

ICO Workshop R & RStudio

Part 3

Data manipulation with `dplyr`

Sven De Maeyer & Tine van Daal

23th - 25th November, 2022

1 / 28

Overview

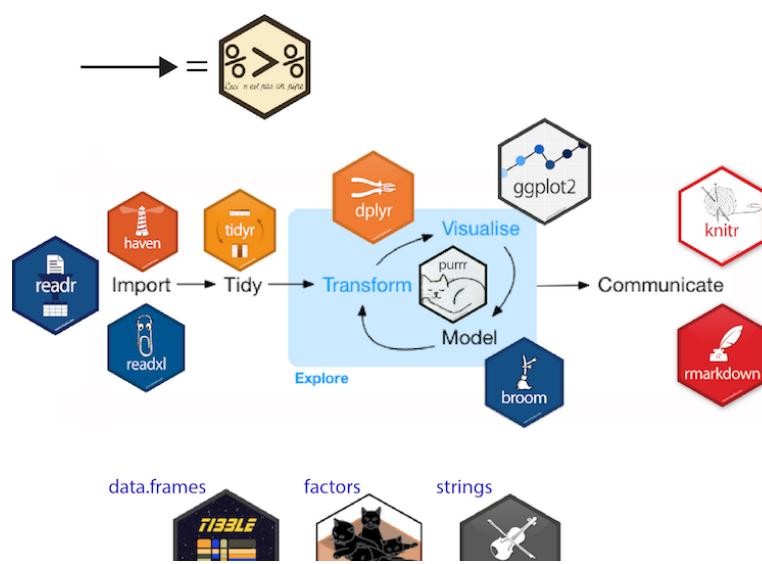
- Tidyverse --- ([Click here](#))
- The `dplyr` package --- ([Click here](#))

2 / 28

1. Tidyverse

3 / 28

Welcome in the tidyverse



4 / 28

Why tidyverse?

- more accessible for beginners
- consistent approach for all potential tasks
- powerful potential applications with minimum 'effort'
- can give confidence to explore R

5 / 28

Tibble

Normally we work with a **dataframe** in R but we can have very complex data-structures as well (e.g., lists, matrices, ...)

In the tidyverse ecosystem we work with a simple form of data-structure: a **tibble**

A tibble is a dataframe that fits the **tidy data** principle

Friends

```
## # A tibble: 108 × 4
##   student occasion condition fluency
##   <dbl>     <dbl>     <dbl>    <dbl>
## 1 1         1         1      101.
## 2 2         1         2      104.
## 3 1         3         1      117.
## 4 2         1         2      98.8
## 5 2         2         2      107.
## 6 2         3         2      111.
## 7 3         1         3      105.
## 8 3         2         3      102.
## 9 3         3         3      101.
## 10 4        1         1      102.
## # ... with 98 more rows
```

6 / 28

What is tidy data?

“**TIDY DATA** is a standard way of mapping the meaning of a dataset to its structure.”

—HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

each column a variable		
id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

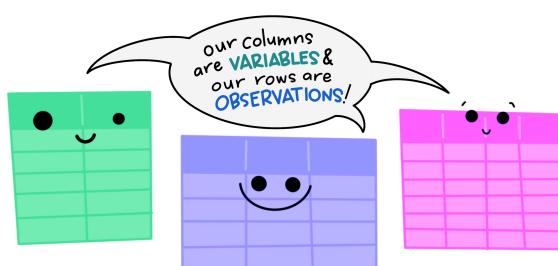
each row an observation

Wickham, H. (2014). Tidy Data. Journal of Statistical Software 59 (10). DOI: 10.18637/jss.v059.i10

Artwork by @allison_horst 7 / 28

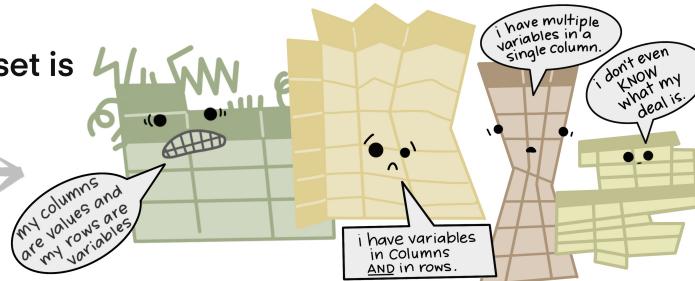
What is tidy data?

The standard structure of tidy data means that “tidy datasets are all alike...”



“...but every messy dataset is messy in its own way.”

—HADLEY WICKHAM

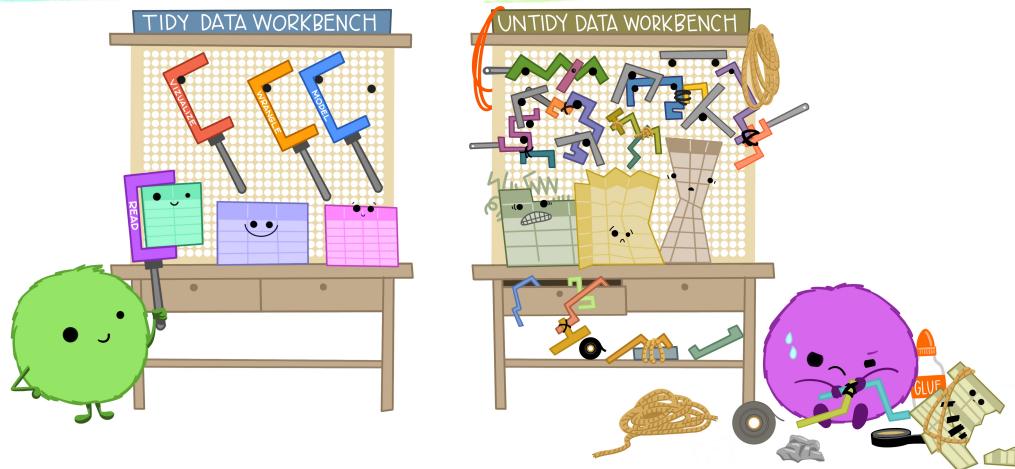


Artwork by @allison_horst 8 / 28

What is tidy data?

When working with tidy data,
we can use the same tools in
similar ways for different datasets...

...but working with untidy data often means
reinventing the wheel with one-time
approaches that are hard to iterate or reuse.



Artwork by @allison_horst 9 / 28

2. The `dplyr` package

dplyr ...

is THE package to work with tidy data !

VERBS are at the core:

- filter()
- mutate()
- select()
- group_by() + summarise()
- arrange()
- rename()
- relocate()
- join()

11 / 28

Data transformation with dplyr :: CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each variable is in its own column & Each observation, or case, is in its own row 

Summarise Cases

Apply `summary` functions to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

→ `summarise(data, ...)`
Compute multiple summaries.
`summarise(mtcars, avg = mean(mpg))`

→ `count(data, ..., wt = NULL, sort = FALSE, name = "N")`
NULL means count of rows in each group defined by the variables in ... Also `tally`!
`count(mtcars, cyl)`

Group Cases

Use `group_by`, `add` = FALSE, `drop` = TRUE to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

→ `mtcars %>% group_by(cyl) %>% summarise(avg = mean(mpg))`

Use `rowwise`(`data, ...`) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidy cheat sheet for list-column workflow.

→ `starwars %>% rowwise() %>% mutate(film_count = length(films))`

`ungroup(x, ...)` Returns ungrouped copy of table.
`ungroup(g_mtcars)`

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

→ `filter(data, ..., preserve = FALSE)` Extract rows that meet logical criteria.
`filter(mtcars, mpg > 20)`

→ `distinct(data, ..., keep_all = FALSE)` Remove rows with duplicate values.
`distinct(mtcars, gear)`

→ `slice(data, ..., preserve = FALSE)` Select rows by position.
`slice(mtcars, 10:15)`

→ `slice_n(data, ..., n, prop, weight_by = "wt", replace = FALSE)` Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
`slice_n(mtcars, 10, prop = 0.25)`

→ `slice_head(data, ..., n, prop, with_ties = TRUE)` and `slice_tail` Select rows with the lowest and highest values.
`slice_min(mtcars, mpg, prop = 0.25)`

→ `slice_head(data, ..., n, prop) and slice_tail()` Select the first or last rows.
`slice_head(mtcars, n = 5)`

Logical and boolean operators to use with `filter()`

`= < <= > >= | & | & | xor|`
See `?base::Logic` and `?Comparison` for help.

ARRANGE CASES

→ `arrange(data, ..., by = group = FALSE)` Order rows by values of a column or columns (low to high), use with `desc`() to order from high to low.
`arrange(mtcars, mpg)`

ADD CASES

→ `add_row(data, ..., before = NULL, after = NULL)` Add one or more rows to a table.
`add_row(cars, speed = 1, dist = 1)`

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

→ `pull(data, var = 1, name = NULL, ...)` Extract column(s) as a vector, by name or index.
`pull(mtcars, wt)`

→ `select(data, ...)` Extract columns as a table.
`select(mtcars, mpg, wt)`

→ `relocate(data, ..., before = NULL, after = NULL)` Move columns to new position.
`relocate(mtcars, mpg, cyl, after = last_col())`

Use these helpers with `select()` and `across()`

e.g. `select(mtcars, mpg:cyl)`
`contains(match)` `num_range(prefix, range)` ; e.g. `mpg:cyl`
`ends_with(match)` `all_of(contains, vars)` ; e.g. `gear`
`starts_with(match)` `matche(match)` `everything()`

MANIPULATE MULTIPLE VARIABLES AT ONCE

→ `across(cols, func, ..., names = NULL)` Summarise or mutate multiple columns in the same way.
`summarise(mtcars, across(cyl, everything), mean)`

→ `c_across(cols)` Compute across columns in row-wise data.
`transmute(rowwise(iris), total = sum(c_across(1:2)))`

MAKE NEW VARIABLES

Apply `vectorized` functions to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

→ `vectorized function`

→ `mutate(data, ..., keep = "all", before = NULL, after = NULL)` Compute new column(s). Also `add_column`, `add_count`, and `add_tally`.
`mutate(mtcars, gmm = 1 / mpg)`

→ `transmute(data, ...)` Compute new column(s), drop others.
`transmute(mtcars, gmm = 1 / mpg)`

→ `rename(data, ...)` Rename columns. Use `rename` with `b` to rename with a function.
`rename(cars, distance = dist)`



R Studio

RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-3222 • rstudio.com • Learn more at dplyr.tidyverse.org • dplyr 1.0.7 • Updated: 2021-07

<https://raw.githubusercontent.com/rstudio/cheatsheets/master/data-transformation.pdf>

12 / 28

The `%>%` operator (a 'pipe')



To create
a chain of functions

Instead of

```
mean(c(1,2,3,4))
```

or

```
Numbers <- c(1,2,3,4)  
mean(Numbers)
```

you can do

```
c(1,2,3,4) %>%  
  mean()
```

With the `%>%` you can write a sentence like:

I `%>%` woke up `%>%`, took a shower `%>%`, got breakfast `%>%`, took the train `%>%` and arrived at the ICO course `%>%` ...

filter()

dplyr::filter()

KEEP ROWS THAT
satisfy
your CONDITIONS

keep rows from... this data... ONLY IF... type is "otter" AND site is "bay"
`filter(df, type == "otter" & site == "bay")`

	type	food	site
otter	urchin	bay	
Shark	seal	channel	X
otter	abalone	bay	V
otter	crab	wharf	X

Artwork by @allison_horst

Let's apply filter()

With the FRIENDS data:

| We only select observations from the first measurement occasion in condition 1

```
Friends_Occ1 <- Friends %>%  
  filter(occasion == 1 & condition == 1)
```

== is equals (notice the 2 = signs!)

| Let's clean some data, and only keep observations with fluency values lower than 300 and that do not equal 0

```
Friends_clean <- Friends %>%  
  filter(fluency < 300 & fluency != 0)
```

!= means not equal to

15 / 28

mutate()



Artwork by @allison_horst 16 / 28

Let's apply `mutate()`

With the Friends data:

| *We calculate a new variable containing the fluency scores minus the average of fluency*

```
Friends <- Friends %>%
  mutate(
    fluency_centered = fluency - mean(fluency, na.rm = T)
  )
```

17 / 28

Let's apply `mutate()`

With the Friends data:

| *We create a factor for condition*

```
Friends <- Friends %>%
  mutate(
    condition_factor = as.factor(condition)
  )
str(Friends$condition_factor)

## Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 3 3 3 1 ...
```

18 / 28

Let's apply `select()`

To select variables.

Some examples with the Friends data:

We only select condition and occasion and inspect the result with the `str()`function

```
Friends %>%
  select(
    condition, occasion
  ) %>%
  str()

## # tibble [108 x 2] (S3:tbl_df/tbl/data.frame)
## $ condition: num [1:108] 1 1 1 2 2 2 3 3 3 1 ...
##   ..- attr(*, "value.labels")= Named chr [1:3] "3" "2" "1"
##   ... ..- attr(*, "names")= chr [1:3] "No subtitles" "Spanish" "English"
## $ occasion: num [1:108] 1 2 3 1 2 3 1 2 3 1 ...
##   ..- attr(*, "variable.labels")= Named chr(0)
##   ... ..- attr(*, "names")= chr(0)
##   - attr(*, "codepage")= int 1252
```

19 / 28

Rename variables with `rename()`

Notice how the variable `occassion` is misspelled! Pretty enoying when coding... But we can easily rename variables.

Function `rename(new_name = old_name)`

Rename the variable `occassion` to `occasion`

```
Friends <- Friends %>%
  rename(
    occasion = occassion
  )
```

20 / 28

Super combo 1: `group_by()` + `summarize()`

- transform a tibble to a *grouped tibble* making use of `group_by()`
- calculate summary stats per group making use of `summarize()`

Calculate the average fluency and standard deviation per condition

```
Friends %>%
  group_by(
    condition
  ) %>%
  summarize(
    mean_fluency = mean(fluency),
    sd_fluency   = sd(fluency)
  )

## # A tibble: 3 × 3
##   condition mean_fluency sd_fluency
##       <dbl>        <dbl>      <dbl>
## 1         1        109.     9.08
## 2         2        108.     6.02
## 3         3        103.     4.17
```

21 / 28

Super combo 1: `group_by()` + `summarize()`

Calculate the number of observations for each combination of condition and occasion

```
Friends %>%
  group_by(
    occasion, condition
  ) %>%
  summarize(
    n_observations = n()
  )

## # A tibble: 9 × 3
## Groups:   occasion [3]
##   occasion condition n_observations
##       <dbl>        <dbl>          <int>
## 1         1            1            12
## 2         1            2            12
## 3         1            3            12
## 4         2            1            12
## 5         2            2            12
## 6         2            3            12
## 7         3            1            12
## 8         3            2            12
## 9         3            3            12
```

22 / 28

Super combo 2: `mutate()` + `case_when()`



Artwork by @allison_horst

23 / 28

Super combo 2: `mutate()` + `case_when()`

To recode variables into new variables!

We create a new categorical variant of fluency with 3 groups, then we select this new variable and have a look to the top 5 observations...

```
Friends %>%
  mutate(
    fluency_grouped = case_when(
      fluency < 106.625 - 7.1 ~ 'low',
      fluency >= 106.625 - 7.1 & fluency < 106.625 + 7.1 ~ 'average',
      fluency >= 106.625 + 7.1 ~ 'high'
    )
  ) %>%
  select(
    fluency,
    fluency_grouped
  ) %>%
  head(5)
```

```
## # A tibble: 5 × 2
##   fluency fluency_grouped
##       <dbl> <chr>
## 1     101. average
## 2     104. average
## 3     117. high
## 4     98.8 low
## 5     107. average
```

24 / 28

How to define conditions

- `x == y` → 'x is equal to y'
- `x != y` → 'x is NOT equal to y'
- `x < y` → 'x is smaller than y'
- `x <= y` → 'x is smaller or equal to y'
- `x > y` → 'x is higher than y'
- `x >= y` → 'x is higher or equal to y'

25 / 28

Boolean operators

We can combine conditions!

- `&` → 'and' → example: `gender == 1 & age <=18`
- `|` → 'or' → example: `gender == 1 | gender == 2`
- `!` → 'not' → example: `gender == 1 & !age <=18`

26 / 28

Interactive tutorial about `dplyr()`

If you want some more material and a place to exercise your skills? This online and free tutorial (made with the package `learnr`) is strongly advised!

The screenshot shows a web-based interactive tutorial. On the left, a sidebar lists sections: 'Wrangling penguins: some basic data wrangling in R with dplyr', 'ALLISON HORST', and 'Resources'. Under 'Resources', there is a link to 'Start Over' with a small icon. The main content area has a title '1. Welcome' and a sub-section 'WHAT IS THE TIDYVERSE?'. It includes text about the tidyverse and a note that `dplyr` is part of it. A central image is a hexagonal logo for `dplyr` featuring a colorful rocket ship against a dark background with stars. Below the logo is a brief description of what `dplyr` does.

[27 / 28](https://allisonhorst.shinyapps.io/dplyr-learnr/#section>Welcome</p></div><div data-bbox=)

Exercise `dplyr`



- You can find the qmd-file `Exercises_dplyr.qmd` on the Open Science Framework (Exercises > Exercise2_dplyr)
- Open this document in an R project for the Exercises folder
- You get a set of tasks with empty code blocks to start coding
- Write and test the necessary code
- Stuck? No Worries!
 - We are there
 - Help each other
 - There is a solution key (`Exercises_dplyr_solutions.qmd`)

28 / 28