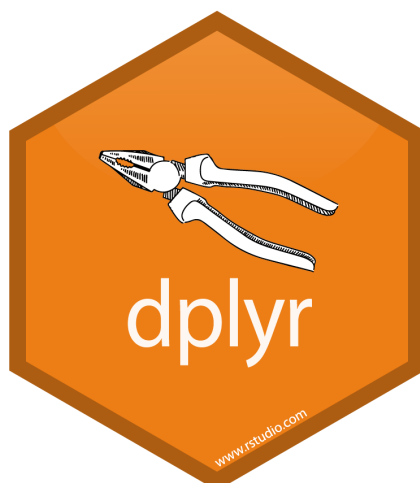


# Data wrangling with dplyr

*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](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
<code>filter()</code>	Select rows based on some criteria	<code>data %&gt;% filter(age &gt; 40 &amp; sex == "m")</code>
<code>arrange()</code>	Sort rows	<code>data %&gt;% arrange(date, group)</code>
<code>select()</code>	Select columns (and ignore all others)	<code>data %&gt;% select(age, sex)</code>
<code>rename()</code>	Rename columns	<code>data %&gt;% rename(DATE_MONTHS_X24, date)</code>
<code>mutate()</code>	Add new columns	<code>data %&gt;% mutate(height.m = height.cm / 100)</code>

verb	action	example
<code>case_when()</code>	Recode values of a column	<code>data %&gt;% sex.n = case_when(sex == 0 ~ "m", sex == 1 ~ "f")</code>
<code>group_by()</code> , <code>summarise()</code>	Group data and then calculate summary statistics	<code>data %&gt;% group_by(...) %&gt;% summarise(...)</code>

## 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!

```

# -----
# 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 then 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. Then, load the set of `tidyverse` packages with `library()`
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.)
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.)
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)`.
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.
7. The `ACTG175` data is comes from the `speff2trial` package. If you don't have the `speff2trial` package already, 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>))
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!).
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)
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`
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`)

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

## 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.
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
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`

## group\_by() and summarise()

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

## 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()` )
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.
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` )

## 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: