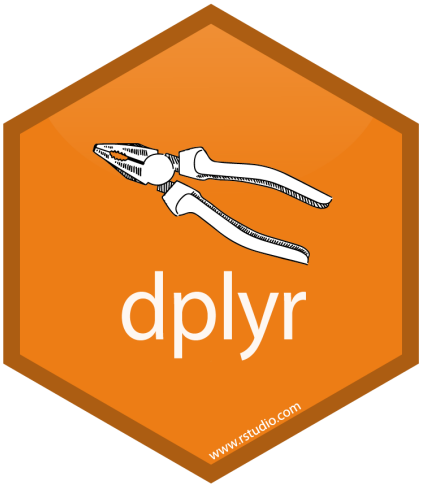
Data wrangling with dplyr

TheRBootcamp



Source: https://www.rstudio.com/ (https://www.rstudio.com/)

Slides

- Here are the introduction slides for this practical on data wrangling! (https://therbootcamp.github.io/_sessions/D2S1_Wrangling/Wrangling.html)

Overview

In this practical you'll practice "data wrangling" with the `dplyr` package. Data wrangling refers to modifying and summarizing data. For example, sorting, adding columns, recoding values,

If you don't have it already, you can access the dplyr cheatsheet here https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf). This has a nice overview of all the major functions in dplyr.

Glossary

Here are the main verbs you will be using in `dplyr` :

verb	action	example
`filter()`	Select rows based on some criteria	data %>% filter(age > 40 & sex == "m")
`arrange()`	Sort rows	data %>% arrange(date, group)
`select()`	Select columns (and ignore all others)	data %>% select(age, sex)
`rename()`	Rename columns	data %>% rename(DATE_MONTHS_X24, date)
`mutate()`	Add new columns	data %>% mutate(height.m = height.cm / 100)

verb	action	example
`case_when()`	Recode values of a column	data %>% sex.n = case_when(sex == 0 ~ “m”, sex == 1 ~ “f”)
`group_by(), summarise()`	Group data and then calculate summary statistics	data %>% group_by(…) %>% summarise(…)

Examples

- The following examples will take you through the steps of doing data wrangling with dplyr. Try to go through each line of code and see how it works!

```r
# ------------------------------------------------
# Examples of using dplyr on the ChickWeight data
# ------------------------------------------------

library(tidyverse)         # Load tidyverse

chick <- as_tibble(ChickWeight)   # Save a copy of the ChickWeight data as chick

# Add columns with mutate()

chick <- chick %>%
  mutate(
    weight.kg = weight / 1000,  # Add new column of weight in kilograms
    time.week = Time / 7        # Add time.week as time in weeks
  )

# Sort rows with arrange()

chick <- chick %>%
  arrange(Diet, weight)   # sort rows by Diet and then weight


# Recode variables with case_when()

chick <- chick %>%
  mutate(
    diet.name = case_when(
      Diet == 1 ~ "vegetables",
      Diet == 2 ~ "fruit",
      Diet == 3 ~ "candy",
      Diet == 4 ~ "meat"

    )
  )

# Grouped statistics with group_by() and summarise()

chick %>%
  group_by(Diet) %>%
  summarise(
    weight.mean = mean(weight),
    weight.max = max(weight),
    time.mean = mean(Time),
    N = n()
  ) %>%
  arrange(weight.mean)


# Many sequential functions

chick %>%
  filter(Chick > 10) %>%              # Only chicks with values larger than 10
  group_by(Time, Diet) %>%            # Group by Time and Diet
  summarise(
    weight.median = median(weight),
    weight.sd = sd(weight),
    N = n()                           # Counts of cases
  )
```

Tasks

Getting the data and project setup

1. Open an R project (or create a new one). In the main directory where that project is located, create two new empty folders: one called `data` and one called `R`. (Hint: You can create, and them rename, new empty folders in the `Files` pane of RStudio.)
2. Open a new R script and save it in the `R` folder you just created as a new file called `datawrangling.R`. At the top of the script, using comments, write your name and the date. The, load the set of `tidyverse` packages with `library()`

```
library(tidyverse)
```

3. For this practical, we'll use the `ACTG175` data, this is the result of a randomized clinical trial comparing the effects of different medications on adults infected with the human immunodeficiency virus. The data are stored at "https://raw.githubusercontent.com/therbootcamp/BaselRBootcamp2017/master/tutorials/data/ACTG175.csv (https://raw.githubusercontent.com/therbootcamp/BaselRBootcamp2017/master/tutorials/data/ACTG175.csv)". Using `read_csv()` load the data into R and store it as a new object called `ACTG175`. (Hint: This is really easy! Just run `read_csv(file = LINK)`, where `LINK` is the file URL.)

```
ACTG175 <- read_csv(file = "https://raw.githubusercontent.com/therbootcamp/BaselRBootcamp
2017/master/tutorials/data/ACTG175.csv")
```

```
library(tidyverse)
```

```
library(speff2trial)
write_csv(ACTG175, path = "tutorials/data/ACTG175.csv")
```

4. Ok you've got the data from the internet into your R session. But let's save a local copy of the data as a `.csv` file so we'll always have access to it. Using `write_csv()`, save a copy of the data as a `.csv` file called `ACTG175.csv`, also make sure to put it in the data folder you just created (Hint: the main arguments to `write_csv()` are `x`, the object you are writing, and `path`, the directory and file name you're saving the object to.)

```
write_csv(ACTG175, path = "data/ACTG175.csv")
```

5. Now that you have a local copy of the data saved, we try loading the data back into R from that file. But first, delete the object `ACTG175` from your workspace by running `rm(ACTG175)`.

```
rm(ACTG175)
```

6. Now it's time to get the data back from the local file! To do this, use `read_csv()` to load the `ACTG175.csv` file data back into your R session and again store it as the object `ACTG175`. Now, your code should be able to access your local copy of the data, even when you are not online.

```
ACTG175 <- read_csv("data/ACTG175.csv")
```

7. The `ACTG175` data is comes from the `speff2trial` package. If you don't have the `speff2trial` package allready, install it using `install.packages()`. Then, load the package with `library()` and then look at the help menu for the `ACTG175` data by running `?ACTG175`. (Note: If you become really interested in the data, you can also read an article discussing the trial here:

http://www.nejm.org/doi/full/10.1056/nejm199610103351501#t=article (http://www.nejm.org/doi/full/10.1056/nejm199610103351501#t=article))

```
library(speff2trial)
?ACTG175
```

8. First thing's first, take a look at the first few rows of the data by printing the `ACTG175` object. It should look like this:

```
# A tibble: 2,139 x 27
   pidnum   age    wtkg  hemo  homo drugs karnof oprior   z30 zprior
    <int> <int>   <dbl> <int> <int> <int>  <int>  <int> <int>  <int>
 1  10056    48 89.8128     0     0     0    100      0     0      1
 2  10059    61 49.4424     0     0     0     90      0     1      1
 3  10089    45 88.4520     0     1     1     90      0     1      1
 4  10093    47 85.2768     0     1     0    100      0     1      1
 5  10124    43 66.6792     0     1     0    100      0     1      1
 6  10140    46 88.9056     0     1     1    100      0     1      1
 7  10165    31 73.0296     0     1     0    100      0     1      1
 8  10190    41 66.2256     0     1     1    100      0     1      1
 9  10198    40 82.5552     0     1     0     90      0     1      1
10  10229    35 78.0192     0     1     0    100      0     1      1
# ... with 2,129 more rows, and 17 more variables: preanti <int>,
#   race <int>, gender <int>, str2 <int>, strat <int>, symptom <int>,
#   treat <int>, offtrt <int>, cd40 <int>, cd420 <int>, cd496 <int>,
#   r <int>, cd80 <int>, cd820 <int>, cens <int>, days <int>, arms <int>
```

mutate()

9. Add a new column to the dataframe called `age_months` that shows each patient's age in months instead of years (Hint: do simple arithmetic!).

```
ACTG175 <- ACTG175 %>% mutate(
  age_months = age * 12
)
```

10. The column `karnof` shows each patient's Karnofsky score (http://emedicine.medscape.com/article/2172510-overview) on a scale from 0 to 100, create a new variable called `karnof_b` that shows the score on a scale from 0 to 1 (hint: just divide the original column by 100)

```
ACTG175 <- ACTG175 %>%
  mutate(
         karnof_b = karnof / 100
        )
```

11. Now, do the previous two questions in one chunk of code. That is, using one call to `mutate()` create both `age_months` and `karnof_b`

```
ACTG175 <- ACTG175 %>%
 mutate(
    age_months = age * 12,
    karnof_b = karnof / 100
    )
```

12. A physician wants to see a new score called the `sparrow` which is equal to the Karnofsky score divided by a person's age plus the person's weight in kg. Add each participant's sparrow score as a new column called `sparrow` (Hint: Just take `karnof / age + wtkg)

```
ACTG175 <- ACTG175 %>%
 mutate(
    sparrow = karnof / age + wtkg
    )
```

arrange()

13. Arrange the `ACTG175` data in ascending order of age (from lowest to highest). After, look at the first few rows of the data with `head()` to make sure it worked.

```
ACTG175 <- ACTG175 %>%
 arrange(age)
```

14. Now arrange the data in *descending* order of age (from highest to lowest). After, look at the first few rows of the data with `head()` to make sure it worked.

```
ACTG175 <- ACTG175 %>%
 arrange(desc(age))
```

- To arrange data in descending order, just include `desc()` around the variable. E.g.; `data %>% arrrange(desc(height))`

15. You can sort the rows of dataframes with multiple columns by including many arguments to `arrange()`. Now sort the data by `karnof` and then age (`age`), and then arms (`arms`)

```
ACTG175 <- ACTG175 %>%
 arrange(karnof, age, arms)
```

case_when()

16. Create a new column `gender_char` that shows gender as a string.

- Look at the help file with `?ACTG175` to see how gender is coded.

```
ACTG175 <- ACTG175 %>%
  mutate(
  gender.char = case_when(
    gender == 0 ~ "female",
    gender == 1 ~ "male"
  )
  )
```

