Data Transformation using **dplyr** and **forcats** packages

Kamarul Imran Musa
Assoc Prof (Epidemiology and Statistics)

2019-02-17

Contents

1 Data transformation

1.1 Definition of data transformation

Data transformation is also known as Data Munging or Data Wrangling. It is loosely the process of manually converting or mapping data from one "raw" form into another format. The process allows for more convenient consumption of the data. In doing so, we will be using semi-automated tools in RStudio.

For more information, please refer https://community.modeanalytics.com/sql/tutorial/data-wrangling-with-sql/

1.2 Data transformation with dplyr package

1.2.1 dplyr package

dplyr is a package grouped inside **tidyverse** collection of packages. **dplyr** package is a very useful package to munge or wrangle or to tranform your data. It is a grammar of data manipulation. It provides a consistent set of verbs that help you solve the most common data manipulation challenges

For more information, please read https://github.com/tidyverse/dplyr

1.3 Common procedures for doing data transformation

When we begin to work with data, common procedures include transforming variables in the dataset.

The common procedures that data analyst does include:

1. reducing the size of dataset by selecting certain variables (or columns)
2. generating new variable from existing variables
3. sorting observation of a variable
4. grouping observations based on certain criteria
5. reducing variable to groups to in order to estimate summary statistic

1.4 Some dplyr functions

For the procedures listed above, the corresponding **dplyr** functions are

1. `dplyr::select()` - to select a number of variables from a dataframe
2. `dplyr::mutate()` - to generate a new variable from existing variables
3. `dplyr::arrange()` - to sort observation of a variable
4. `dplyr::filter()` - to group observations that fulfil certain criteria
5. `dplyr::group_by()` and `dplyr::summarize()` - to reduce variable to groups in order to provide summary statistic

2 Hands-on 1: Create a new project, set up working directory and read your data

2.1 Create a new project or set your working directory

It is very important to ensure you know where your working directory is.

To do so, the best practice is *is to create a new project everytime you want to start new analysis with R.* To do so, create a new project by `File -> New Project`.

If you do not start with a new project, you still need to know **Where is my working directory?**.

So, I will emphasize again, every time you want to start processing your data, please make sure:

1. to use R project to work with your data or analysis
2. if you are not using R project, make sure you are inside the correct working directory. Type `getwd()` to display the active **working directory**. And to set a working directory use `setwd()`.
3. once you are know where your working directory is, you can start read or import data into your working directory. Remember, there are a number of packages you can use to read the data into R. It depends on the format of your data.

For example, we know that data format can be in:

1. SPSS (`.sav`) format,
2. Stata (`.dta`) format,
3. SAS format,
4. MS Excel (`.xlsx`) format
5. Comma-separated-values `.csv` format.
6. other formats

Three packages - **haven**, **readr** and **foreign** packages - are very useful to read or import your data into R memory.

1. **readr** provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf).
2. **readxl** reads .xls and .xlsx sheets.
3. **haven** reads SPSS, Stata, and SAS data.

2.2 *starwars* data

To make life easier and to facilitate reproducibility, we will use examples available from the public domains.

We will produce and reproduce the outputs demonstrated on **tidyverse** website (https://github.com/tidyverse/dplyr).

One of the useful datasets is `starwars` dataset. The `starwars` data comes together with **dplyr** package. This original source of data is from SWAPI, the Star Wars API accessible at http://swapi.co/.

The `starwars` data is class of `tibble`. The data have:

- 87 rows (observations)
- 13 columns (variables)

Now, let us:

1. load the **tidyverse** package
2. examine the column names (variable names)

Loading **tidyverse** packages will load **dplyr** automatically. If you want to load only **dplyr**, just type `library(dplyr)`.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
```

```
##     intersect, setdiff, setequal, union
```

Take a peek at the `starwars` data

```
glimpse(starwars)
```

```
## Observations: 87
## Variables: 13
## $ name       <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Darth Vader", ...
## $ height     <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, 182, 188...
## $ mass       <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, 32.0, 8...
## $ hair_color <chr> "blond", NA, NA, "none", "brown", "brown, grey", "b...
## $ skin_color <chr> "fair", "gold", "white, blue", "white", "light", "l...
## $ eye_color  <chr> "blue", "yellow", "red", "yellow", "brown", "blue",...
## $ birth_year <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, NA, 24.0...
## $ gender     <chr> "male", NA, NA, "male", "female", "male", "female",...
## $ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine", "Alder...
## $ species    <chr> "Human", "Droid", "Droid", "Human", "Human", "Human...
## $ films      <list> [<"Revenge of the Sith", "Return of the Jedi", "Th...
## $ vehicles   <list> [<"Snowspeeder", "Imperial Speeder Bike">, <>, <>,...
## $ starships  <list> [<"X-wing", "Imperial shuttle">, <>, <>, "TIE Adva...
```

Next, we examine the first 10 observations of the data. There are 77 more rows NOT SHOWN. You can also see the types of the variables:

1. `chr` (character),
2. `int` (integer),
3. `dbl` (double)

```
starwars
```

```
## # A tibble: 87 x 13
##    name  height  mass hair_color skin_color eye_color birth_year gender
##    <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
##  1 Luke~    172    77 blond      fair       blue            19   male
##  2 C-3PO    167    75 <NA>       gold       yellow         112   <NA>
##  3 R2-D2     96    32 <NA>       white, bl~ red             33   <NA>
##  4 Dart~    202   136 none       white      yellow          41.9 male
##  5 Leia~    150    49 brown      light      brown           19   female
##  6 Owen~    178   120 brown, gr~ light      blue            52   male
##  7 Beru~    165    75 brown      light      blue            47   female
##  8 R5-D4     97    32 <NA>       white, red red             NA   <NA>
##  9 Bigg~    183    84 black      light      brown           24   male
## 10 Obi-~    182    77 auburn, w~ fair       blue-gray       57   male
## # ... with 77 more rows, and 5 more variables: homeworld <chr>,
## #   species <chr>, films <list>, vehicles <list>, starships <list>
```

3 Hands-on 2: `dplyr::select()` , `dplyr::mutate()` and `dplyr::rename()`

3.1 `dplyr::select()`

When you work with large datasets with many columns, sometimes it is easier to select only the necessary columns to reduce the size of dataset.

This is possible by creating a smaller dataset (less variables). Then you can work on at the initial part of data analysis with this smaller dataset. This will greatly help data exploration.

To create smaller datasets, select some of the columns (variables) in the dataset.

In `starwars` data, we have 13 variables. From this dataset, let us generate a new dataset named as `mysw` with only these 4 variables ,

1. name
2. height
3. mass
4. gender

```
mysw <- starwars %>% select(name, gender, height, mass)
mysw
```

```
## # A tibble: 87 x 4
##    name               gender height  mass
##    <chr>              <chr>   <int> <dbl>
##  1 Luke Skywalker     male      172    77
##  2 C-3PO              <NA>      167    75
##  3 R2-D2              <NA>       96    32
##  4 Darth Vader        male      202   136
##  5 Leia Organa        female    150    49
##  6 Owen Lars          male      178   120
##  7 Beru Whitesun lars female    165    75
##  8 R5-D4              <NA>       97    32
##  9 Biggs Darklighter  male      183    84
## 10 Obi-Wan Kenobi     male      182    77
## # ... with 77 more rows
```

The new dataset `mysw` is now created. You can see it in the `Environment` pane.

3.2 `dplyr::mutate()`

With `dplyr::mutate()`, you can generate new variable.

For example, in the dataset `mysw`, we want to create a new variable named as `bmi`. This variable equals mass in kg divided by squared height (in meter)

$$bmi = \frac{kg}{m^2}$$

```
mysw <- mysw %>% mutate(bmi = mass/(height/100)^2)
mysw
```

```
## # A tibble: 87 x 5
##    name               gender height  mass   bmi
##    <chr>              <chr>   <int> <dbl> <dbl>
##  1 Luke Skywalker     male      172    77  26.0
##  2 C-3PO              <NA>      167    75  26.9
##  3 R2-D2              <NA>       96    32  34.7
##  4 Darth Vader        male      202   136  33.3
##  5 Leia Organa        female    150    49  21.8
##  6 Owen Lars          male      178   120  37.9
##  7 Beru Whitesun lars female    165    75  27.5
##  8 R5-D4              <NA>       97    32  34.0
##  9 Biggs Darklighter  male      183    84  25.1
## 10 Obi-Wan Kenobi     male      182    77  23.2
```

```
## # ... with 77 more rows
```

Now, your dataset `mysw` has 5 columns (variables). The last variable is `bmi`

3.3 `dplyr::rename()`

Now,

1. create a new variable *bmi2* which equals bmi^2.
2. rename *bmi2* to *bmisq*

```
mysw <- mysw %>% mutate(bmi2 = bmi^2)
mysw
```

```
## # A tibble: 87 x 6
##    name               gender height  mass   bmi  bmi2
##    <chr>              <chr>   <int> <dbl> <dbl> <dbl>
##  1 Luke Skywalker     male      172    77  26.0  677.
##  2 C-3PO              <NA>      167    75  26.9  723.
##  3 R2-D2              <NA>       96    32  34.7 1206.
##  4 Darth Vader        male      202   136  33.3 1111.
##  5 Leia Organa        female    150    49  21.8  474.
##  6 Owen Lars          male      178   120  37.9 1434.
##  7 Beru Whitesun lars female    165    75  27.5  759.
##  8 R5-D4              <NA>       97    32  34.0 1157.
##  9 Biggs Darklighter  male      183    84  25.1  629.
## 10 Obi-Wan Kenobi     male      182    77  23.2  540.
## # ... with 77 more rows
```

```
mysw <- mysw %>% rename(bmisq = bmi2)
mysw
```

```
## # A tibble: 87 x 6
##    name               gender height  mass   bmi bmisq
##    <chr>              <chr>   <int> <dbl> <dbl> <dbl>
##  1 Luke Skywalker     male      172    77  26.0  677.
##  2 C-3PO              <NA>      167    75  26.9  723.
##  3 R2-D2              <NA>       96    32  34.7 1206.
##  4 Darth Vader        male      202   136  33.3 1111.
##  5 Leia Organa        female    150    49  21.8  474.
##  6 Owen Lars          male      178   120  37.9 1434.
##  7 Beru Whitesun lars female    165    75  27.5  759.
##  8 R5-D4              <NA>       97    32  34.0 1157.
##  9 Biggs Darklighter  male      183    84  25.1  629.
## 10 Obi-Wan Kenobi     male      182    77  23.2  540.
## # ... with 77 more rows
```

4 Hands-on 3: `dplyr::arrange()` and `dplyr::filter()`

4.1 `dplyr::arrange()`

We can sort data in ascending or descending order.

To do that, we will use `dplyr::arrange()`. It will sort the observation based on the values of the specified variable.

For dataset `mysw`, let us sort the `bmi` from the biggest bmi (descending).

```
mysw <- mysw %>% arrange(desc(bmi))
mysw
```

```
## # A tibble: 87 x 6
##    name                  gender        height  mass   bmi   bmisq
##    <chr>                 <chr>          <int> <dbl> <dbl>   <dbl>
##  1 Jabba Desilijic Tiure hermaphrodite    175  1358 443.  196629.
##  2 Dud Bolt              male              94    45  50.9   2594.
##  3 Yoda                  male              66    17  39.0   1523.
##  4 Owen Lars             male             178   120  37.9   1434.
##  5 IG-88                 none             200   140  35     1225
##  6 R2-D2                 <NA>              96    32  34.7   1206.
##  7 Grievous              male             216   159  34.1   1161.
##  8 R5-D4                 <NA>              97    32  34.0   1157.
##  9 Jek Tono Porkins      male             180   110  34.0   1153.
## 10 Darth Vader           male             202   136  33.3   1111.
## # ... with 77 more rows
```

Now, we will replace the dataset `mysw` with data that contain the `bmi` values from the lowest to the biggest bmi (ascending).

```
mysw <- mysw %>% arrange(bmi)
mysw
```

```
## # A tibble: 87 x 6
##    name          gender height  mass   bmi bmisq
##    <chr>         <chr>   <int> <dbl> <dbl> <dbl>
##  1 Wat Tambor    male      193    48  12.9  166.
##  2 Adi Gallia    female    184    50  14.8  218.
##  3 Sly Moore     female    178    48  15.1  230.
##  4 Roos Tarpals  male      224    82  16.3  267.
##  5 Padmé Amidala female    165    45  16.5  273.
##  6 Lama Su       male      229    88  16.8  282.
##  7 Jar Jar Binks male      196    66  17.2  295.
##  8 Ayla Secura   female    178    55  17.4  301.
##  9 Shaak Ti      female    178    57  18.0  324.
## 10 Barriss Offee female    166    50  18.1  329.
## # ... with 77 more rows
```

4.2 dplyr::filter()

To select observations based on certain criteria, we use the `dplyr::filter()` function.

Here, we will create a new dataset (which we will name as `mysw_m_40`) that contains observations with these criteria:

- gender is male AND
- bmi at or above 40

```
mysw_m_40 <- mysw %>% filter(gender == 'male', bmi >= 40)
mysw_m_40
```

```
## # A tibble: 1 x 6
##   name     gender height  mass   bmi bmisq
##   <chr>    <chr>   <int> <dbl> <dbl> <dbl>
## 1 Dud Bolt male       94    45  50.9 2594.
```

Next, we will create a new dataset (named as `mysw_ht_45`) that contain

- `height` above 200 OR `BMI` above 45, AND
- does not include `NA` (which is missing value) observation for `bmi`

```
mysw_ht_45 <- mysw %>% filter(height >200 | bmi >45, bmi != 'NA')
mysw_ht_45
```

```
## # A tibble: 9 x 6
##   name                  gender        height  mass   bmi   bmisq
##   <chr>                 <chr>          <int> <dbl> <dbl>   <dbl>
## 1 Roos Tarpals          male             224    82  16.3    267.
## 2 Lama Su               male             229    88  16.8    282.
## 3 Tion Medon            male             206    80  18.9    355.
## 4 Chewbacca             male             228   112  21.5    464.
## 5 Tarfful               male             234   136  24.8    617.
## 6 Darth Vader           male             202   136  33.3   1111.
## 7 Grievous              male             216   159  34.1   1161.
## 8 Dud Bolt              male              94    45  50.9   2594.
## 9 Jabba Desilijic Tiure hermaphrodite    175  1358 443.  196629.
```

5 Hands-on 4: `dplyr::group_by()` and `dplyr::summarize`

5.1 `dplyr::group_by()`

The `group_by` function will prepare the data for group analysis.

For example,

1. to get summary values for mean `bmi`, mean `ht` and mean `mass`
2. for male, female, hermaphrodite and none (`gender` variable)

```
mysw_g <- mysw %>% group_by(gender)
mysw_g
```

```
## # A tibble: 87 x 6
## # Groups:   gender [5]
##    name          gender height  mass   bmi bmisq
##    <chr>         <chr>   <int> <dbl> <dbl> <dbl>
##  1 Wat Tambor    male      193    48  12.9  166.
##  2 Adi Gallia    female    184    50  14.8  218.
##  3 Sly Moore     female    178    48  15.1  230.
##  4 Roos Tarpals  male      224    82  16.3  267.
##  5 Padmé Amidala female    165    45  16.5  273.
##  6 Lama Su       male      229    88  16.8  282.
##  7 Jar Jar Binks male      196    66  17.2  295.
##  8 Ayla Secura   female    178    55  17.4  301.
##  9 Shaak Ti      female    178    57  18.0  324.
## 10 Barriss Offee female    166    50  18.1  329.
## # ... with 77 more rows
```

5.2 dplyr::summarize()

Now that we have a group data named mysw_g, now, we would summarize our data using the mean and standard deviation (SD).

```
mysw_g %>% summarise(meanbmi = mean(bmi, na.rm = TRUE),
                     meanht  = mean(height, na.rm = TRUE),
                     meanmass = mean(mass, na.rm = TRUE),
                     sdbmi = sd(bmi, na.rm = TRUE),
                     sdht = sd(height, na.rm = TRUE),
                     sdmass = sd(mass, na.rm = TRUE))
```

```
## # A tibble: 5 x 7
##   gender        meanbmi meanht meanmass  sdbmi  sdht sdmass
##   <chr>           <dbl>  <dbl>    <dbl>  <dbl> <dbl>  <dbl>
## 1 female           18.8   165.     54.0   3.71  23.0   8.37
## 2 hermaphrodite   443.    175    1358   NaN    NaN   NaN
## 3 male             25.7   179.     81.0   6.49  35.4  28.2
## 4 none             35     200     140   NaN    NaN   NaN
## 5 <NA>             31.9   120      46.3   4.33  40.7  24.8
```

To calculate the frequencies

- with one variable

```
freq_species <- starwars %>% count(species, sort = TRUE)
freq_species
```

```
## # A tibble: 38 x 2
##    species      n
##    <chr>    <int>
##  1 Human       35
##  2 Droid        5
##  3 <NA>         5
##  4 Gungan       3
##  5 Kaminoan     2
##  6 Mirialan     2
##  7 Twi'lek      2
##  8 Wookiee      2
##  9 Zabrak       2
## 10 Aleena       1
## # ... with 28 more rows
```

- with two variables

```
freq_species_home <- starwars %>% count(species, homeworld, sort = TRUE)
freq_species_home
```

