Data Wrangling with R: Day 2

Advanced data wrangling starring `purrr`

Presented by Emi Tanaka

Department of Econometrics and Business Statistics

MONASH University

2nd December 2020 @ Statistical Society of Australia | Zoom

Repetitive tasks

```r
df <- forcats::gss_cat
```

- 🎯 Let's calculate the number of distinct values for each column and store it in `n`

```r
n <- vector("integer", length = ncol(df))
n[1] <- n_distinct(df[[1]])
n[2] <- n_distinct(df[[2]])
n[3] <- n_distinct(df[[3]])
n[4] <- n_distinct(df[[4]])
n[5] <- n_distinct(df[[5]])
n[6] <- n_distinct(df[[6]])
n[7] <- n_distinct(df[[6]])
n[8] <- n_distinct(df[[8]])
n[9] <- n_distinct(df[[9]])
```

- Anything you notice from above?

Using for loops

```
n <- vector("integer", length = ncol(df))
for(i in 1:ncol(df)) {
  n[i] <- n_distinct(df[[i]])
}
```

- But R is notoriously known for being slow using for loops
- The new `across` function in `dplyr` can do this without for loops

```
df %>%
  summarise(across(everything(), n_distinct))

## # A tibble: 1 x 9
##    year marital   age  race rincome partyid relig denom tvhours
##   <int>   <int> <int> <int>   <int>   <int> <int> <int>   <int>
## 1     8       6    73     3      16      10    15    30      25
```

- But the result is a data.frame

Functional programming purrr Part 1

- `purrr` is part of the core `tidyverse` packages
- It contains a series of `map` and `walk` functions

```
map_int(df, n_distinct)

##     year marital      age     race rincome partyid   relig   denom tvhours
##        8       6       73        3      16      10      15      30      25

map_chr(df, n_distinct)

##     year marital      age
##      "8"     "6"     "73"

map_df(df, n_distinct)

## # A tibble: 1 x 9
##     year marital   age
##    <int>   <int> <int>
## 1      8       6    73
```

- The related functions in `purrr` have been design so that that input are consistent
- The user is required to think of the expected output before seeing the output, e.g.
  - `map_int` returns a vector of integer
  - `map_df` returns a data frame

map functions in purrr

- `map(.x, .f, ...)` returns list
- `map_chr(.x, .f, ...)` returns a vector of character
- `map_dbl(.x, .f, ...)` returns a vector of numeric
- `map_int(.x, .f, ...)` returns a vector of integer
- `map_lgl(.x, .f, ...)` returns a vector of logical
- `map_raw(.x, .f, ...)` returns a vector of raw
- `map_df(.x, .f, ...)`, `map_dfr(.x, .f, ...)` returns a data frame, combining data.frame by row
- `map_dfc(.x, .f, ...)` returns a data frame, combining data.frame by column

Conditional maps in purrr

- `map_if(.x, .p, .f, ...)` uses `.p` to determine if `.f` will be applied to `.x`
- `map_at(.x, .at, .f, ...)` applies `.f` to `.x` at `.at` (name or position)
- `map_depth(.x, .depth, .f, ...)` apples `.f` to `.x` at a specific depth level of a nested vector
- **The return object is always a list**

```
map_if(df, is.factor, as.character) %>% as_tibble()

## # A tibble: 21,483 x 9
##      year marital      age race  rincome      partyid      relig        denom        tvhc
##     <int> <chr>      <int> <chr> <chr>        <chr>        <chr>        <chr>        <i
##  1   2000 Never ma…     26 White $8000 to … Ind,near r… Protesta… Souther…
##  2   2000 Divorced      48 White $8000 to … Not str re… Protesta… Baptist…
##  3   2000 Widowed       67 White Not appli… Independent Protesta… No deno…
##  4   2000 Never ma…     39 White Not appli… Ind,near r… Orthodox… Not app…
```

Functional programming Base R

- `lapply`, `Map`, `mapply`, `sapply`, `tapply`, `apply`, and `vapply` are variants of functional programming in Base R
- Some function outputs in Base R are more predictable than others:
  - `purrr::map` is a variant of `lapply` (which always returns list)
  - `purrr::pmap` is a variant of `Map` (which takes more than one input)
- `sapply` doesn't require users to specify the output type, instead it'll try to figure out what looks best for the user... great for interactive use but require great caution for programming
- So setting return type expectation is the reason to use `purrr`?