17. Create a new column `over50` that is 1 when patients are older than 50, and 0 when they are younger than or equal to 50

```
ACTG175 <- ACTG175 %>%
  mutate(
  over50 = case_when(
    age > 50 ~ 1,
    age <= 50 ~ 0
  )
  )
```

18. Now, repeat the previous two questions, but do them both in the same call to `mutate()`. That is, in one block of code, create `gender_char` and `over50`

```
ACTG175 <- ACTG175 %>%
  mutate(
    gender.char = case_when(
    gender == 0 ~ "female",
    gender == 1 ~ "male"),
  over50 = case_when(
    age > 50 ~ 1,
    age <= 50 ~ 0)
  )
```

`group_by()` and `summarise()`

19. For each arm, calculate the mean participant age.

```
ACTG175 %>%
  group_by(arms) %>%
  summarise(
    age.mean = mean(age)
  )
```

20. For each arm, calculate the mean participant age *and* the median Karnofsky score

```
ACTG175 %>%
  group_by(arms) %>%
  summarise(
    age.mean = mean(age),
    karnof.median = median(karnof)
  )
```

21. Separately for male and female patients, calculate the percent who have a history of intravenous drug use.

- To calculate a percent of a binary variable with 0s and 1s, just calculate the mean

```
ACTG175 %>%
  group_by(gender) %>%
  summarise(
    drug.percent = mean(drugs)
  )
```

22. Separately for male and female patients (`gender`), calculate the percent who have a history of intravenous drug use (`drugs`), and the percent of patients who indicate homosexual activity

```
ACTG175 %>%
  group_by(gender) %>%
  summarise(
    drug.percent = mean(drugs),
    homo.percent = mean(homo)
  )
```

23. Separately for all combinations of gender and race, calculate the mean age and mean CD4 T cell count at baseline (`cd40`) (Hint: group by `gender` and `race`)

```
ACTG175 %>%
  group_by(gender, race) %>%
  summarise(
    age.mean = mean(age),
    cd40.mean = mean(cd40)
  )
```

Challenges

24. Now let's check the major differences between the treatment arms. For each arm, calculate the following:

- Mean days until a a major negative event (`days`)
- Mean CD4 T cell count at baseline.
- Mean CD4 T cell count at 20 weeks.
- Mean CD4 T cell count at 96 weeks.
- Mean *change* in CD4 T cell count between baseline and 96 weeks
- Number of patients (Hint: use `N = n()`)

```
ACTG175 %>%
  group_by(arms) %>%
  summarise(
    days_mean = mean(days),
    cd4_bl = mean(cd40),
    cd4_20 = mean(cd420),
    cd4_96 = mean(cd496, na.rm = TRUE),
    cd4_change = mean(cd496 - cd40, na.rm = TRUE),
    N = n()
  )
```

25. Repeat the previous analysis, but before you do the grouping and summary statistics, recode the values of `arms` to text that reflect what the values actually represent. For example, looking at the help file `?ACTG175` , I can see that the treatment arm of - is "zidovudine". I might call this arm `"Z"` . Do this in the all in the same chunk of code.

```
ACTG175 %>%
  mutate(
    arms = case_when(
      arms == 0 ~ "Z",
      arms == 1 ~ "ZD",
      arms == 2 ~ "ZZ",
      arms == 3 ~ "D"
    )
  ) %>%
  group_by(arms) %>%
  summarise(
    days_mean = mean(days),
    cd4_bl = mean(cd40),
    cd4_20 = mean(cd420),
    cd4_96 = mean(cd496, na.rm = TRUE),
    cd4_change = mean(cd496 - cd40, na.rm = TRUE)
  )
```

26. Repeat the previous analysis, but only include patients with a Karnofsky score equal to 100, and who did not use zidovudine in the 30 days prior to the treatment initiation (`z30`)

```
ACTG175 %>%
  filter(karnof == 100 & z30 == 0) %>%
  mutate(
    arms = case_when(
      arms == 0 ~ "Z",
      arms == 1 ~ "ZD",
      arms == 2 ~ "ZZ",
      arms == 3 ~ "D"
    )
  ) %>%
  group_by(arms) %>%
  summarise(
    days_mean = mean(days),
    cd4_bl = mean(cd40),
    cd4_20 = mean(cd420),
    cd4_96 = mean(cd496, na.rm = TRUE),
    cd4_change = mean(cd496 - cd40, na.rm = TRUE)
  )
```

Additional reading

- For more details on data wrangling with R, check out the chapter in YaRrr! The Pirate's Guide to R YaRrr! Chapter Link (https://bookdown.org/ndphillips/YaRrr/advanceddataframe.html)
- Hadley Wickham, the author of dplyr, also has great examples in the dplyr vignette here: dplyr vignette link (https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html)
- The `tidyr` package is a natural extension to `dplyr` that allows you to reshape your data into a format that is easier to manage. Check out the tidyr vignette (https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html) for examples: