Chapter 3

Data Analytics 1: Data Cleaning and Transformation

Prepared by Dr Yuslina Zakaria

In this session, you will be learning the data cleaning and transforming phase of data analytics phase. Data cleaning refers to the phase of preparing data for analysis to ensure that the data is consistent and correct. The `dplyr` package developed by Hadley Wickham (http://hadley.nz/) makes the data cleaning process much easier for data scientists.

R Packages:

For this lesson, the following packages are used :

```
library(magrittr)
library(dplyr)
library(haven)
```

Note: These packages are parts of `tidyverse` package. Install tidyverse package if you have not done so.

Data set:

Download these datasets from http://tiny.cc/phc410_da1 and place the files in the `input` folder. Create the `input` folder if it does not exist in your project workspace.

- `bmi_adult.csv`
- `subs_nhanes1718.xpt`
- `nhanes_desc.txt`

Instructions:

- 🧑‍🏫 explains the steps for activity you need to follow.
- 🧠 section contains practice questions for you to work on and submit as your lab report.

3.1 Data Cleaning and Transformation using `dplyr`

For the purpose of managing data set, we are going to focus on how to use the `dplyr` package, another core member of the `tidyverse` package. We will explore a few `dplyr` verbs:

- `rename()` to rename columns
- `recode()` and `recode_factor()` to recode values in a column
- `count()` to count discrete values
- `select()` to choose columns
- `filter()` to extract data on conditions
- `arrange()` to order or sort data
- `mutate()` to compute new values
- `group_by` to select specific data
- `summarise()` to create summary statistics on grouped data

In the following activity, data set `bmi_adult.csv` will be used. The description of the data set is as follows:

1. GNDR: Gender of respondents (0: Female, 1: Male)
2. HGHT: Height of respondents (in cm)
3. WGHT: Weight of respondents (in kg)
4. IDX: Category of respondents' BMI
    - 0 - Extremely Underweight
    - 1 - Underweight
    - 2 - Normal
    - 3 - Overweight
    - 4 - Obesity
    - 5 - Extreme Obesity

Import the `bmi_adult.csv` as object `bmi_adult` using `read.csv()`.

```
bmi_adult <- read.csv("input/bmi_adult.csv")
```

Observe the data using `str()`

```
str(bmi_adult)
## 'data.frame':    500 obs. of  4 variables:
##  $ GNDR: int  1 1 0 0 1 1 1 1 1 0 ...
##  $ HGHT: int  174 189 185 195 149 189 147 154 174 169 ...
##  $ WGHT: int  96 87 110 104 61 104 92 111 90 103 ...
##  $ IDX : int  4 2 4 3 3 3 5 5 3 4 ...
```

This data set contains 500 observations with 4 variables i.e. `Gender`, `Height`, `Weight` and `Index`.

3.1.1 The pipe operator %>%

Before we start using `dplyr`, it is crucial for you to know the forward pipe (`%>%`) function provided by `magrittr` package. This package comes with the installation of `tidyverse` suite. The function greatly simplifies the process of data management and will be used throughout the exercises in this chapter.

In computing a "pipe" is a method to create a data stream in the memory of the computer without the need to create intermediary files or R objects. In R, the pipe is represented by:

object %>% operation() ⟶ result

Once started with data from an object, the resulting stream of data can be modified by a function and then passed on to the next function, and then the next etc. The flow of data can be conceptualized as a flow of water going through pipes until it exits.

In the following sections of `dplyr` functions, you will see an alternative approach using pipe, indicated by pipe.

3.1.2 Alter the name of variable using `rename()`

To change the column or variable names of a data frame, `rename()` function can be used.

Observe the column names of `bmi_adult` by printing the names.

```
names(bmi_adult)
## [1] "GNDR" "HGHT" "WGHT" "IDX"
```

As you can see, the column names are not particularly helpful. You can use `rename()` to replace a single or multiple column names. Suppose you want to replace all column names in `bmi_adult` with a new set of names i.e. Gender for GNDR, Height for HGHT, Weight for WGHT, and Index for IDX.

To replace the column names.

```
rename(bmi_adult,
        Gender=GNDR,
        Height=HGHT,
        Weight=WGHT,
        Index=IDX
        )
##     Gender Height Weight Index
## 1        1    174     96     4
## 2        1    189     87     2
## 3        0    185    110     4
## 4        0    195    104     3
## 5        1    149     61     3
...
```

Using the %>% function:

```
#display the colum names
names(bmi_adult)
## [1] "GNDR" "HGHT" "WGHT" "IDX"

#rename all columns
bmi_adult %>% rename(Gender=GNDR,
        Height=HGHT,
        Weight=WGHT,
        Index=IDX
        )
##     Gender Height Weight Index
## 1        1    174     96     4
## 2        1    189     87     2
## 3        0    185    110     4
## 4        0    195    104     3
## 5        1    149     61     3
...
```

Do not forget to use (<-), to return the changes you have made to the column names to the data frame `bmi_adult`.

```
#rename all columns
bmi_adult <- bmi_adult %>% rename(Gender=GNDR,
        Height=HGHT,
        Weight=WGHT,
        Index=IDX
        )
```

3.1.3 Count discrete values using count()

The `count()` function counts the number of observations (or rows) for a specific variable.