Anonymous functions Part 1

- Also called **lambda expression** in computer programming
- These are functions without names

```
map_int(df, function(x) length(unique(x)))

##     year marital     age    race rincome partyid   relig   denom tvhours 
##        8       6      73       3      16      10      15      30      25
```

- Tidyverse often employs a special shorthand using a formula and `.x` as a special placeholder for input

```
map_int(df, ~length(unique(.x)))

##     year marital     age    race rincome partyid   relig   denom tvhours 
##        8       6      73       3      16      10      15      30      25
```

Anonymous functions Part 2

- And yes, it's not just for `purrr` functions:

```
df %>%
  summarise(across(everything(), ~length(unique(.x))))

## # A tibble: 1 x 9
##      year marital   age  race rincome partyid relig denom tvhours
##     <int>   <int> <int> <int>   <int>   <int> <int> <int>   <int>
## 1       8       6    73     3      16      10    15    30      25
```

- This formula anonymous function is expanded to an actual anonymous function under the hood using `rlang::as_function()`
- Most `tidyverse` functions would support this formula approach to anonymous function, but likely not outside of that ecosystem unless developers adopt the same system

Functions with two inputs

- For functions with two inputs, you can use the `map2` variants in `purrr`

```
x <- c(1, 2, 3)
y <- c(0.1, 0.2, 0.3)
map2_dbl(x, y, function(.x, .y)  .x + .y)

## [1] 1.1 2.2 3.3
```

- For anonymous functions with two inputs, the first input is `.x` (as before) and the second is `.y`

```
map2_dbl(x, y, ~.x + .y)

## [1] 1.1 2.2 3.3
```

Functions with more than two inputs

- What about if there are more than two input?
- You can use `pmap` variants in `purrr`
- But no formula anonymous function supported:

```
x <- c(1, 2, 3)
y <- c(0.1, 0.2, 0.3)
z <- c(10, 20, 30)
pmap_dbl(list(x, y, z), function(.x, .y, .z)  .x + .y + .z)

## [1] 11.1 22.2 33.3
```

Other functions in `purrr`

Using names of input

- The `imap(x)` variants are shorthand for `map2(x, names(x))`

```
imap_chr(list(x = 1L, y = 4L), ~paste0(.y, ":", .x))

##     x     y
## "x:1" "y:4"
```

Expecting no return object

- If you are looking to get a side effect rather than return, you can use the `walk` variants

```
iwalk(df, ~write.csv(.x, file = paste0(.y, ".csv")))
```

A simulated study Part 1

- Let's simulate response of size 400 from a data generating process where it is a simple linear model with slope as -2 and intercept as 1 with error from $N(0, 1)$

```
set.seed(1)
dat <- tibble(id = 1:400) %>% # 400 observations
  mutate(x = runif(n(), 0, 10), # independent covariate
         y = 1 - 2 * x + rnorm(n())) # dependent response
```

- Let's then fit a linear model and a robust linear model to compare their estimates

```
fit_lm <- lm(y ~ x, data = dat)
fit_rlm <- MASS::rlm(y ~ x, data = dat)
```

```
coef(fit_lm)
```

```
## (Intercept)           x
##   0.9965522  -1.9994216
```

```
coef(fit_rlm)
```

```
## (Intercept)           x
##    1.012063   -1.999890
```

A simulated study Part 2

- Let's make it interesting but adding a few outliers

```r
set.seed(1)
dat$y[1:3] <- 100
```

- Let's refit the model

```r
fit_lm <- lm(y ~ x, data = dat)
fit_rlm <- MASS::rlm(y ~ x, data = dat)
```

```r
coef(fit_lm)
```

```
## (Intercept)           x
##    2.233095   -2.089164
```

```r
coef(fit_rlm)
```

```
## (Intercept)           x
##    1.019283   -2.000755
```

