UCLA QCBio Collaboratory

# Advanced data visualization with ggplot2

Wenbin Guo
Bioinformatics IDP, UCLA
wbguo@ucla.edu

2024 Spring

UCLA Bioinformatics

# Notation of the slides

- `Code` or `Pseudo-Code` chunk starts with " ➢ ", e.g.
  - ➢ `print("Hello world!")`

- Link is underlined

- Important terminology is in **bold** font

- Practice comes with

# Workshop goals

- Master the syntax and grammar of ggplot2

- Use online resources to to generate publication-quality figures

- Develop a mindset and taste for better data visualizations

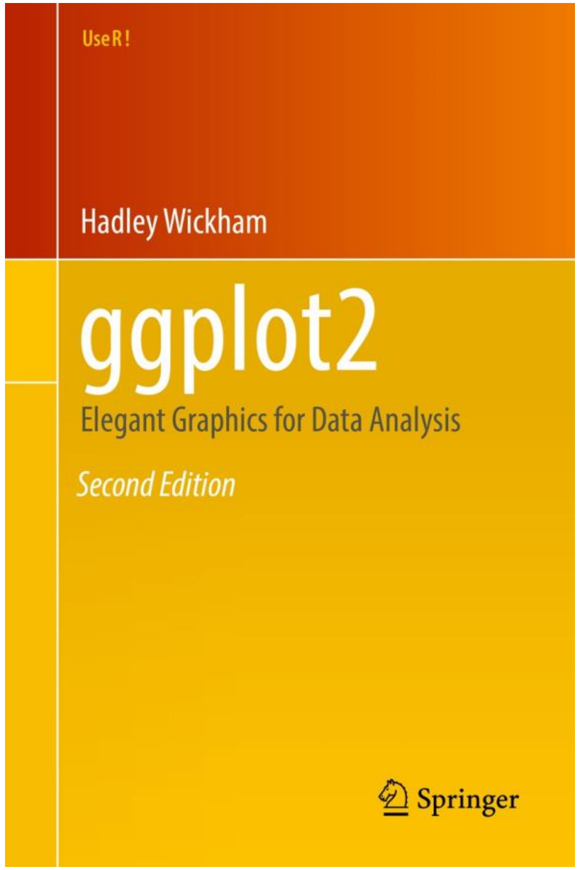
# Agenda

- Day 1: Data visualization basics
  - Getting started with ggplot2
  - Recap of data wrangling functions

- Day 2: Building a plot layer by layer
  - Exploring different plot types
  - Getting more control on the plots

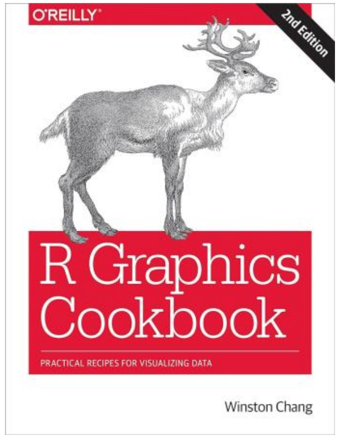
- Day 3: Examples and useful packages
  - Practical examples and principles
  - Introducing some useful packages

# Reference

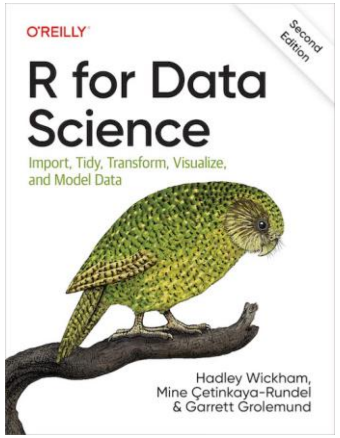



link

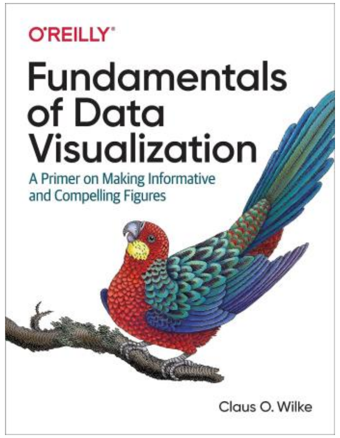
- Other useful references




link




link




link

# The creator

"***for influential work in statistical computing, visualization, graphics, and data analysis***; *for developing and implementing an impressively comprehensive* ***computational infrastructure for data analysis through R*** *software; for making statistical thinking and computing accessible to large audience; and for enhancing an appreciation for the important role of statistics among data scientists."*

Hadley Wickham
Chief Scientist, Rstudio/posit

**—— 2019 COPSS President's Award**



Image from Committee of Presidents of Statistical Societies

# Environment setup

- Go to the official download [website]

posit PRODUCTS SOLUTIONS LEARN & SUPPORT EXPLORE MORE PRICING

DOWNLOAD

## RStudio Desktop

Used by millions of people weekly, the RStudio integrated development environment (IDE) is a set of tools built to help you be more productive with R and Python.

Don't want to download or install anything? Get started with RStudio on Posit Cloud for free. If you're a professional data scientist looking to download RStudio and also need common enterprise features, don't hesitate to book a call with us.

### 1: Install R

RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

DOWNLOAD AND INSTALL R

### 2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR WINDOWS

Size: 214.34 MB | SHA-256: FE62B784 | Version: 2023.09.1+494 | Released: 2023-10-17

# Environment setup

- Go to the official download website

- Install R and RStudio desktop based on your operating system

The Comprehensive R Archive Network

**Download and Install R**

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

**Source Code for all Platforms**

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2023-10-31, Eye Holes) R-4.3.2.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

**Questions About R**

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

# Environment setup

- Go to the official download website

- Install R and RStudio desktop based on your operating system

- Install the necessary package(s) in RStudio Console
  - `install.packages("tidyverse")`

UCLA QCBio Collaboratory

# Day 1: Data visualization basics

Wenbin Guo

Bioinformatics IDP, UCLA

wbguo@ucla.edu

2024 Spring

# Overview

## Time

- 3-hour workshop (45min + 45min + 30min + practice/Q&A)

