Tidyverse workshop

Sander Wuyts
Stijn Wittouck

Overview

1. Introduction
2. Introduction to ggplot2
3. Introduction to table manipulation
4. Introduction to tidy data
5. Additional materials
    a. Additional verbs
    b. Advanced ggplot2
    c. Other tidyverse packages

1. Introduction

Who are we?

Stijn Wittouck

M.Sc. Bioscience Engineering: Bioinformatics
KU Leuven

PhD Student
Lab of Applied Microbiology & Biotechnology
UAntwerp

@s_wittouck

Sander Wuyts

M.Sc. Bioscience Engineering: Cell & Gene technology
KU Leuven

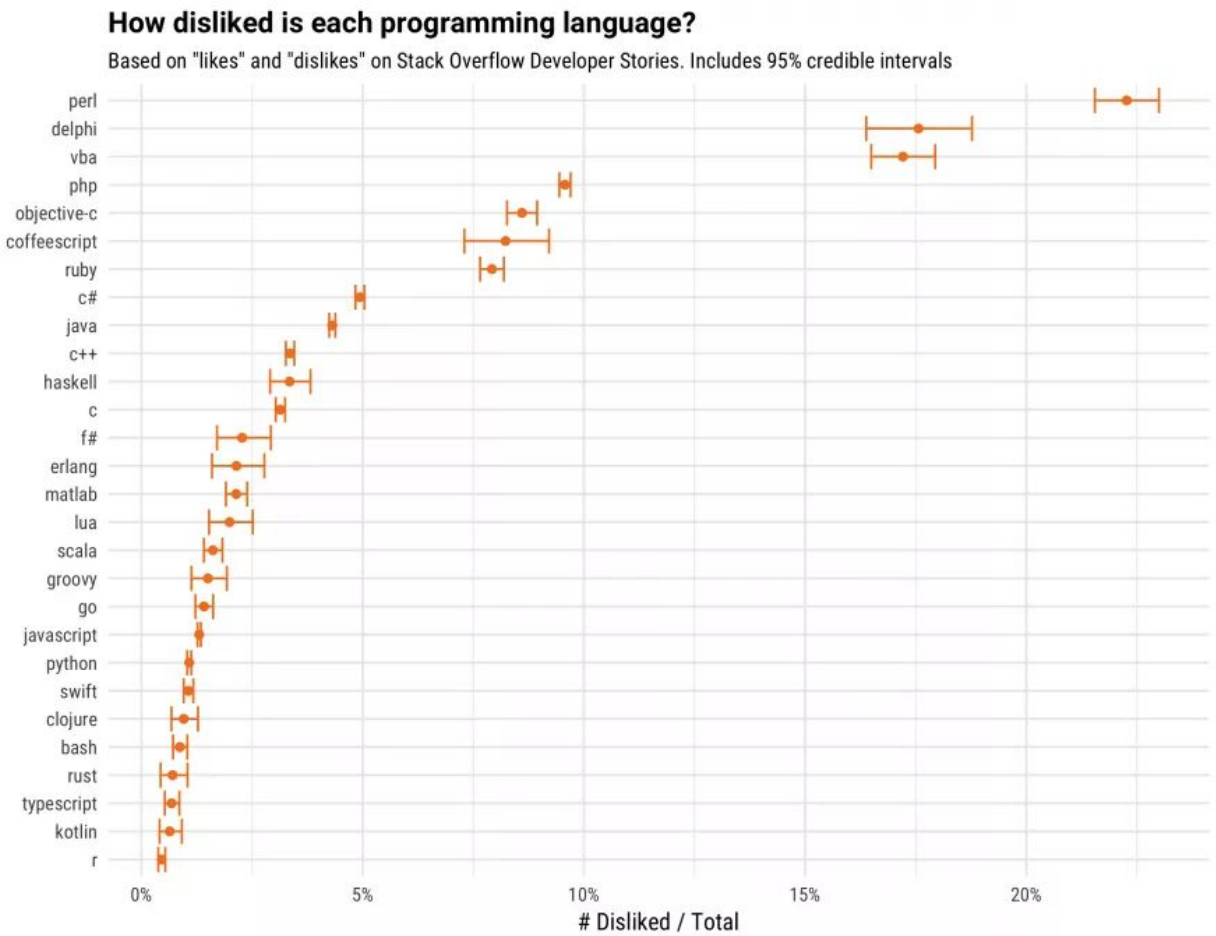
PhD Student
Lab of Applied Microbiology & Biotechnology
UAntwerp

Industrial Microbiology and Food Biotechnology
VUB

@s_wuyts

R

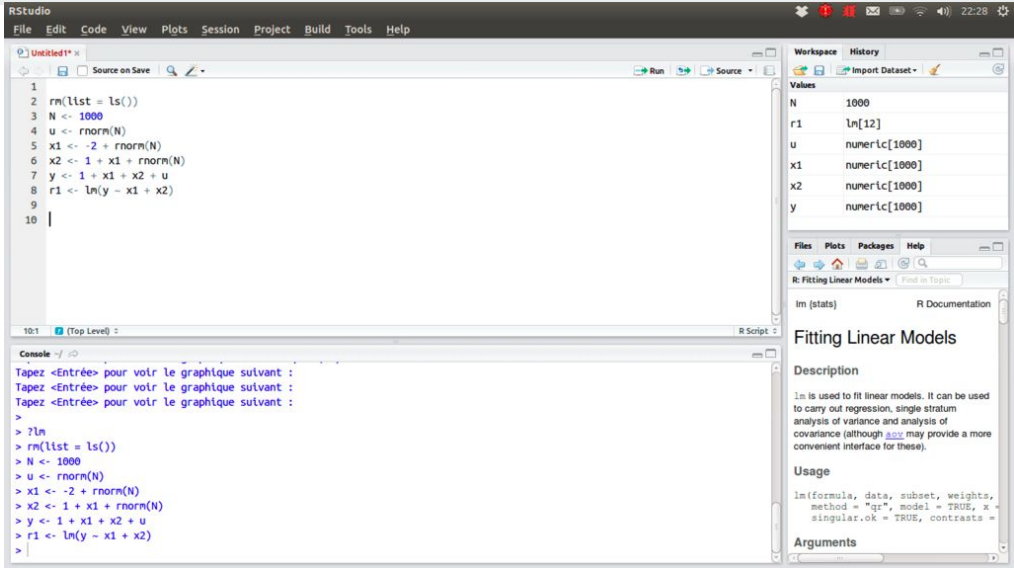
- Open source programming language
- Mostly known as software environment for statistical computing
- Rising popularity in the data sciences
- Capability is expandable by importing *packages*
  - > 11,000 packages available through CRAN, Bioconductor, Github, …
- Most of the analyses are centered around dataframes (~ spreadsheets or tables in SQL)



https://boingboing.net/2017/10/31/perl-is-the-most-hated-program.html

RStudio

- Integrated development environment (IDE)
- Free and open-source
- Cross platform (Windows, macOS & Linux)
- Also available for servers

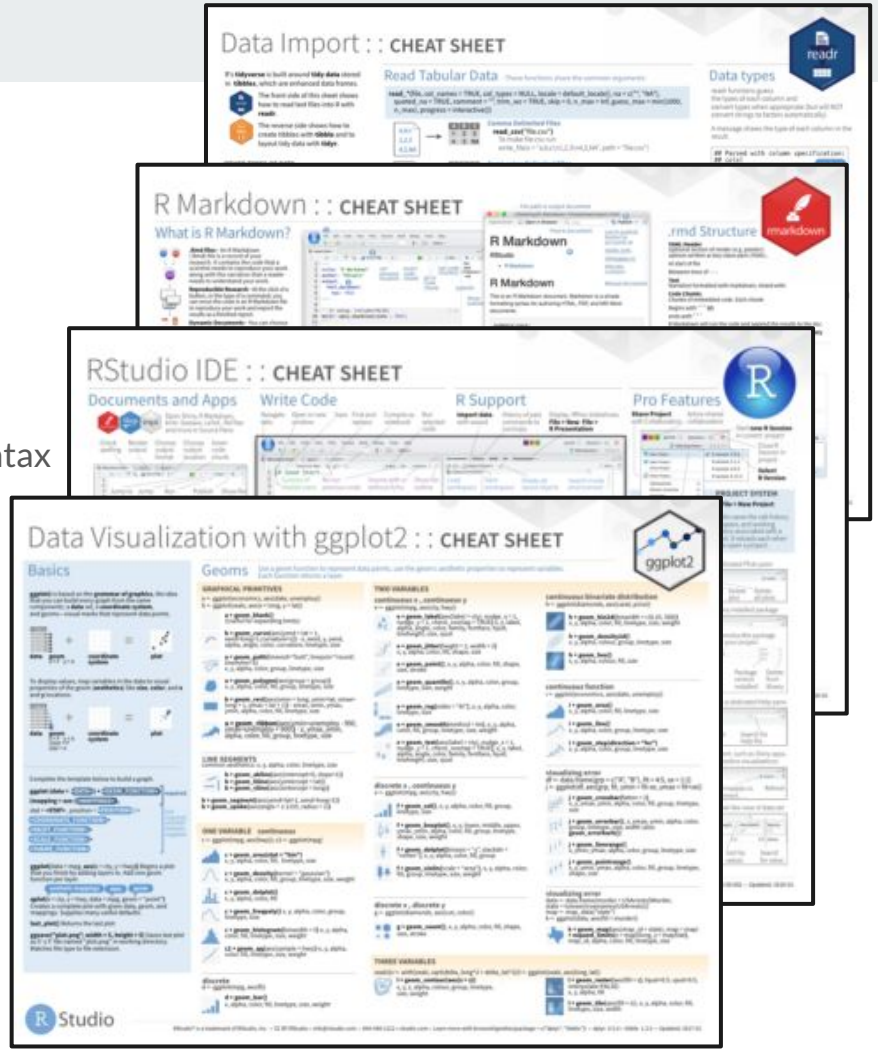


RStudio®

RStudio cheat sheets

Very good reference if you can't remember the right syntax

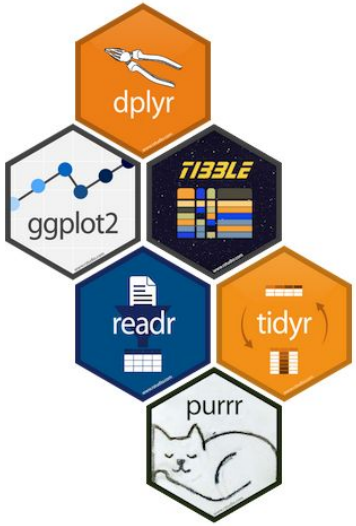
https://www.rstudio.com/resources/cheatsheets/



The tidyverse

R packages for data science

- Set of tools to transform and visualise data
- All packages share an underlying philosophy
- Most of them are created by Hadley Wickham
- *packages*:
  - ggplot2
  - dplyr
  - tidyr
  - readr
  - ...