A simulated study Part 3

- For a proper simulation, you need to run it a multiple times
- Let's do the same simulation 200 times

```
set.seed(1)
map_dfr(1:200, ~{
  dat <- tibble(id = 1:400) %>% # 400 observations
    mutate(x = runif(n(), 0, 10), # independent covariate
           y = 1 - 2 * x + rnorm(n())) # dependent response
  dat$y[1:3] <- 100
  fit_lm <- lm(y ~ x, data = dat)
  fit_rlm <- MASS::rlm(y ~ x, data = dat)

  tibble(intercept_lm = coef(fit_lm)[1], intercept_rlm = coef(fit_rlm)[1],
         slope_lm = coef(fit_lm)[2], slope_rlm = coef(fit_rlm)[2], sim = .x)})
```

```
## # A tibble: 200 x 5
##    intercept_lm intercept_rlm slope_lm slope_rlm   sim
##           <dbl>         <dbl>    <dbl>     <dbl> <int>
##  1        2.23          1.02     -2.09     -2.00     1
```

A simulated study Part 4

- You can speed up your simulations by using parallel processing with `furrr` package

```
set.seed(1)
library(furrr)
plan(multisession, workers = 2)
future_map_dfr(1:200, ~{
  dat <- tibble(id = 1:400) %>% # 400 observations
    mutate(x = runif(n(), 0, 10), # independent covariate
           y = 1 - 2 * x + rnorm(n())) # dependent response
  dat$y[1:3] <- 100
  fit_lm <- lm(y ~ x, data = dat)
  fit_rlm <- MASS::rlm(y ~ x, data = dat)

  tibble(intercept_lm = coef(fit_lm)[1], intercept_rlm = coef(fit_rlm)[1],
         slope_lm = coef(fit_lm)[2], slope_rlm = coef(fit_rlm)[2], sim = .x)})

## # A tibble: 200 x 5
##    intercept_lm intercept_rlm slope_lm slope_rlm   sim
```

- There are many functions in `tidyverse`, the key is *not* to remember them all but remember key points to have a mental trigger where to look
- There are multiple of ways to data wrangle to get to the same result
- What's more important that code is readable and done in a way to satisfy your objective to you *and* others that you share your work with, e.g. in production, backward compatibility in Base R may be more important

Top downloaded CRAN packages

rank	package	count	from	to
1	jsonlite	2322852	2020-11-01	2020-11-30
2	rlang	1813779	2020-11-01	2020-11-30
3	ggplot2	1801691	2020-11-01	2020-11-30
4	vctrs	1651019	2020-11-01	2020-11-30
5	fs	1618762	2020-11-01	2020-11-30
6	devtools	1590942	2020-11-01	2020-11-30
7	usethis	1520368	2020-11-01	2020-11-30
8	dplyr	1394366	2020-11-01	2020-11-30
9	tibble	1250293	2020-11-01	2020-11-30
10	glue	1190160	2020-11-01	2020-11-30

scroll for more

- How many of the top downloaded package in the past month is `tidyverse` related?
- It's hard to ignore what is widely used
- Also a duty to be aware with the latest if you work with data?

Software language is an evolving language

Tidyverse evolution has at least now slowed down compared to before

Enjoy your evolving data wrangling journey!

</> If you installed the `dwexercise` package, run below in your R console

```
learnr::run_tutorial("day2-exercise-04", package = "dwexercise")
```

🔗 If the above doesn't work for you, go here.

❓ Questions or issues, let us know!

15:00

Session Information

```
devtools::session_info()

## ─ Session info ───────────────────────────────────────────────────────────────
##  setting  value
##  version  R version 4.0.1 (2020-06-06)
##  os       macOS Catalina 10.15.7
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_AU.UTF-8
##  ctype    en_AU.UTF-8
##  tz       Australia/Melbourne
##  date     2020-12-02
##
## ─ Packages ───────────────────────────────────────────────────────────────────
##  package     * version    date       lib source
##  anicon        0.1.0      2020-06-21 [1] Github (emitanaka/anicon@0b756df)
##  assertthat    0.2.1      2019-03-21 [2] CRAN (R 4.0.0)
```