## Topics

❑ Introduction

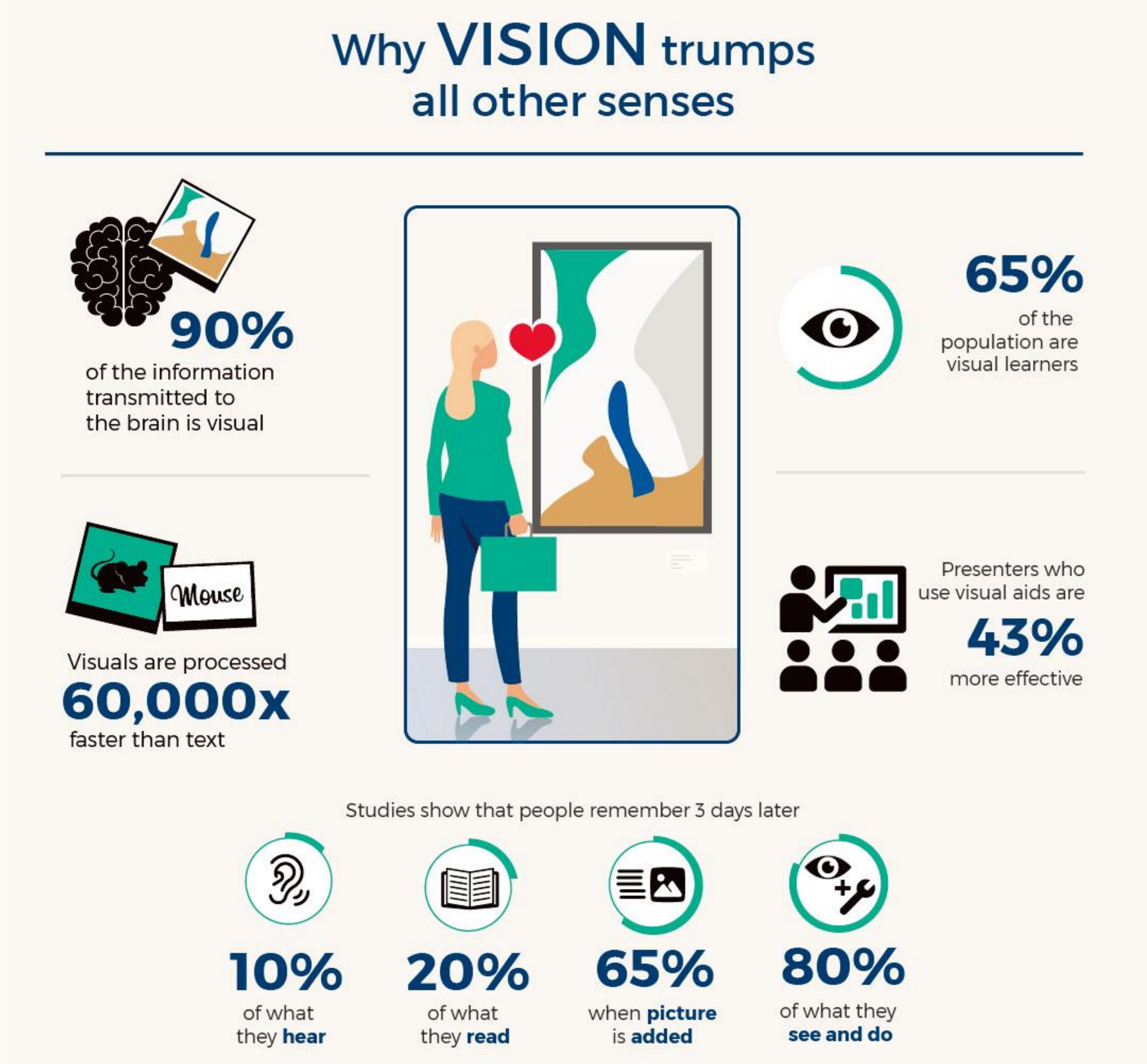
❑ Getting started with `ggplot2`

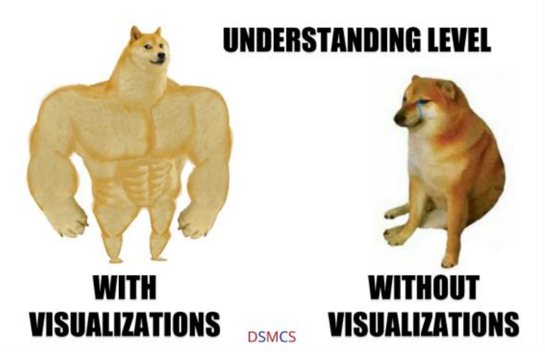
❑ Recap of data wrangling functions

  - Variable types, factors, data frame
  - `dplyr::filter(), select(), mutate(), left_join(), bind_rows() …`
  - `tidyr::pivot_longer(), pivot_wider() …`

❑ Aesthetic mapping

# Facts on how our brain reacts to visuals



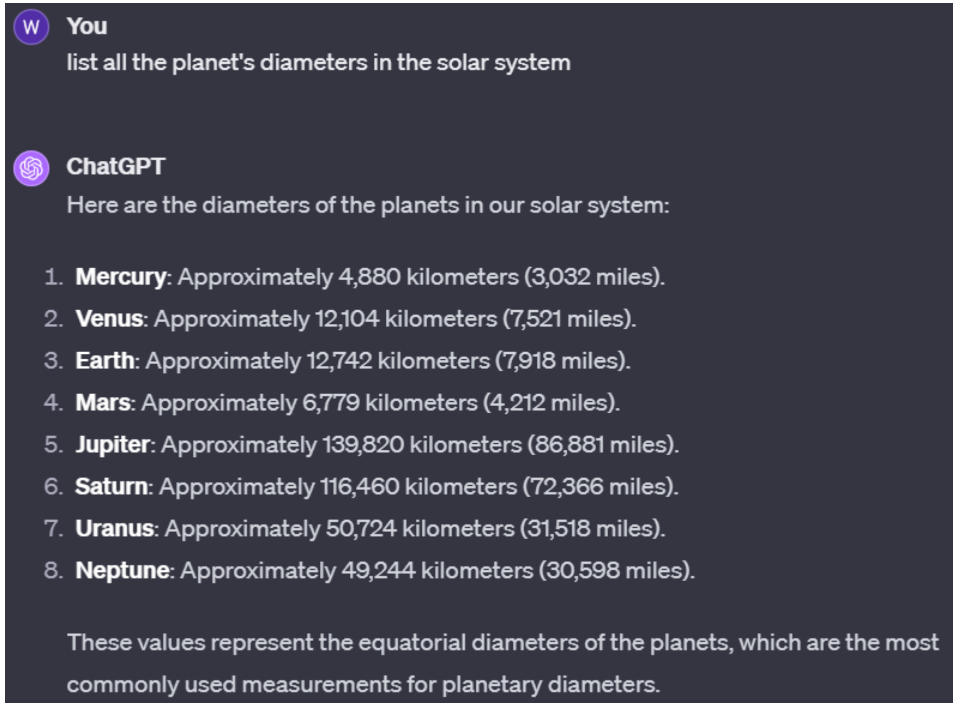



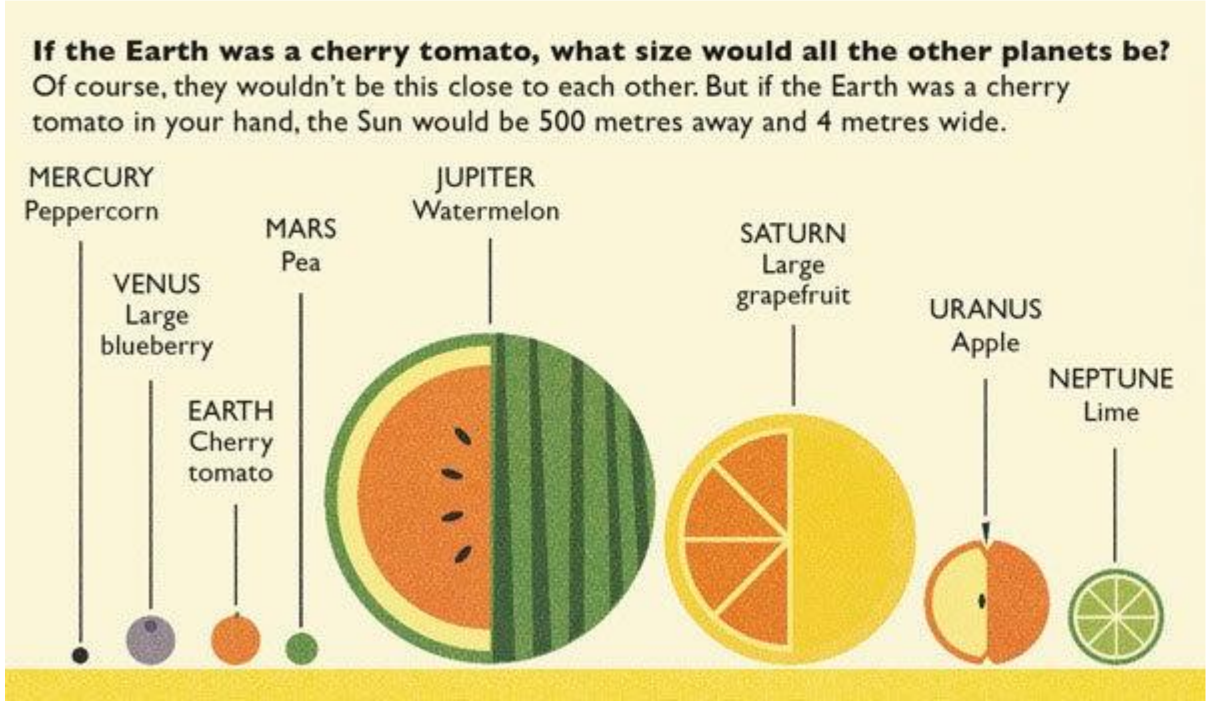



Martin, K. *Medical Writing* (2020)

# "A picture may be worth a thousand words"

## Example




But the visualization can be imprecise (e.g. the distance between planets)



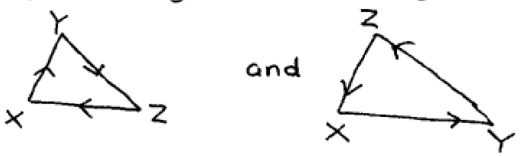
Image from Internet

# "A picture may be worth a thousand words"

<u>Remark</u> Whether the traversal is clockwise, is undefined in the case of the degenerate triangle, i.e. a "triangle" with its vertices on a straight line, but in that case its area equals zero and its sign is irrelevant. Also, please note that the theorems

$$\triangle XYZ = -\triangle ZYX$$
$$\triangle XYZ = \triangle YZX$$

hold independently of the sign of $\triangle XYZ$, i.e., for both




A picture may be worth a thousand words, a formula is worth a thousand pictures. (End of Remark.)

—— *A first exploration of effective reasoning* (1996)

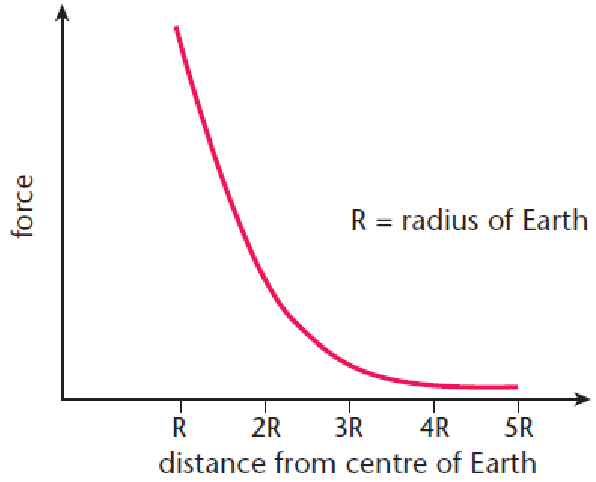
Edsger Dijkstra
(1930-2002)

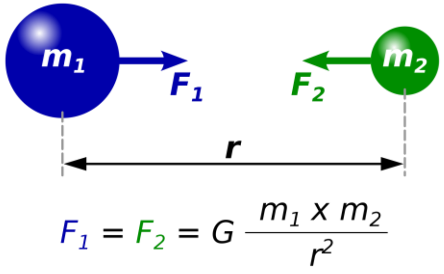
# "A formula is worth a thousand pictures"

A picture may be worth a thousand words,
a formula is worth a thousand pictures.
(End of Remark.)

Edsger Dijkstra
(1930-2002)

Newton's law of gravitation:



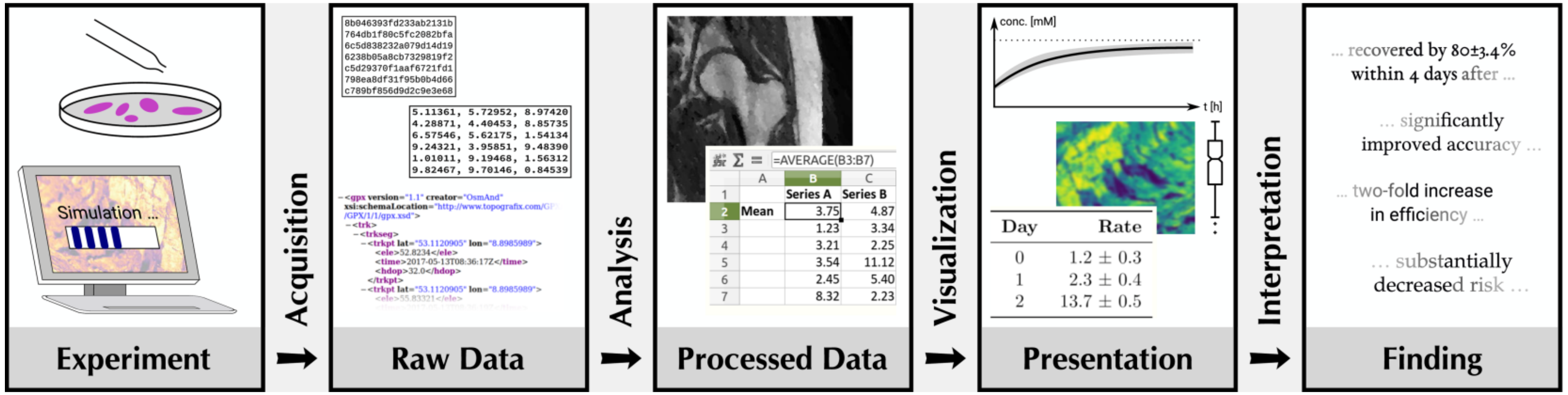




Question: What is worth than a thousand formulas?



