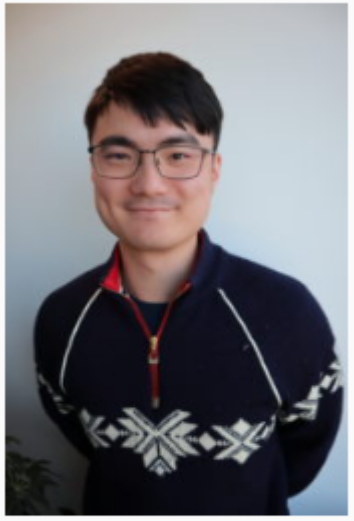
R workshop

Data manupulation & Rmarkdown

Bolin Wu
NEAR, Aging Research Center

2023/01/26 (updated: 2023-02-01)

About me



Bolin Wu. Photo: Maria Yohuang

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

workshopr package

Install

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

Load

```
library(workshopr)
library(tidyverse)
```

Data manupulation

tidyverse, assign

Introduction

This session is to share useful data manipulation skills at daily epidemiology 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.

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

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*)
- **filter** variables based on name pattern (*select*)
- **extract** the label from DTA and SPSS in R (*filter*)
- **check** missing values (*summarise & across*)
- **mutate data** based on column types (*mutate & across*)
- **bin** variables by percentiles (*cut*)
- **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:

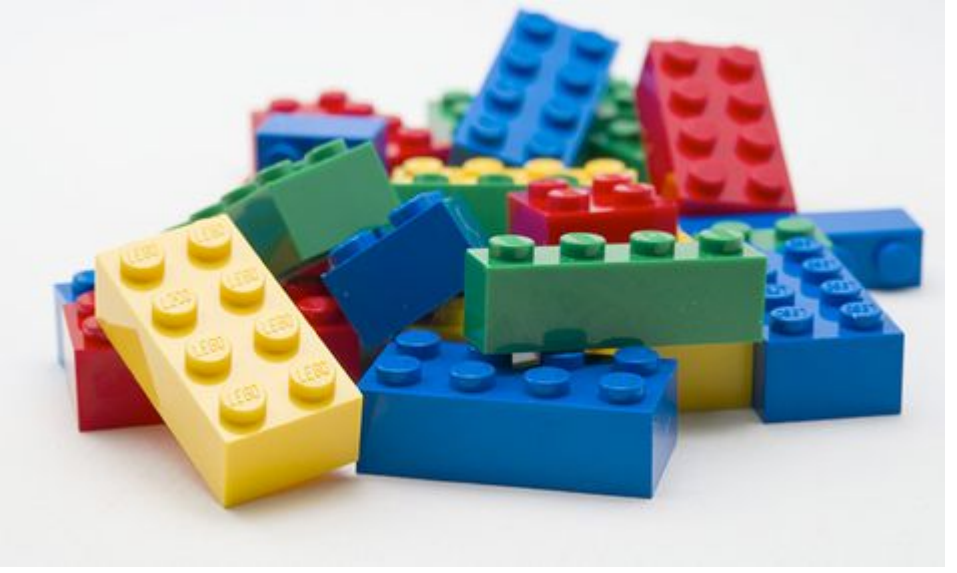
- magrittr: pipeline operator %>%.
- ggplot2: `ggplot()`.
- dplyr: `select()`, `filter()`, `mutate()`

Notes

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

Pipeline operator %>%

Beautiful syntax with pipeline, just like playing LEGO.



Example

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

```
# A tibble: 3,365 x 2
   Lopnr Date_wave1
   <int> <date>
 1  1284 1848-01-16
 2  2013 1788-07-16
 3  1618 1833-09-17
 4  3190 1803-04-08
 5  1103 1705-06-10
 6  2193 1733-11-07
 7  2555 1792-08-31
 8   254 1768-04-16
 9  2250 1810-02-09
10  2258 1722-08-02
# ... with 3,355 more rows
```

- In addition, filter on the date column

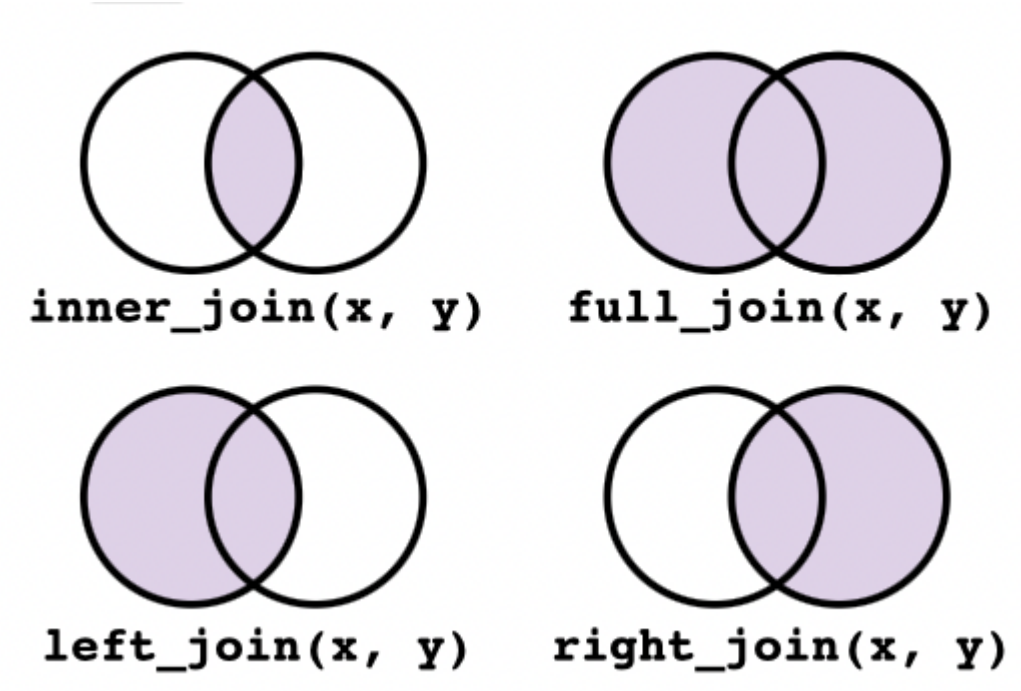
```
fake_snack_df %>%
  select(Lopnr, Date_wave1) %>%
  filter(Date_wave1 > "1800-01-01") %>%
  slice(1:5)
```

```
# A tibble: 5 x 2
  Lopnr Date_wave1
  <int> <date>
1  1284 1848-01-16
2  1618 1833-09-17
3  3190 1803-04-08
4  2250 1810-02-09
5  2835 1813-12-12
```

- 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:

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

```
Joining, by = "x"
```

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

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_snack_df, n = 5)
```

```
# A tibble: 5 x 15
  Lopnr    sex Date_wave1 education Date_wave2 Date_wave3 Dat
  <int>  <dbl> <date>         <dbl> <date>     <date>     <da
1  1284 -1.62  1848-01-16   -1.00   1703-06-26 1807-08-10 176
2  2013  0.384 1788-07-16    0.883  1785-09-24 1811-08-26 181
3  1618 -0.363 1833-09-17    0.0753 1792-10-19 1781-01-06 174
4  3190  0.682 1803-04-08    0.0943 1829-04-27 1744-02-04 184
5  1103  0.521 1705-06-10    1.37   1734-12-24 1746-03-13 182
# ... with 8 more variables: Date_wave5 <date>, Date_wave6 <d
#   dementia_wave1 <dbl>, dementia_wave2 <dbl>, dementia_wave
#   dementia_wave4 <dbl>, dementia_wave5 <dbl>, dementia_wave
```

Example

The column names are:

```
sort(colnames(fake_snack_df))
```

```
 [1] "Date_wave1"     "Date_wave2"     "Date_wave3"     "Date
 [5] "Date_wave5"     "Date_wave6"     "dementia_wave1" "deme
 [9] "dementia_wave3" "dementia_wave4" "dementia_wave5" "deme
[13] "education"      "Lopnr"          "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 two variables with prefix should be formatted: 'Date' and 'dementia'. For beginners, I would recommend to start small.

