Data Wrangling (Data Preprocessing)

Code

Practical assessment 2

Huu Thien Ngoc Phung

29/09/2021

Setup

Hide

```
library(kableExtra)
library(readr)
library(tidyr)
library(dplyr)
library(lubridate)
library(editrules)
library(deductive)
library(validate)
library(MVN)
library(forecast)

cap <- function(x){
  quantiles <- quantile( x, c(.05, 0.25, 0.75, .95 ) )
  x[ x < quantiles[2] - 1.5*IQR(x) ] <- quantiles[1]
  x[ x > quantiles[3] + 1.5*IQR(x) ] <- quantiles[4]
  x}
```

Student names, numbers and percentage of contributions

Group information

Student name	Student number	Percentage of contribution
Huu Thien Ngoc Phung	s3643808	100%

Executive Summary

This report will preprocess 3 data sets taken from FitBit Fitness Tracker Data (Kaggle 2020) to prepare the data for analysis of calories consumption and physical activity of FitBit users. Following are the steps taken for preprocessing:

- **Data**: Import the 3 data sets to R
- **Understand**: inspect data structure and variable types, then apply proper data type conversion for date time and factor variables where applicable for each data set.
- **Tidy & Manipulate I**: convert the untidy data set from wide format to long format and create a new activity time variable so that the it can be merged with the other 2 data sets using common keys which are 'Id' and 'Activity Minute'.
- **Scan I**: scan for missing values and NaN values and safely exclude found missing values from the data frame.

- **Tidy & Manipulate II**: group the observations by each user, activity date, and activity intensity level to find the daily total calories consumed, total steps that the users taken, and calories per step ratio for each user per day in each level of intensity.
- **Scan II**:
    - *Scan for obvious errors* by setting constraints for the numeric variables.
    - *Scan for multivariate outliers* using the Mahalanobis distance approach for numeric variables.
    - *Analyse the univariate distribution and univariate outliers* of each numeric variable using the Tukey's method of outlier detection.
    - *Capping*: it is found that capping method with (the 5th and 95th percentile values of the distributions) would be a reasonable approach to handle these univariate outliers.
- **Transformation**: for each numeric variable, applying data transformation via Mathematical operations with logarithm, square root, and reciprocal tranformation. Another attempt is taken using BoxCox transformation to find the best approach to transform each of these variables.

Data

There are 3 data sets to be used for this report which were taken from FitBit Fitness Tracker Data https://www.kaggle.com/arashnic/fitbit (https://www.kaggle.com/arashnic/fitbit). The data set contains the records of the personal tracker data of 30 eligible FitBit users from 12/04/2016 to 12/05/2016. The data files to be used are 'minuteCaloriesWide_merged.csv', 'minuteIntensitiesNarrow_merged.csv', 'minuteStepsNarrow_merged.csv'.

Following code chunks are to import the data sets to R as 3 data frames namely **"minuteCaloriesWide", "minuteIntensities" and "minuteSteps"**.

Data Set 1: minuteCaloriesWide_merged.csv

The dataset contains 62 variables and 21,645 rows of data which recorded the calories consumed every minutes of the FitBit users. The variables of this data set are:

- Id [Numeric]: Identified number of each the FitBit user
- ActivityHour [POSIXct]: activity time and date, time is rounded down to the nearest hour.
- Calories00 [Numeric]: calories consumed in the first minute of the activity hour.
- Calories01 [Numeric]: calories consumed in the second minute of the activity hour.
- Calories02 [Numeric]: calories consumed in the third minute of the activity hour.
- …
- Calories58 [Numeric]: calories consumed in the fifty-ninth minute of the activity hour.
- Calories59 [Numeric]: calories consumed in the sixtieth minute of the activity hour.

Hide

```
minuteCaloriesWide <- read_csv("Fitabase Data 4.12.16-5.12.16/minuteCaloriesWide_merged.csv")
```

```


-- Column specification ------------------------------------------------------------------------------------
--------------------
cols(
  .default = col_double(),
  ActivityHour = col_character()
)
i Use `spec()` for the full column specifications.
```

Hide

```
head(minuteCaloriesWide)
```

Id <dbl>	ActivityHour <chr>	Calories00 <dbl>	Calories01 <dbl>	Calories02 <dbl>	Calories03 <dbl>	Calories04 <dbl>
1503960366	4/13/2016 12:00:00 AM	1.8876	2.2022	0.9438	0.9438	0.9438
1503960366	4/13/2016 1:00:00 AM	0.7865	0.7865	0.7865	0.7865	0.9438
1503960366	4/13/2016 2:00:00 AM	0.7865	0.7865	0.7865	0.7865	0.7865
1503960366	4/13/2016 3:00:00 AM	0.7865	0.7865	0.7865	0.7865	0.7865
1503960366	4/13/2016 4:00:00 AM	0.7865	0.7865	0.7865	0.7865	0.7865
1503960366	4/13/2016 5:00:00 AM	0.7865	0.7865	0.7865	0.7865	0.7865

6 rows | 1-7 of 62 columns

Import the dataset to R and save it as a dataframe named “minuteCaloriesWide”, then inspect the first 6 observations of the dataset.

Data Set 2: minuteIntensitiesNarrow_merged.csv

The dataset contains 3 variables with 1,325,580 rows of data, which recorded the intensity level of the activity every minute taken by the FitBit users. The variables of this dataset are:

- Id [Numeric]: Identified number of each the FitBit user
- ActivityMinute [POSIXct]: activity time and date, recorded every minute
- Intensity [Factor]: activity intensity level, the 4 levels are “0” for Sedentary, “1” for Lightly Active, “2” for Fairly Active, “3” for Very Active

Hide

```
minuteIntensities <- read_csv("Fitabase Data 4.12.16-5.12.16/minuteIntensitiesNarrow_merged.csv")
```

```


-- Column specification --------------------------------------------------------------------------------
---------------------
cols(
  Id = col_double(),
  ActivityMinute = col_character(),
  Intensity = col_double()
)
```

Hide

```
head(minuteIntensities)
```

Id <dbl>	ActivityMinute <chr>	Intensity <dbl>
1503960366	4/12/2016 12:00:00 AM	0
1503960366	4/12/2016 12:01:00 AM	0

Id <dbl>	ActivityMinute <chr>	Intensity <dbl>
1503960366	4/12/2016 12:02:00 AM	0
1503960366	4/12/2016 12:03:00 AM	0
1503960366	4/12/2016 12:04:00 AM	0
1503960366	4/12/2016 12:05:00 AM	0

6 rows

Import the dataset to R and save it as a dataframe named “minuteIntensities”, then inspect the first 6 observations of the dataset.

Data Set 3: minuteStepsNarrow_merged.csv

The dataset contains 3 variables with 1,325,580 rows of data, which recorded the number of steps taken every minute by the FitBit users. The variables of this dataset are:

- Id [Numeric]: Identified number of each the FitBit user
- ActivityMinute [POSIXct]: activity time and date, recorded every minute
- Steps [Numeric]: the number of steps detected

Hide

```
minuteSteps <- read_csv("Fitabase Data 4.12.16-5.12.16/minuteStepsNarrow_merged.csv")
```

```


-- Column specification ------------------------------------------------------------------------------
---------------------
cols(
  Id = col_double(),
  ActivityMinute = col_character(),
  Steps = col_double()
)
```

Hide

```
head(minuteSteps)
```

Id <dbl>	ActivityMinute <chr>	Steps <dbl>
1503960366	4/12/2016 12:00:00 AM	0
1503960366	4/12/2016 12:01:00 AM	0
1503960366	4/12/2016 12:02:00 AM	0
1503960366	4/12/2016 12:03:00 AM	0
1503960366	4/12/2016 12:04:00 AM	0
1503960366	4/12/2016 12:05:00 AM	0

6 rows

Import the dataset to R and save it as a dataframe named “minuteSteps”, then inspect the first 6 observations of the data set.

Understand

Data frame 1: minuteCaloriesWide

Hide

```
str(minuteCaloriesWide)
```

```
spec_tbl_df [21,645 x 62] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ Id          : num [1:21645] 1.5e+09 1.5e+09 1.5e+09 1.5e+09 1.5e+09 ...
 $ ActivityHour: chr [1:21645] "4/13/2016 12:00:00 AM" "4/13/2016 1:00:00 AM" "4/13/2016 2:00:00 AM"
"4/13/2016 3:00:00 AM" ...
 $ Calories00  : num [1:21645] 1.888 0.786 0.786 0.786 0.786 ...
 $ Calories01  : num [1:21645] 2.202 0.786 0.786 0.786 0.786 ...
 $ Calories02  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories03  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories04  : num [1:21645] 0.944 0.944 0.786 0.786 0.786 ...
 $ Calories05  : num [1:21645] 2.045 0.944 0.786 0.786 0.786 ...
 $ Calories06  : num [1:21645] 0.944 0.944 0.786 0.786 0.786 ...
 $ Calories07  : num [1:21645] 2.202 0.786 0.786 0.786 0.786 ...
 $ Calories08  : num [1:21645] 0.944 0.944 0.786 0.786 0.786 ...
 $ Calories09  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories10  : num [1:21645] 0.944 0.944 0.786 0.786 0.786 ...
 $ Calories11  : num [1:21645] 0.786 0.786 0.786 2.045 0.786 ...
 $ Calories12  : num [1:21645] 0.786 0.944 0.786 0.944 0.786 ...
 $ Calories13  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories14  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories15  : num [1:21645] 0.944 0.786 0.786 0.944 0.786 ...
 $ Calories16  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories17  : num [1:21645] 0.786 0.786 0.786 0.944 0.786 ...
 $ Calories18  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories19  : num [1:21645] 0.786 0.786 0.786 0.786 0.944 ...
 $ Calories20  : num [1:21645] 1.888 0.786 0.786 0.786 0.786 ...
 $ Calories21  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories22  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories23  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories24  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories25  : num [1:21645] 2.045 0.786 0.786 0.786 0.786 ...
 $ Calories26  : num [1:21645] 2.359 0.786 0.786 0.786 0.786 ...
 $ Calories27  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories28  : num [1:21645] 2.045 0.786 0.786 0.786 0.786 ...
 $ Calories29  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories30  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories31  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories32  : num [1:21645] 2.045 0.786 0.786 0.786 0.786 ...
 $ Calories33  : num [1:21645] 1.888 0.786 0.786 0.944 0.786 ...
 $ Calories34  : num [1:21645] 0.944 0.786 0.786 2.045 0.786 ...
 $ Calories35  : num [1:21645] 0.786 0.786 0.786 2.045 0.786 ...
 $ Calories36  : num [1:21645] 0.786 0.786 0.786 1.888 0.786 ...
 $ Calories37  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories38  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories39  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories40  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories41  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories42  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories43  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories44  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories45  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories46  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories47  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories48  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories49  : num [1:21645] 0.786 0.786 0.786 0.786 0.786 ...
 $ Calories50  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories51  : num [1:21645] 2.045 0.786 0.786 0.786 0.786 ...
 $ Calories52  : num [1:21645] 2.045 0.786 0.786 0.786 0.786 ...
 $ Calories53  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories54  : num [1:21645] 2.359 0.786 0.786 0.786 0.786 ...
 $ Calories55  : num [1:21645] 1.888 0.786 0.786 0.786 0.786 ...
 $ Calories56  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories57  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories58  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 $ Calories59  : num [1:21645] 0.944 0.786 0.786 0.786 0.786 ...
 - attr(*, "spec")=
  .. cols(
  ..   Id = col_double(),
  ..   ActivityHour = col_character(),
  ..   Calories00 = col_double(),
  ..   Calories01 = col_double(),
  ..   Calories02 = col_double(),
  ..   Calories03 = col_double(),
  ..   Calories04 = col_double(),
  ..   Calories05 = col_double(),
  ..   Calories06 = col_double(),
  ..   Calories07 = col_double(),
  ..   Calories08 = col_double(),
  ..   Calories09 = col_double(),
  ..   Calories10 = col_double(),
  ..   Calories11 = col_double(),
  ..   Calories12 = col_double(),
  ..   Calories13 = col_double(),
  ..   Calories14 = col_double(),
  ..   Calories15 = col_double(),
  ..   Calories16 = col_double(),
  ..   Calories17 = col_double(),
  ..   Calories18 = col_double(),
  ..   Calories19 = col_double(),
  ..   Calories20 = col_double(),
  ..   Calories21 = col_double(),
  ..   Calories22 = col_double(),
  ..   Calories23 = col_double(),
  ..   Calories24 = col_double(),
  ..   Calories25 = col_double(),
  ..   Calories26 = col_double(),
  ..   Calories27 = col_double(),
  ..   Calories28 = col_double(),
  ..   Calories29 = col_double(),
  ..   Calories30 = col_double(),
  ..   Calories31 = col_double(),
  ..   Calories32 = col_double(),
  ..   Calories33 = col_double(),
  ..   Calories34 = col_double(),
  ..   Calories35 = col_double(),
  ..   Calories36 = col_double(),
  ..   Calories37 = col_double(),
  ..   Calories38 = col_double(),
  ..   Calories39 = col_double(),
  ..   Calories40 = col_double(),
  ..   Calories41 = col_double(),
  ..   Calories42 = col_double(),
  ..   Calories43 = col_double(),
  ..   Calories44 = col_double(),
  ..   Calories45 = col_double(),
  ..   Calories46 = col_double(),
  ..   Calories47 = col_double(),
  ..   Calories48 = col_double(),
  ..   Calories49 = col_double(),
  ..   Calories50 = col_double(),
  ..   Calories51 = col_double(),
  ..   Calories52 = col_double(),
  ..   Calories53 = col_double(),
  ..   Calories54 = col_double(),
  ..   Calories55 = col_double(),
  ..   Calories56 = col_double(),
  ..   Calories57 = col_double(),
  ..   Calories58 = col_double(),
  ..   Calories59 = col_double()
  .. )
```

Check the structure including the attributes of the data set using str(). From the console output above, we can see that minuteCaloriesWide is a data frame with the dimensions of 62 variables and 21,645 rows. The current column names and data types are also shown. It is noted that the current data type for ActivityHour is character, which should be POSIXct instead.

Hide

```
minuteCaloriesWide$ActivityHour <- parse_date_time(
  minuteCaloriesWide$ActivityHour, "%m/%d/%y %I:%M:%S %p")
class(minuteCaloriesWide$ActivityHour)
```

```
[1] "POSIXct" "POSIXt"
```

Convert the data type for ActivityHour from character to POSIXct format and confirm the result.

Data frame 2: minuteIntensities

Hide

```
str(minuteIntensities)
```

```
spec_tbl_df [1,325,580 x 3] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ Id            : num [1:1325580] 1.5e+09 1.5e+09 1.5e+09 1.5e+09 1.5e+09 ...
 $ ActivityMinute: chr [1:1325580] "4/12/2016 12:00:00 AM" "4/12/2016 12:01:00 AM" "4/12/2016 12:02:00
AM" "4/12/2016 12:03:00 AM" ...
 $ Intensity     : num [1:1325580] 0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "spec")=
  .. cols(
  ..   Id = col_double(),
  ..   ActivityMinute = col_character(),
  ..   Intensity = col_double()
  .. )
```

Check the structure including the attributes of the data set using str(). From the console outputs above, we can see that minuteIntensities is a data frame with the dimensions of 3 variables with 1,325,580 observations. The current column names and data types are also shown. It is noted that the current data type for ActivityMinute is character, which should be datetime instead. Intensity column also needs to be factorised.

Hide

```
minuteIntensities$ActivityMinute <- parse_date_time(
  minuteIntensities$ActivityMinute, "%m/%d/%y %I:%M:%S %p")
class(minuteIntensities$ActivityMinute)
```

```
[1] "POSIXct" "POSIXt"
```

Convert the data type for ActivityMinute from character to POSIXct format and confirm the result.

Hide

```
unique(minuteIntensities$Intensity)
```

```
[1] 0 1 2 3
```

Hide

```
minuteIntensities$Intensity <- minuteIntensities$Intensity %>%
  factor(levels = c(0:3),
         labels = c("Sedentary", "Lightly Active",
                    "Fairly Active", "Very Active"),
         ordered = TRUE)
levels(minuteIntensities$Intensity)
```

```
[1] "Sedentary"      "Lightly Active" "Fairly Active"  "Very Active"
```

Show unique values of Intensity. Factorise the Intensity variable, label and order the levels as “Sedentary” < “Lightly Active” < “Fairly Active” < “Very Active” and show the levels.

Data frame 3: minuteSteps

Hide

```
str(minuteSteps)
```

```
spec_tbl_df [1,325,580 x 3] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ Id            : num [1:1325580] 1.5e+09 1.5e+09 1.5e+09 1.5e+09 1.5e+09 ...
 $ ActivityMinute: chr [1:1325580] "4/12/2016 12:00:00 AM" "4/12/2016 12:01:00 AM" "4/12/2016 12:02:00 
AM" "4/12/2016 12:03:00 AM" ...
 $ Steps         : num [1:1325580] 0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "spec")=
  .. cols(
  ..   Id = col_double(),
  ..   ActivityMinute = col_character(),
  ..   Steps = col_double()
  .. )
```

Check the structure including the attributes of the data set using str(). From the console outputs above, we can see that minuteSteps is a data frame with the dimensions of 3 variables with 1,325,580 observations. The current column names and data types are also shown. It is noted that the current data type for ActivityMinute is character, which should be POSIXct instead.

Hide

```
minuteSteps$ActivityMinute <- parse_date_time(
  minuteSteps$ActivityMinute, "%m/%d/%y %I:%M:%S %p")
class(minuteSteps$ActivityMinute)
```

```
[1] "POSIXct" "POSIXt"
```

Convert the data type for ActivityMinute from character to POSIXct format and confirm the result.

Note: Before merging the 3 data frames (minuteIntensities, minuteSteps and minCaloriesWide) together, we need to tidy minuteCaloriesWide in order to create the common keys (user ID and time of activity) to merge.

Tidy & Manipulate Data I

Reshape minuteCaloriesWide into tidy format

According to Wickham and Grolemund (2016), tidy data rules are variables are in columns, obseravations are in rows, and values are in cells. Therefore, the data frame minuteCaloriesWide is untidy because some column headers [Calories00..Calories59] are observations, not variable names.

Hide

```
minCalories <- minuteCaloriesWide %>% pivot_longer(-c(1,2),
                                                   names_to = "Minute",
                                                   values_to = "Calories")
dim(minCalories)
```

```
[1] 1298700       4
```

Hide

```
head(minCalories)
```

Id <dbl>	ActivityHour <S3: POSIXct>	Minute <chr>	Calories <dbl>
1503960366	2016-04-13	Calories00	1.8876
1503960366	2016-04-13	Calories01	2.2022
1503960366	2016-04-13	Calories02	0.9438
1503960366	2016-04-13	Calories03	0.9438
1503960366	2016-04-13	Calories04	0.9438
1503960366	2016-04-13	Calories05	2.0449

6 rows

Convert minuteCaloriesWide from wide format to a new data frame named ‘minuteCalories’ with long format. 60 Column headers represented the activity minute were gathered to “Minute” column, while the calories consumed each minute is kept in “Calories” column. ‘minuteCalories’ has 1,298,700 rows and 4 variables.

Next, in order to have a common key to merge ‘minuteCalories’ with the other 2 data frames (‘minuteSteps’ & ‘minuteIntensities’), we will need to create a column represent the date time of the activity in minute level beside the user Id details.

Hide

```
str(minCalories$ActivityHour)
```

```
 POSIXct[1:1298700], format: "2016-04-13 00:00:00" "2016-04-13 00:00:00" "2016-04-13 00:00:00" "2016-04
-13 00:00:00" "2016-04-13 00:00:00" ...
```

Hide

```
summary(minCalories$ActivityHour)
```

```
                 Min.                 1st Qu.                  Median                    Mean                 3
rd Qu. 
"2016-04-13 00:00:00" "2016-04-19 23:00:00" "2016-04-27 00:00:00" "2016-04-27 06:10:30" "2016-05-04 10:
00:00" 
                 Max. 
"2016-05-13 08:00:00" 
```

Inspect the structure of ActivityHour column, which currently has the time rounded down to the nearest hour.

Hide

```
minCalories$Minute <- minCalories$Minute %>% substr(9,10) %>% as.numeric()
head(minCalories)
```

Id <dbl>	ActivityHour <S3: POSIXct>	Minute <dbl>	Calories <dbl>
1503960366	2016-04-13	0	1.8876
1503960366	2016-04-13	1	2.2022
1503960366	2016-04-13	2	0.9438
1503960366	2016-04-13	3	0.9438
1503960366	2016-04-13	4	0.9438
1503960366	2016-04-13	5	2.0449

6 rows

Use substring to extract the last 2 numbers in Minute column, which represented the minute level of the activity. Then convert these values to numeric and replace the old values in ‘Minute’ to prepare for the next step.

Hide

```
minute(minCalories$ActivityHour) <- 
  minute(minCalories$ActivityHour) + minCalories$Minute
dim(minCalories)
```

```
[1] 1298700       4
```

Hide

```
head(minCalories)
```

Id <dbl>	ActivityHour <S3: POSIXct>	Minute <dbl>	Calories <dbl>
1503960366	2016-04-13 00:00:00	0	1.8876
1503960366	2016-04-13 00:01:00	1	2.2022

Id	ActivityHour	Minute	Calories
<dbl>	<S3: POSIXct>	<dbl>	<dbl>
1503960366	2016-04-13 00:02:00	2	0.9438
1503960366	2016-04-13 00:03:00	3	0.9438
1503960366	2016-04-13 00:04:00	4	0.9438
1503960366	2016-04-13 00:05:00	5	2.0449

6 rows

The “Minute” was added to “ActivityHour” to create new ActivityHour values, which reflects the activity time recorded every minute for each observation (all timedate details are in one column ActivityHour)

Merge 3 data frames

Hide

```
merged <- minuteIntensities %>%
  left_join(minuteSteps, by = c("Id", "ActivityMinute")) %>%
  right_join(minCalories, by = c("Id", "ActivityMinute" = "ActivityHour")) %>%
  select(-Minute)
dim(merged)
```

```
[1] 1298700       5
```

Hide

```
head(merged)
```

Id	ActivityMinute	Intensity	Steps	Calories
<dbl>	<S3: POSIXct>	<ord>	<dbl>	<dbl>
1503960366	2016-04-13 00:00:00	Lightly Active	4	1.8876
1503960366	2016-04-13 00:01:00	Lightly Active	16	2.2022
1503960366	2016-04-13 00:02:00	Sedentary	0	0.9438
1503960366	2016-04-13 00:03:00	Sedentary	0	0.9438
1503960366	2016-04-13 00:04:00	Sedentary	0	0.9438
1503960366	2016-04-13 00:05:00	Lightly Active	9	2.0449

6 rows

Merge 3 data frames minuteIntensities, minuteSteps and minCalories using common keys that are user Id and time of activity at minute level. For this report, which prepare the data for the analysis of the calories consumption with physical activity, we keep all the record in ‘minCalories’ and find its matching values in ‘minuteSteps’ and ‘minuteIntensity’.

The new data frame is named ‘merged’, which has dimension of 1,298,700 rows and 5 variables

Note: Before performing the Tidy & Manipulate II which might involve with some calculations, we first scan the newly merged data for missing values and special values to handle these values properly.

Scan I

Scan missing values and special value NaN

Hide

```
colSums(is.na(merged))
```

```
          Id ActivityMinute      Intensity          Steps       Calories 
           0              0          20640          20640              0 
```

Hide

```
sapply(merged, function(x) sum( is.nan(x) ))
```

```
          Id ActivityMinute      Intensity          Steps       Calories 
           0              0              0              0              0 
```

Scan the total number of missing values (NA) and special value NaN for each column. The outputs showed that there are 20,640 missing values in Intensity and 20,640 missing values in Steps. There are no NaN (Not a Number) values found.