Image from Internet

# The role of data visualization in science




**Typical steps in scientific work involving data analysis**

(Schwen et al. *Plos Computational Biology*, 2018)

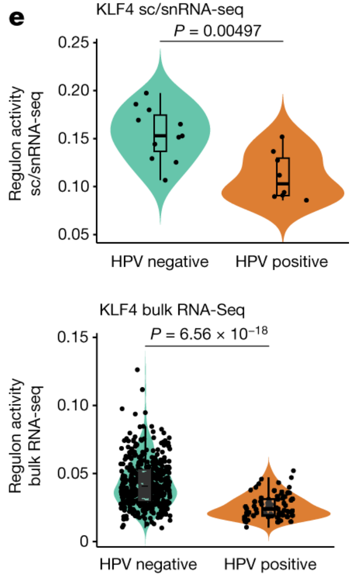

As a scientist, we use data visualization techniques to explore and explain



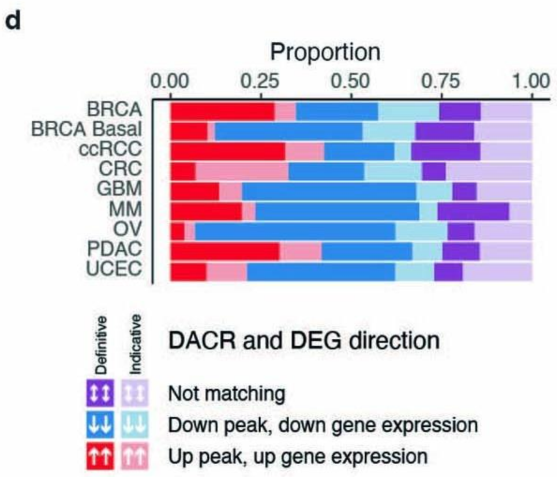
Image from Internet

# The role of data visualization in science (example)

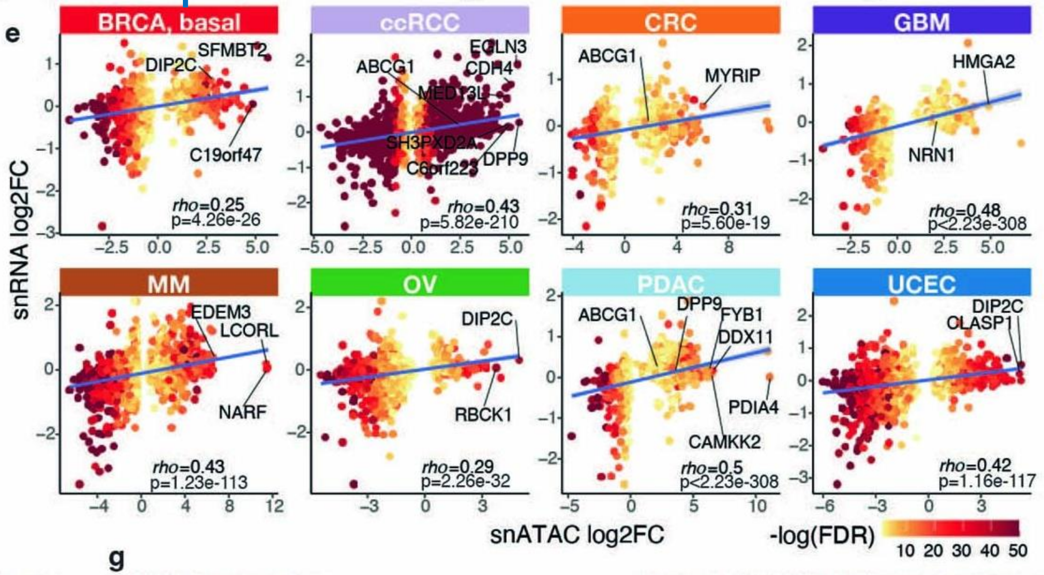
## Boxplot/Violin plot



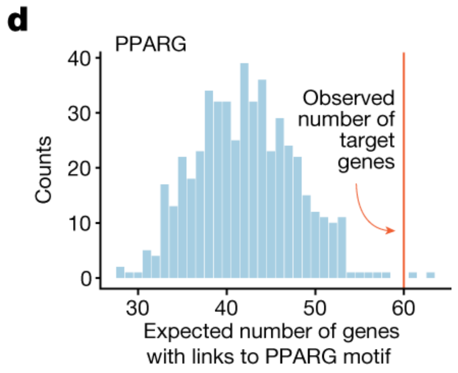

## Bar plot



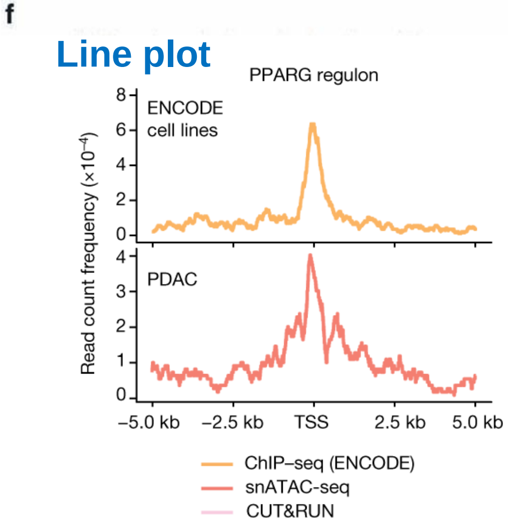

## Scatter plot



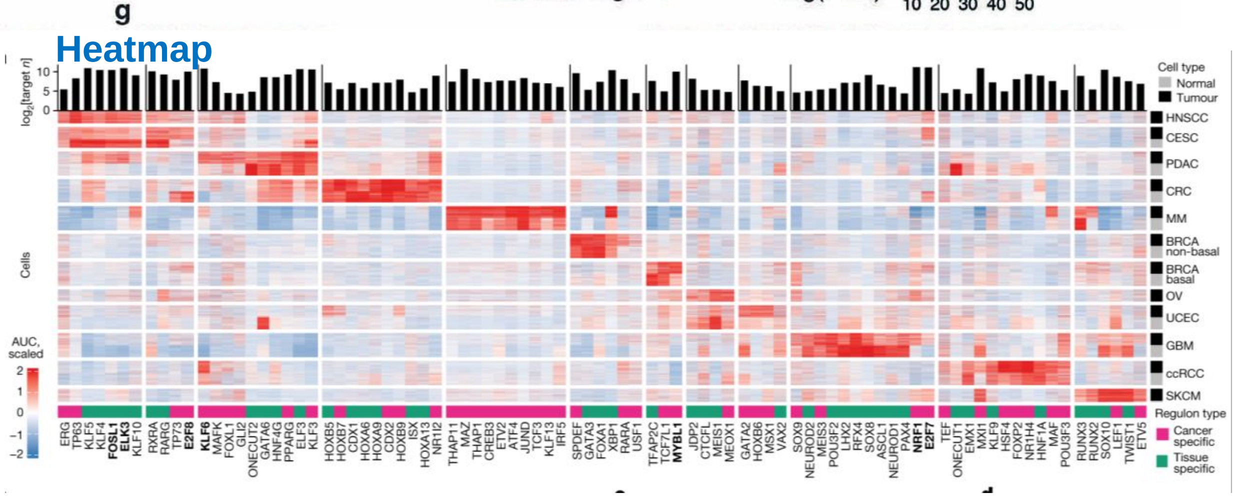

## Histogram




## Line plot




## Heatmap

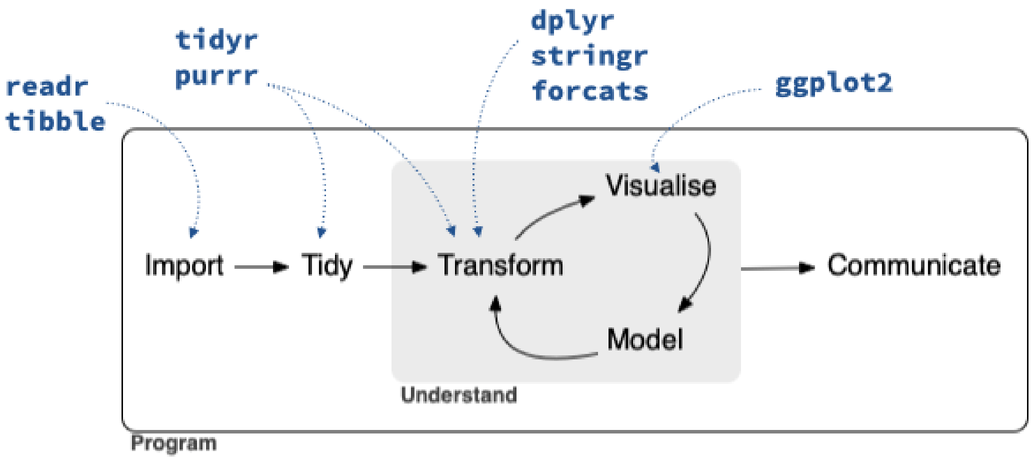



## And more …



Terekhanova, et al. *Nature* (2023)

# ggplot2: a core member of tidyverse




a typical data science project


tidyverse: a collection of R packages for data science



Cetinkaya-Rundel et al. *arXiv* (2021)

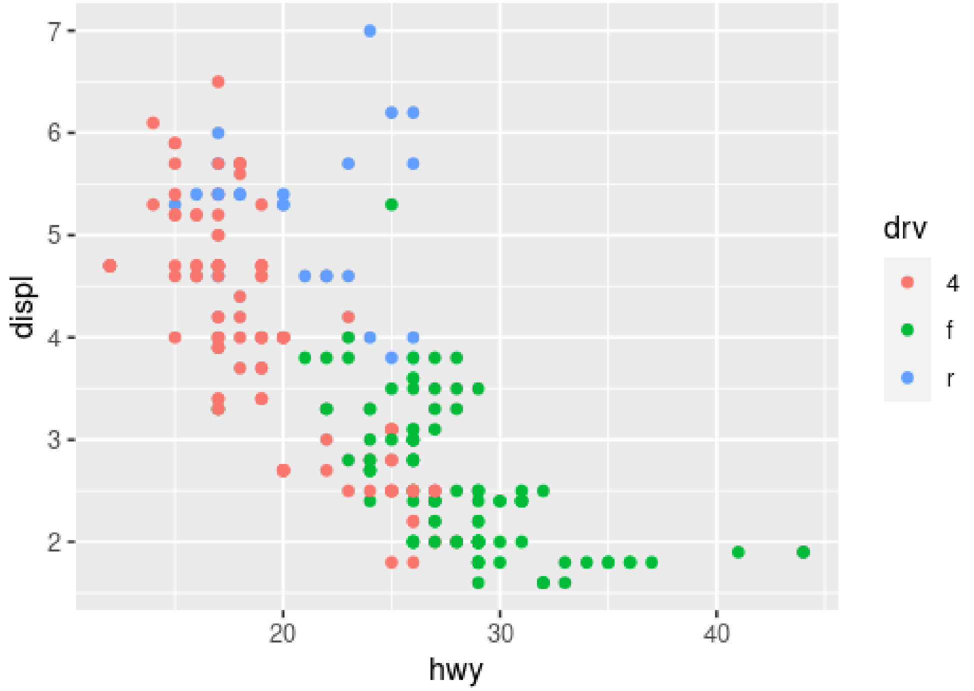
# Idea behind ggplot2 visualization

## Take scatter plot as an example

Filter

| | manufacturer | model | displ | year | cyl | trans | drv | cty | hwy | fl | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | audi | a4 | 1.8 | 1999 | 4 | auto(l5) | f | 18 | 29 | p | compact |
| 2 | audi | a4 | 1.8 | 1999 | 4 | manual(m5) | f | 21 | 29 | p | compact |
| 3 | audi | a4 | 2.0 | 2008 | 4 | manual(m6) | f | 20 | 31 | p | compact |
| 4 | audi | a4 | 2.0 | 2008 | 4 | auto(av) | f | 21 | 30 | p | compact |
| 5 | audi | a4 | 2.8 | 1999 | 6 | auto(l5) | f | 16 | 26 | p | compact |
| 6 | audi | a4 | 2.8 | 1999 | 6 | manual(m5) | f | 18 | 26 | p | compact |
| 7 | audi | a4 | 3.1 | 2008 | 6 | auto(av) | f | 18 | 27 | p | compact |
| 8 | audi | a4 quattro | 1.8 | 1999 | 4 | manual(m5) | 4 | 18 | 26 | p | compact |
| 9 | audi | a4 quattro | 1.8 | 1999 | 4 | auto(l5) | 4 | 16 | 25 | p | compact |
| 10 | audi | a4 quattro | 2.0 | 2008 | 4 | manual(m6) | 4 | 20 | 28 | p | compact |
| 11 | audi | a4 quattro | 2.0 | 2008 | 4 | auto(s6) | 4 | 19 | 27 | p | compact |
| 12 | audi | a4 quattro | 2.8 | 1999 | 6 | auto(l5) | 4 | 15 | 25 | p | compact |
| 13 | audi | a4 quattro | 2.8 | 1999 | 6 | manual(m5) | 4 | 17 | 25 | p | compact |
| 14 | audi | a4 quattro | 3.1 | 2008 | 6 | auto(s6) | 4 | 17 | 25 | p | compact |
| 15 | audi | a4 quattro | 3.1 | 2008 | 6 | manual(m6) | 4 | 15 | 25 | p | compact |

Showing 1 to 16 of 234 entries, 11 total columns






Data source: mpg

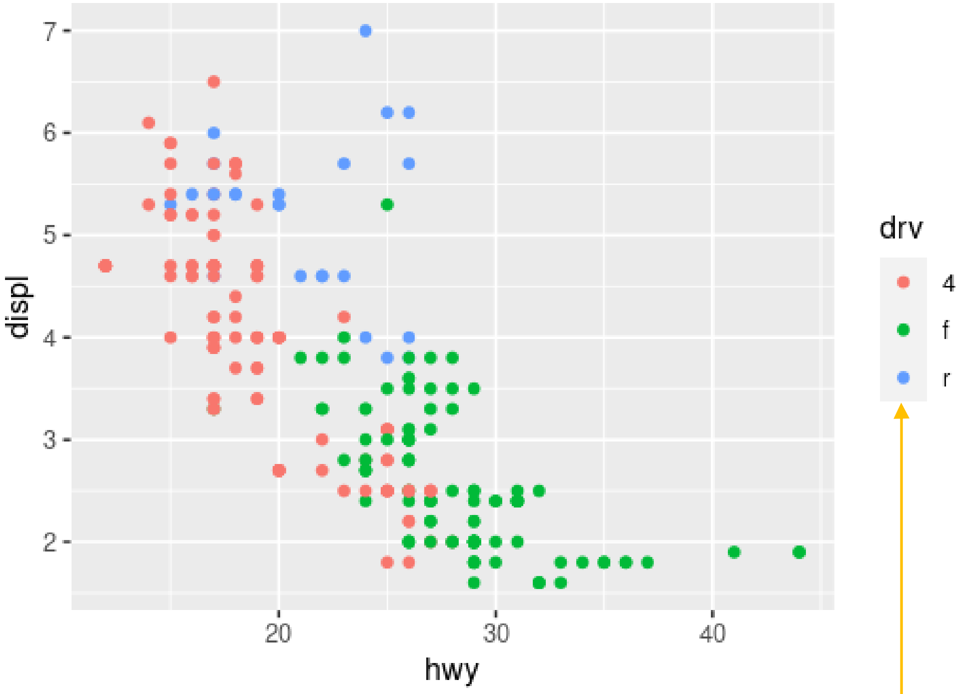
# Idea behind ggplot2 visualization

## Take scatter plot as an example

hwy → x
displ → y
drv → colour

| | manufacturer | model | displ | year | cyl | trans | drv | cty | hwy | fl | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | audi | a4 | 1.8 | 1999 | 4 | auto(l5) | f | 18 | 29 | p | compact |
| 2 | audi | a4 | 1.8 | 1999 | 4 | manual(m5) | f | 21 | 29 | p | compact |
| 3 | audi | a4 | 2.0 | 2008 | 4 | manual(m6) | f | 20 | 31 | p | compact |
| 4 | audi | a4 | 2.0 | 2008 | 4 | auto(av) | f | 21 | 30 | p | compact |
| 5 | audi | a4 | 2.8 | 1999 | 6 | auto(l5) | f | 16 | 26 | p | compact |
| 6 | audi | a4 | 2.8 | 1999 | 6 | manual(m5) | f | 18 | 26 | p | compact |
| 7 | audi | a4 | 3.1 | 2008 | 6 | auto(av) | f | 18 | 27 | p | compact |
| 8 | audi | a4 quattro | 1.8 | 1999 | 4 | manual(m5) | 4 | 18 | 26 | p | compact |
| 9 | audi | a4 quattro | 1.8 | 1999 | 4 | auto(l5) | 4 | 16 | 25 | p | compact |
| 10 | audi | a4 quattro | 2.0 | 2008 | 4 | manual(m6) | 4 | 20 | 28 | p | compact |
| 11 | audi | a4 quattro | 2.0 | 2008 | 4 | auto(s6) | 4 | 19 | 27 | p | compact |
| 12 | audi | a4 quattro | 2.8 | 1999 | 6 | auto(l5) | 4 | 15 | 25 | p | compact |
| 13 | audi | a4 quattro | 2.8 | 1999 | 6 | manual(m5) | 4 | 17 | 25 | p | compact |
| 14 | audi | a4 quattro | 3.1 | 2008 | 6 | auto(s6) | 4 | 17 | 25 | p | compact |
| 15 | audi | a4 quattro | 3.1 | 2008 | 6 | manual(m6) | 4 | 15 | 25 | p | compact |

Showing 1 to 16 of 234 entries, 11 total columns






Data source: mpg

# Create a ggplot object

## Create the initial plot object

- `ggplot(data = NULL, mapping = aes(), ...)`
  - `data`: dataset to use for plot
  - `mapping`: list of aesthetic mappings to use for plot
  - `...` : other arguments

- The `data =` and `mapping =` specifications are optional, so long as the data and mapping are passed into the function in the right order

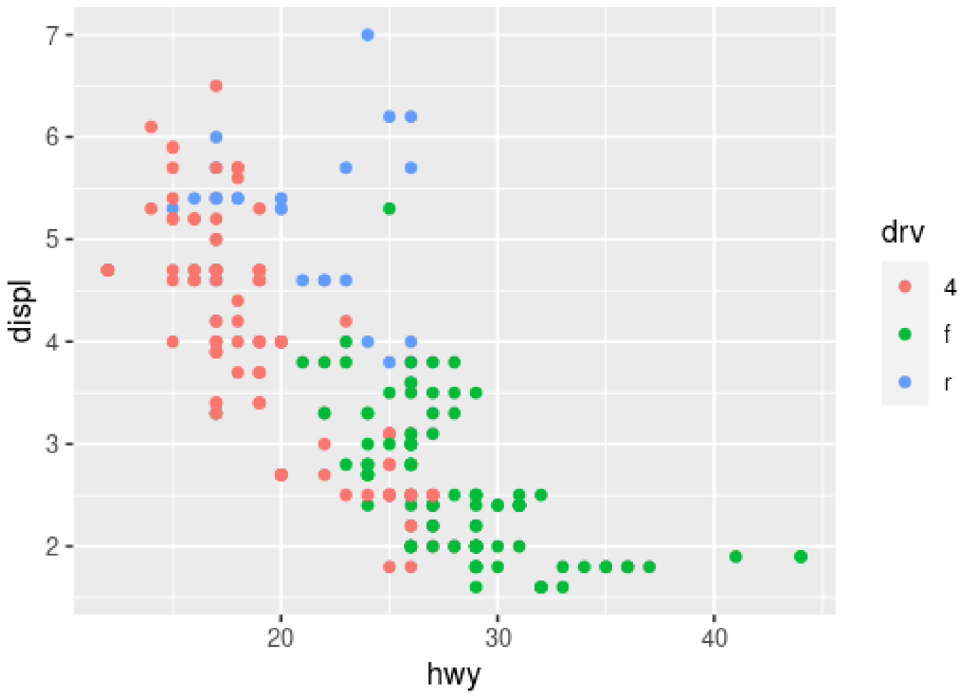
- `ggplot()` is usually followed by a plus sign (+) to add additional components/**layers** to the plot



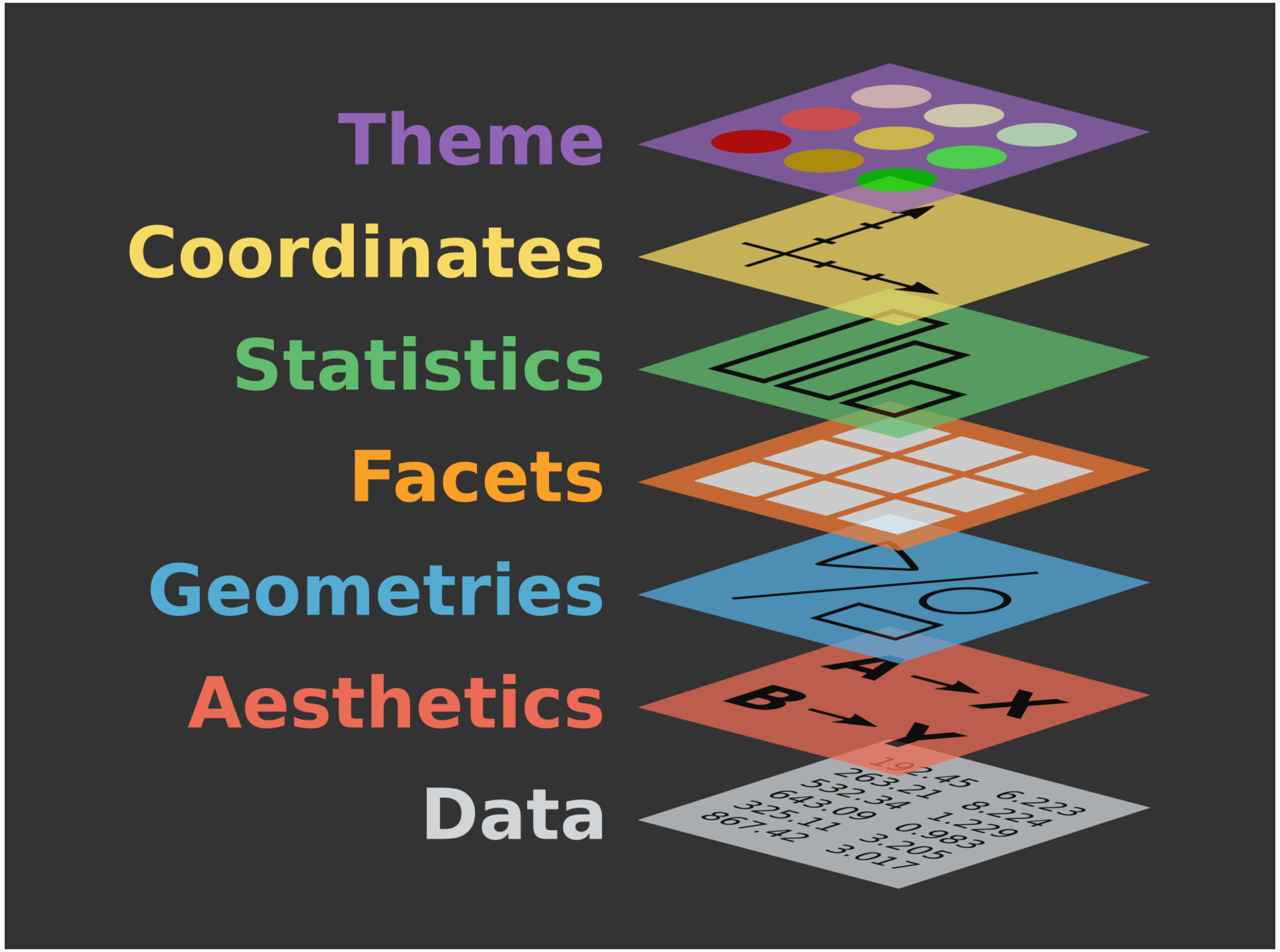
https://ggplot2.tidyverse.org/reference/ggplot.html

# Key components of a ggplot object

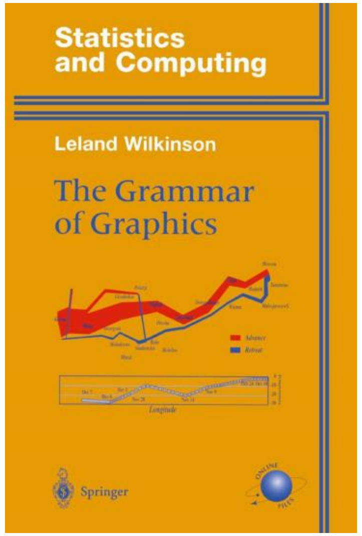
- The **data**
- A set of **aesthetic mappings** between *variables* and *visual properties*
- At least one **layer** describing how to render the observations



# The layered grammar of graphics

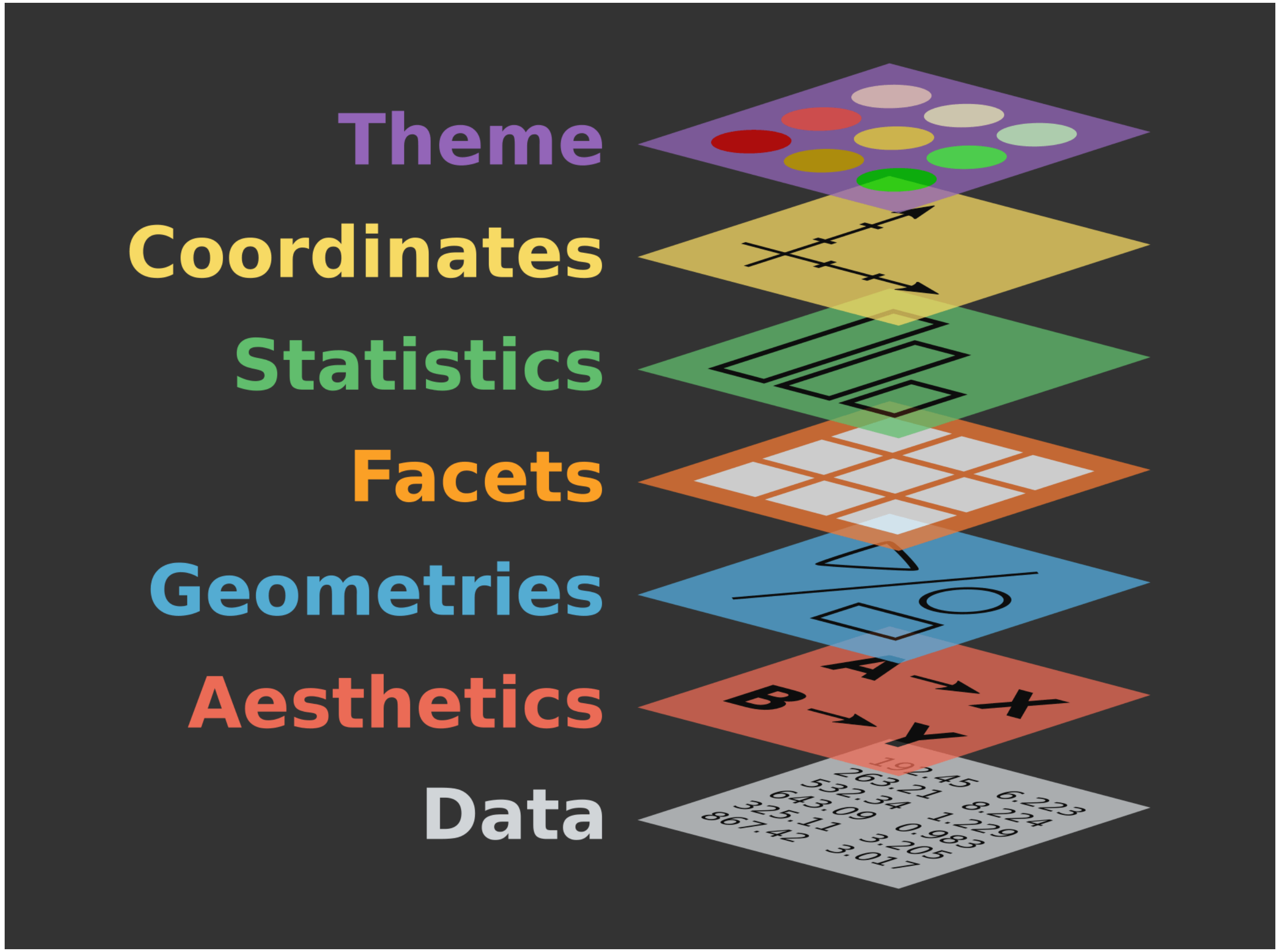



Leland Wilkinson
(1944-2021)




*The Grammar of Graphics* (1999)

# The layered grammar of graphics




Describes all the non-data ink

Plotting space for the data

Statistical models and summaries

Rows and columns of sub-plots

Shapes used to represent the data

Scales onto which data is mapped

The actual variables to be plotted

# Components of the layered grammar

- A default dataset and set of mappings from variables to aesthetics

- One or more layers, each composed of
  - a geometric object (visual object in the plot)
  - a statistical transformation
  - a position adjustment
  - optionally, a dataset and aesthetic mappings

```
p + layer(
  mapping = NULL,
  data = NULL,
  geom = "point",
  stat = "identity",
  position = "identity"
)
```

