Intro to R for Biologists
Session 3
Data exploration in depth

Irina & Rao

14/07/2021

Summer 2021

INTRO TO R FOR BIOLOGISTS

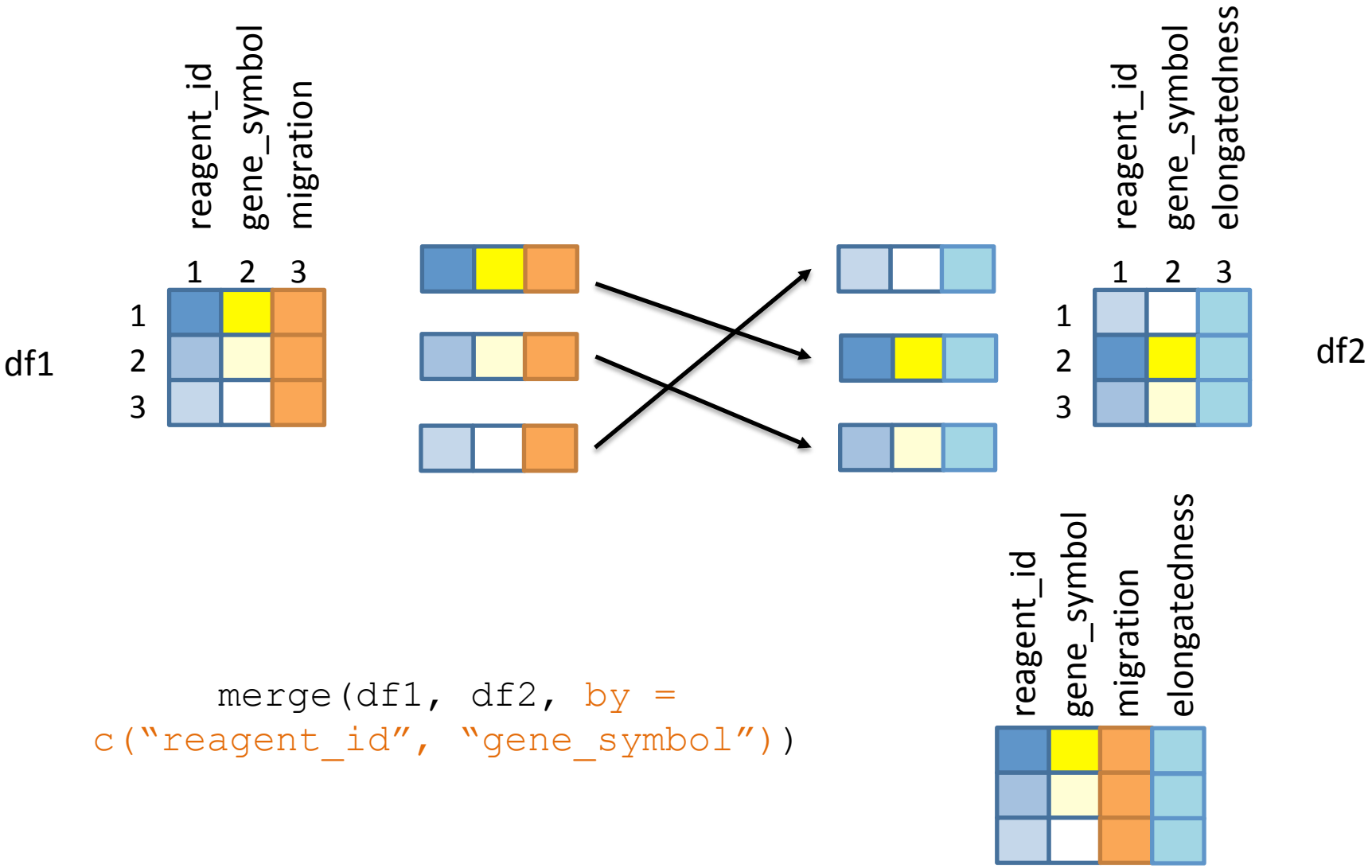
- **Data exploration**
  - Loop functions
  - Merging
  - Filtering and logical ops
  - Intro to Tidyverse
  - Dplyr verbs – mutate, select, filter, summarise, arrange
  - Group and summarise data
  - Reshape data
  - Analyse Covid data (demo)
  - Practical (breakout rooms)
    - Covid data
    - Wet-lab Covid-vaccine data

Loop functions

- `lapply()` – perform an action on each element of a vector: returns list
- `sapply()` – as above, returns a simplified object (variable)
- `apply()` – loop over rows or columns of a matrix or df
- `tapply()` – loop over a vector, split based on a factor
- `mapply()` – loop over more than one vector

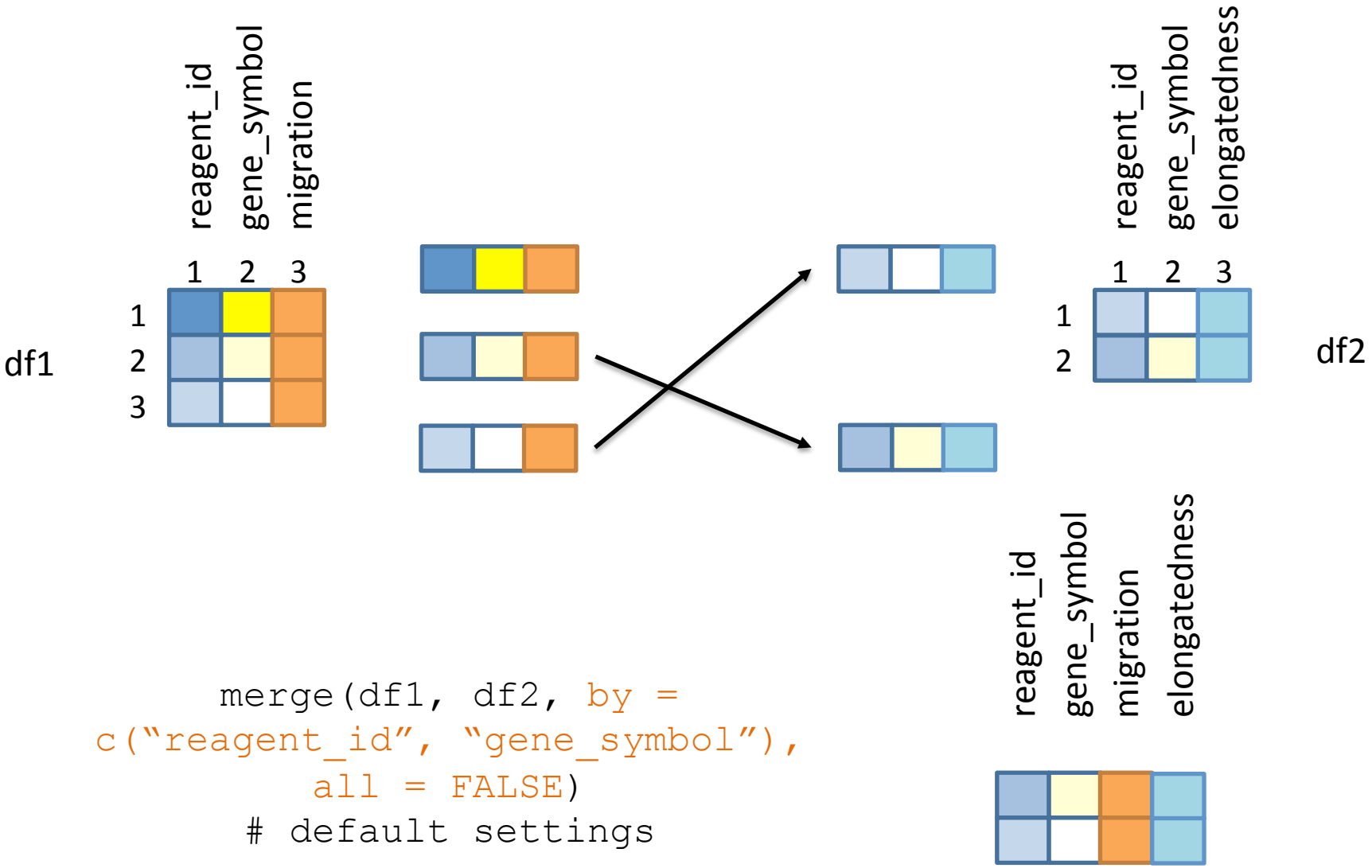
```
> my_list = list(a = c(1, 2, 3), b = c(4, 5, 6), c = c(7, 8, 9))
> lapply(my_list, mean)
$a
[1] 2

$b
[1] 5

$c
[1] 8
```

Merging data.frames



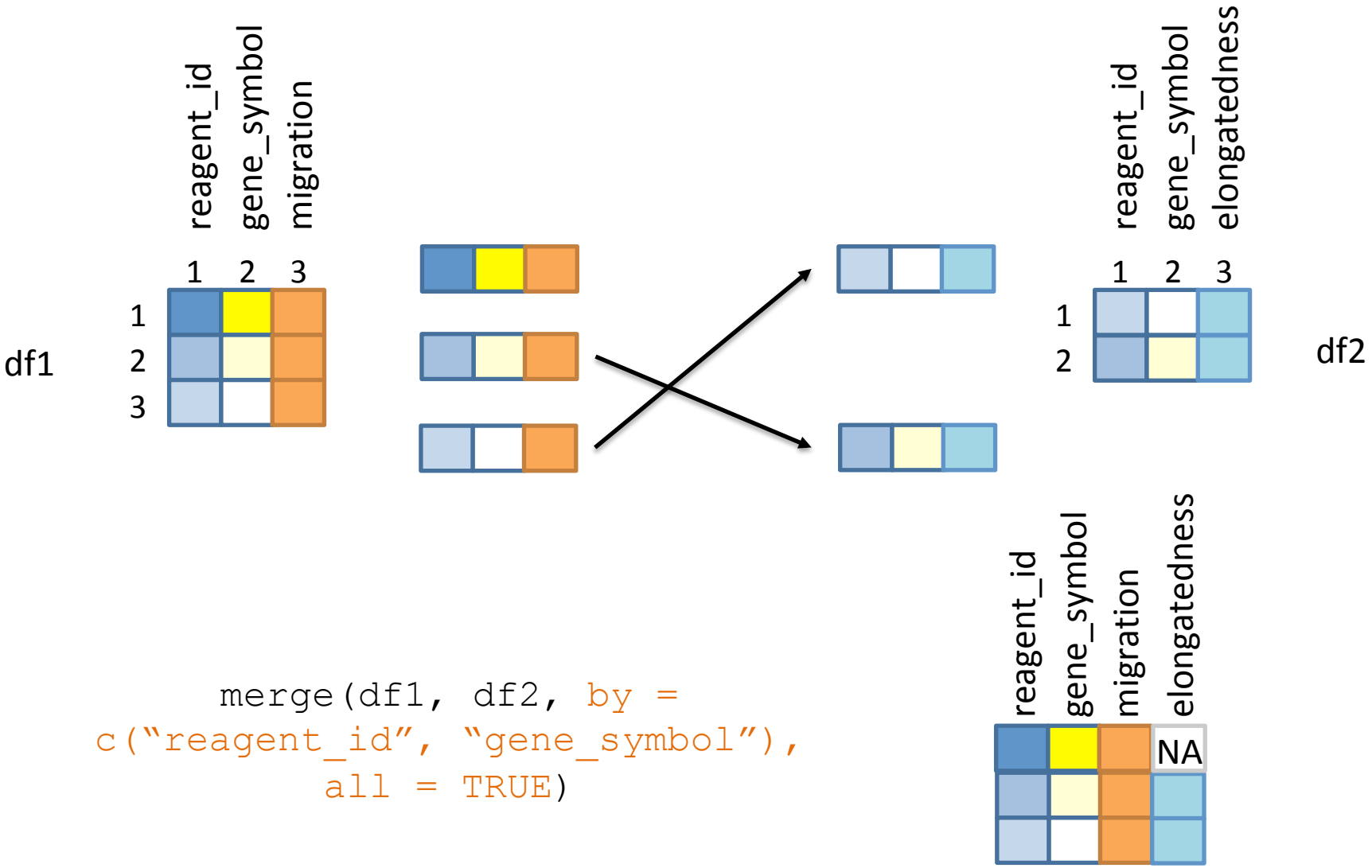

```
merge(df1, df2, by =
c("reagent_id", "gene_symbol"))
```

Merging data.frames




```
merge(df1, df2, by =
c("reagent_id", "gene_symbol"),
        all = FALSE)
    # default settings
```

Merging data.frames




```
merge(df1, df2, by =
c("reagent_id", "gene_symbol"),
        all = TRUE)
```

Filtering data

- Keep/remove data that satisfies one or more conditions

```
> my_vec = c(1, 2, 3, 4, 5, 6)
> my_vec < 4
TRUE TRUE TRUE FALSE FALSE FALSE
> my_vec[my_vec < 4]
1 2 3
```

- For more than one condition, we need to use logical operators
  - & (AND)
  - | (OR)

Logical operators

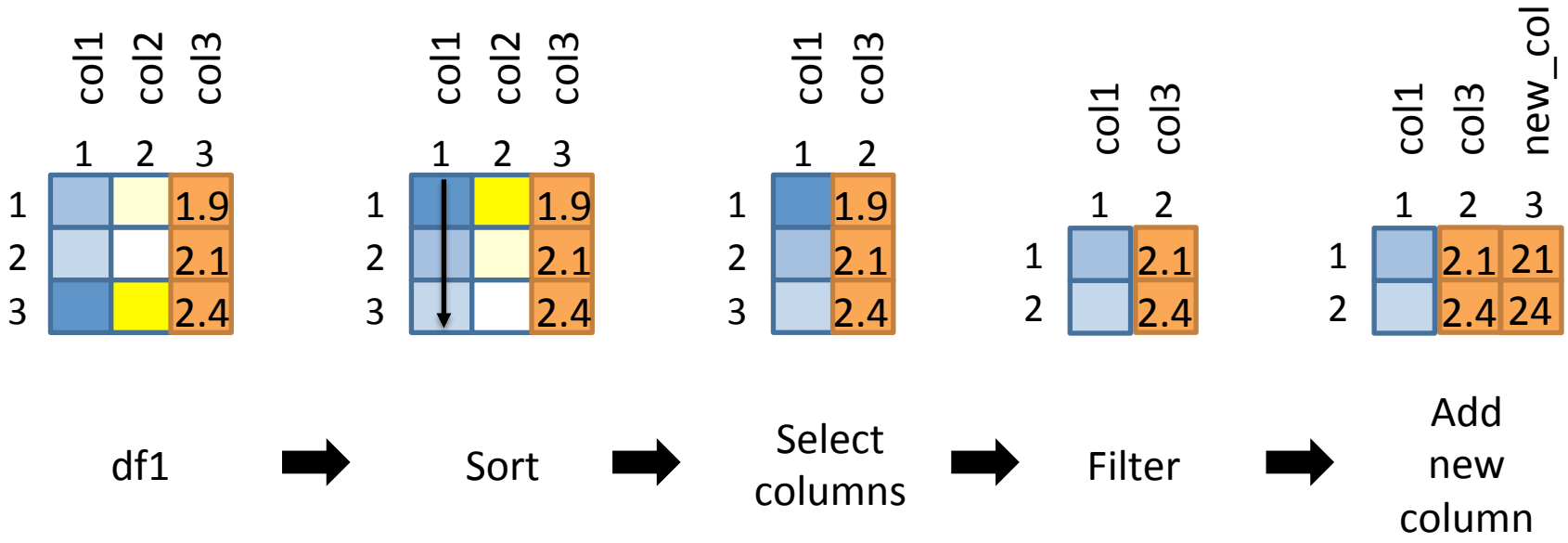
- TRUE & TRUE # evaluates to TRUE
- TRUE & FALSE # evaluates to FALSE
- FALSE & FALSE # evaluates to FALSE
- TRUE | FALSE # evaluates to TRUE

```
> my_vec = c(1, 2, 3, 4, 5, 6)
> my_vec < 4 & my_vec > 2
FALSE FALSE TRUE FALSE FALSE FALSE
> my_vec[my_vec < 4 & my_vec > 2]
3
```

Data exploration and cleaning

- Look at the data
  - `head(), tail(), class(), str(), View()`
- Are the data types correct? If not, convert to appropriate type
  - `as.numeric(), as.character(), as.logical()`
- Is there any missing data? NA or NaN are missing data
  - `na.omit(), complete.cases()`
- Is there unnecessary data? Rows or columns you don't need?
  - Subset the data
  - Filter the data
- Visualise the data with graphs

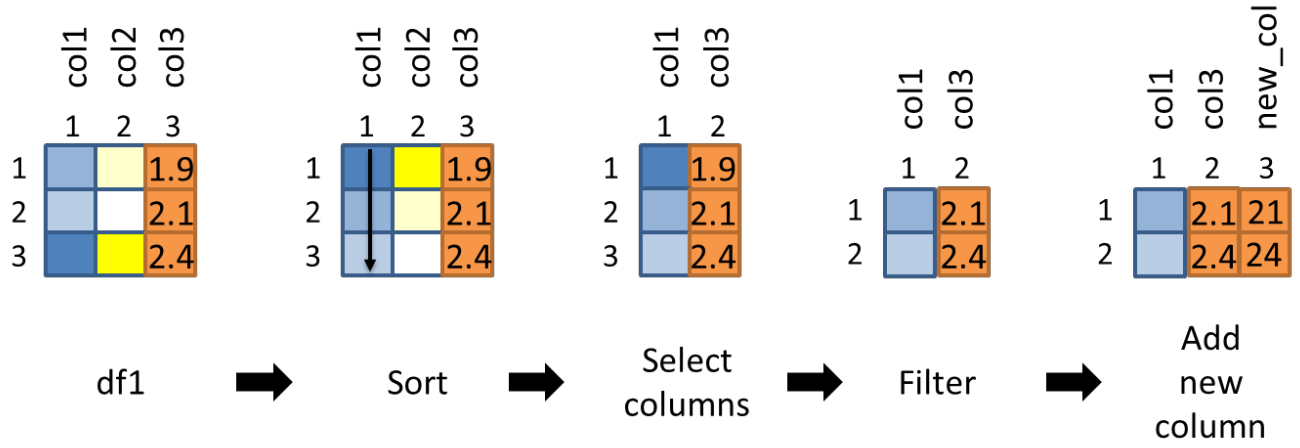
A typical workflow



This can be achieved by using:

- Base R – just the functions that come pre-installed in R
- External packages – to make the analysis faster, more readable, more intuitive, etc.
  - The data.table package – speed, conciseness
  - The dplyr package (Tidyverse) – readability, beginner-friendly
  - Other packages are available, but more help available online for the popular ones

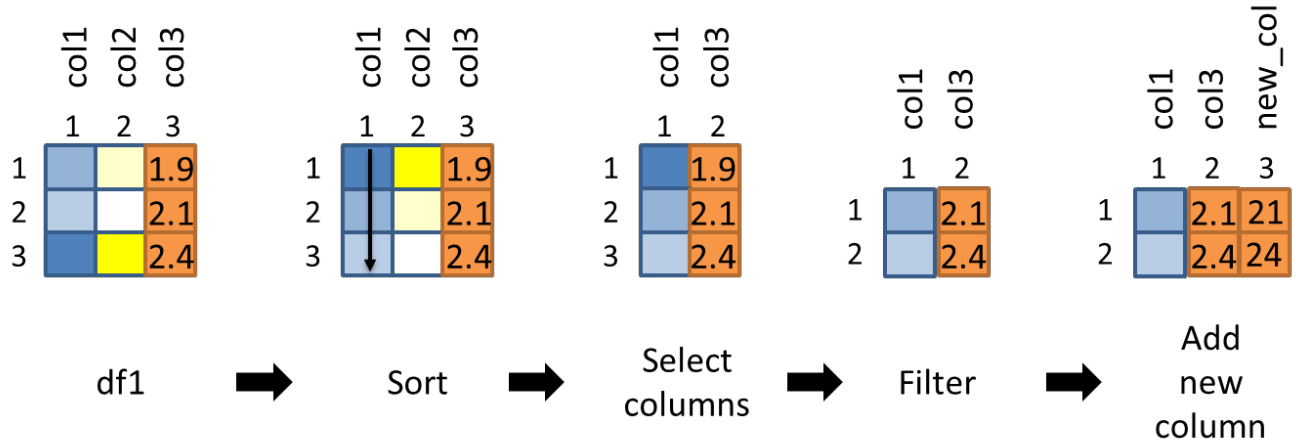
A typical workflow – base R




```
df1_sorted = df1[order(df1$col1), ]
df1_sel_vars = df1_sorted[, c("col1", "col3")]
df1_filtered = df1_sel_vars[df1_sel_vars$col3 > 2.0, ]
df1_filtered$new_col = df1_filtered$col3 * 10
```

```
df1_sort_sel_filtered = df1[order(df1$col1), ][df1$col3 > 2.0, c("col1", "col3")]
df1_sort_sel_filtered$new_col = df1_sort_sel_filtered$col3 * 10
```

A typical workflow – data.table

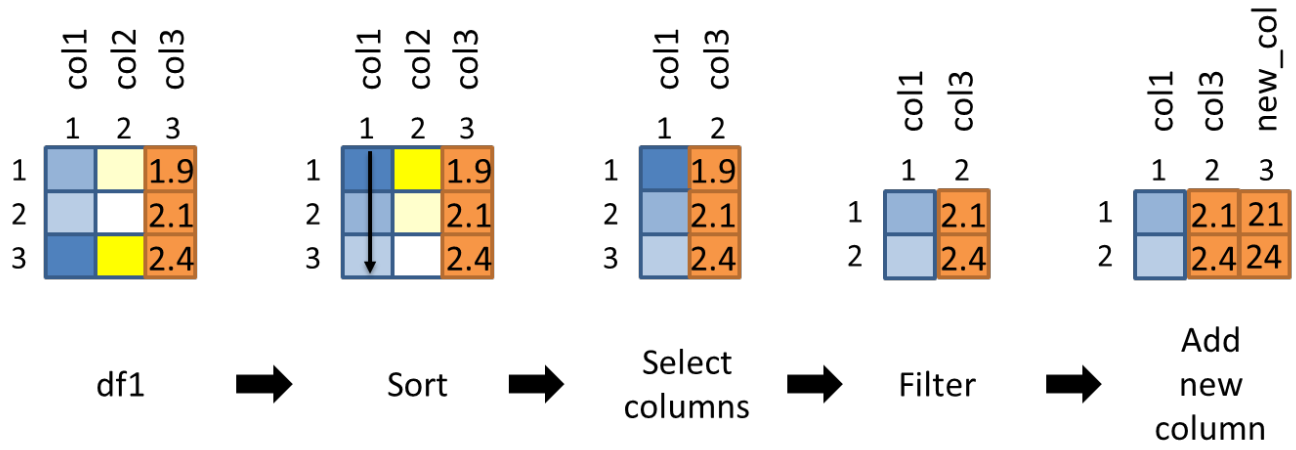



```
df1_sort_sel_filtered = df1[order(col1), ][
                              col3 > 2.0, .(col1, col3)][
                                        , new_col := col3 * 10]
```

Install with `install.packages("data.table")`
We won't discuss data.table in detail, except for some special functions (more from Irina later) but:

- More intuitive for some people (similarity to base R)
- Ultra-fast reading and writing and sorting (look up `fread` and `fwrite` – changed my life!)

A typical workflow – dplyr




```
df1 %>%
      arrange(col1) %>%
      select(col1, col3) %>%
      filter(col3 > 2.0) %>%
      mutate(new_col = col3 * 10) -> df_sort_sel_filtered
```

* Use spacing and indentation effectively to make your code readable

Dplyr is part of the 'Tidyverse'

- Tabular data structures - one observation per row, one variable per column
- Simple functions that do one thing (filter, mutate, arrange, etc.)
- Use the 'pipe' %>% to chain functions – imagine a flow of data from left to right
- The Tidyverse package is a 'meta-package' consisting of
  - **dplyr** – data manipulation
  - **ggplot2** – plotting ('Grammar of Graphics')
  - tidyr – functions to create 'tidy data'
  - readr – reading and writing several file formats
  - purrr – expanded set of loop functions
  - tibble – tidy data.frames
  - stringr – handling strings (joining, searching, splitting, etc.)
  - forcats – handling categorical data (factors)

Arrange

```
df1_sorted = arrange(df1, col1)
```
OR
```
arrange(df1, col1) -> df1_sorted
```
OR
```
df1 %>%
      arrange(col1) -> df1_sorted
```

```
df1_sorted = arrange(df1, col1, col2)
```
OR
```
df1 %>%
      arrange(col1, col2) -> df1_sorted
```

Select columns

```
df1_selected = select(df1, col1, col3)
```

OR

```
df1 %>%
       select(col1, col3) -> df1_selected
```

```
df1_selected = select(df1, -col2)
```

OR

```
df1 %>%
       select(-col2) -> df1_selected
```

Filter

```
df1 %>%
      filter(col3 > 2.0) -> df1_filtered

df1 %>%
      filter(col2 < 100 & col3 > 2.0) -> df1_filtered

df1 %>%
      filter(col2 < 100 | col3 > 2.0) -> df1_filtered
```

Mutate

```
df1 %>%
      mutate(new_col = col3 * 2.0) -> df1_mutated


df1 %>%
      mutate(new_col = col3 * 2.0) %>%
      mutate(col4 = new_col/col3) -> df1_mutated

df1 %>%
      mutate(new_col = col3 * 2.0, col4 = new_col/col3) ->
df1_mutated
```

Base R vs Dplyr syntax

- Column names in quotations in base R, but not in Dplyr

```
Df1[, "col1"] # base R syntax
select(df1, col1) # dplyr syntax
```

- Dplyr functions *always* have the data as the first argument

```
select(df1, col1)
arrange(df1, col1)
mutate(df1, col4 = col1 * 10)
```

- When the 'pipe' %>% is used, the first argument of any function is passed invisibly, so we drop it in the actual function call

```
df1 %>% arrange(df1, col1) %>% select(df1, col1, col3)
```

Let’s explore practically

Try out the PROBLEM SET in breakout rooms

Summarise

Reduces multiple values to a single value

- One column:

```
summarise(df1, sum(col1))
```
or
```
summarise(df1, mean(col2))
```

- Several columns:

```
summarise(df1, sum(col1), mean(col2), sd(col3))
```

- Assign column names:

```
df1 %>%
      summarise(sum_col1 = sum(col1), mean_col2 =
mean(col2), sd_col3 = sd(col3)) -> df1_sum
```

Note: **`summarise()`** can be used separately, but typically used on grouped data created by **`group_by()`** - see next slides.

Useful functions for summarisation

- Center: `mean(), median()`
- Spread: `sd(), IQR()`
- Range: `min(), max(), quantile()`
- Position: `first(), last()`
- Count: `n(), n_distinct()`
- Logical: `any(), all()`

`...`

Group by one or more variables

```
df1 %>% group_by(col2)
```

Grouping itself doesn't change how the data looks!
It means that further operations will always be performed "by group".

```
df1 %>%
      group_by(col2) %>%
      summarise(sum_col4 = sum(col4)) -> df1_grouped

df1 %>%
      group_by(col2, col3) %>%
      summarise(sum_col4 = sum(col4)) -> df1_grouped

df1 %>%
      group_by(new_col2 = toupper(col2)) %>%
      summarise(sum_col4 = sum(col4)) -> df1_grouped
```

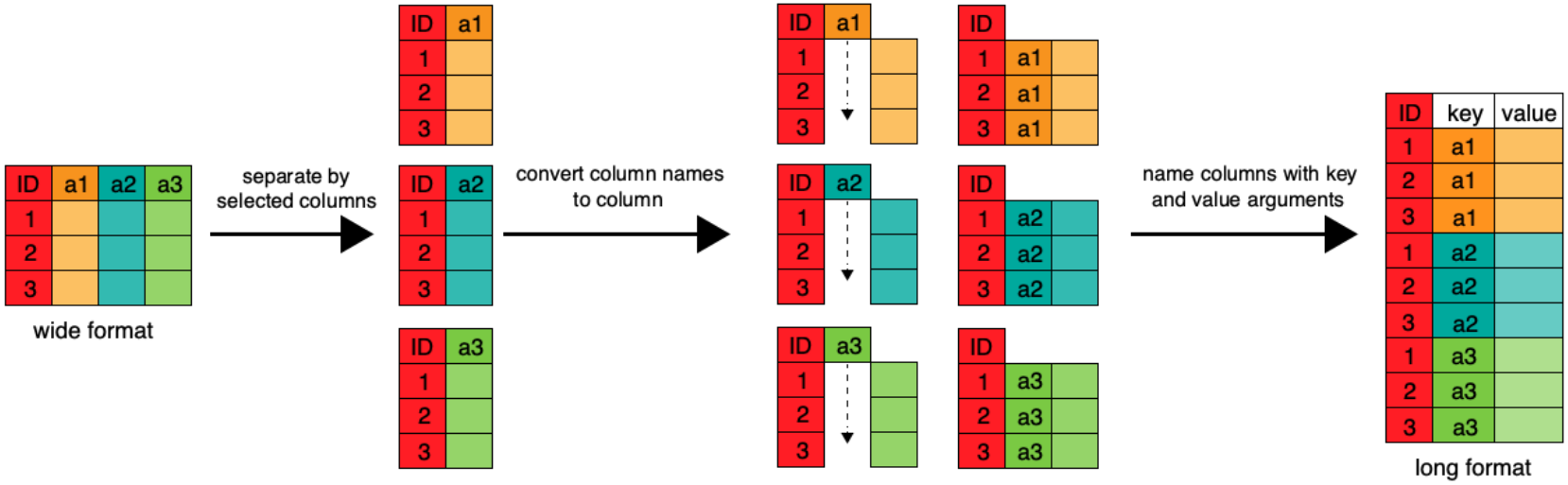
Explore it yourself

Scoped grouping - three scoped variants:
`group_by_all()`, `group_by_if()` and `group_by_at()` make it easy to group a dataset by a selection of variables.

`across()` - function, superseding the `_all`, `_at`, and `_if` versions of `summarise()` and `mutate()`

```
df1 %>%
        group_by(col2, col3) %>%
        summarise(across(c(col1, col4), mean)) ->
df1_grouped
```

Wide vs. Long data format



Reshaping data with data.table

```
melt(data,
id.vars,
measure.vars,
variable.name = "variable",
value.name = "value",…)
```

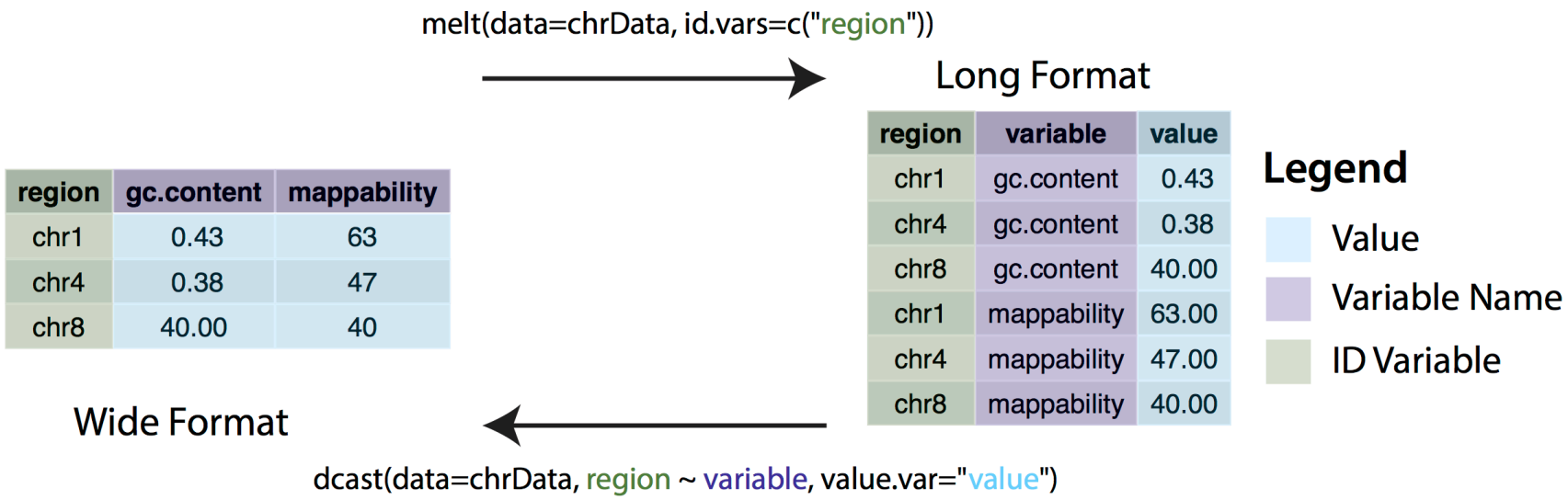
Wide-to-long format

```
dcast(data,
formula,
value.var = guess(data),…)
```

Long-to-wide format

Hint: **tidyr** – part of Tidyverse (introduced by Rao) has synonymous reshaping functions **gather** and **spread**

Reshaping data - example



Real dataset

Letter | Published: 17 December 2020

T cell and antibody responses induced by a single dose of ChAdOx1 nCoV-19 (AZD1222) vaccine in a phase 1/2 clinical trial

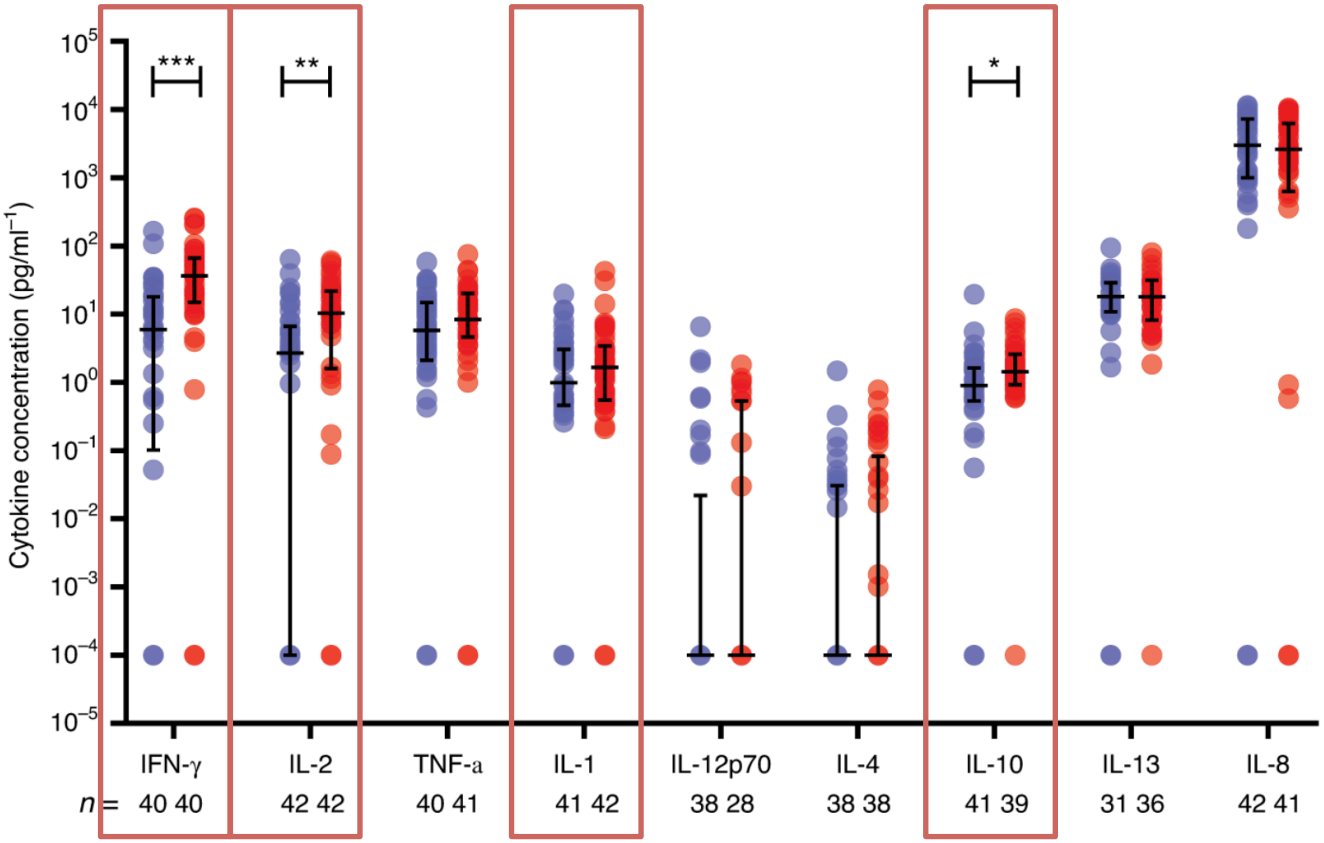
Katie J. Ewer ✉, Jordan R. Barrett, Sandra Belij-Rammerstorfer, Hannah Sharpe, Rebecca Makinson, Richard Morter, Amy Flaxman, Daniel Wright, Duncan Bellamy, Mustapha Bittaye, Christina Dold, Nicholas M. Provine, Jeremy Aboagye, Jamie Fowler, Sarah E. Silk, Jennifer Alderson, Parvinder K. Aley, Brian Angus, Eleanor Berrie, Sagida Bibi, Paola Cicconi, Elizabeth A. Clutterbuck, Irina Chelysheva, Pedro M. Folegatti, Michelle Fuskova, Catherine M. Green, Daniel Jenkin, Simon Kerridge, Alison Lawrie, Angela M. Minassian, Maria Moore, Yama Mujadidi, Emma Plested, Ian Poulton, Maheshi N. Ramasamy, Hannah Robinson, Rinn Song, Matthew D. Snape, Richard Tarrant, Merryn Voysey, Marion E. E. Watson, Alexander D. Douglas, Adrian V. S. Hill, Sarah C. Gilbert, Andrew J. Pollard, Teresa Lambe ✉ & the Oxford COVID Vaccine Trial Group -Show fewer authors

Nature Medicine (2020) | Cite this article

DOI: 10.1038/s41591-020-01194-5

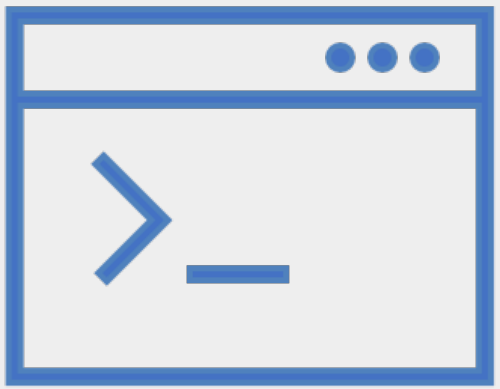
Data to explore - MSD

Fig.1 i



A multiplex cytokine analysis was performed on day 7 after vaccination using supernatants after antigen-specific stimulation of PBMCs from ChAdOx1 nCov-19 (red) and MenACWY (blue). Number of samples presented: MenACWY–ChAdOx1 nCov-19: IFN-γ (n = 40,40); IL-2 (n = 42,42); TNF-α (n = 40,41); IL-1β (n = 41,42); IL-12p70 (n = 38,28); IL-4 (n = 38,38); IL-10 (n = 41,39); IL-13 (n = 31,36); and IL-8 (n = 42,41). Individual data points are shown here as an aligned dot plot with lines showing the median with IQR. Significant differences were determined by two-tailed Mann–Whitney test with Bonferroni correction for multiple comparisons (***P < 0.001; **P < 0.01; *P < 0.05).

Let’s explore practically



Address the tasks in breakout rooms!

Useful reference

- R Programming for Data Science (Roger Peng) – Ch. 3, 4, 5, 9
- Swirl – Interactive learning
- R for Data Science (Hadley Wickham) – Using Dplyr verbs
- The data.table package – Intro
- A data.table and Dplyr tour – Side by side comparison of functions
- Cheatsheets – quick reference for tasks like data wrangling, visualisation, and several packages