Hide

```
merged[!complete.cases(merged), ]
```

Id <dbl>	ActivityMinute <S3: POSIXct>	Intensity <ord>	Steps <dbl>	Calories <dbl>
1503960366	2016-05-11 21:00:00	NA	NA	0.78140
1503960366	2016-05-11 21:01:00	NA	NA	0.78140
1503960366	2016-05-11 21:02:00	NA	NA	0.78140
1503960366	2016-05-11 21:03:00	NA	NA	0.93768
1503960366	2016-05-11 21:04:00	NA	NA	0.93768
1503960366	2016-05-11 21:05:00	NA	NA	0.78140
1503960366	2016-05-11 21:06:00	NA	NA	0.78140
1503960366	2016-05-11 21:07:00	NA	NA	0.78140
1503960366	2016-05-11 21:08:00	NA	NA	0.78140
1503960366	2016-05-11 21:09:00	NA	NA	0.78140

1-10 of 20,640 rows Previous **1** 2 3 4 5 6 ... 100 Next

Hide

```
dim(merged[!complete.cases(merged), ])
```

```
[1] 20640     5
```

Subset the ‘merged’ data to obtain a data frame with incomplete cases. There are 20,640 incomplete cases found.

Excluding missing data

Hide

```
minact <- merged[complete.cases(merged), ]
colSums(is.na(minact))
```

```
            Id ActivityMinute      Intensity          Steps       Calories 
             0              0              0              0              0 
```

Hide

```
dim(minact)
```

```
[1] 1278060       5
```

Hide

```
head(minact)
```

Id <dbl>	ActivityMinute <S3: POSIXct>	Intensity <ord>	Steps <dbl>	Calories <dbl>
1503960366	2016-04-13 00:00:00	Lightly Active	4	1.8876
1503960366	2016-04-13 00:01:00	Lightly Active	16	2.2022
1503960366	2016-04-13 00:02:00	Sedentary	0	0.9438
1503960366	2016-04-13 00:03:00	Sedentary	0	0.9438
1503960366	2016-04-13 00:04:00	Sedentary	0	0.9438
1503960366	2016-04-13 00:05:00	Lightly Active	9	2.0449

6 rows

Subset the ‘merged’ data with only complete cases to exclude the missing values and saved it as a new data frame named ‘minact’. According to Taheri 2021a, as the amount of missing data is 20,640 which is less than 2% of the ‘merged’ data set size (1,298,700 observations), these missing values can be excluded as the best strategy to not bias the analysis.

Scan for infinite values

Hide

```
sapply(minact, function(x) sum( is.infinite(x) ))
```

```
            Id ActivityMinute      Intensity          Steps       Calories 
             0              0              0              0              0 
```

Scan for infinite values. There are none infinite values found.

Note: Scan for obvious errors will be performed after Tidy & Manipulate Data II.

Tidy & Manipulate Data II

Hide

```
dayact <- minact %>%
  group_by(Id, floor_date(ActivityMinute, unit = "day"), Intensity) %>%
  summarise(TotalSteps = sum(Steps, na.rm = TRUE),
            TotalCalories = round(sum(Calories, na.rm = TRUE),2),
            CalperStep = round(ifelse(
              TotalSteps == 0 | TotalCalories == 0, 0,
              TotalCalories/TotalSteps),2)
  ) %>%
  ungroup()
```

```
`summarise()` has grouped output by 'Id', 'floor_date(ActivityMinute, unit = "day")'. You can override
using the `.groups` argument.
```

Hide

```
head(dayact)
```

Id <dbl>	floor_date(ActivityMinute, unit = "day") <S3: POSIXct>	Intensity <ord>	TotalSteps <dbl>	TotalCalo <
1503960366	2016-04-13	Sedentary	0	100
1503960366	2016-04-13	Lightly Active	7246	60
1503960366	2016-04-13	Fairly Active	1102	6
1503960366	2016-04-13	Very Active	2387	12
1503960366	2016-04-14	Sedentary	0	104
1503960366	2016-04-14	Lightly Active	6210	50

6 rows | 1-5 of 6 columns

Hide

```
str(dayact)
```

```
tibble [2,773 x 6] (S3: tbl_df/tbl/data.frame)
 $ Id                                      : num [1:2773] 1.5e+09 1.5e+09 1.5e+09 1.5e+09 1.5e+09 ...
 $ floor_date(ActivityMinute, unit = "day"): POSIXct[1:2773], format: "2016-04-13" "2016-04-13" "2016-0
4-13" "2016-04-13" ...
 $ Intensity                               : Ord.factor w/ 4 levels "Sedentary"<"Lightly Active"<..: 1
2 3 4 1 2 3 4 1 2 ...
 $ TotalSteps                              : num [1:2773] 0 7246 1102 2387 0 ...
 $ TotalCalories                           : num [1:2773] 1003.1 607.6 63.4 123.7 1043.7 ...
 $ CalperStep                              : num [1:2773] 0 0.08 0.06 0.05 0 0.08 0.06 0.05 0 0.11 ...
```

From ‘minact’ data frame, creating ‘dayact’ data frame which contains the daily records (TotalSteps, TotalCalories, CalperStep) of each FitBit user (Id) grouped by Id, date and Intensity level. The time of the activity were rounded down to the nearest date(`floor_date(ActivityMinute, unit = "day")`). The ‘dayact’ data frame has dimension of 2,773 rows and 6 variables.

Hide

```
dayact <- rename(dayact,
                 ActivityDate = `floor_date(ActivityMinute, unit = "day")`)
head(dayact)
```

Id <dbl>	ActivityDate <S3: POSIXct>	Intensity <ord>	TotalSteps <dbl>	TotalCalories <dbl>	CalperStep <dbl>
1503960366	2016-04-13	Sedentary	0	1003.10	0.00
1503960366	2016-04-13	Lightly Active	7246	607.65	0.08
1503960366	2016-04-13	Fairly Active	1102	63.39	0.06
1503960366	2016-04-13	Very Active	2387	123.72	0.05
1503960366	2016-04-14	Sedentary	0	1043.69	0.00
1503960366	2016-04-14	Lightly Active	6210	508.87	0.08

6 rows

Rename column `floor_date(ActivityMinute, unit = "day")` to ActivityDate.

Scan II

Scan for obvious errors

Hide

```
Rule <- editset(c("TotalSteps >= 0",
                  "TotalCalories >= 0",
                  "CalperStep >= 0"))
violated <- violatedEdits(Rule, dayact)
summary(violated)
```

```
No violations detected, 0 checks evaluated to NA
```

```
NULL
```

Set constraints for the numeric data in dayact, the TotalSteps, TotalCalories, and CalperStep can only be equal to or greater than 0. Check the dayact for violated values and there is no violation found.

Scan for outliers

Multivariate Outliers

The ‘dayact’ data frame has 3 numeric variables. According to Taheri 2021b, in multivariate setting, the Mahalanobis distance approach is commonly used to detect outliers.

