Using tidyverse

Recap

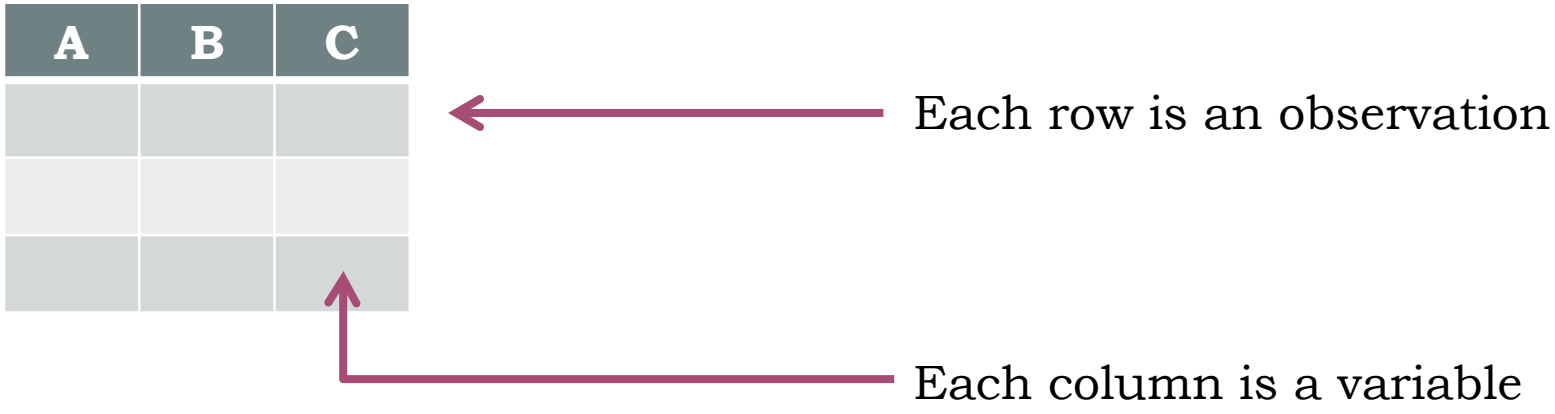
- Last lesson we learned about the grammar of graphics, and how to build a plot
  - geom_s, aesthetic mappings, scale_s, facets, and more
- We solved some visualization challenges
- We also got to play with data munging (transforming the scraped google play data in various ways)
- We talked about the five basic **dplyr** verbs (mutate, filter, select, summarize, arrange)
- We talked about some additional functions like: count, add_count, group_by, tribble

What we're going to do today?

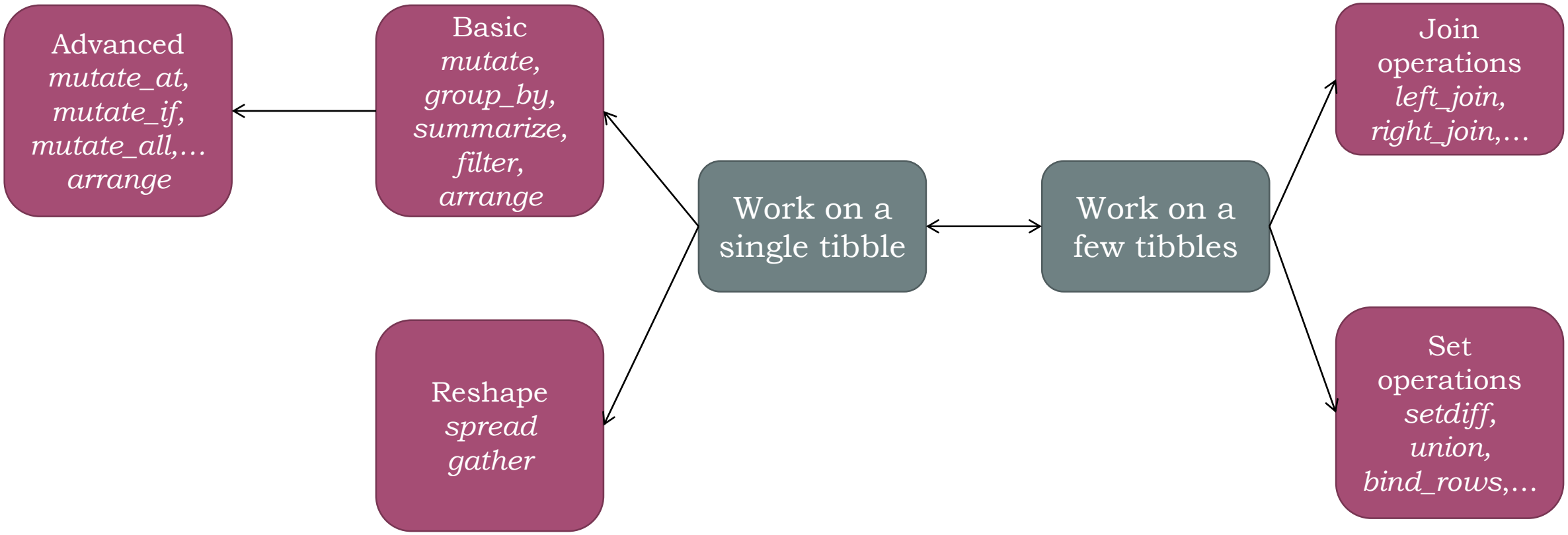
- Check homework from previous lesson
- Exercise on parametrized RMarkdown reports + Two small exercises about visualization "do and don't do"
  - 02-Plotting.Rmd, exercise 1.5 + 2
- A short note about interactivity of charts (plotly, ggvis)
- We're going to expand our knowledge about tidyverse in the following packages and functions:
  - dplyr – reshaping data *spread/gather* + using *mutate_at*, *mutate_if*, *mutate_all* (and equivalently *summarize_**, *rename_**, etc.)
  - rlang – building "tidy like" functions with non-standard evaluation (NSE)
  - purrr – iterating using the *map* and *walk* family
  - forcats, stringr, lubridate – some more examples

What is tidyverse?
What is "tidy"?

- *"An opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures"*
  – https://www.tidyverse.org

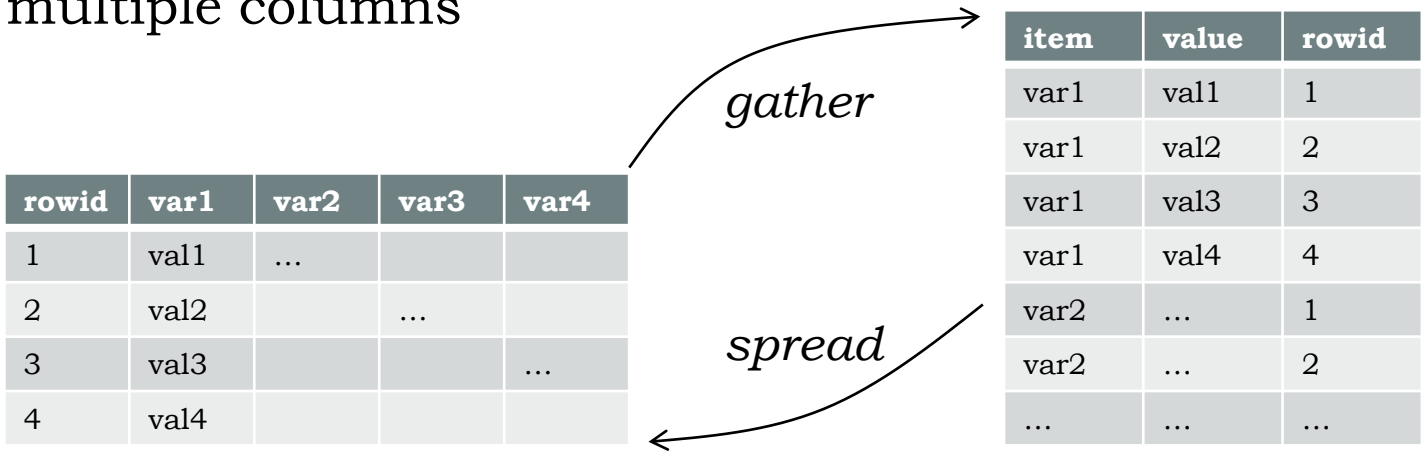


Conceptual map of dplyr



Reshaping data (*gather* and *spread*)

- *gather* takes a wide table (lots of columns) and **gathers** it to "2-columns".
  - (Not necessarily 2)
- *spread* takes a long table (few columns, lots of rows) and **spreads** it to multiple columns



rowid	var1	var2	var3	var4
1	val1	…		
2	val2		…	
3	val3			…
4	val4			

item	value	rowid
var1	val1	1
var1	val2	2
var1	val3	3
var1	val4	4
var2	…	1
var2	…	2
…	…	…

* The two functions are bound to be improved in the near future (with *pivot_wide*/*pivot_long* which are still in the development version), but for now spread and gather are still "the gold standard"

Reshaping data – quick quiz

- In pairs: find two examples for datasets you used/worked on in the past in which you had to transform one form into the other.
- If both of you did not used/worked on such a dataset, pair up with the pair next to you
- How did you solve the problem (i.e., did you use *dplyr::gather/spread*? *reshape2*? something else?

3 minutes

Reshaping data – mini exercise

- In pairs: what's wrong with the following code?
  - The code is also available in the exercise folder under: "03-Example for spread gather.R"
- Read the documentation of *gather* and try to fix the problem

```
library(tidyverse)

wide_dataset <- tribble(
  ~merchant, ~day1, ~day2, ~day3, ~day4, ~day5,
  "fizzbizz",   9, 3, 5, 1, 6,
  "wizzmizzy",  5, 1, 7, 1, 8,
  "lollipoppy", 4, 9, 2, 7, 1
)

wide_dataset %>%
   gather(key = "day", value = "frauds_detected")
```

6 minutes

Reshaping data – mini exercise (2)

- In pairs: bring this long format back into a wide format, using *spread*
  - The code is also available in the exercise folder under: "03-Example for spread gather.R"

```
library(tidyverse)

long_dataset <- expand.grid(merchant = c("fizzbizz", "wizzmizzy", "lollipoppy"),
                            day = paste0("day", 1:5)) %>%
  mutate(frauds_detected = floor(runif(n = 15, min = 1, max = 10)))

long_dataset %>%
   spread(???)
```

- What would be the proper form to use as a basis of a ggplot2 graph? Why?

6 minutes

Selecting variables (*select*)

- How does variable *selection* works? key for the rest of this presentation
- Name the specific variables:

```
some_tibble %>%
   select(specific_name1, specific_name2, -unwanted_name3)
```

- Use a range

```
some_tibble %>%
   select(specific_name1:specific_name100)
```

- Select helper functions (*starts_with*, *ends_with*, *contains*, *matches*):

```
some_tibble %>%
   select(one_of(c("specific_name1", "specific_name2")))
```

Mental note:
These will do the same, but are fundamentally different

Working on specific variables using select helper functions

- With *mutate_at* you can specify a function which will run on all columns which adhere to a select helper criteria
- *mutate_all* will run over the entire dataset
- *mutate_if* lets you specify a custom condition (instead of using a helper function)
- *summarize_**, *rename_** work in a similar manner

```
library(tidyverse)

some_tbl %>%
   mutate_at(vars(starts_with("some_string")), funs(some_function))
```

funs lets *mutate_at* know that we want it to treat *some_function* as a function

Two table operations
Joining and set operations
(*left_join, right_join, full_join, semi_join, anti_join, setdiff, intersect, union*)

- These functions join two datasets by a common variable(s) as the key
  - *left_join(x, y)* – join all in y into x
  - *right_join(x, y)* – join all in x into y
  - *full_join(x, y)* – join all in x and all in y, retain all values, all rows
  - *inner_join(x, y)* – retain matches only
  - …
- Functions that add or omit rows
  - *intersect(x, y)* – retain rows that appear in x and y
  - *setdiff(x, y)* – rows that appear in x but not in y
  - *union(x, y)* – rows that appear either in x or in y
  - *bind_rows(x, y)* – add y to x
- **Question – what is the difference between *bind_rows* and *union*?**

Exercise

- Open 03-tidyverse.Rmd and solve exercise 1