       3        120          NA  7.01       98        70         2
17  50       yes       1        140          90 15.00       98        90         1
18  25       yes       1        140         100  7.01       98        80         1
19  30  marginal       2        120          80  6.90      101        76         3
20  10        no       3         70          50  6.90       98        70         2
21  40       yes       1        140         100 18.00       98        90         1
22  50  marginal       2        140          80  6.70       98        70         3
```

Explanation:

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

Normalization method:

Code:

```
age <- mydata$Age
min_max_normalization <- function(x) {
  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}
normalized_age <- min_max_normalization(age)

normalized_age
```

Output:

```
> # Print the normalized 'Age' values
> normalized_age
  [1] 0.09375 0.15625 0.11875 0.12500 0.15625 0.08125 0.13750 0.20000 0.08125 0.05625 0.09375 0.23750 0.25000 0.09375
 [15] 0.12500 0.00000 0.18750 0.25000 0.06875 0.05000 0.03750 0.07500 0.24375 0.11250 0.06250 0.08125 0.07500 0.06875
 [29] 0.01250 0.31250 0.28125 0.07500 0.08125 0.09375 0.12500 0.08125 0.13750 0.20000 0.08125 0.03125 0.03125 0.09375
 [43] 0.07500 0.15625 0.05625 0.31250 0.08125 0.13750 0.20000 0.03125 0.03125 0.01250 0.11875 0.11875 0.04375 0.05625
 [57] 0.13750 0.10000 0.11875 0.05625 0.27500 0.21250 0.08125 0.07500 0.28125 0.06875 0.03750 0.14375 0.01250 0.11250
 [71] 0.06875 0.05000 0.06875 0.03750 0.05625 0.08125 0.07500 0.31250 0.01875 0.08125 0.11250 0.25000 0.11875 0.05625
 [85] 0.05625 0.31250 0.09375 0.09375 0.08125 0.15000 0.09375 0.20000 0.13750 0.25000 0.17500 0.18125 0.12500 0.33125
 [99] 0.09375 0.12500 0.28125 0.13750 0.12500 0.23750 0.24375 0.09375 0.18750 0.13750 0.15625 0.27500 0.28125 0.11875
[113] 0.86250 0.18750 0.13750 0.15625 0.27500 0.18750 0.07500 0.18750 0.28125 0.25000 0.05000 0.13750 0.04375 0.04375
[127] 0.09375 0.04375 0.02500 0.03125 0.03125 0.01250 0.16875 0.05000 0.94375 0.09375 0.08125 0.01250 0.11250 0.18750
[141] 0.28125 0.09375 0.15625 0.06875 0.05000 0.06875 0.03750 0.05625 1.00000 0.13750 0.07500 0.24375 0.11250 0.01250
[155] 0.06250 0.08125 0.07500 0.06875 0.15625 0.27500 0.18750 0.01250 0.31250 0.28125 0.25000 0.31250 0.28125 0.21875
[169] 0.15625 0.07500 0.01875 0.08125 0.04375 0.11250 0.25000 0.09375 0.12500 0.13125 0.08125 0.11875 0.04375
> 
```

Explanation :

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

DiastolicBP:

Code:

```
DiastolicBP <- mydata$DiastolicBP
min_max_normalization <- function(x) {
  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}
normalized_age <- min_max_normalization(DiastolicBP)
normalized_age
```

Output:

```
> normalized_age <- min_max_normalization(DiastolicBP)
>
> # Print the normalized 'Age' values
> normalized_age
  [1] 0.60784314 0.80392157 0.41176471 0.70588235 0.21568627 0.60784314 0.80392157 0.60784314 0.21568627 0.60784314
 [11] 0.78431373 0.60784314 0.80392157 1.00000000 0.60784314 0.01960784 1.00000000 0.60784314 0.31372549 0.21568627
 [21] 0.41176471 0.31372549 0.80392157 0.21568627 0.80392157 0.70588235 0.80392157 0.01960784 0.21568627 0.60784314
 [31] 0.31372549 0.70588235 0.80392157 0.41176471 0.60784314 0.80392157 0.80392157 0.60784314 0.21568627 0.00000000
 [41] 0.60784314 0.60784314 0.31372549 0.41176471 0.70588235 0.31372549 0.80392157 0.80392157 0.60784314 0.00000000
 [51] 0.21568627 0.21568627 0.41176471 0.41176471 0.21568627 0.60784314 0.31372549 0.21568627 0.41176471 0.60784314
 [61] 0.41176471 0.80392157 0.41176471 0.21568627 0.80392157 0.21568627 0.31372549 0.31372549 0.21568627 0.80392157
 [71] 0.31372549 0.21568627 0.60784314 0.41176471 0.50980392 0.70588235 0.80392157 0.70588235 0.31372549 0.80392157
 [81] 0.21568627 0.60784314 0.41176471 0.60784314 0.70588235 0.31372549 0.60784314 1.00000000 0.80392157 0.21568627
 [91] 1.00000000 1.00000000 1.00000000 0.90196078 0.21568627 0.41176471 1.00000000 0.80392157 1.00000000 0.60784314
[101] 1.00000000 1.00000000 1.00000000 0.60784314 0.80392157 1.00000000 1.00000000 0.80392157 1.00000000 1.00000000
[111] 0.90196078 0.41176471 0.60784314 1.00000000 0.80392157 1.00000000 1.00000000 0.90196078 0.21568627 0.70588235
[121] 0.90196078 1.00000000 0.60784314 1.00000000 0.21568627 0.27450980 0.80392157 0.60784314 0.31372549 0.21568627
[131] 0.31372549 0.21568627 0.80392157 0.41176471 0.70588235 0.80392157 0.31372549 0.21568627 0.80392157 0.80392157
[141] 0.70588235 0.80392157 0.60784314 0.31372549 0.21568627 0.60784314 0.41176471 0.50980392 1.00000000 0.80392157
[151] 0.31372549 0.80392157 0.21568627 0.21568627 0.80392157 0.70588235 0.80392157 0.60784314 1.00000000 1.00000000
[161] 0.90196078 0.21568627 0.70588235 0.90196078 1.00000000 0.60784314 0.31372549 0.90196078 0.41176471 0.70588235
[171] 0.31372549 0.80392157 0.31372549 0.21568627 0.60784314 0.41176471 0.60784314 0.21568627 0.80392157 0.41176471
[181] 0.21568627
>
```

Explanation:

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

SystolicBP:

Code:

```
SystolicBP<- mydata$SystolicBP

min_max_normalization <- function(x) {
  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}
normalized_age <- min_max_normalization(SystolicBP)
normalized_age
```

Output:

```
R 4.3.3 · ~/
> normalized_age <- min_max_normalization(SystolicBP)
>
> # Print the normalized 'Age' values
> normalized_age
  [1] 0.81818182 1.00000000 0.09090909 1.00000000 0.63636364 1.00000000 0.63636364 0.81818182 0.09090909 0.63636364
 [11] 0.45454545 0.63636364 1.00000000 1.00000000 0.63636364 0.00000000 1.00000000 1.00000000 0.09090909 0.09090909
 [21] 0.27272727 0.27272727 0.63636364 0.09090909 0.27272727 0.27272727 0.63636364 0.00000000 0.18181818 0.63636364
 [31] 0.27272727 0.63636364 0.63636364 0.09090909 0.63636364 0.63636364 0.63636364 0.63636364 0.09090909 0.00000000
 [41] 0.63636364 0.63636364 0.27272727 0.27272727 0.63636364 0.09090909 0.63636364 0.63636364 0.63636364 0.00000000
 [51] 0.00000000 0.18181818 0.09090909 0.81818182 0.00000000 0.63636364 0.63636364 0.00000000 0.81818182 0.63636364
 [61] 0.81818182 0.63636364 0.81818182 0.00000000 0.63636364 0.09090909 0.09090909 0.54545455 0.18181818 0.63636364
 [71] 0.09090909 0.09090909 0.63636364 0.27272727 0.63636364 0.27272727 0.63636364 0.63636364 0.09090909 0.63636364
 [81] 0.54545455 0.63636364 0.81818182 0.63636364 0.63636364 0.09090909 0.63636364 1.00000000 1.00000000 0.00000000
 [91] 1.00000000 1.00000000 1.00000000 1.00000000 0.90909091 0.09090909 1.00000000 1.00000000 1.00000000 0.63636364
[101] 1.00000000 1.00000000 1.00000000 0.63636364 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
[111] 1.00000000 0.63636364 0.63636364 1.00000000 1.00000000 1.00000000 1.00000000 0.63636364 0.09090909 0.63636364
[121] 1.00000000 0.81818182 0.63636364 1.00000000 0.09090909 0.09090909 0.63636364 0.63636364 0.09090909 0.00000000
[131] 0.27272727 0.18181818 0.63636364 0.27272727 0.27272727 0.63636364 0.00000000 0.18181818 0.63636364 0.63636364
[141] 0.80000000 0.27272727 0.63636364 0.09090909 0.09090909 0.63636364 0.27272727 0.63636364 1.00000000 1.00000000
[151] 0.27272727 0.63636364 0.09090909 0.09090909 0.27272727 0.27272727 0.63636364 0.63636364 1.00000000 1.00000000
[161] 0.63636364 0.18181818 0.63636364 1.00000000 0.81818182 0.63636364 0.27272727 0.63636364 0.27272727 0.63636364
[171] 0.09090909 0.63636364 0.09090909 0.00000000 0.63636364 0.09090909 0.63636364 0.63636364 0.63636364 0.81818182
[181] 0.00000000
>
```

Explanation:

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

BS:

Code:

```
BS<- mydata$BS

min_max_normalization <- function(x) {
  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}
normalized_age <- min_max_normalization(BS)
normalized_age
```

Output:

```
+ }
>
>
> normalized_age <- min_max_normalization(BS)
> normalized_age
  [1] 0.153846154 0.153846154 0.153846154 0.076923077 0.007692308 0.077692308 0.069230769 0.153846154 0.077692308
 [10] 0.076923077 0.077692308 0.153846154 0.692307692 0.077692308 0.069230769 0.069230769 0.923076923 0.053846154
 [19] 0.115384615 0.115384615 0.092307692 0.092307692 0.092307692 0.092307692 0.084615385 0.084615385 0.084615385
 [28] 0.007692308 0.007692308 0.007692308 0.007692308 0.007692308 0.007692308 0.007692308 0.007692308 0.007692308
 [37] 0.115384615 0.115384615 0.115384615 0.115384615 0.076923077 0.076923077 0.076923077 0.076923077 0.076923077
 [46] 0.076923077 0.053846154 0.030769231 0.030769231 0.030769231 0.076923077 0.092307692 0.053846154 0.053846154
 [55] 0.230769231 0.076923077 0.000000000 0.000000000 0.130769231 0.076923077 0.153846154 0.153846154 0.069230769
 [64] 0.069230769 0.461538462 0.069230769 0.069230769 0.076923077 0.069230769 0.069230769 0.069230769 0.069230769
 [73] 0.069230769 0.069230769 0.069230769 0.069230769 0.138461538 0.692307692 0.138461538 0.138461538 0.138461538
 [82] 0.138461538 0.138461538 0.076923077 0.138461538 0.061538462 0.061538462 0.061538462 0.061538462 0.384615385
 [91] 0.061538462 0.923076923 0.146153846 0.153846154 0.146153846 0.230769231 0.692307692 0.692307692 0.146153846
[100] 0.146153846 0.923076923 0.146153846 0.692307692 0.384615385 0.692307692 0.115384615 0.153846154 0.923076923
[109] 0.115384615 0.692307692 1.000000000 0.230769231 0.384615385 1.000000000 0.923076923 0.115384615 0.692307692
[118] 0.384615385 0.115384615 0.692307692 1.000000000 0.769230769 0.069230769 0.069230769 0.069230769 0.069230769
[127] 0.053846154 0.053846154 0.076923077 0.053846154 0.053846154 0.053846154 0.384615385 0.053846154 0.053846154
[136] 0.115384615 0.115384615 0.115384615 0.115384615 0.461538462 0.115384615 0.115384615 0.115384615 0.115384615
[145] 0.115384615 0.115384615 0.092307692 0.092307692 1.000000000 0.923076923 0.092307692 0.092307692 0.092307692
[154] 0.146153846 0.084615385 0.084615385 0.084615385 0.084615385 0.153846154 0.692307692 0.384615385 0.007692308
[163] 0.692307692 1.000000000 0.769230769 0.007692308 0.007692308 0.007692308 0.007692308 0.007692308 0.146153846
[172] 0.007692308 0.007692308 0.153846154 0.692307692 0.007692308 0.007692308 0.007692308 0.007692308 0.007692308
[181] 0.230769231
> |
```

Explanation:

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

BodyTemp:

Code:

```
BodyTemp<- mydata$BodyTemp
min_max_normalization <- function(x) {
  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}
normalized_age <- min_max_normalization(BodyTemp)
normalized_age
```

Output:

```
> BodyTemp<- mydata$BodyTemp
>
>
> min_max_normalization <- function(x) {
+   (x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
+ }
> normalized_age <- min_max_normalization(BodyTemp)
> normalized_age
  [1] 0.4285714 0.4285714 0.0000000 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.428571
 [12] 0.4285714 0.4285714 0.4285714 0.0000000 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.428571
 [23] 0.4285714 0.0000000 0.4285714 0.4285714 0.4285714 0.4285714 0.0000000 0.4285714 0.4285714 0.4285714 0.428571
 [34] 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.428571
 [45] 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 1.000000
 [56] 0.4285714 0.8571429 0.8571429 0.0000000 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.428571
 [67] 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.428571
 [78] 0.4285714 0.8571429 0.4285714 0.8571429 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.428571
 [89] 0.4285714 1.0000000 0.4285714 0.4285714 0.4285714 0.4285714 0.8571429 0.4285714 0.4285714 0.4285714 0.428571
[100] 0.8571429 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.428571
[111] 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 1.0000000 0.4285714 0.428571
[122] 0.4285714 1.0000000 0.4285714 0.8571429 0.8571429 0.8571429 1.0000000 0.8571429 0.4285714 0.4285714 0.428571
[133] 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.428571
[144] 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 1.000000
[155] 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 1.0000000 0.4285714 0.4285714 0.428571
[166] 0.4285714 0.4285714 0.4285714 0.4285714 0.4285714 0.8571429 0.4285714 0.0000000 0.8571429 0.4285714 0.428571
[177] 0.4285714 0.4285714 0.4285714 0.4285714 1.0000000
> |
```

Explanation:

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

HeartRate

Code:

```
HeartRate<- mydata$HeartRate
min_max_normalization <- function(x) {
  (x - min(x, na.rm = TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm =
TRUE))
}
normalized_age <- min_max_normalization(HeartRate)
normalized_age
```

Output:

```
> HeartRate<- mydata$HeartRate
>
> min_max_normalization <- function(x) {
+   (x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
+ }
> normalized_age <- min_max_normalization(HeartRate)
> normalized_age
  [1] 0.8666667 0.3333333 0.6666667 0.3333333 0.5333333 0.3333333 0.3333333 0.3333333 0.5333333 0.3333333 0.5666
 [12] 0.9333333 1.0000000 0.6666667 0.5333333 0.3333333 1.0000000 0.3333333 0.5333333 0.3333333 0.6666667 0.3333
 [23] 0.5666667 0.7333333 0.9333333 0.2000000 0.7333333 0.3333333 0.0000000 0.5000000 0.2000000 0.9333333 0.0000
 [34] 0.6666667 0.3333333 0.3333333 0.3333333 0.3333333 0.5333333 0.5666667 0.3333333 0.2000000 0.6666667 0.0000
 [45] 0.0000000 0.5666667 0.3333333 0.3333333 0.3333333 0.5666667 0.6666667 0.5666667 0.6666667 0.6000000 0.8666
 [56] 0.3333333 0.5333333 0.8666667 0.6000000 0.3333333 0.2333333 0.6666667 0.3333333 0.5333333 0.3333333 0.8666
 [67] 0.5333333 0.3333333 0.1666667 0.3333333 0.5333333 0.3333333 0.5333333 0.6666667 0.2000000 0.2000000 0.7333
 [78] 0.0000000 0.6666667 0.0000000 0.8666667 0.3333333 0.6000000 0.3333333 0.0000000 0.5666667 0.2000000 0.6666
 [89] 0.3333333 0.8666667 0.6666667 1.0000000 0.6000000 0.0000000 0.8666667 0.6666667 0.3333333 1.0000000 0.6666
[100] 0.5333333 1.0000000 0.6000000 0.3333333 0.9333333 1.0000000 0.6666667 0.5666667 0.9333333 0.2000000 0.2000
[111] 0.5666667 0.6666667 0.9333333 0.5666667 0.9333333 0.2000000 0.2000000 0.6666667 0.0000000 0.0000000 0.5666
[122] 0.5000000 0.5333333 0.6000000 0.5333333 0.3333333 0.6666667 0.5333333 0.3333333 0.6666667 0.5333333 0.5666
[133] 0.9333333 0.5333333 0.3333333 0.6666667 0.3333333 0.1666667 0.3333333 0.6666667 0.9333333 0.5333333 0.6666
[144] 0.5333333 0.3333333 0.5333333 0.6666667 0.2000000 0.5666667 0.9333333 0.3333333 0.5666667 0.7333333 0.2000
[155] 0.9333333 0.2000000 0.7333333 0.5666667 0.2000000 0.2000000 0.6666667 0.0000000 0.0000000 0.5666667 0.5000
[166] 0.5000000 0.2000000 0.2000000 0.2000000 0.9333333 0.6666667 0.0000000 0.2333333 0.8666667 0.3333333 0.6666
[177] 0.3333333 0.5333333 0.3333333 0.6000000 0.8666667
> 
```

Explanation:

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

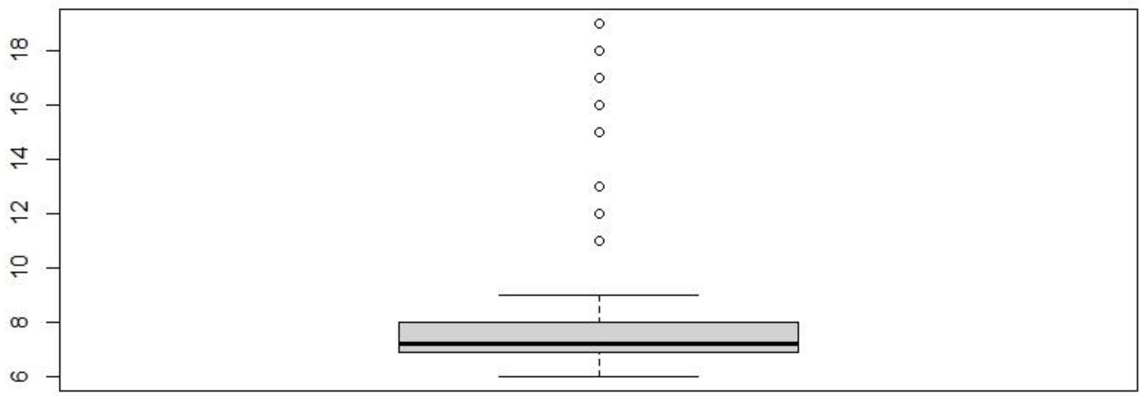
Replacing BS outliers with mean value:

Code:

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

Output:

```
> ageBoxplot <- boxplot(mydata$BS)
> outliers <- ageBoxplot$out
> cat("Outliers are", outliers)
Outliers are 15 13 18 11 15 18 12 16 12 15 11 18 17 15 15 18 15 11 15 19 18 15 19 11 19 18 15 11 15 19 16 11 12 19 18 15 11
15 19 16 15> ageMean <- mean(mydata$BS, na.rm = TRUE)
> outlierPositions <- match(outliers, mydata$BS)
> mydata$BS[outlierPositions] <- as.integer (ageMean)
> 
```



Explanation:

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

Replacing BodyTemp outliers with mean value:

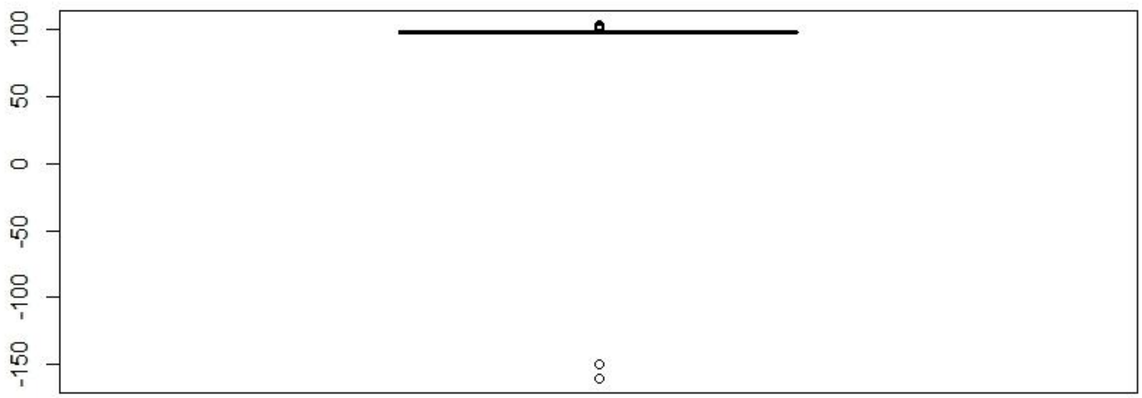
Code:

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

```
mydata$BodyTemp[outlierPositions] <- as.integer (ageMean)
```

Output:

```
> outlierPositions <- match(outliers, mydata$BS)
> mydata$BS[outlierPositions] <- as.integer (ageMean)
> ageBoxplot <- boxplot(mydata$BodyTemp)
> outliers <- ageBoxplot$out
> cat("Outliers are", outliers)
Outliers are 100 101 -150 102 102 101 101 -160 101 101 102 101 101 102 102 101 101 101 102 101 102 102 101 103 101 102> age
ean <- mean(mydata$BodyTemp, na.rm = TRUE)
> outlierPositions <- match(outliers, mydata$BodyTemp)
> mydata$BodyTemp[outlierPositions] <- as.integer (ageMean)
> |
```



Explanation:

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

Find Invalid values:

Code:

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

Output:

```
> negative_indices <- which(dataset < 0, arr.ind = TRUE)
> print("Indices of negative values:")
[1] "Indices of negative values:"
> negative_indices
     row col
[1,]  30   7
[2,]  72   7
>
```

Explanation:

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

Remove Invalid Value

Code:

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

Output:

```
> min_non_negative <- min(mydata[mydata >= 0], na.rm = TRUE)
> mydata[mydata < 0] <- min_non_negative
> print("Invalid (negative) values have been successfully recovered.")
[1] "Invalid (negative) values have been successfully recovered."
> print(mydata)
   Age Infection Smoking SystolicBP DiastolicBP    BS BodyTemp HeartRate RiskLevel
1   25       yes       1        130          80 15.00       98        86 high risk
2   35       yes       1        140          90 13.00       98        70 high risk
3   29       yes       1         90          70  8.00      100        80 high risk
4   30       yes       1        140          85  7.00       98        70 high risk
5   35        no       3        120          60  6.10       98        76  low risk
6   23       yes       1        140          80  7.01       98        70 high risk
9   32  marginal       2        120          90  6.90       98        70  mid risk
10  42       yes       1        130          80 18.00       98        70 high risk
11  23        no       3         90          60  7.01       98        76  low risk
12  19  marginal       2        120          80  7.00       98        70  mid risk
13  25        no       3        110          89  7.01       98        77  low risk
15  48  marginal       2        120          80 11.00       98        88  mid risk
17  50       yes       1        140          90 15.00       98        90 high risk
18  25       yes       1        140         100  7.01       98        80 high risk
19  30  marginal       2        120          80  6.90      101        76  mid risk
20  10        no       3         70          50  6.90       98        70  low risk
21  40       yes       1        140         100 18.00       98        90 high risk
22  50  marginal       2        140          80  6.70       98        70  mid risk
23  21        no       3         90          65  7.50       98        76  low risk
24  18        no       3         90          60  7.50       98        70  low risk
26  16        no       3        100          70  7.20       98        80  low risk
28  22        no       3        100          65  7.20       98        70  low risk
```

Explanation:

This code finds the minimum non-negative value (`min_non_negative`) in the dataset `mydata`, replaces all negative values in `mydata` with this minimum non-negative value, prints a message indicating successful recovery of negative values, and finally prints the updated dataset `mydata`.