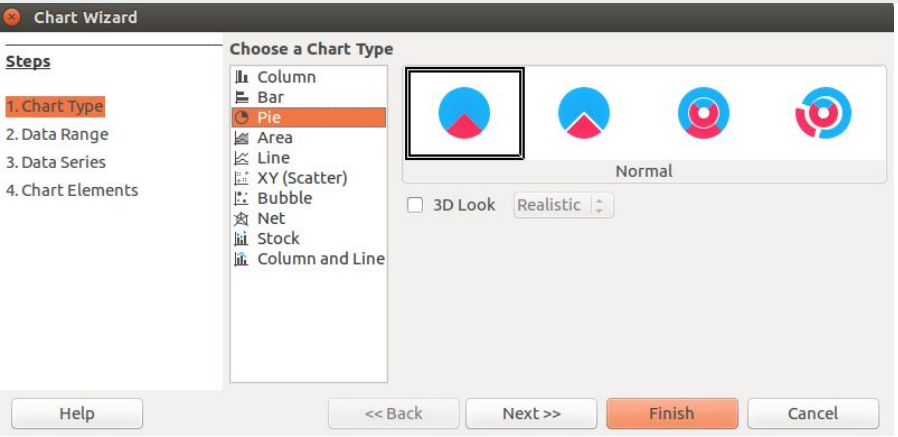
2. Introduction to ggplot2

Grammar of Graphics

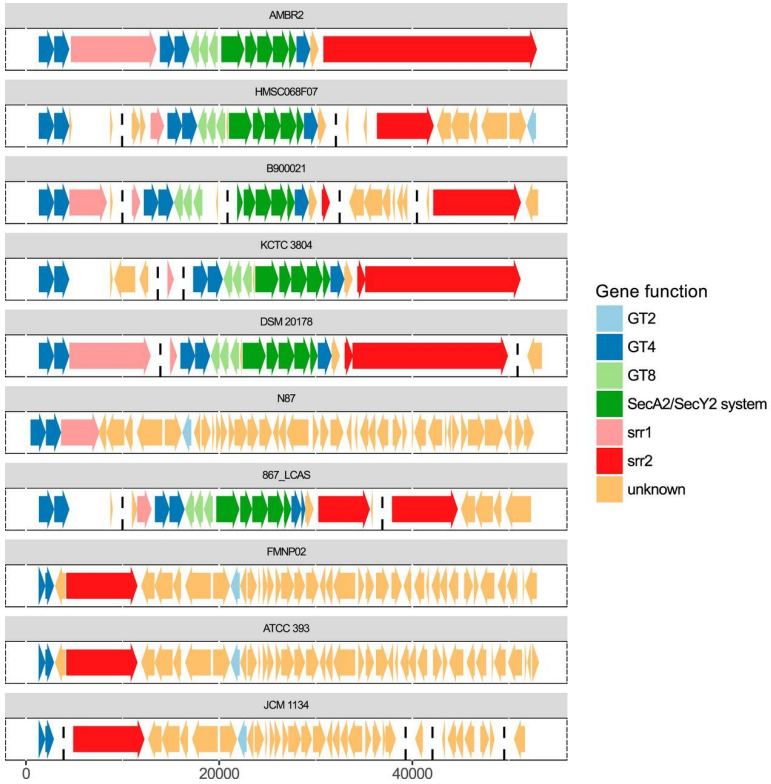
"An abstraction which makes thinking, reasoning and communicating graphics easier"

- First described by Leland Wilkinson (**G**rammar of **G**raphics, 1999)
- Implemented in **gg**plot2 (Hadley Wickham)
- Divide your graphics in different layers based on grammar

=> Use building blocks to create your visualisation



“Click button” visualizations

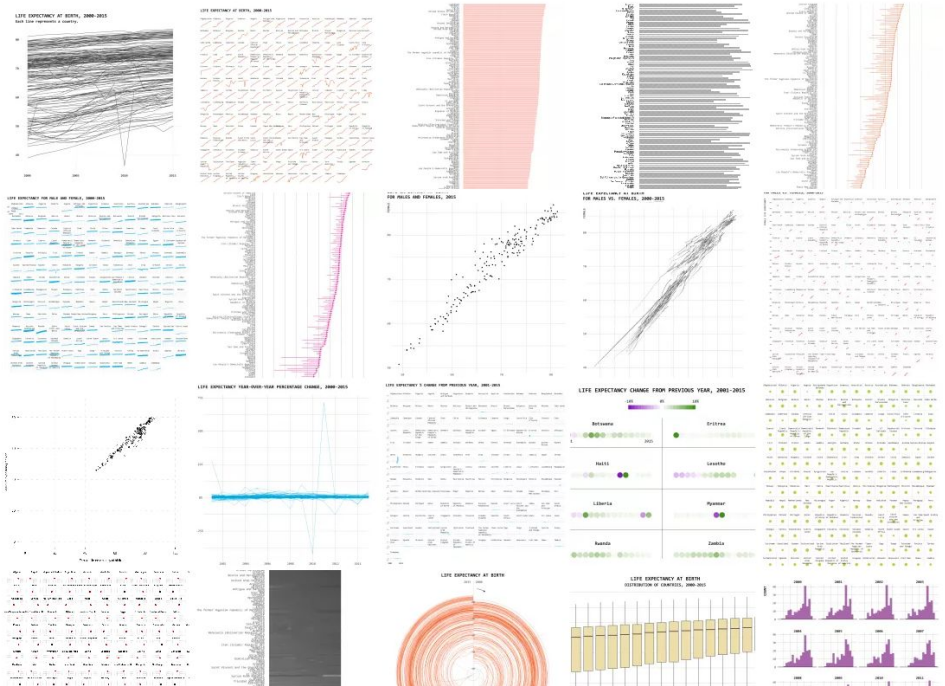


Grammar of graphics visualisation
ggplot2

Explore the design space

- Once you know the grammar
  => lot's of possibilities!

- Same dataset visualized 25 times



https://flowingdata.com/2017/01/24/one-dataset-visualized-25-ways/

Build your own ggplot graph

Three main parts of a ggplot graph

1. **Data**
   => your dataset of interest

2. **Aesthetic mapping**
   => An aesthetic is a visual dimension of your graph that can be used to communicate information
   (*e.g.* x-axis and y-axis in a scatterplot, color, shape, ...)

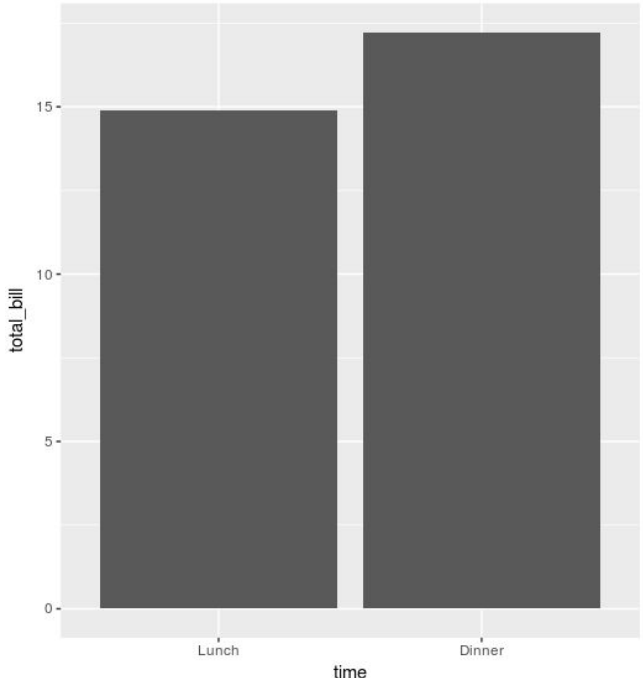
3. **Geoms**
   => Add a layer of geometric objects
   (*e.g.* points, lines, bars, ...)

Build your own ggplot graph

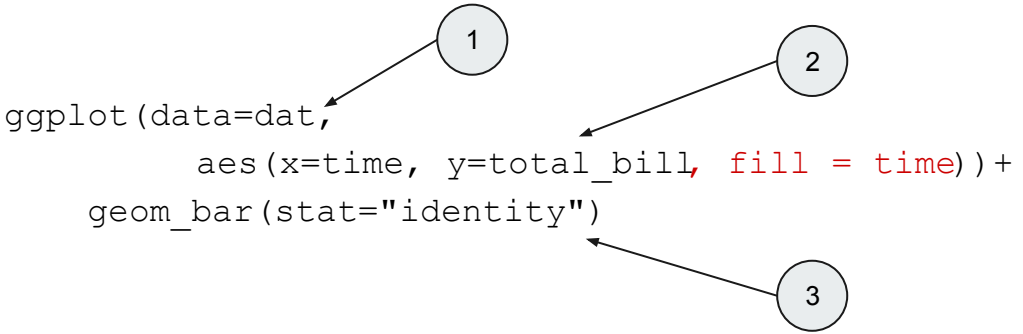
```
ggplot(data=dat,
          aes(x=time, y=total_bill))+
    geom_bar(stat="identity")
```

1
2
3

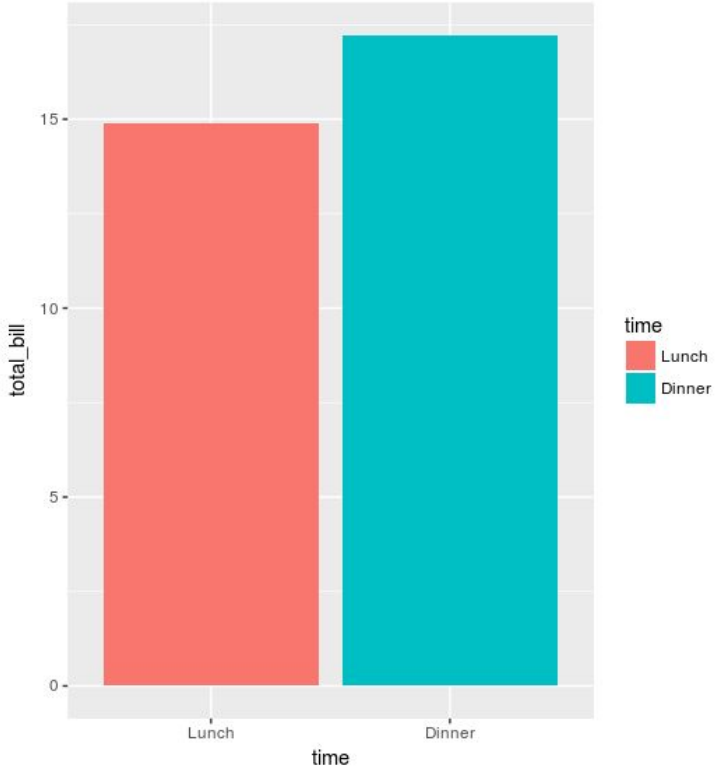
1. Data
2. Aesthetic mapping
3. Geoms



Build your own ggplot graph



1. Data
2. Aesthetic mapping
3. Geoms



Build your own ggplot graph

Additional layers

4. Stats => Statistical transformations
5. Position adjustments => Resolves overlapping geoms
6. Scales => Tweak details like the axis labels or legend keys
7. Facets => Display different subsets of the data
8. Coord => Change how the *x* and *y* aesthetic combine
9. Themes => Control the display of all non-data elements of the plot

1. data

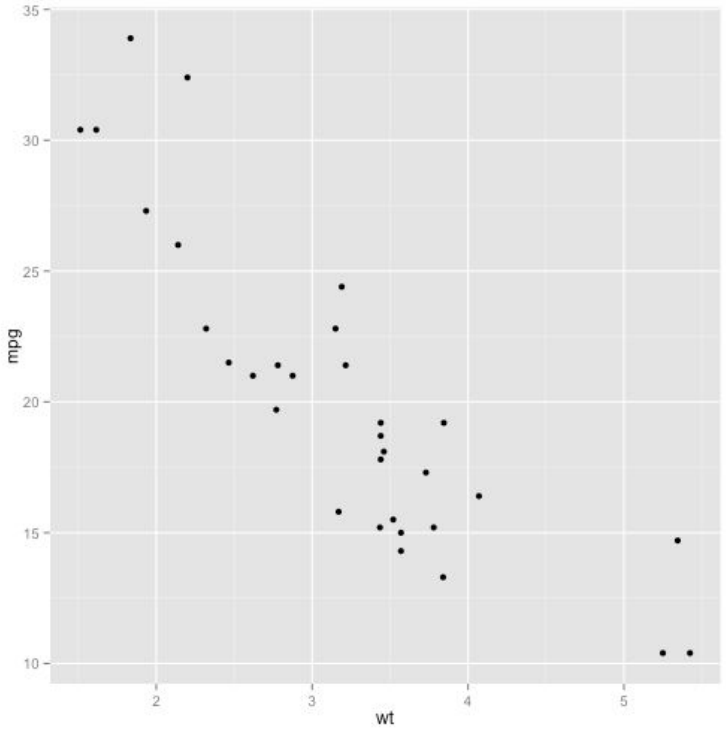
More about data formatting and data handling in the following chapters

For now read in data using:

```
read_tsv()

read_csv()

read_table()

...
```

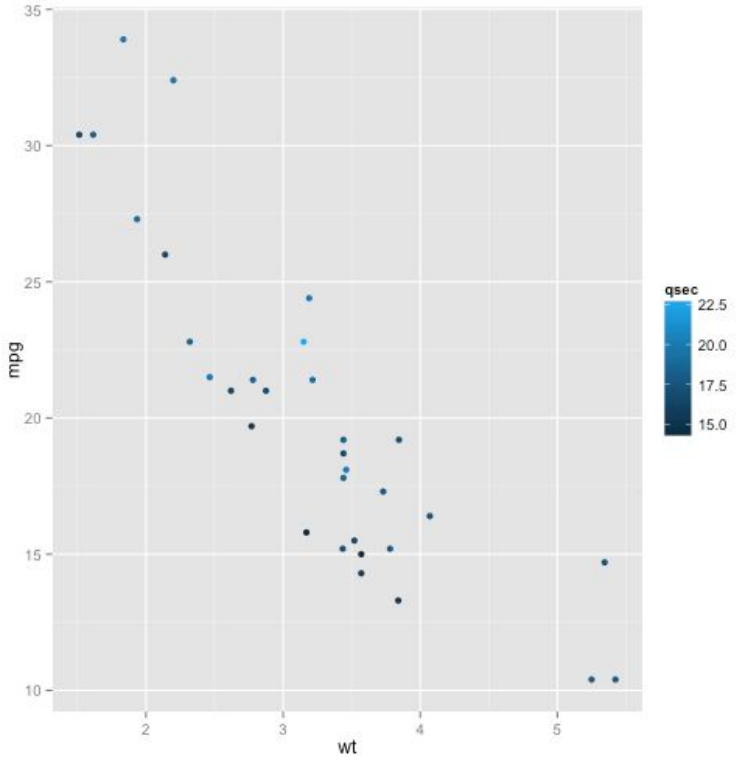
2. Aesthetic mapping

- x- and y-axis



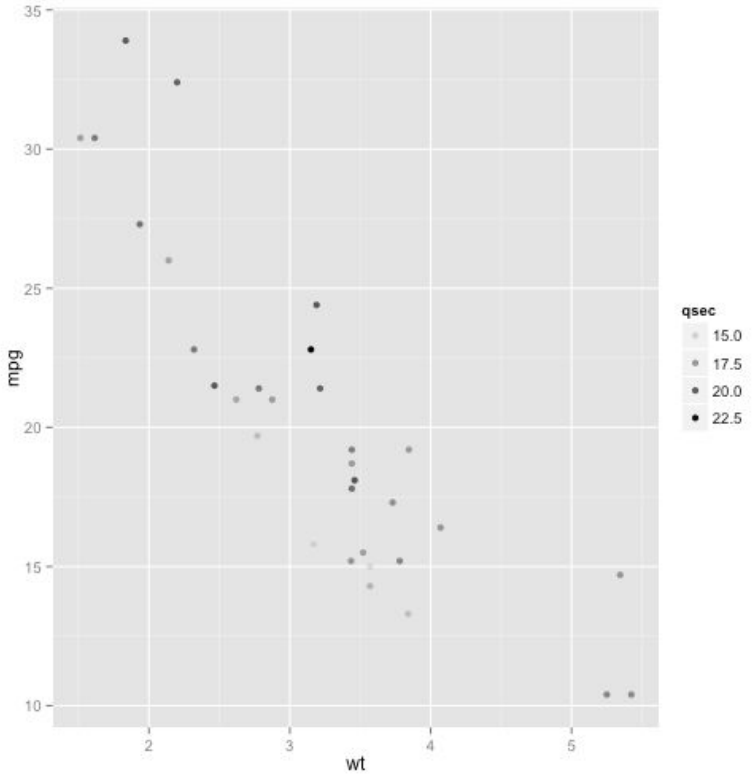
2. Aesthetic mapping

- x- and y-axis
- color



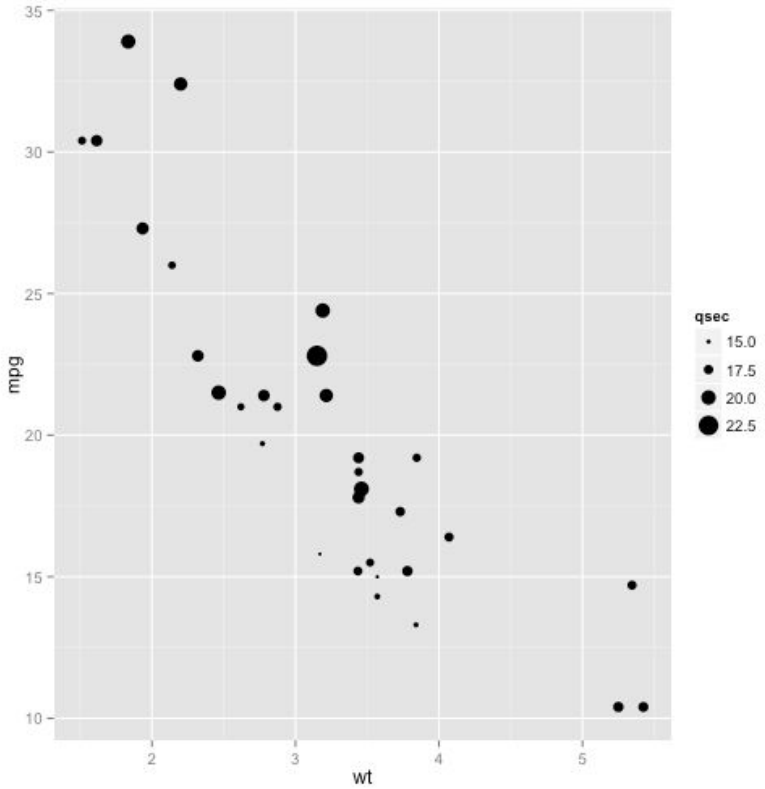
2. Aesthetic mapping

- x- and y-axis
- color
- alpha (transparency)



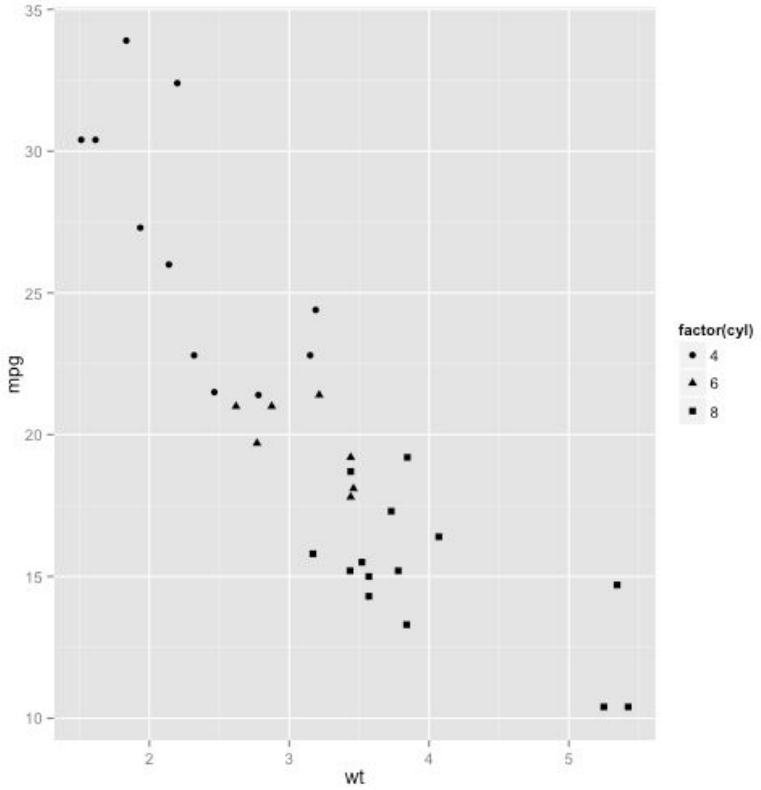
2. Aesthetic mapping

- x- and y-axis
- color
- alpha (transparency)
- size



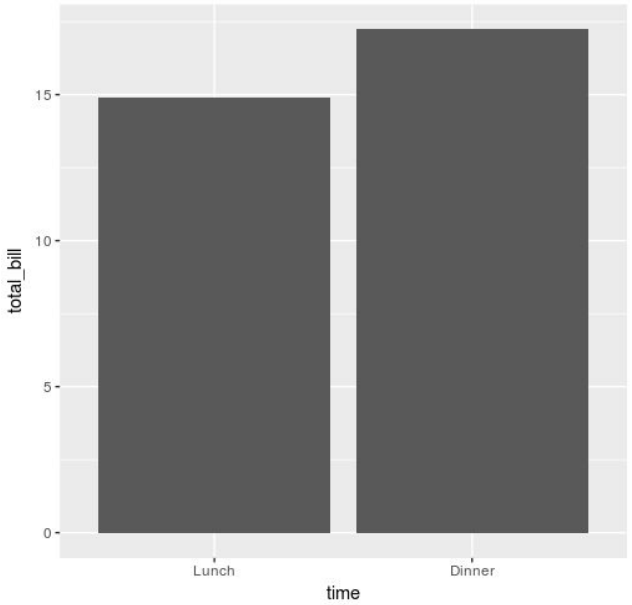
2. Aesthetic mapping

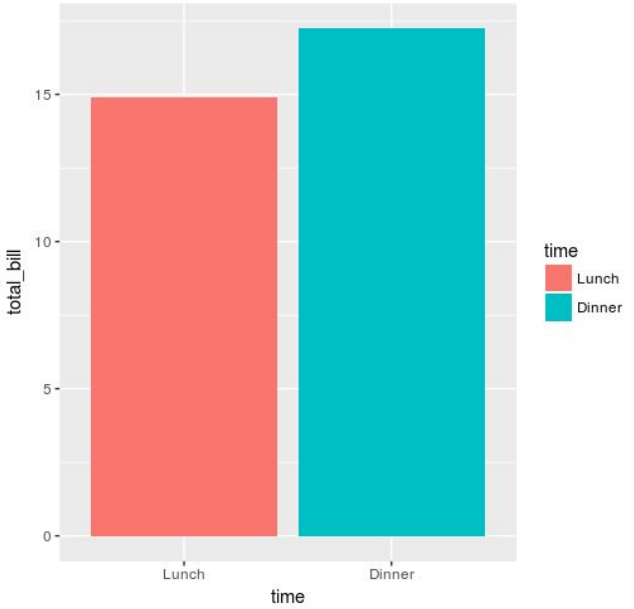
- x- and y-axis
- color
- alpha (transparency)
- size
- shape



2. Aesthetic mapping

- x- and y-axis
- color
- alpha (transparency)
- size
- shape
- fill
- ...





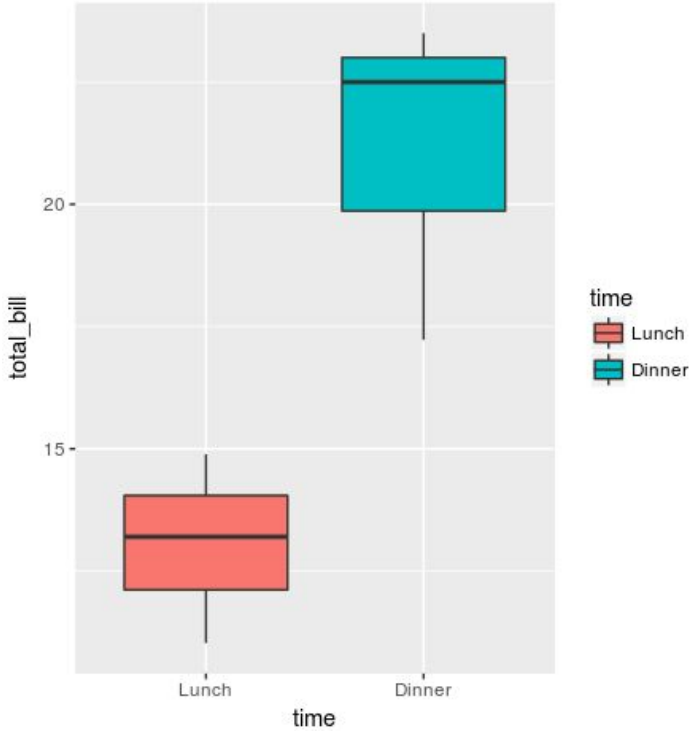
3. Geoms

```
geom_point()

geom_bar()

geom_boxplot()

geom_violin()

geom_line()

...
```

3. Geoms

You can build different layers of geoms!

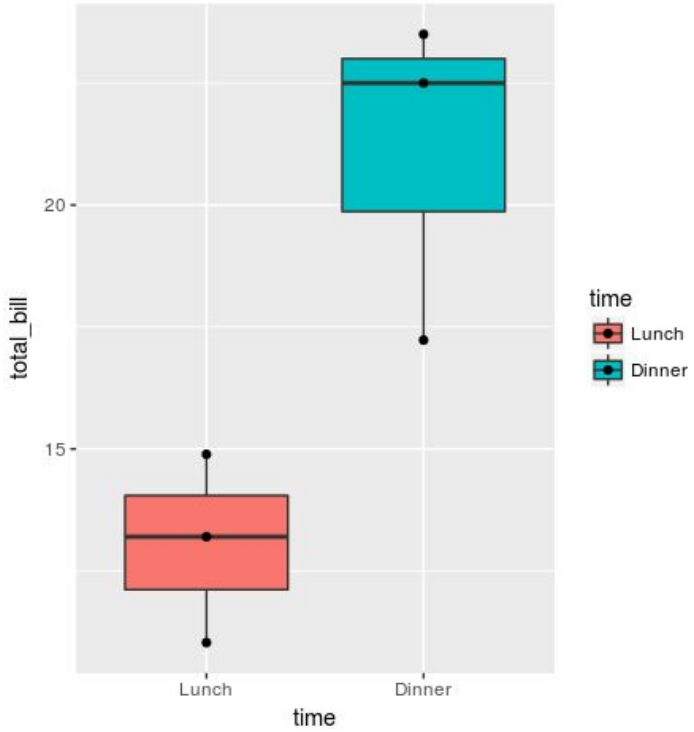
```
ggplot(data=dat,
          aes(x=time, y=total_bill, fill = time)) +
     geom_boxplot()
```



3. Geoms

You can build different layers of geoms!

```
ggplot(data=dat,
          aes(x=time, y=total_bill, fill = time)) +
     geom_boxplot() +
     geom_point()
```



Demonstration

Demonstration dataset is gathered from NCBI’s Eukaryote genome data

Exercise chapter 2

Exercise dataset is the enterotype dataset (Arumugam, M. *et al. Nature, 2011)* obtained from the *Phyloseq* package

1. Read in “sampledata.tsv”
2. Explore the dataset
3. Plot the amount of males and females in this study using a barchart
4. Do the same but for the nationality of the participants instead
5. Create a boxplot showing the age distribution of each nationality. Use the fill aesthetic to make it a little bit more colorful
6. Add an extra layer to 5. with plotting points over the boxplot. Remove that layer again and explore the difference with geom_jitter()
7. Advanced: Plot the age density of all participants coloured by gender, faceted per nationality.

3. Introduction to table manipulation

Grammar of data manipulation

ggplot: grammar of graphics → similar “grammar of data manipulation”?

Verbs:

- functions that perform one task
- tibble as input (first argument)
- tibble as output

Complex tasks can be expressed as sequences of simple verbs

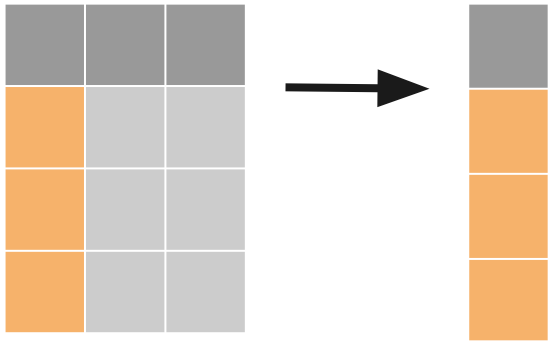
Grammar of data manipulation

Package dplyr

5 essential verbs:

- `select()`: select columns
- `mutate()`: make new columns
- `filter()`: select rows
- `arrange()`: order rows
- `summarize()`: summarize rows

Select columns: `select()`



```
tibble <- select(tibble, var_1, var_2, …)
```

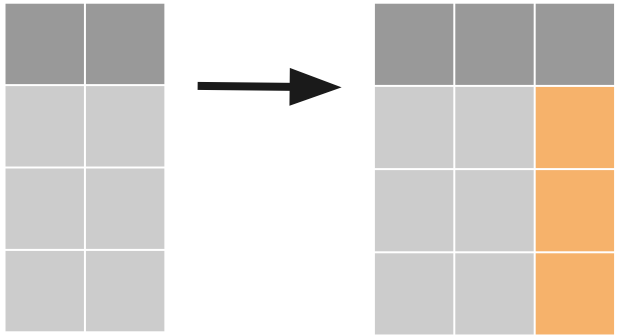
arguments:

- tibble or data frame
- bare variable names: no quotes!

helper verbs for variable selection:

- `contains()`, `starts_with()`, `ends_with()`, `-`, …

Make columns: `mutate()`



```
tibble <- mutate(tibble,
new_var_1 = expression_1,
new_var_2 = expression_2, ...)
```

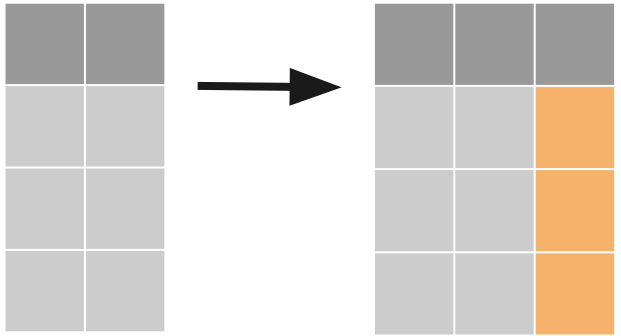
arguments:

- tibble or data frame
- expressions to compute new variables
  - use “=”, not “<-”
  - bare variable names (new and old)

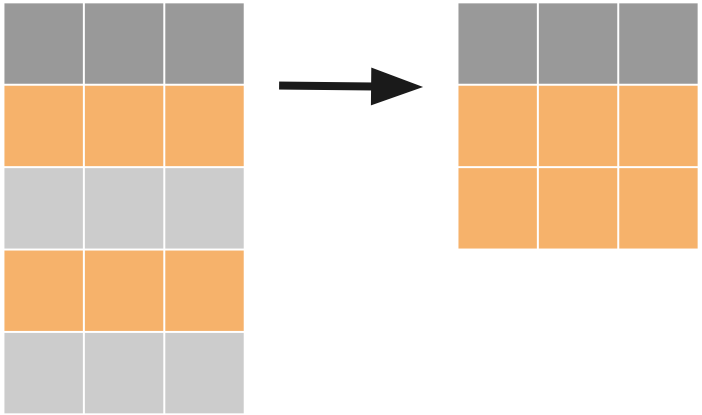
Make columns: `mutate()`

what to compute?

- `+, -, *, /, ^`
- `mean(),sd(),sum()`



Select rows: `filter()`



```
tibble <- filter(tibble, logical_variable)
```

arguments:

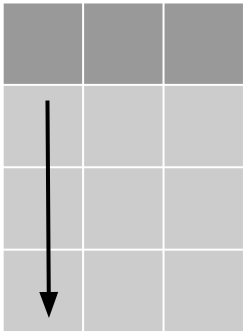
- tibble or data frame
- logical variable
  - type directly, e.g. `c(T, T, F, T, ...)`
  - compute from existing (bare) variables, e.g. `var > 10`
  - to create logical variables: `==, !=, >, <, >=, <=, %in%, is.na()`

Order rows: `arrange()`

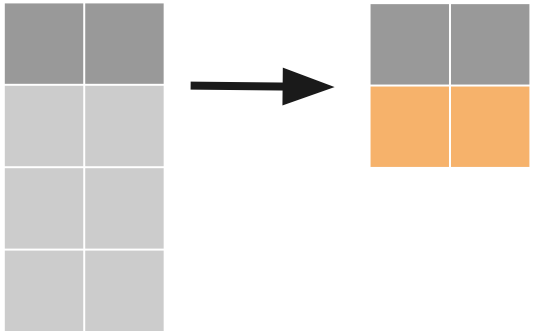
```
tibble <- arrange(tibble, var_1, - var_2)
```

arguments:

- tibble or data frame
- bare variable names to sort on
    - use “-” to sort in inverse order



Summarize rows: `summarize()`



```
tibble <- summarize(tibble,
aggregated_var = expression)
```

arguments:

- tibble or data frame
- expressions that result in one value
  - use “=”, not “<-”
  - bare variable names (new and old)
  - e.g. `mean_var = mean(var)`

Group-wise analysis

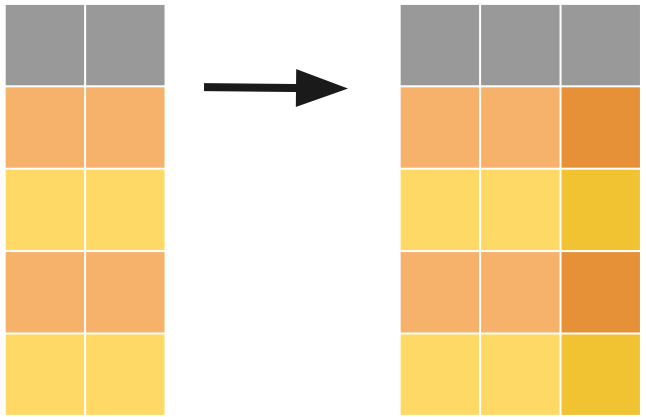
You can add grouping structure to a tibble

- computations within other verbs (e.g. `mutate()`, `summarize()`) will happen per group
- verbs:
  - `group_by()` : add grouping
  - `ungroup()` : remove grouping
- helper verb:
  - `n()` gives the number of rows in that group

Group-wise analysis

Workflow for group-wise **computations**:

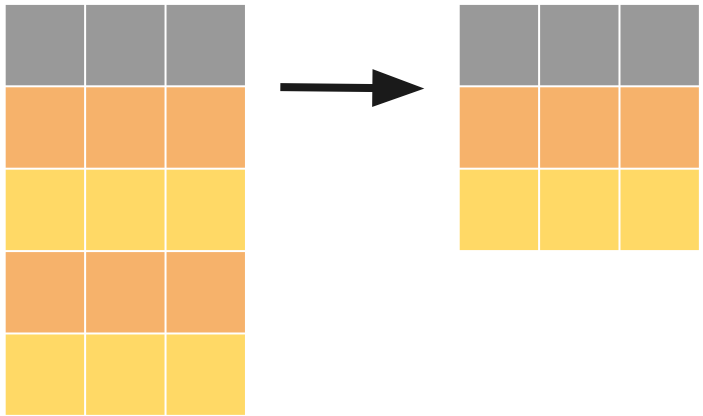
1. `group_by()`
2. `mutate()`
3. `ungroup()`



Group-wise analysis

Workflow for group-wise **summaries**:

1. group_by()
2. summarize()



The pipe operator (%>%)

pass output of LHS as first argument to RHS

`x %>% f(y)` becomes `f(x, y)`

advantages:

- less typing
- less redundancy (easier to change object names)
- more readable code

Exercise chapter 3

1. Import the file "sampledata.tsv" as a tibble.
2. Filter out all rows where the variables nationality or bmi_group are NA. Store the resulting tibble as “sample_data_filtered.tsv”.
3. Make a tibble "sample_data_summary" with a count of participants per nationality - bmi_group combination. Sort the table first by nationality, then inversely by count (largest count first). Start from “sample_data_filtered”.
4. Make a bar plot to inspect whether some nationalities have more obese participants than others.

4. Introduction to tidy data

Untidy data

Try to make a simple ggplot figure:

- day on the x-axis
- give the names a different color

name	day_1	day_2	day_3
edmund	10	11	11
baldrick	19	21	17
percy	3	5	6

Untidy data

Plotting impossible!

Why?

- Turnip count should be one variable, but it is spread over multiple columns
- Day should be a variable, but this information is now in the column headers

name	day_1	day_2	day_3
edmund	10	11	11
baldrick	19	21	17
percy	3	5	6

Tidy data

What changed?

1. The variable “turnip count” is now in one column
2. The variable “day” is now a separate column
3. Values in all other columns are duplicated

name	day	turnips
edmund	day_1	10
edmund	day_2	11
edmund	day_3	11
baldrick	day_1	19
baldrick	day_2	21
baldrick	day_3	17
percy	day_1	3
percy	day_2	5
percy	day_3	6

name	day_1	day_2	day_3
edmund	10	11	11
baldrick	19	21	17
percy	3	5	6

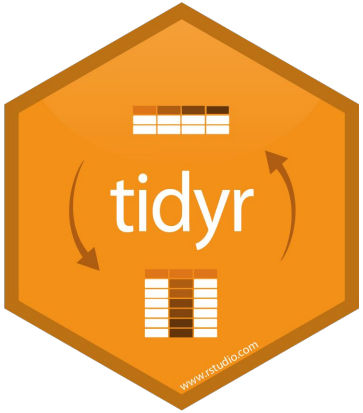
```
data <- gather(data,
  day_1, day_2, day_3,
  value = turnips,
  key = day
)
```

name	day	turnips
edmund	day_1	10
edmund	day_2	11
edmund	day_3	11
baldrick	day_1	19
baldrick	day_2	21
baldrick	day_3	17
percy	day_1	3
percy	day_2	5
percy	day_3	6

Tidy data

	untidy	tidy
rows	contain **observations in different conditions** of variable x	contain **one observation** of variable x
columns	**variable x in multiple columns** ● some columns are different measurement conditions of x ● other columns are group properties	**each column is one variable** ● measurements of x in one column ● measurement conditions of x in one column ● other columns give the group properties of the observation

Tidying verbs



Package “tidyr”

- `gather()`: make table tidy (wide to long)
- `spread()`: make table untidy (long to wide)

Tidying: `gather()`

```
tibble <- gather(tibble, value = “variable”, key = “variable”, var_1, var_2, …)
```

Input:

- tibble
- var_1, var_2, …: variables with observations of x
- value: name of new variable for observations of x
- key: name of new variable containing conditions

Why tidy data?

Easier to create ggplot visualizations

Easier to manipulate (e.g. aggregating levels)

More scalable format (adding more “value” variables possible)

Exercise chapter 4

1. Import the file otutable.tsv as a tibble.
2. Tidy the tibble. The result should be a tibble with three columns: sample, taxon, abundance.
3. Add a fourth column with relative abundances within a sample. Call it “rel_abundance”.
4. Filter the tibble so that only taxa are retained with a mean relative abundance of at least 1%.
5. Make a tile plot to visualize the relative abundances. Put the samples on the x-axis and taxa on the y-axis.

Feedback

Thoughts? Suggestions?

- Contents: quantity, level, …
- Didactics
- Exercises
- ...