```
## # A tibble: 58 x 3
##    species  homeworld     n
##    <chr>    <chr>     <int>
##  1 Human    Tatooine      8
##  2 Human    Naboo         5
##  3 Human    <NA>          5
##  4 Gungan   Naboo         3
##  5 Human    Alderaan      3
##  6 Droid    Tatooine      2
##  7 Droid    <NA>          2
```

```
##  8 Human    Corellia      2
##  9 Human    Coruscant     2
## 10 Kaminoan Kamino        2
## # ... with 48 more rows
```

6 Hands-on 5: More complicated dplyr verbs

To be more efficent, use multiple **dplyr** functions in one line of R code

```
starwars %>% filter(gender == "male", height > 100, mass > 100) %>%
  select(height, mass, species) %>%
  group_by(species) %>%
  summarize(mean_ht = mean(height, na.rm = TRUE),
            mean_mass = mean(mass, na.rm = TRUE),
            freq = n())
```

```
## # A tibble: 5 x 4
##   species    mean_ht mean_mass  freq
##   <chr>        <dbl>     <dbl> <int>
## 1 Besalisk      198        102     1
## 2 Human         187.       122     3
## 3 Kaleesh       216        159     1
## 4 Trandoshan    190        113     1
## 5 Wookiee       231        124     2
```

7 Data transformation for categorical variables

7.1 forcats package

Data transformation for categorical variables (factor variables) can be facilitated using the **forcats** package.

8 Hands-on 6: `forcats()`

8.1 Create a dataset

Let us create create a dataset to demonstrate **forcats** package. The dataset will contain

1. a vector column named as **sex1** , values = 0,1
2. a vector column named as **race1** , values = 1,2,3,4
3. a tibble dataframe (dataset) named as **data_f**

```
sex1 <- rbinom(n = 100, size = 1, prob = 0.5)
str(sex1)
```

```
##  int [1:100] 1 1 1 1 0 1 1 0 1 0 ...
```

```
race1 <- rep(seq(1:4), 25)
str(race1)
```

```
##  int [1:100] 1 2 3 4 1 2 3 4 1 2 ...
```

```
data_f <- tibble(sex1, race1)
head(data_f)
```

```
## # A tibble: 6 x 2
##    sex1 race1
##   <int> <int>
## 1     1     1
## 2     1     2
## 3     1     3
## 4     1     4
## 5     0     1
## 6     1     2
```

Now let us see the structure of the dataset. You should see that they are all in the integer (numerical) format

```
str(data_f)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    100 obs. of  2 variables:
##  $ sex1 : int  1 1 1 1 0 1 1 0 1 0 ...
##  $ race1: int  1 2 3 4 1 2 3 4 1 2 ...
```

8.2 Conversion from numeric to factor variables

Now, we will convert the integer (numerical) variable to a factor (categorical) variable.

For example, we will generate a new factor (categorical) variable named as `male` from variable `sex1` (which is an integer variable). We will label `male`as *No* or *Yes*.

We then generate a new factor (categorical) variable named as `race2` from `race1` (which is an integer variable) and label as *Mal, Chi, Ind, Others*

```
data_f$male <- factor(data_f$sex1, labels = c('No', 'Yes'))
data_f$race2 <- factor(data_f$race1, labels = c('Mal', 'Chi', 'Ind', 'Others'))
str(data_f)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    100 obs. of  4 variables:
##  $ sex1 : int  1 1 1 1 0 1 1 0 1 0 ...
##  $ race1: int  1 2 3 4 1 2 3 4 1 2 ...
##  $ male : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 2 1 2 1 ...
##  $ race2: Factor w/ 4 levels "Mal","Chi","Ind",..: 1 2 3 4 1 2 3 4 1 2 ...
```

```
head(data_f) ; tail(data_f)
```

```
## # A tibble: 6 x 4
##    sex1 race1 male  race2
##   <int> <int> <fct> <fct>
## 1     1     1 Yes   Mal
## 2     1     2 Yes   Chi
## 3     1     3 Yes   Ind
## 4     1     4 Yes   Others
## 5     0     1 No    Mal
## 6     1     2 Yes   Chi
```

```
## # A tibble: 6 x 4
##    sex1 race1 male  race2
##   <int> <int> <fct> <fct>
## 1     1     3 Yes   Ind
```

```
## 2     0     4 No    Others
## 3     0     1 No    Mal
## 4     0     2 No    Chi
## 5     0     3 No    Ind
## 6     0     4 No    Others
```

8.3 forcats::fct_recode()

Recode old levels to new levels

Our objectives:

1. For variable **male**, change from `No` vs `Yes` to `Fem` and `Male`
2. Create a new variable **malay** from variable **race2** and label `Chi` to `Non-Malay`, `Ind` to `Non-Malay` and `Others` to `Non-Malay`. But we keep `Mal` as it is

We will use **forcats** packages for that. Below we show two ways of recoding the variables.

```
library(forcats)
data_f$male2 <- data_f$male %>% fct_recode('Fem' = 'No', 'Male' = 'Yes')
data_f <- data_f %>% mutate(malay = fct_recode(race2,
                                               'Non-Malay' = 'Chi',
                                               'Non-Malay' = 'Ind',
                                               'Non-Malay' = 'Others'))
head(data_f) ; tail(data_f)
```

```
## # A tibble: 6 x 6
##    sex1 race1 male  race2  male2 malay
##   <int> <int> <fct> <fct>  <fct> <fct>
## 1     1     1 Yes   Mal    Male  Mal
## 2     1     2 Yes   Chi    Male  Non-Malay
## 3     1     3 Yes   Ind    Male  Non-Malay
## 4     1     4 Yes   Others Male  Non-Malay
## 5     0     1 No    Mal    Fem   Mal
## 6     1     2 Yes   Chi    Male  Non-Malay
```

```
## # A tibble: 6 x 6
##    sex1 race1 male  race2  male2 malay
##   <int> <int> <fct> <fct>  <fct> <fct>
## 1     1     3 Yes   Ind    Male  Non-Malay
## 2     0     4 No    Others Fem   Non-Malay
## 3     0     1 No    Mal    Fem   Mal
## 4     0     2 No    Chi    Fem   Non-Malay
## 5     0     3 No    Ind    Fem   Non-Malay
## 6     0     4 No    Others Fem   Non-Malay
```

9 Summary

dplyr package is a very useful package that encourages users to use proper verb when manipulating variables (columns) and observations (rows).

We have learned to use 5 functions but there are more functions available. Other useful functions include:

1. `dplyr::distinct()`
2. `dplyr::transmutate()`

3. `dplyr::sample_n()` and `dplyr::sample_frac()`

Also note that, package **dplyr** is very useful when it is combined with another function that is `group_by`

If you working with database, you can use **dbplyr** which has been developed to perform very effectively with databases.

For categorical variables, you can use **forcats** package.

10 Self-practice

If you have completed the tutorial above, you may:

1. Read your own data (hints: **haven**, **foreign**) or you can download data from https://www.kaggle.com/datasets . Maybe can try this dataset https://www.kaggle.com/blastchar/telco-customer-churn
2. Create a smaller dataset by selecting some variable (hints: `dplyr::select()`)
3. Creating a dataset with some selection (hints: `dplyr::filter()`)
4. Generate a new variable (hints`dplyr::mutate()`)
5. Creata an object using pipe and combining `dplyr::select()`, `dplyr::filter()` and `dplyr::mutate()` in one single line of R code
6. Summarise the mean, standard deviation and median for numerical variables `dplyr::group_by()` and `dplyr::summarize()`
7. Calculare the number of observations for categorical variables (hints: `dplyr::count()`)
8. Recode a categorical variable (hints: `forcats::fct_recode()`)

11 References

1. dplyr vignettes here https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html
2. forcats examples here http://r4ds.had.co.nz/factors.html
3. reading data into R https://garthtarr.github.io/meatR/rio.html

12 Session

```
sessionInfo()
```

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 17763)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
```

```
## other attached packages:
## [1] forcats_0.3.0  bindrcpp_0.2.2 dplyr_0.7.8
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.0       knitr_1.21       bindr_0.1.1      magrittr_1.5
##  [5] tidyselect_0.2.5 R6_2.3.0         rlang_0.3.0.1    fansi_0.4.0
##  [9] stringr_1.3.1    tools_3.5.2      xfun_0.4         utf8_1.1.4
## [13] cli_1.0.1        htmltools_0.3.6  yaml_2.2.0       assertthat_0.2.0
## [17] digest_0.6.18    tibble_1.4.2     crayon_1.3.4     purrr_0.2.5
## [21] glue_1.3.0       evaluate_0.12    rmarkdown_1.11   stringi_1.2.4
## [25] compiler_3.5.2   pillar_1.3.1     pkgconfig_2.0.2
```