Count observations for `Gender`

```
count(bmi_adult,Gender)
##   Gender   n
## 1      0 255
## 2      1 245
```

`count()` (*pipe alternative*):

```
bmi_adult %>% count(Gender)
##   Gender   n
## 1      0 255
## 2      1 245
```

3.1.4 Selecting columns using select()

You can use this function to display only specific columns, as follows:

Select `Height` and `Weight` from `bmi_adult`.

```
select(bmi_adult,Height,Weight)
##     Height Weight
## 1      174     96
## 2      189     87
## 3      185    110
## 4      195    104
## 5      149     61
...
```

`select()` (*pipe alternative*):

```
bmi_adult %>% select(Height,Weight)
##     Height Weight
## 1      174     96
## 2      189     87
## 3      185    110
## 4      195    104
## 5      149     61
...
```

3.1.5 Filtering rows (`filter()`)

`filter()` is used to extract subsets of rows from a data frame by giving a specified condition. It is similar to `subset()` provided by the base package. In conjunction with `filter()`, conditional selections uses comparison operators such as (`==`, `>`, `<`, `!=`, `&`, `|`) to get the subset of data. These operators can be used to create multiple arguments using logical operators, which will be explained in detail in Section 3.1.5.1.

In R, the comparison operators are usually used to effectively filter the observations that users want.

Suppose we want to get all data for female respondents only.

```
#(1) female respondents
filter(bmi_adult,Gender==0)
##     Gender Height Weight Index
## 1        0    185    110     4
## 2        0    195    104     3
## 3        0    169    103     4
## 4        0    159     80     4
## 5        0    192    101     3
...
```

`filter()` (*pipe alternative*):

```
#female respondents
bmi_adult %>% filter(Gender==0)
##     Gender Height Weight Index
## 1        0    185    110     4
## 2        0    195    104     3
## 3        0    169    103     4
## 4        0    159     80     4
## 5        0    192    101     3
...
```

It is worth to mention here that using `=` instead of `==` for comparison of equality will result in errors, as in the following example:

`filter()` (*pipe alternative*):

```
#female respondents
bmi_adult %>% filter(Gender=0)
## Error: Problem with `filter()` input `..1`.
## x Input `..1` is named.
## i This usually means that you've used `=` instead of `==`.
## i Did you mean `Gender == 0`?
```

3.1.5.1 Logical operators

To combine arguments for selecting observations, you will need to use logical or Boolean operators i.e. `&` for **and**, `|` for **or**, and `!` for **not**.

Suppose we want to get `Gender`, `Height` and `Index` columns of female respondents who are overweight.

```
#select Gender, Height and Index for overweight female respondents
select(filter(bmi_adult,Gender==0 & Index==3),Gender, Height, Index)
##    Gender Height Index
## 1       0    195     3
## 2       0    192     3
## 3       0    151     3
## 4       0    197     3
## 5       0    187     3
...
```

`filter()` (*pipe alternative*):

```
#select Gender, Height and Index for overweight female respondents
bmi_adult %>% filter(Gender==0 & Index==3) %>% select(Gender, Height, Index)
##    Gender Height Index
## 1       0    195     3
## 2       0    192     3
## 3       0    151     3
## 4       0    197     3
## 5       0    187     3
...
```

3.1.6 Order data (`arrange()`)

The `arrange()` function is used to reorder the rows of a data frame according to the given variables or columns.

To arrange the data set according to ascending and descending order of `Index` variable.

```
#ascending order
arrange(bmi_adult,Index)
##     Gender Height Weight Index
## 1        0    191     54     0
## 2        1    193     54     0
## 3        1    181     51     0
## 4        1    198     50     0
## 5        0    190     50     0
...

#descending order
arrange(bmi_adult,desc(Index))
##     Gender Height Weight Index
## 1        1    147     92     5
## 2        1    154    111     5
## 3        0    153    107     5
## 4        0    157    110     5
## 5        1    140    129     5
...
```

`arrange()` (*pipe alternative*):

```
#ascending order
bmi_adult %>% arrange(bmi_adult,Index)
##     Gender Height Weight Index
## 1        0    140     76     4
## 2        0    140    146     5
## 3        0    141    126     5
## 4        0    141    136     5
## 5        0    141    143     5
...

#descending order
bmi_adult %>% arrange(desc(Index))
##     Gender Height Weight Index
## 1        1    147     92     5
## 2        1    154    111     5
## 3        0    153    107     5
## 4        0    157    110     5
## 5        1    140    129     5
...
```

3.1.7 mutate() to compute new values

The `mutate()` function is used to compute a new variable or values derived from existing variables in a data frame.

Suppose you want to create a new variable `BMI` that contains BMI values calculated from `Height` (in metre) and `Weight` using the following equation:

$$BMI = \frac{Weight}{(\frac{Height}{100})^2} \tag{3.1}$$

Note: the result from `mutate()` *will be used in the next section thus here it will be stored as* `m.bmi_adult`*.*

```
#create BMI variable using the formula
mutate(bmi_adult,BMI=Weight/(Height/100)^2)
##     Gender Height Weight Index      BMI
## 1        1    174     96     4 31.70828
## 2        1    189     87     2 24.35542
## 3        0    185    110     4 32.14025
## 4        0    195    104     3 27.35043
## 5        1    149     61     3 27.47624
...

#to round the value into two decimal places
new_bmi <- mutate(bmi_adult,BMI=round(Weight/(Height/100)^2,1))
```

`mutate()` (*pipe alternative*):

```
#create BMI variable using the formula
bmi_adult %>% mutate(BMI=Weight/(Height/100)^2)
##     Gender Height Weight Index      BMI
## 1        1    174     96     4 31.70828
## 2        1    189     87     2 24.35542
## 3        0    185    110     4 32.14025
## 4        0    195    104     3 27.35043
## 5        1    149     61     3 27.47624
```

```
...

#to round the value into two decimal places
new_bmi <- bmi_adult %>% mutate(BMI=round(Weight/(Height/100)^2,1))
```

3.1.8 recode() and recode_factor() to recode values in a column

Recode data is one of the most important steps in data analysis that you need to do for data preparation. It is not often to see the original data are set up the way we need them for analysis.

In `bmi_adult`, all variables are imported as numeric variables whereas `Gender` and `Index` are categorical variables that need to be treated differently. The `recode()` function is used if you want to recode the values without transforming the variable into factor. Whereas, `recode_factor()` is used to recode the values and set the variable as factor.

Since `Gender` is nominal and it is known as factor in R, we will first recode `Gender` using `recode_factor()`. Then, we recode the ordinal variable of `Index` based on the description in Section 3.1.

Additional parameter `.ordered=TRUE` must be set for ordinal variables. By default, R will transform other values than the ones we set in `recode()` to NA.

To recode the values in `Gender`.

```
# first print all of the unique values you will need to recode
unique(new_bmi$Gender)
## [1] 1 0

#1) recode nominal Gender: 0 as Female and 1 as Male
new_bmi$Gender <- recode_factor(new_bmi$Gender,`0`= "Female",`1` = "Male")

#2) recode ordinal Index based on the description
#create factors with levels ordered as they appear in the recode call.
new_bmi$Index <- recode_factor(new_bmi$Index,
                               `0`="Extremely Underweight",
                               `1` = "Underweight",
                               `2` = "Normal",
                               `3` = "Overweight",
                               `4` = "Obesity",
                               `5` = "Extreme Obesity",.ordered=TRUE)

#3) verify the new structure of new_bmi
str(new_bmi)
## 'data.frame':    500 obs. of  5 variables:
##  $ Gender: Factor w/ 2 levels "Female","Male": 2 2 1 1 2 2 2 2 2 1 ...
##  $ Height: int  174 189 185 195 149 189 147 154 174 169 ...
##  $ Weight: int  96 87 110 104 61 104 92 111 90 103 ...
##  $ Index : Ord.factor w/ 6 levels "Extremely Underweight"<..: 5 3 5 4 4 4 6 6 4 5 ...
##  $ BMI   : num  31.7 24.4 32.1 27.4 27.5 29.1 42.6 46.8 29.7 36.1 ...

#4) display summary of new_bmi$Index
summary(new_bmi$Index)
## Extremely Underweight           Underweight                Normal
##                    13                    21                    68
##            Overweight               Obesity       Extreme Obesity
##                    68                     130                    196
##                  NA's
##                     4
```

3.1.9 `group_by()` and `summarize()`

The `group_by()` function splits data into group of observations using a factor or categorical variable first. When the data is grouped, `summarize()` (or `summarise()`) can be used to create a summary of the grouped values.

Split the `bmi_adult` into group of gender and get the count for each gender. Then use `n()` function to find the count for each group.

```
test <- group_by(new_bmi,Gender)
summarize(test,Count=n())
## `summarise()` ungrouping output (override with `.groups` argument)
## # A tibble: 2 x 2
##   Gender Count
##   <fct>  <int>
## 1 Female   255
## 2 Male     245
```

`group()` and `summarize()` (*pipe alternative*):

```
new_bmi %>% group_by(Gender) %>% summarize(Count=n())
## `summarise()` ungrouping output (override with `.groups` argument)
## # A tibble: 2 x 2
##   Gender Count
##   <fct>  <int>
## 1 Female   255
## 2 Male     245
```

We can also use built-functions e.g. min, max, median and mean to summarize the group. However, these functions will return `NA` if the variables contain `NA` or missing data as a way to notify user that they have to deal with it. To ignore the missing data, user can set `na.rm=TRUE` to remove the associated observations in the function parameter.

In the following examples, we will use the mutated data frame `new_bmi` which contains a new variable `BMI` created from previous section.

Find the `mean()` of BMI and remove missing data (`NA`). Round the calculated mean into 1 decimal place.

```
new_bmi %>% group_by(Gender) %>% summarize(Average=round(mean(BMI,na.rm=TRUE),1))
## `summarise()` ungrouping output (override with `.groups` argument)
## # A tibble: 2 x 2
##   Gender Average
##   <fct>    <dbl>
## 1 Female    37.5
## 2 Male      38.2
```

3.2 Missing Values

The `is.na()` and `is.nan()` functions can be used to check for both types of missing values.

Check missing value of `Height` in `new_bmi` data frame, report the count and verify the affected rows.

```
#display observations of missing Height
new_bmi %>% filter(is.na(Height))
##   Gender Height Weight Index BMI
## 1   Male     NA    139  <NA>  NA
## 2   Male     NA     52  <NA>  NA
## 3 Female     NA     51  <NA>  NA

#report the count of observations containing empty cell of Height
new_bmi %>% filter(is.na(Height)) %>% count()
##   n
## 1 3

#get which rows containing empty cell of Height to verify the missing value
which(is.na(new_bmi$Height))
## [1] 24 32 71
```

From the displayed results, we can see that there are 3 observations with missing values of `Height`. These missing values are from row 24, 32 and 71.

3.2.1 Complete cases

The `complete.cases()` function can be used to display a logical vector that indicates complete rows (i.e. rows without NA). R will return `TRUE` if the row has no missing cells but `FALSE` for rows containing missing cells.

Return complete rows (or cases).

```
#return logical vector
new_bmi %>% complete.cases()
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
##  [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [49]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [61]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
...
```

Keep only complete rows and store into new object `comp_new_bmi`.

```
#display the complete rows
new_bmi[complete.cases(new_bmi),]
##     Gender Height Weight               Index  BMI
## 1     Male    174     96             Obesity 31.7
## 2     Male    189     87              Normal 24.4
## 3   Female    185    110             Obesity 32.1
## 4   Female    195    104          Overweight 27.4
## 5     Male    149     61          Overweight 27.5
...
```

```
#assign the complete rows to comp_weight
comp_new_bmi <- new_bmi[complete.cases(new_bmi),]
```

The new `comp_weight` remove 4 rows containing missing values and store only complete rows of `new_bmi`.

3.3 Practice

Data set:

This practice will use demographic data set from the 2017-2018 National Health and Nutrition Examination Survey (NHANES) from the Centers for Disease Control website (https://wwwn.cdc.gov/nchs/nhanes/). It is a survey program conducted by the National Center for Health Statistics (NCHS) to assess the health and nutritional status of adults and children in the United States.

Put your answers in the Rmarkdown file `da1_practice.Rmd` found in http://tiny.cc/phc410_da1 and knit to PDF to generate a PDF report.

Use pipe `%>%` function whenever appropriate.

1. Load `dplyr` and `haven` packages.
2. Import SAS transport file `subs_nhanes1718.xpt` from your `input` folder into your RStudio (use `read_xpt()`). Assign the object as `nhanes`. *Ensure that the input file subs_nhanes1718.xpt is in your current working directory*
3. Print the column names of `nhanes`.
4. Using the following names, replace the column names and save your changes to `nhanes`:
   - `RIAGENDR` : Gender
   - `DMDCITZN` : Citizenship
   - `DMDHREDZ` : EducationL
   - `DMDHRMAZ` : MaritalS
   - `DMDHRAGZ` : Age
   - `INDFMPIR` : PovertyR
5. Based on the description in `nhanes_desc.txt`, recode `Gender`, `Citizenship`, `EducationL` and `MaritalS`.
6. Display the sequence no, education level and poverty-income ratio for the first 10 male participants.
7. Count the number of female and male in `nhanes`, grouped by education level.
8. Count the number of rows in `nhanes` with no missing values. *Hint: use complete.cases()*