Example

- Select the interested columns

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

```
# A tibble: 5 x 6
  Date_wave1 Date_wave2 Date_wave3 Date_wave4 Date_wave5 Date
  <date>     <date>     <date>     <date>     <date>     <dat
1 1848-01-16 1703-06-26 1807-08-10 1764-12-13 1774-04-04 1739
2 1788-07-16 1785-09-24 1811-08-26 1812-05-24 1777-11-08 1825
3 1833-09-17 1792-10-19 1781-01-06 1746-09-20 1804-04-12 1764
4 1803-04-08 1829-04-27 1744-02-04 1846-08-24 1766-03-26 1735
5 1705-06-10 1734-12-24 1746-03-13 1829-11-30 1711-02-15 1710
```

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

```
# A tibble: 20,190 x 3
   Lopnr wave   date
   <int> <chr>  <date>
 1  1284 _wave1 1848-01-16
 2  1284 _wave2 1703-06-26
 3  1284 _wave3 1807-08-10
 4  1284 _wave4 1764-12-13
 5  1284 _wave5 1774-04-04
 6  1284 _wave6 1739-04-12
```

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

```
# A tibble: 20,190 x 3
   Lopnr wave   dementia
   <int> <chr>     <dbl>
 1  1284 _wave1   -0.246
 2  1284 _wave2   -0.704
 3  1284 _wave3   -0.508
```

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 x 3
  Lopnr wave  date
  <int> <fct> <date>
1  1284 1     1848-01-16
2  1284 2     1703-06-26
3  1284 3     1807-08-10
4  1284 4     1764-12-13
5  1284 5     1774-04-04
```

- 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_snack_df %>%
  select(starts_with("Date"))
```

contains

```
fake_snack_df %>%
  select(contains("Date") & contains("wave"))
```

- Imagine if only using `contains('Date')` without '&' operator. What result would look like? Then try it.

match

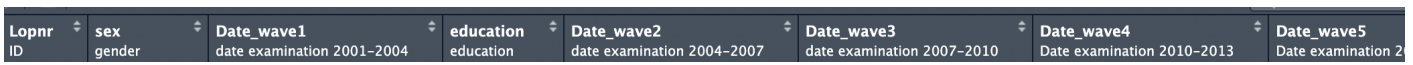
- `match()` is more advanced since it uses regular expression (regex).
- There are many regex cheatsheets online. For exmaple this.
- One useful website to test regex is here.

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

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_snack_df)
```



Lopnr ID	sex gender	Date_wave1 date examination 2001–2004	education education	Date_wave2 date examination 2004–2007	Date_wave3 date examination 2007–2010	Date_wave4 Date examination 2010–2013	Date_wave5 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_snack_df)
```

The one I use is the sjlabelled R package.

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

```
                     Lopnr                        sex
                      "ID"                   "gender"
                Date_wave1                  education
"date examination 2001-2004"                "education"
                Date_wave2                  Date_wave3
"date examination 2004-2007" "date examination 2007-2010"
                Date_wave4                  Date_wave5
"Date examination 2010-2013" "Date examination 2013-2016"
                Date_wave6              dementia_wave1
"Date examination 2016-2019"        "dementia 2001-2004"
```

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_df <- tibble::as_tibble(label_df)
label_df
```

```
# A tibble: 15 x 2
   variable       label_char
   <chr>          <chr>
 1 Lopnr          ID
 2 sex            gender
 3 Date_wave1     date examination 2001-2004
 4 education      education
 5 Date_wave2     date examination 2004-2007
 6 Date_wave3     date examination 2007-2010
 7 Date_wave4     Date examination 2010-2013
 8 Date_wave5     Date examination 2013-2016
 9 Date_wave6     Date examination 2016-2019
10 dementia_wave1 dementia 2001-2004
11 dementia_wave2 dementia 2004-2007
```

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 x 2
  variable       label_char
  <chr>          <chr>
1 dementia_wave1 dementia 2001-2004
2 dementia_wave2 dementia 2004-2007
3 dementia_wave3 dementia 2007-2010
4 dementia_wave4 dementia 2010-2013
5 dementia_wave5 dementia 2013-2016
6 dementia_wave6 dementia 2016-2019
```

- filter the variable contains 'wave'

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

```
# A tibble: 12 x 2
   variable       label_char
   <chr>          <chr>
 1 Date_wave1     date examination 2001-2004
 2 Date_wave2     date examination 2004-2007
 3 Date_wave3     date examination 2007-2010
 4 Date_wave4     Date examination 2010-2013
 5 Date_wave5     Date examination 2013-2016
 6 Date_wave6     Date examination 2016-2019
 7 dementia_wave1 dementia 2001-2004
 8 dementia_wave2 dementia 2004-2007
 9 dementia_wave3 dementia 2007-2010
10 dementia_wave4 dementia 2010-2013
11 dementia_wave5 dementia 2013-2016
12 dementia_wave6 dementia 2016-2019
```

Missing values

Count the NA

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

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

```
[1] 0
```

- What if multiple columns?

Count NA in multiple columns

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

```
# A tibble: 1 x 6
  Date_wave1 Date_wave2 Date_wave3 Date_wave4 Date_wave5 Date
       <int>      <int>      <int>      <int>      <int>
1          0          0          0          0          0
```

- You may wonder: what is this `~ sum(is.na(.))`?
- It is a purrr-style lambda function.
- One can also use `colSums(is.na(fake_snack_df))`.

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 x 9
  Lopnr   sex education dementi~1 demen~2 demen~3 demen~4 dem
  <int> <int>     <int>     <int>   <int>   <int>   <int>   <
1     0     0         0         0       0       0       0
# ... with abbreviated variable names 1: dementia_wave1,
#   2: dementia_wave2, 3: dementia_wave3, 4: dementia_wave4,
#   5: dementia_wave5, 6: dementia_wave6

# A tibble: 1 x 1
  education
```

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

```
# A tibble: 3,365 x 15
   Lopnr   sex Date_wave1 education Date_wave2 Date_wave3 Dat
   <dbl> <dbl> <date>          <dbl> <date>     <date>     <da
 1  1284 -1.62 1848-01-16     -1    1703-06-26 1807-08-10 176
 2  2013  0.38 1788-07-16      0.88 1785-09-24 1811-08-26 181
 3  1618 -0.36 1833-09-17      0.08 1792-10-19 1781-01-06 174
 4  3190  0.68 1803-04-08      0.09 1829-04-27 1744-02-04 184
 5  1103  0.52 1705-06-10      1.37 1734-12-24 1746-03-13 182
```

Bin variables

- `cut()` 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_snack_df %>%
  transmute(education, edu_bin = cut(education,
    breaks = 3,
    labels = c("low", "medium", "high")
  ))
```

```
# A tibble: 3,365 x 2
   education edu_bin
       <dbl> <fct>
 1   -1.00   medium
 2    0.883  medium
 3    0.0753 medium
 4    0.0943 medium
 5    1.37   high
```

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", "Cohort1
  "Cohort1FU3_Cohort2FU2", "Cohort2_Baseline_BMI", "C
  "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", "SNA
  "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.

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"
[3] "Cohort1_FU2_BMI"       "Cohort1FU3_Cohort2FU2"
[5] "Cohort2_Baseline_BMI"
```

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

R markdown

Basics and daily work uses

Introduction

- A complete Rmarkdown documentation could be found here.
- My understanding is:

Rmarkdown = markdown + yaml heading + code chunk options.

Markdown

code chunk options

YAML

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

Final exercise

- Calculate the NA of all columns except for id column.
- Bin the education variable with your own defined cut points.
- (optional) Pivot the original 'fake_snack_df' by using only one `pivot_longer()` function. Put wave and date after 'Lopnr' column.

Hint: check the `names_to` argument and `relocate()` function.

```
# A tibble: 3 x 6
  Lopnr wave  Date         sex education dementia
  <int> <chr> <date>     <dbl>     <dbl>    <dbl>
1  1284 wave1 1848-01-16 -1.62     -1.00   -0.246
2  1284 wave2 1703-06-26 -1.62     -1.00   -0.704
3  1284 wave3 1807-08-10 -1.62     -1.00   -0.508
```

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. Check documentation here. The R example code is pulled from Rmarkdown, I do not need to copy and paste.

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