Hide

```
sub_dayact <- dayact %>% select(c("TotalSteps",
                                  "TotalCalories",
                                  "CalperStep"))
head(sub_dayact)
```

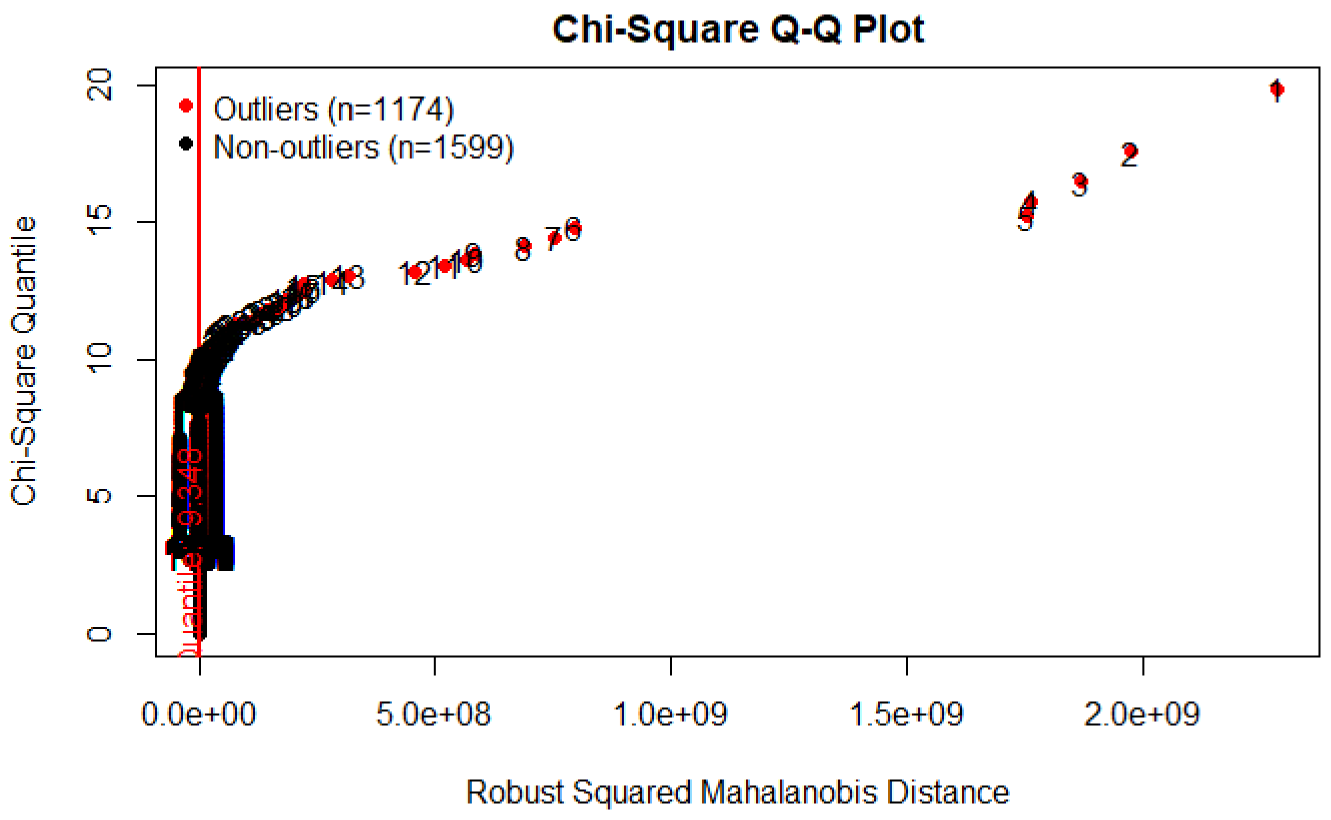
TotalSteps <dbl>	TotalCalories <dbl>	CalperStep <dbl>
0	1003.10	0.00
7246	607.65	0.08
1102	63.39	0.06
2387	123.72	0.05
0	1043.69	0.00
6210	508.87	0.08

6 rows

Subset dayact for the 3 numeric variables and saved it as sub_dayact data frame to prepare for the next step. sub_dayact has 2,773 rows.

Hide

```
results <- mvn(data = sub_dayact, multivariateOutlierMethod = "quan",
               showOutliers = TRUE)
```



Using the Mahalanobis distance method, there are 1,174 observations found to be outliers, which account for more than 40% of the data set. As the data were collected by personal smart Fitbit tracker of each user, the variation among observations might result from the use of different tracker devices and different tracking device using habits among the users.

As the number of possible multivariate outliers is large compared to the data set size, excluding all of these observations might bias the analysis as there is a possibility that these detected outliers by Mahalanobis distance method might not be actual outliers, but instead valuable and interesting points which need to be further investigated.

However, we still can see that there are a few extreme outliers (top right of Chi-Square Q-Q Plot) which are far away from the rest of the observations. Thus, using univariate outliers detecting method for each of the variables might give us more information about the each variable distribution before deciding appropriate method for handling the outliers.

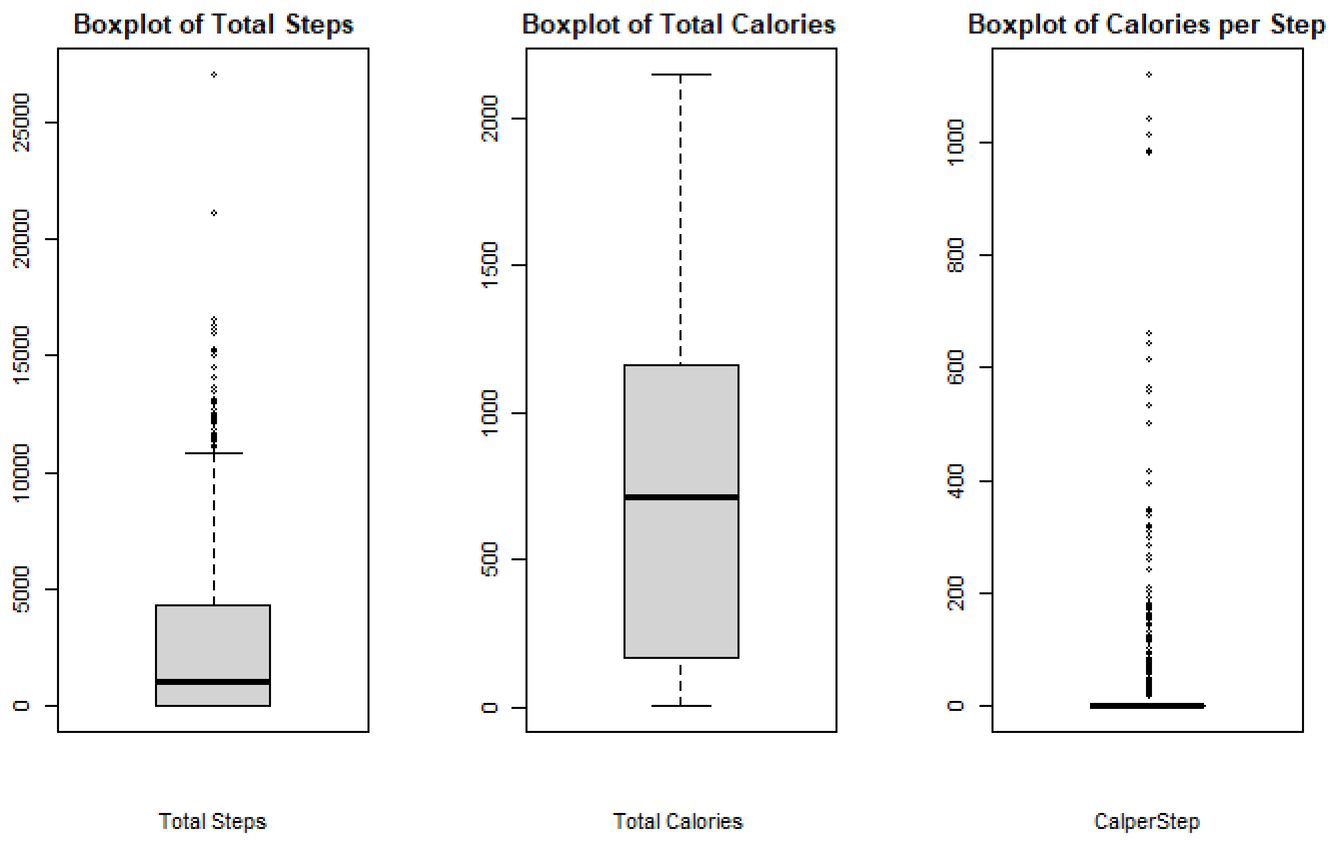
Univariate Outliers

Hide

```
par(mfrow=c(1,3))
Step <- boxplot(dayact$TotalSteps, xlab = "Total Steps",
        main = "Boxplot of Total Steps")
Cal <- boxplot(dayact$TotalCalories, xlab = "Total Calories",
        main = "Boxplot of Total Calories")
```

Hide

```
CalStep <- boxplot(dayact$CalperStep, xlab = "CalperStep",
        main = "Boxplot of Calories per Step")
par(mfrow=c(1,1))
```



Applying the Tukey’s method of outlier detection for each of numeric variable, the 3 boxplots are shown above. All 3 distribution are positive skewed. TotalSteps and CalperStep seem to have many outliers lying outside the upper fence of each distribution while TotalCalories doesn’t seem to have any outliers.

Hide

```
length(Step$out)
```

```
[1] 47
```

Hide

```
length(Cal$out)
```

```
[1] 0
```

Hide

```
length(CalStep$out)
```

```
[1] 137
```

Show the number of outliers detected by Tukey's method in each variables from the 3 boxplots. Out of 2,773 observations of 'dayact', TotalSteps has 47 outliers, TotalCalories doesn't have any outlier and CalperStep has 137 outliers.

Hide

```
summary(sub_dayact)
```

```
   TotalSteps    TotalCalories       CalperStep     
 Min.   :    0   Min.   :   2.25   Min.   :   0.00  
 1st Qu.:    0   1st Qu.: 168.92   1st Qu.:   0.00  
 Median : 1051   Median : 709.81   Median :   0.09  
 Mean   : 2459   Mean   : 748.09   Mean   :   7.62  
 3rd Qu.: 4325   3rd Qu.:1158.30   3rd Qu.:   0.14  
 Max.   :27068   Max.   :2149.25   Max.   :1121.02  
```

The summary of these numeric variables indicated that the maximum value of TotalSteps(27,068) and CalperStep(1,121) are too far away compare to their respective median values (1,051 and 0.09) and the 3rd quantile values (4,325 and 0.14). In fact, it is unlikely for a person to consume 1,121 calories per step, which highly likely to be a result of data processing error as data was extracted from multiple tracking devices.

As the calories consumed (TotalCalories) doesn't have any outliers, it is unlikely for there to be extreme values in corresponding user physical activity (TotalSteps), and extreme values in the calories per physical activities ratios (CalperStep). To phrase it another way, within 2,773 obervations, as 47 observations with extremely high level of physical activity (47 outliers in TotalSteps) have similar level of calories consumption with the majority of the group (no outliers in Calories).

Therefore, we can use capping method to handle these suggested 47 outliers in Total Calories and 137 outliers in CalperStep instead of removing the whole corresponding obsavations and lose information from other variables.

Capping

Hide

```
capped <- as.data.frame(sapply(sub_dayact, FUN = cap))
summary(capped)
```

```
  TotalSteps     TotalCalories       CalperStep     
 Min.   :    0   Min.   :   2.25   Min.   :0.00000  
 1st Qu.:    0   1st Qu.: 168.92   1st Qu.:0.00000  
 Median : 1051   Median : 709.81   Median :0.09000  
 Mean   : 2374   Mean   : 748.09   Mean   :0.09636  
 3rd Qu.: 4325   3rd Qu.:1158.30   3rd Qu.:0.14000  
 Max.   :10812   Max.   :2149.25   Max.   :0.35000  
```

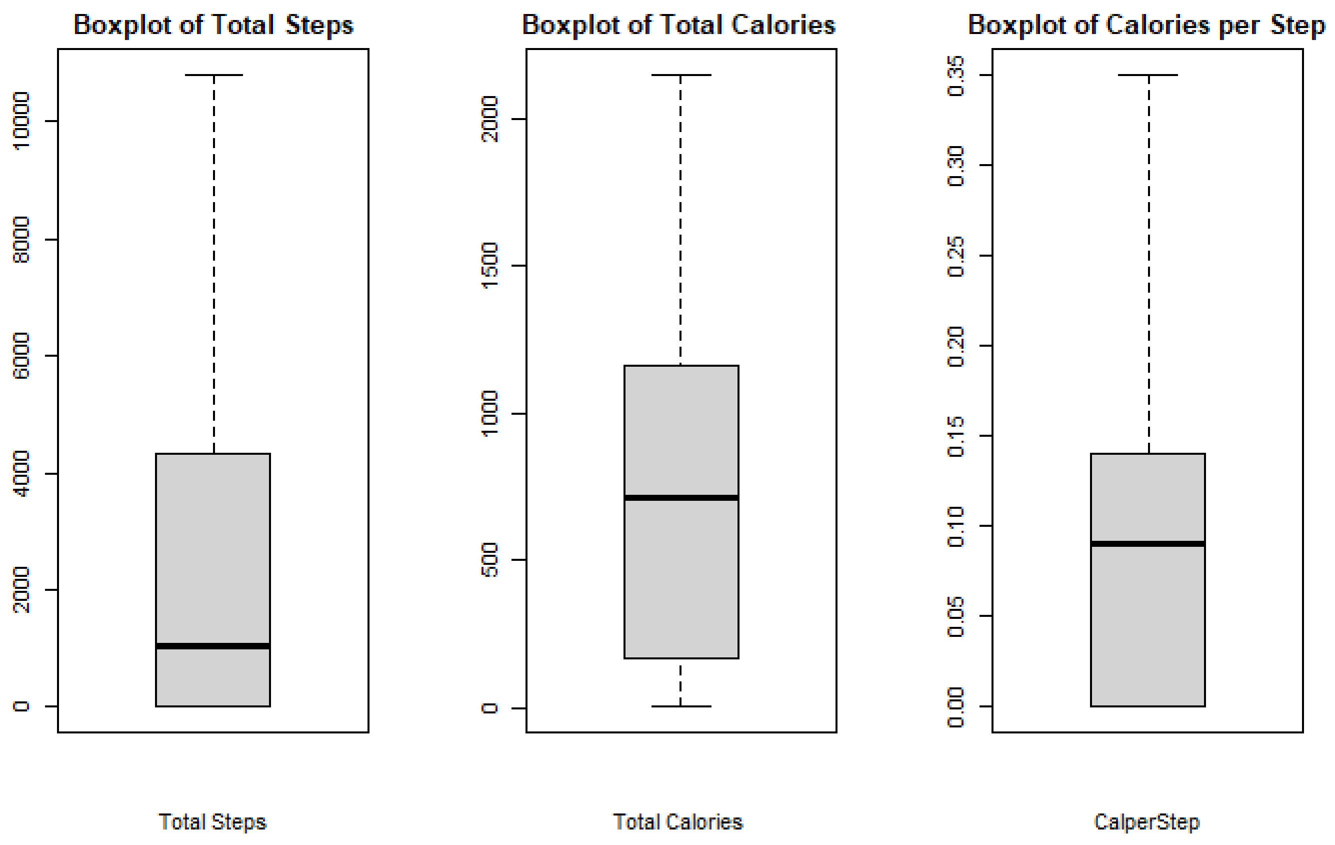
The univariate outliers in sub_dayact is replaced with its nearest non-outlier neighbours which can either be the 5th percentile value or 95th percentile value of the data distribution. The new maximum values for TotalSteps and CalperStep (10,812 and 0.35) have now closer to median and 3rd quantile values of the variable distributions.

Hide

```
par(mfrow=c(1,3))
boxplot(capped$TotalSteps, xlab = "Total Steps",
        main = "Boxplot of Total Steps")
boxplot(capped$TotalCalories, xlab = "Total Calories",
        main = "Boxplot of Total Calories")
```

Hide

```
boxplot(capped$CalperStep, xlab = "CalperStep",
        main = "Boxplot of Calories per Step")
par(mfrow=c(1,1))
```

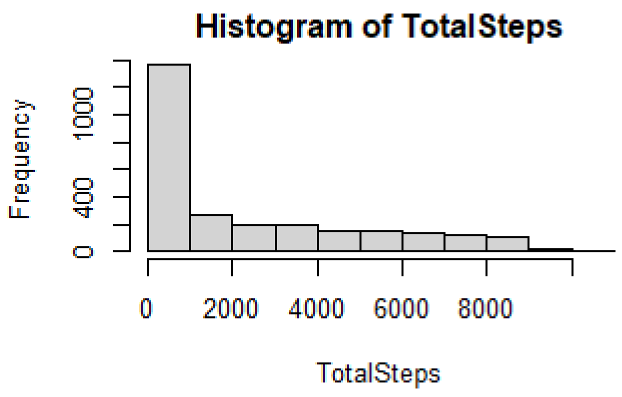


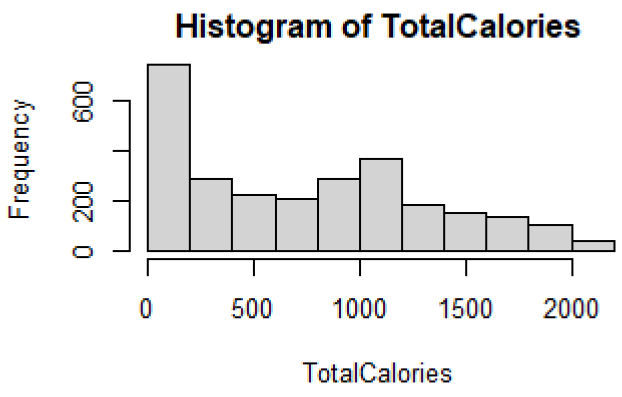
These are the boxplots for the capped data, which showed no outlier.

