R workshop

Data manupulation & Rmarkdown

Bolin Wu
NEAR, Aging Research Center
2023/01/26 (updated: 2023-04-14)

About me

- Statistician at NEAR, KI.
- MSc in Statistics, Uppsala University.
- Interests:
  - R package dev
  - Statistics
  - Data manipulation
  - guitar

Photo: Maria Yohuang

workshopr package

Install

```r
install.packages("remotes")
remotes::install_github("Bolin-Wu/workshopr",
  subdir = "rpackage",
  force = TRUE
)
```

Load

```r
library(workshopr)
library(tidyverse)
library(here)
```

Data manupulation

tidyverse, assign

Introduction

This session is to share useful data manipulation skills at daily data harmonization work. My main goal is to follow the "don't repeat yourself" (DRY) principle.

It can make our code more readable and reduce our chance of making mistakes.

Suppose you want to change column class.

```
df$cohort1_edu = as.factor(df$cohort1_edu )
df$cohort2_edu  = as.factor(df$cohort2_edu)
# and so on...
```

Get the code

Note: Data frames may seem to be unfit in the PDF. I do not user any html widges since the code will be pulled out for tutorial purpose. Attendants can run the code on their on machine instead to get better view.

```
workshopr::get_code_2023(session = "tidyverse")
```

Content

The content is selected based on data manipulation in real work scenario.

I hope by the end of the workshop, you will have them in your toolbox:

- %>% syntax
- **join** data frames (*join function*)
- **transform** data shape (*pivot_longer*)
- **select** variables based on name pattern (*select*)
- **extract** the label from DTA and SPSS in R (*filter & sjlabelled*)
- **check** missing values (*summarise & across*)
- **mutate data** based on column types (*mutate & across*)
- **bin** variables by percentiles (*case_when*)
- **assign** function

picture credit: https://www.redarmy.in/

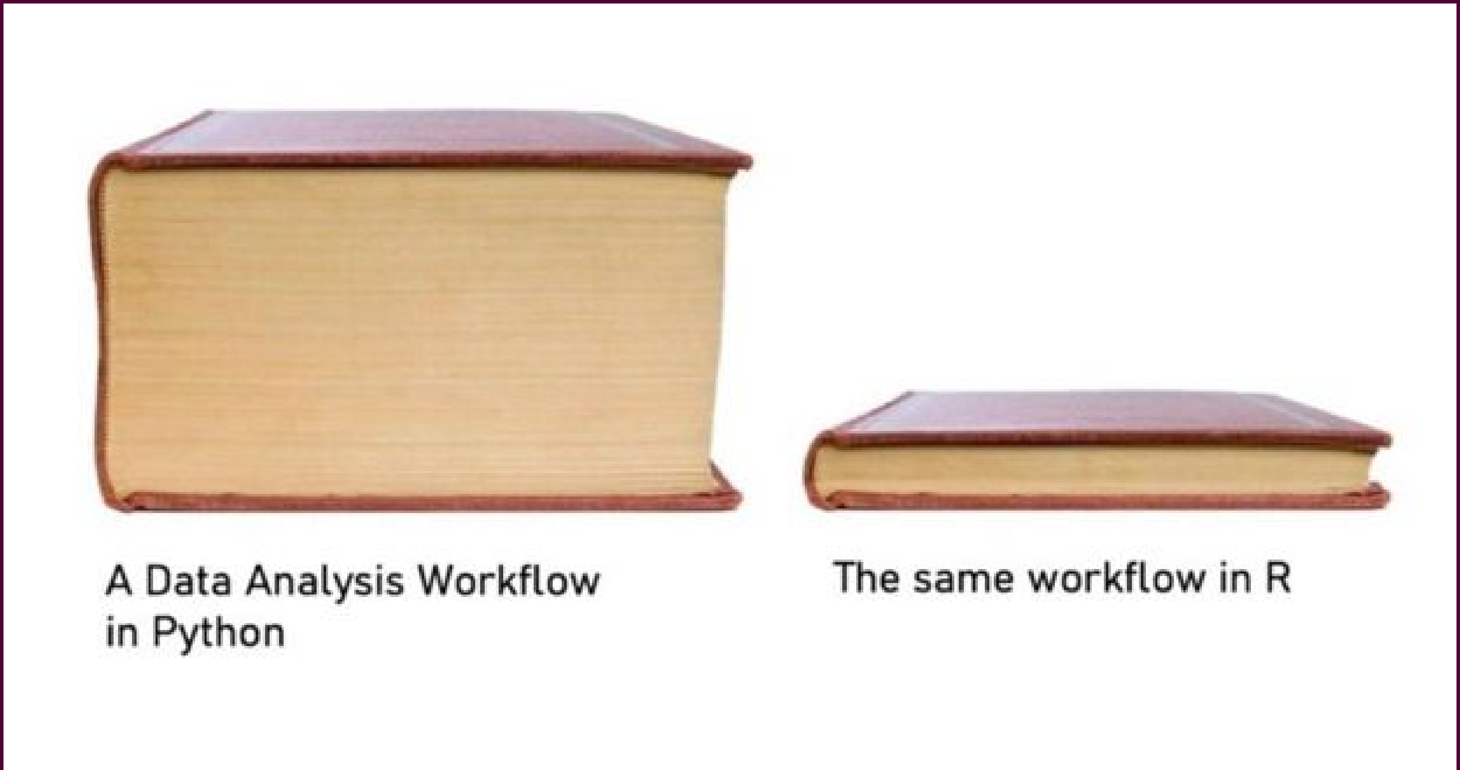
Tidyverse

Many times we just `library(tidyverse)`. Actually Tidyverse is a huge umbrella consists of several powerful visualization and data manipulation package. For example:

- magrittr: pipeline operator `%>%`.
- ggplot2: `ggplot()`.
- dplyr: `select(), filter(), mutate()`.
- stringr: `str_detect(), str_subset()`.

Notes

- Advantage: All at once.
- Disadvantage:
  - Slower to load the whole package.
  - Potential conflicts of function names with other packages. Example here.

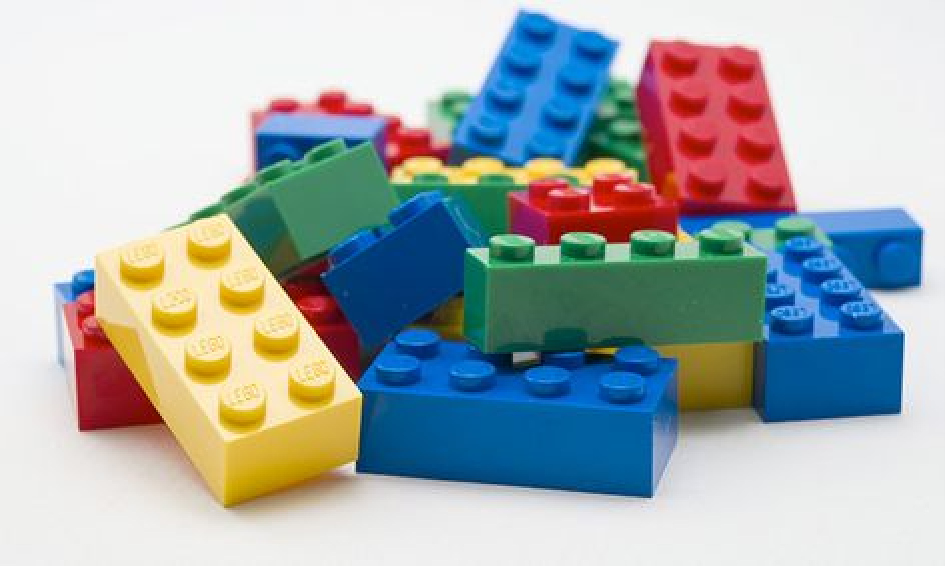


From Matt Dancho's Linkedin

picture credit: Wikipedia of Hadley Wickham

Pipeline operator %>%

Beautiful syntax with pipeline, just like playing LEGO.



Example

```
fake_data %>%
  select(Lopnr, Date_wave1)
```

```
# A tibble: 3,000 × 2
   Lopnr Date_wave1
   <dbl> <date>
 1     1 2001-06-11
 2     2 2002-04-08
 3     3 2001-09-12
 4     4 2001-06-13
 5     5 2001-02-12
 6     6 2001-07-05
 7     7 2001-12-28
 8     8 2002-09-17
 9     9 2001-09-14
10    10 2001-09-21
# … with 2,990 more rows
```

- In addition, filter on the date column

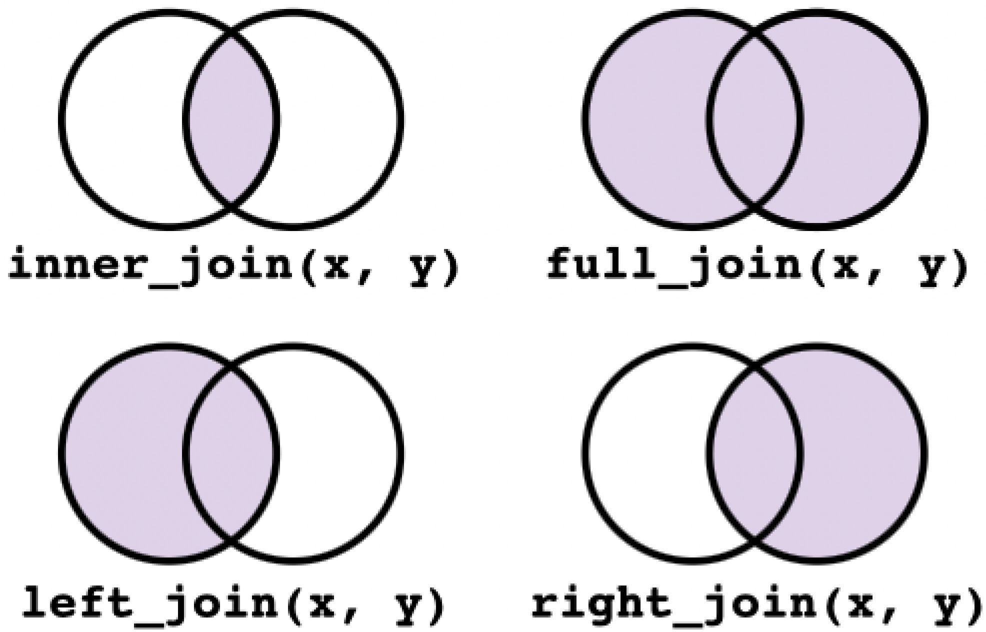
```
fake_data %>%
  select(Lopnr, Date_wave1) %>%
  filter(Date_wave1 > "2002-01-01") %>%
  slice(1:5)
```

```
# A tibble: 5 × 2
  Lopnr Date_wave1
  <dbl> <date>
1     2 2002-04-08
2     8 2002-09-17
3    13 2002-10-13
4    15 2002-04-28
5    19 2002-12-25
```

- You can further stacking group, filter, mutate, etc., with the pipeline operator.

Join data frames

- With `.*_join()` function: There are 4 common types of joins.



- Take `left_join()` as an example:

```
# From  `dplyr` documentation:
df1 <- tibble(x = 1:3)
df2 <- tibble(
  x = c(1, 1, 2),
  y = c("first", "second", "third")
)
df1 %>% left_join(df2)
```

```
# A tibble: 4 × 2
      x y
  <dbl> <chr>
1     1 first
2     1 second
3     2 third
4     3 <NA>
```

By the way, do you know why `tibble()` is better than `data.frame()`?

Transform data shape

Transform data shape is frequently used to clean data. However, for many people, including me, it sounds troublesome. In R, its relevant functions are evolving overtime as well.

In the beginning (2019), I used spread() and gather(). Every time I use `spread()` and `gather()`, it takes me a while to figure out how to fill in 'key' and 'value'. But as you can see from their documentation, their 'lifecycle' is 'superseded'.

Transform data shape

Now I only use `pivot_longer()` and `pivot_wider()` for transforming data. You can find their comprehensive documentation here.

They come with better documentation, more powerful application, and better integration with tidyverse syntax.

Example

Let's assume we received a wide format data:

```
head(fake_data, n = 5)
```

```
# A tibble: 5 × 28
  Lopnr Date_wave1 age_wave1 mmse_w…¹ demen…² Date_wave2 age_
  <dbl> <date>         <dbl>    <dbl>   <dbl> <date>       <d
1     1 2001-06-11      77.6       29       0 2003-06-03     7
2     2 2002-04-08      72.6       18       0 2004-03-07     7
3     3 2001-09-12      79.9       22       0 NA             N
4     4 2001-06-13      79.9       26       0 2003-05-12     8
5     5 2001-02-12      82.6       NA       0 2003-04-20     8
# … with 19 more variables: Date_wave3 <date>, age_wave3 <dbl
#   mmse_wave3 <dbl>, dementia_wave3 <dbl>, Date_wave4 <date>
#   mmse_wave4 <dbl>, dementia_wave4 <dbl>, Date_wave5 <date>
#   mmse_wave5 <dbl>, dementia_wave5 <dbl>, Date_wave6 <date>
#   mmse_wave6 <dbl>, dementia_wave6 <dbl>, age_base <dbl>, s
#   education <dbl+lbl>, and abbreviated variable names ¹mmse
#   ²dementia_wave1, ³age_wave2, ⁴mmse_wave2, ⁵dementia_wave2
```

Example

The column names are:

```
sort(colnames(fake_data))
```

```
 [1] "age_base"        "age_wave1"       "age_wave2"       "age_
 [5] "age_wave4"       "age_wave5"       "age_wave6"       "Date
 [9] "Date_wave2"      "Date_wave3"      "Date_wave4"      "Date
[13] "Date_wave6"      "dementia_wave1" "dementia_wave2" "deme
[17] "dementia_wave4" "dementia_wave5" "dementia_wave6" "educ
[21] "Lopnr"           "mmse_wave1"      "mmse_wave2"      "mmse
[25] "mmse_wave4"      "mmse_wave5"      "mmse_wave6"      "sex"
```

Now, assume for some reason, e.g. merge it with other data set, we want to transform it in a long format.

There are 3 variables with prefix should be formatted: 'age', 'Date' and 'dementia'. For beginners, I would recommend to start small.

Example

- Select the interested columns

```
fake_data %>%
  select(contains("Date")) %>%
  slice(1:5)
```

```
# A tibble: 5 × 6
  Date_wave1 Date_wave2 Date_wave3 Date_wave4 Date_wave5 Date
  <date>     <date>     <date>     <date>     <date>     <dat
1 2001-06-11 2003-06-03 2005-05-27 2007-04-22 2009-04-10 2011
2 2002-04-08 2004-03-07 2006-02-08 2008-02-19 2010-03-26 2012
3 2001-09-12 NA         NA         NA         NA         NA
4 2001-06-13 2003-05-12 2005-04-20 2007-03-28 2009-04-10 2011
5 2001-02-12 2003-04-20 2005-04-06 2007-03-14 2009-03-17 2011
```

Example

Read documentation, try to fill in the arguments.

```
?tidyr::pivot_longer()
```

- The 3 basic arguments are:
  - `cols()`: tells R what variables to pivot.
  - `names_to()`: a new name for columns in `cols()`.
  - `values_to()`: a new name for **values** under the columns in `cols()`.

Example

Let's give a first try:

```
fake_data %>%
  select(Lopnr, contains("Date")) %>%
  pivot_longer(
    cols = contains("Date"),
    names_to = "wave", values_to = "date",
    names_prefix = "Date"
  )
```

```
# A tibble: 18,000 × 3
   Lopnr wave    date
   <dbl> <chr>   <date>
 1     1 _wave1 2001-06-11
 2     1 _wave2 2003-06-03
 3     1 _wave3 2005-05-27
 4     1 _wave4 2007-04-22
 5     1 _wave5 2009-04-10
 6     1 _wave6 2011-04-01
```

Example

The result above looks good, but 'wave' looks a bit strange. I will leave the task to audience to fix this column.

- Do the same with 'dementia' columns

```
fake_data %>%
  select(Lopnr, contains("dementia")) %>%
  pivot_longer(
    cols = contains("dementia"),
    names_to = "wave", names_prefix = "dementia",
    values_to = "dementia"
  )
```

```
# A tibble: 18,000 × 3
   Lopnr wave    dementia
   <dbl> <chr>      <dbl>
 1     1 _wave1         0
 2     1 _wave2         0
 3     1 _wave3         0
```

Data transform exercise (5 - 10 min)

- Read documentation. Change the arguments in the `pivot_longer()` function to get proper wave column.

Example result (first 5 rows):

```
# A tibble: 5 × 3
  Lopnr wave  date
  <dbl> <fct> <date>
1     1 1     2001-06-11
2     1 2     2003-06-03
3     1 3     2005-05-27
4     1 4     2007-04-22
5     1 5     2009-04-10
```

- Merge the two long pivot data sets together.

Consider: is it enough to only join on one column? Why or why not?

Select variables by name pattern

Some times you get data with specific variables. E.g.

- Each wave has its own specific prefix.
- Questionnaire/in person test has its own prefix.
- ...

In this case, the `select()` function can help you. Some useful **selection helpers** are

- `starts_with()`
- `contains()`
- `matches()`
- ...

More details please see documentation:

```
?select()
```

starts_with

```
fake_data %>%
  select(starts_with("Date"))
```

contains

```
fake_data %>%
  select(contains("Date") & contains("1"))
```

- If we want to select variables containing 'Date' or 'age', how to do it? Let's try it.

match

- `match()` is more advanced since it uses regular expression (regex).
- There are many regex cheatsheets online. For exmaple this.
- One useful website to test regex is here.

```
fake_data %>%
  select(matches("Date_wave\\d"))
```

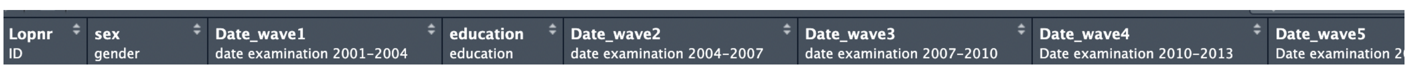
---> Live demo

Get labels from datasets

- When we get SPSS or STATA data sets, usually they come with labels. E.g. in SPSS, one can check them in the "Variable View" tab.

- In R, one can use `view()` function. In the example data, we have:

```
view(fake_data)
```



Lopnr	sex	Date_wave1	education	Date_wave2	Date_wave3	Date_wave4	Date_wave5
ID	gender	date examination 2001-2004	education	date examination 2004-2007	date examination 2007-2010	Date examination 2010-2013	Date examination 2

- A natural question to ask: how to extract the labels in R?

If you run `str()` function, the labels are in the "label" attribute. There are multiple ways to extract the labels.

```
str(fake_data)
```

The one I use is the sjlabelled R package.

```
label_char <- sjlabelled::get_label(fake_data)
label_char
```

```
                       Lopnr                           Date_wave1
                        "ID"  "date examination 2011-2012"
                   age_wave1                           mmse_wave1
     "age at visit 2011-2012"      "mmse at visit 2011-2012"
              dementia_wave1                           Date_wave2
"dementia at visit 2011-2012"  "date examination 2011-2012"
                   age_wave2                           mmse_wave2
     "age at visit 2011-2012"      "mmse at visit 2011-2012"
              dementia_wave2                           Date_wave3
"dementia at visit 2011-2012"  "date examination 2011-2012"
```

The result looks terrible, so we have to fix it by transforming it to be tibble:

```
label_df <- tibble::rownames_to_column(
  as.data.frame(label_char),
  "variable"
)
label_df <- tibble::as_tibble(label_df)
label_df
```

```
# A tibble: 28 × 2
   variable        label_char
   <chr>           <chr>
 1 Lopnr           ID
 2 Date_wave1      date examination 2011-2012
 3 age_wave1       age at visit 2011-2012
 4 mmse_wave1      mmse at visit 2011-2012
 5 dementia_wave1 dementia at visit 2011-2012
 6 Date_wave2      date examination 2011-2012
 7 age_wave2       age at visit 2011-2012
 8 mmse_wave2      mmse at visit 2011-2012
```

The result looks much better! Now we can do lots of things with pipeline.

- filter the label contains 'dementia'

```
label_df %>%
  filter(grepl("dementia", label_char))
```

```
# A tibble: 6 × 2
  variable       label_char
  <chr>          <chr>
1 dementia_wave1 dementia at visit 2011-2012
2 dementia_wave2 dementia at visit 2011-2012
3 dementia_wave3 dementia at visit 2011-2012
4 dementia_wave4 dementia at visit 2011-2012
5 dementia_wave5 dementia at visit 2011-2012
6 dementia_wave6 dementia at visit 2011-2012
```

- filter the variable contains 'wave'

```
label_df %>%
  filter(grepl("wave", variable))
```

```
# A tibble: 24 × 2
   variable        label_char
   <chr>           <chr>
 1 Date_wave1      date examination 2011-2012
 2 age_wave1       age at visit 2011-2012
 3 mmse_wave1      mmse at visit 2011-2012
 4 dementia_wave1 dementia at visit 2011-2012
 5 Date_wave2      date examination 2011-2012
 6 age_wave2       age at visit 2011-2012
 7 mmse_wave2      mmse at visit 2011-2012
 8 dementia_wave2 dementia at visit 2011-2012
 9 Date_wave3      date examination 2011-2012
10 age_wave3       age at visit 2011-2012
# … with 14 more rows
```

Missing values

Count the NA

- If count the NA of one column. It could something like:

```
sum(is.na(fake_data$Date_wave1))
```

```
[1] 0
```

- What if multiple columns?

Count NA in multiple columns

```
fake_data %>%
  summarise(across(
    where(lubridate::is.Date),
    ~ sum(is.na(.))
  ))
```

```
# A tibble: 1 × 6
  Date_wave1 Date_wave2 Date_wave3 Date_wave4 Date_wave5 Date
       <int>      <int>      <int>      <int>      <int>
1          0        207        489        748        983
```

- You may wonder: what is this `~ sum(is.na(.))`?
- It is a purrr-style lambda function.
- You may also wonderL what is `across`?
- It tells `summarise` to do what operations, on which columns.

Missing value exercise (5 - 10 min)

- Read the documentation of `across()` function.
- Count the NA of all numeric/int columns.
- Count the NA of columns contains certain strings, e.g. 'edu'

Example result

```
# A tibble: 1 × 22
  Lopnr age_wa…¹ mmse_…² demen…³ age_w…⁴ mmse_…⁵ demen…⁶ age_
  <int>    <int>   <int>   <int>   <int>   <int>   <int>   <i
1     0        0     135       0     207     323     207
# … with 12 more variables: age_wave4 <int>, mmse_wave4 <int>
#   dementia_wave4 <int>, age_wave5 <int>, mmse_wave5 <int>,
#   dementia_wave5 <int>, age_wave6 <int>, mmse_wave6 <int>,
#   dementia_wave6 <int>, age_base <int>, sex <int>, educatio
#   abbreviated variable names ¹age_wave1, ²mmse_wave1, ³deme
#   ⁴age_wave2, ⁵mmse_wave2, ⁶dementia_wave2, ⁷age_wave3, ⁸mm
```

Mutation based on column type

- Now you should have some understanding of `where()` function.
- When you apply it with `mutate()` function, you can do many manipulations. E.g. round the digits of numeric columns.

```
fake_data %>%
  mutate(across(
    where(is.numeric),
    ~ round(., digits = 2)
  ))
```

```
# A tibble: 3,000 × 28
   Lopnr Date_wave1 age_wave1 mmse_…¹ demen…² Date_wave2 age_
   <dbl> <date>         <dbl>   <dbl>   <dbl> <date>       <d
 1     1 2001-06-11      77.6      29       0 2003-06-03     7
 2     2 2002-04-08      72.6      18       0 2004-03-07     7
```

Bin variables

- `case_when()` function can do the work for us easily, it accommodates well with pipeline syntax.
- For example, if we want to bin the education variable.

```
fake_data %>%
  transmute(mmse_wave1,
    mmse_wave1_bin = case_when(
      between(mmse_wave1, 0, 9) ~ "Severe dementia",
      between(mmse_wave1, 10, 18) ~ "Moderate dement
      between(mmse_wave1, 19, 23) ~ "Mild dementia",
      mmse_wave1 >= 24 ~ "Severe dementia",
      TRUE ~ NA_character_
    )
  )
```

```
# A tibble: 3,000 × 2
   mmse_wave1 mmse_wave1_bin
        <dbl> <chr>
```

Assign function

I would like to spend some time introduce `assign()` function in R.

- Even though it is not part of `tidyverse` family. But it is really useful! For example, when you bulk import databases into R environment; or you want to bulk change the imported data frame names.

- A simple example (run in your R studio):

```
df_names <- c()
set.seed(2023)
for (i in 1:4) {
  df_names[i] <- paste0("edu_", i)
  assign(df_names[i], sample.int(3, 10, replace = T))
  # print the result
  cat(
    "The df name is: ", df_names[i], "\n",
    "its value is: ", get(df_names[i]), "\n"
```

Messy dataframe name

- Imagine in your environment you have these dataframe.
  - Chances are that they are named this way when you get the data files from someone. They have spaces, "-", horrible!

```
objects_name <- c(
  "Cohort1_Baseline_BMI", "Cohort1_FU1_BMI", "Cohort
  "Cohort1FU3_Cohort2FU2", "Cohort2_Baseline_BMI", "
  "Gender data request_20230111", "Gender data reque
  "index", "NEAR_BMI-mortality", "SNAC-K C1 B-F6 coh
  "SNAC-K C1_B", "SNAC-K C1_F1", "SNAC-K C1_F2", "SN
  "SNAC-K C1_F4", "SNAC-K C1_F5"
)

for (i in 1:length(objects_name)) {
  # assign some values to these objects
  assign(objects_name[i], sample(100, 10))
}
```

- Look at your 'Environment' panel. Or `ls()` can retrieve all object names in your R environment. Do you feel familiar?

Let's fix it with `assign()` function (one possible way).

```
clean_names <- gsub(" |-", "_", objects_name)

for (i in 1:length(clean_names)) {
  assign(clean_names[i], get(objects_name[i]))
}

# just show the first few
clean_names[1:5]
```

```
[1] "Cohort1_Baseline_BMI"  "Cohort1_FU1_BMI"        "Cohort1_
[4] "Cohort1FU3_Cohort2FU2" "Cohort2_Baseline_BMI"
```

- Check your 'Environment' panel again.
- The `gsub()` part can be customized with regex syntax. Basically you can fix any kind of messy dataframe names.

R markdown

Basics and daily work uses

Introduction

- A complete Rmarkdown documentation could be found here.
- The key is to understand these three pillows:

Rmarkdown = markdown + yaml heading + code chunk options.

Markdown

- Markdownis a lightweight language for creating formatted text using a plain-text editor.[*]
- Advantage:
  - You do not need to worry about the format. Just focus on writing itself. Could be superfast.
  - Little chance of making the format consistent across the whole documentation.
- Disadvantage:
  - A steep learning curve in the beginning. * However, this disadvantage greatly remedied by many user-friendly editor, e.g. overleaf

Let me give a live example via an online markdown editor

[*] https://en.wikipedia.org/wiki/Markdown

code chunk options

- Code chunk options in RMD can help you control the execution of your code in the file.
- Some useful ones are as follows:[*]

- `include` = FALSE prevents code and results from appearing in the finished file. R Markdown still runs the code in the chunk, and the results can be used by other chunks.
- `echo` = FALSE prevents code, but not the results from appearing in the finished file. This is a useful way to embed figures.
- `message` = FALSE prevents messages that are generated by code from appearing in the finished file.
- `warning` = FALSE prevents warnings that are generated by code from appearing in the finished.

[*] https://rmarkdown.rstudio.com/lesson-3.html

picture credit: https://www.redarmy.in/

```
```{r import package, message=FALSE}
library(tidyverse)
library(Hmisc)
```
```

```
```{r, include=FALSE}
library(DT)
```
```

local option

```
```{r setup, include=FALSE}
# By default, all codes are hidden.
knitr::opts_chunk$set(message = FALSE, warning = FALSE, include = TRUE)
library(workshopr)
library(tidyverse)
library(DT)
```
```

global option

RMD chunk option exercises (1–3 minutes)

- Print the column names of `fake_data`, but do not show the code itself.
- Print the code above, but do not show the result.
- Neither print the code nor the result, but preserve the evaluation of execution for further analysis.

YAML

```
---
title: "A Cool Presentation"
output: html_document
---
```

Or

```
---
title: "A Cool Presentation"
output:
  word_document:
    toc: yes
    toc_depth: 2
---
```

- The YAML heading basically defines the type of output file, layout, etc.

Live demo in R studio

- Get the templates of html or word output:

```
workshopr::get_rmd_2023(
  name = "pretty_template",
  output_file = "word"
)
workshopr::get_rmd_2023(
  name = "pretty_template",
  output_file = "html"
)
```

- Let's take a look together!

Final exercise

Please check this link here.

Wrap up

- Many times, avoiding repetitive coding can help you, also others to review your code.
- In the beginning, the Rmarkdown has steep learning curve, but once you are familiar with it, your life will be easier.
- This whole slide is written in Rmarkdown. Source code is here. The R example code is pulled from Rmarkdown, I do not need to copy and paste.

- The materials are collected in Github repo: https://github.com/Bolin-Wu/workshopr

- Hope today's content could be an enlightment to you. If you have any question, please contact me: bolin.wu@ki.se;