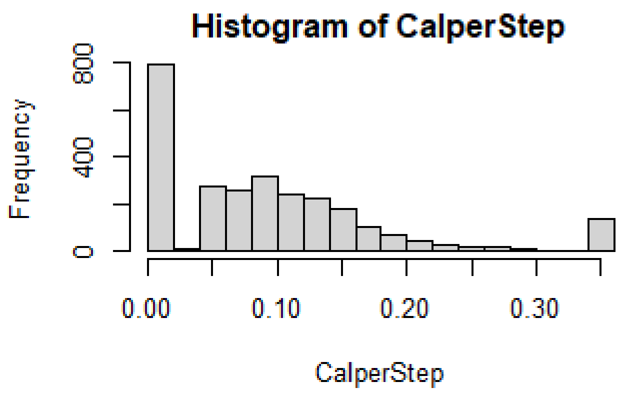
Transform

Hide

```
par(mfrow=c(2,2))
colnames <- colnames(capped)
for (i in 1:3) {
  hist(capped[,i],
       main = paste0("Histogram of ", colnames[i]),
       xlab = colnames[i])
}
par(mfrow=c(1,1))
```







Inspecting the distribution of each variable using histograms. The histograms are showing that TotalSteps, TotalCalories and CalperStep are right skewed. Taheri 2021c suggested that when there are skewed distributions, transformations would help to reduce the skewness and/or heterogeneity of variances and prepare the data for statistical inference with a more symmetric distribution.

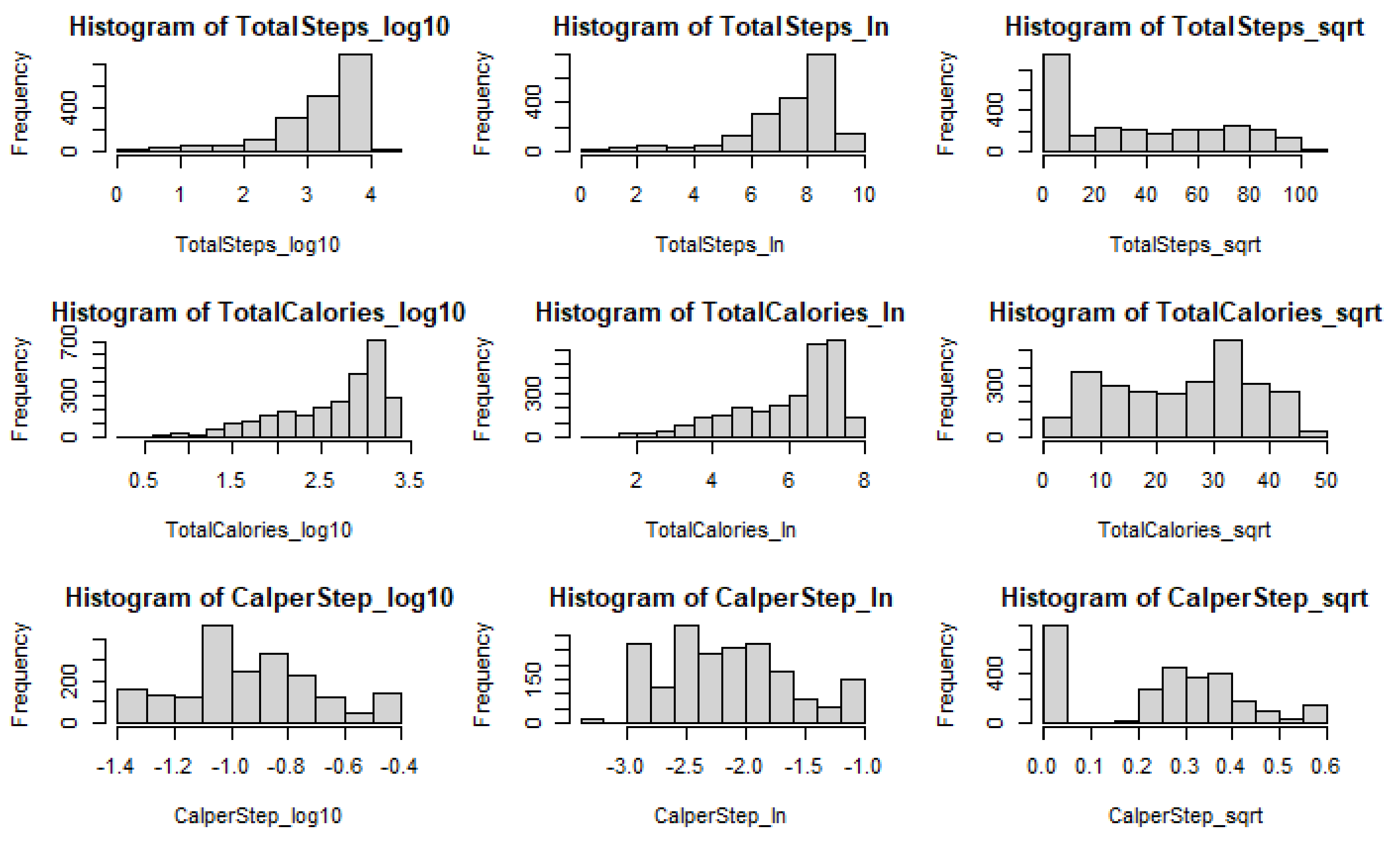
Data transformation via Mathematical operations

As suggested in Taheri 2021c, log transformation base 10 and base e, square root transformation and reciprocal transformation are helpful in reducing the right skewness of the distributions.

Log transformation and Square Root transformation

Hide

```
capped_across <- capped %>%

  mutate(across(everything(), list(log10 = log10, ln = log, sqrt = sqrt)))

par(mfrow=c(3,3))
for (col in names(capped)) {

  for (func in c("log10", "ln", "sqrt")) {

    hist(capped_across[[paste0(col,"_", func)]],

         xlab = paste0(col, "_", func),

         main = paste0("Histogram of ", col, "_", func))
  }
}
par(mfrow=c(1,1))
```

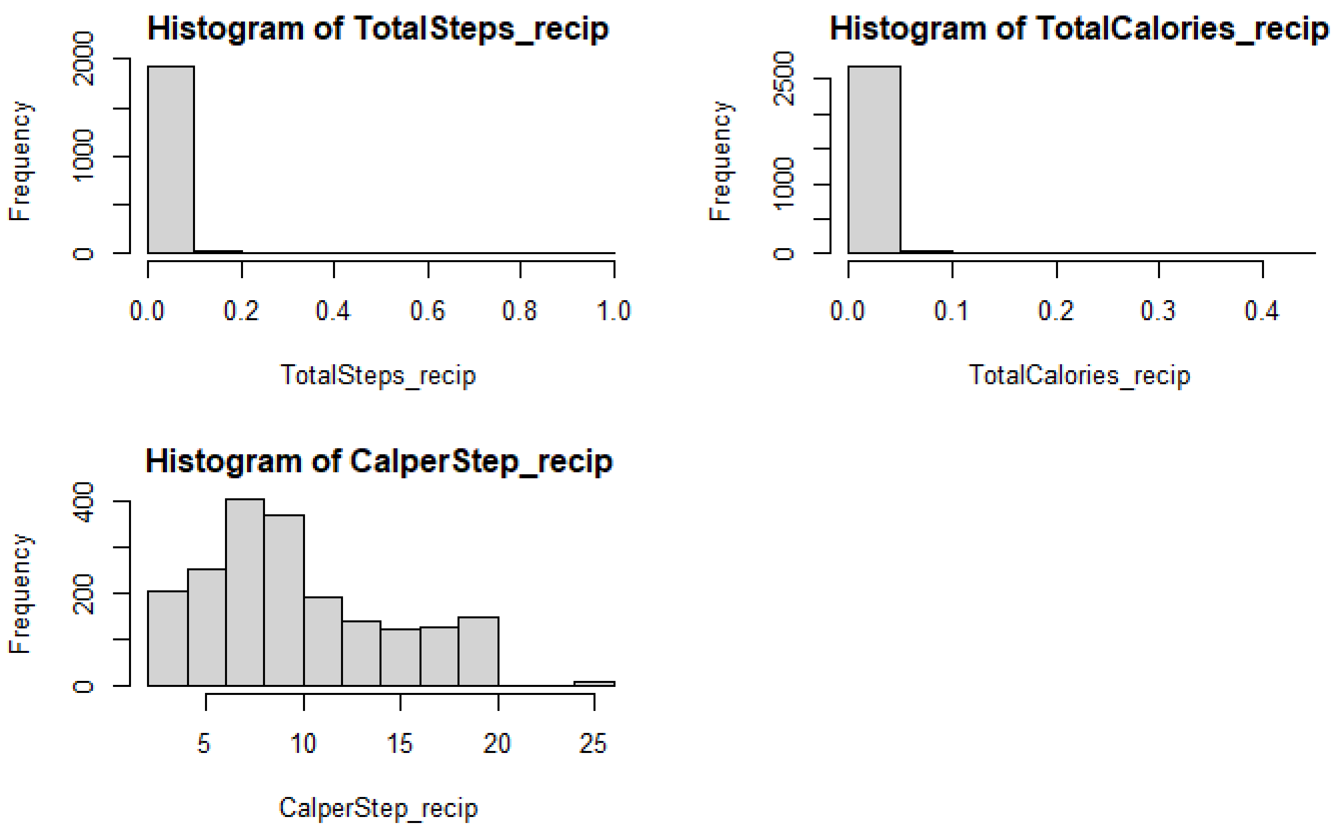


From the histograms, it can be seen that for TotalCalories, square root transformation is more effective in reduce the skewness. For CalpeStep, log base 10 works better than other methods. For TotalSteps, all these 3 mathematical transformation don't seem to be effective.

Reciprocal Transformation

Hide

```
par(mfrow=c(2,2))
capped_recip <- capped^-1
for (i in 1:3) {
  hist(capped_recip[,i],
       main = paste0("Histogram of ", colnames[i],"_recip"),
       xlab = paste0(colnames[i],"_recip"))
}
par(mfrow=c(1,1))
```



Reciprocal transformation helps to improved the distribution for CalperStep, but not for TotalSteps and TotalCalories.

BoxCox transformation

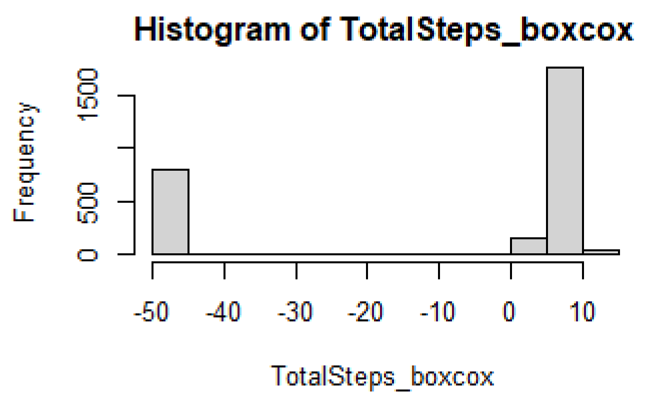
Hide

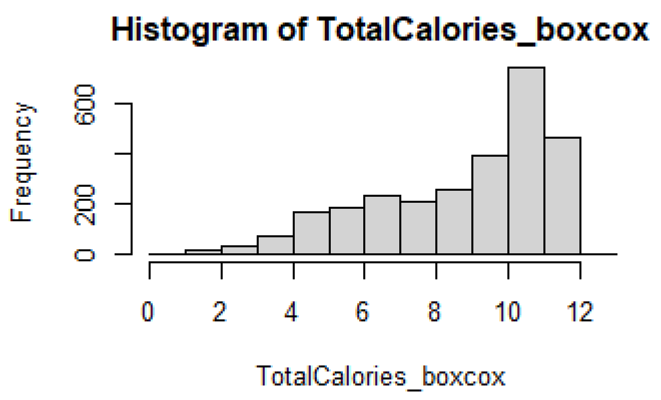
```
par(mfrow=c(2,2))
capped_bc <- as.data.frame(apply(capped, 2, function(x) BoxCox(x, lambda = "auto")))
```

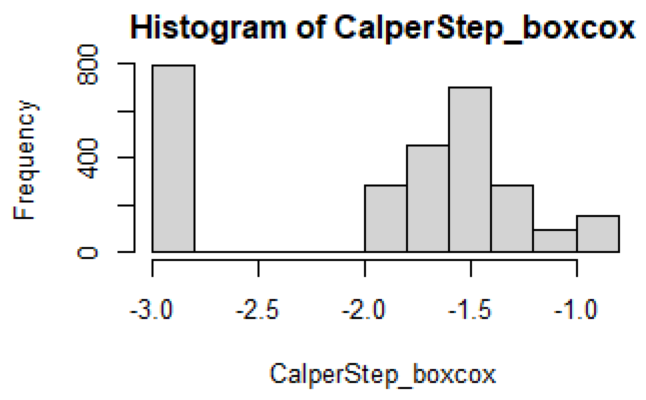
```
Warning in guerrero(x, lower, upper) :
  Guerrero's method for selecting a Box-Cox parameter (lambda) is given for strictly positive data.
Warning in guerrero(x, lower, upper) :
  Guerrero's method for selecting a Box-Cox parameter (lambda) is given for strictly positive data.
```

Hide

```
for (i in 1:3) {
  hist(capped_bc[,i],
       main = paste0("Histogram of ", colnames[i],"_boxcox"),
       xlab = paste0(colnames[i],"_boxcox"))
}
par(mfrow=c(1,1))
```







Applying BoxCox transformation to the dataset and inspect the new distribution with histograms. From the resulted histograms, BoxCox transformation doesn't seem to work well across all the assessed variables.

In summary of data transformation, though square root transformation improved the skewness for Total Calories and log base 10 works for CalpeStep, both transformed distributions are unlikely to be normal. Among the methods used, there are no method can improve the skewness of Total Steps. Therefore, transformation might not bring much benefit in this case.

References

Kaggle 2020, *FitBit Fitness Tracker Data*, data file, Kaggle, viewed 27 September 2021, https://www.kaggle.com/arashnic/fitbit (https://www.kaggle.com/arashnic/fitbit)

Taheri, S 2021a, 'Module 5 Scan: Missing Values', lecture notes, MATH2349, RMIT University, viewed 27 September 2021, http://rare-phoenix-161610.appspot.com/secured/Module_05.html (http://rare-phoenix-161610.appspot.com/secured/Module_05.html)

Taheri, S 2021b, 'Module 6 Scan: Outliers', lecture notes, MATH2349, RMIT University, viewed 27 September 2021, http://rare-phoenix-161610.appspot.com/secured/Module_06.html (http://rare-phoenix-161610.appspot.com/secured/Module_06.html)

Taheri, S 2021c, 'Module 7 Transform: Data Transformation, Standardisation, and Reduction', lecture notes, MATH2349, RMIT University, viewed 27 September 2021, http://rare-phoenix-161610.appspot.com/secured/Module_07.html (http://rare-phoenix-161610.appspot.com/secured/Module_07.html)

Wickham, H & Grolemund, G 2016, *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*, O'Reilly Media Inc, USA