- ➢ `p + geom_point()`

- One scale for each aesthetic mapping
  - map values in the data space to values in the aesthetic space

- A coordinate system
  - map data coordinates to the plane of the graphic

- The faceting specification
  - specifies how to break up and display subsets of data

# 3 ways to invoke `ggplot()`

- ➢ `ggplot(data = df, mapping = aes(), ...)`
  - when all layers use the same data and the same set of aesthetics

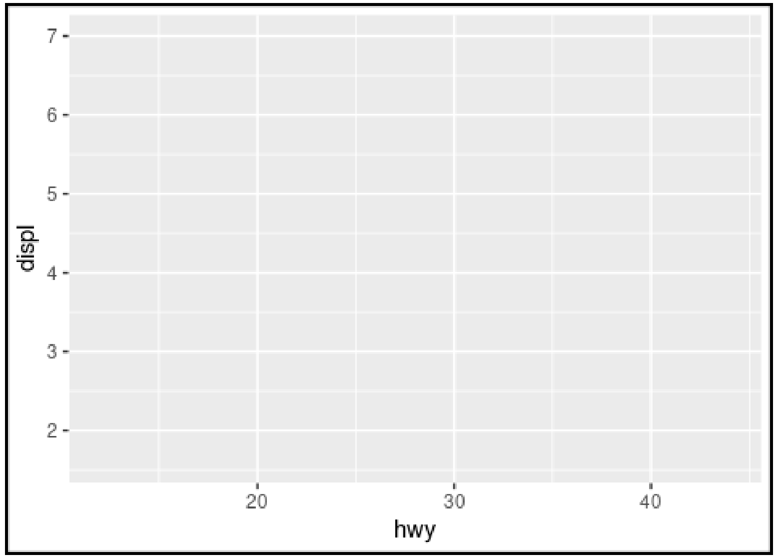
- ➢ `ggplot(data = df)`
  - when layers use the same data, but use different aesthetics

- ➢ `ggplot()`
  - when multiple data frames are used to produce different layers



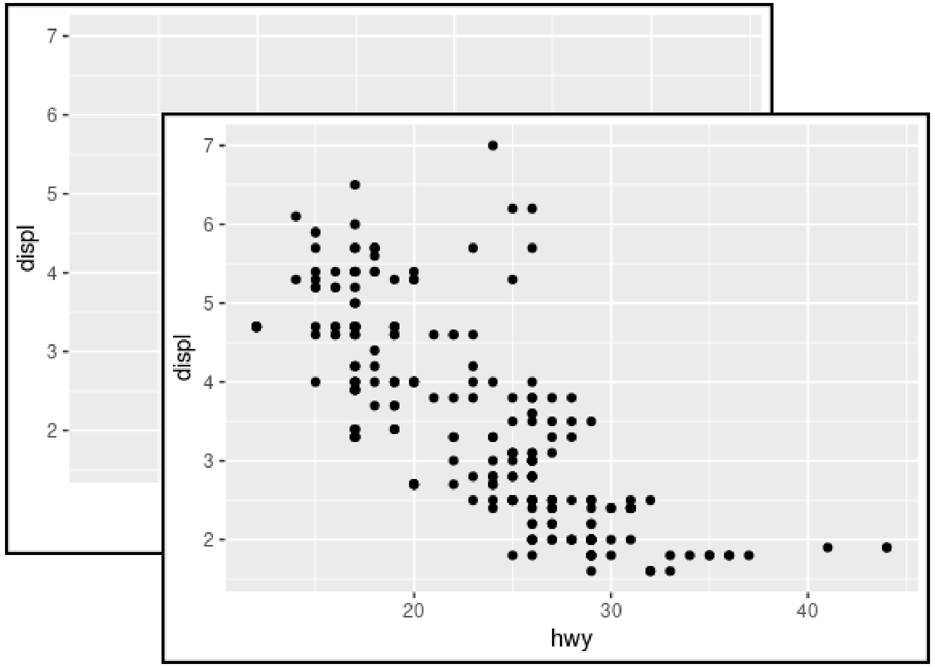
https://ggplot2.tidyverse.org/reference/ggplot.html

# Building plots layer by layer (example)



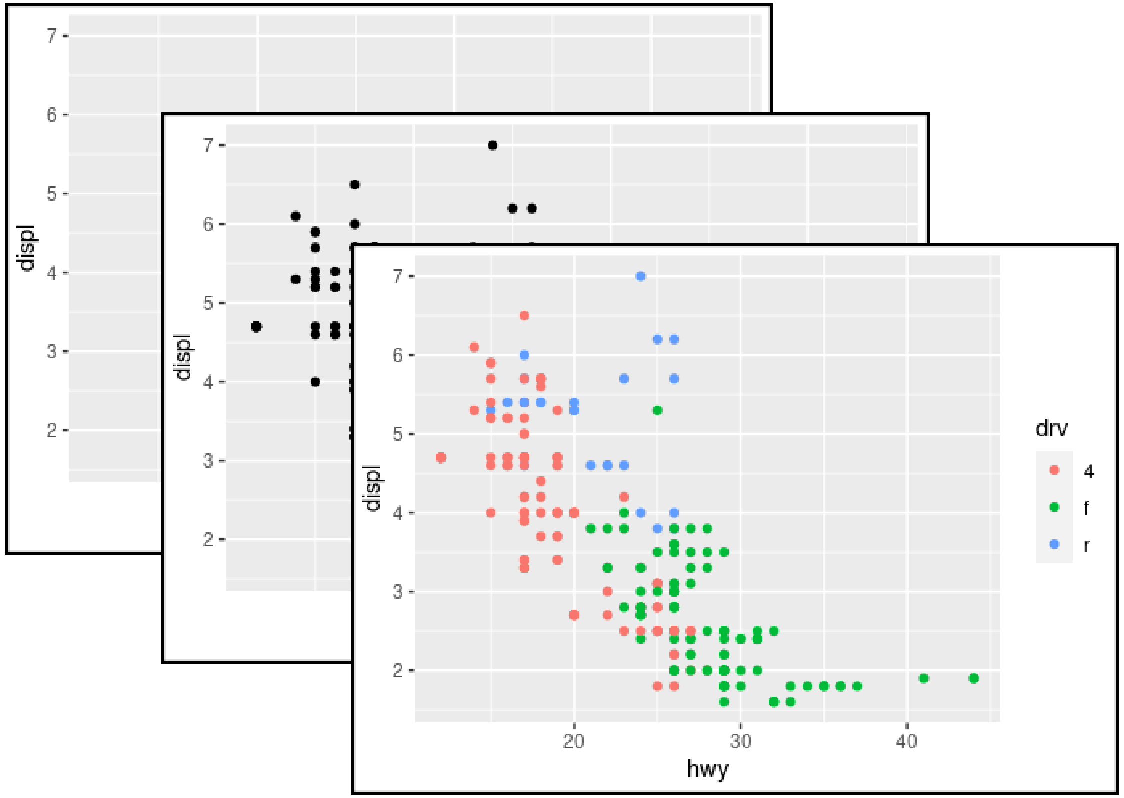

```
base = ggplot(data=mpg, aes(x=hwy, y=displ))
base
```

# Building plots layer by layer (example)



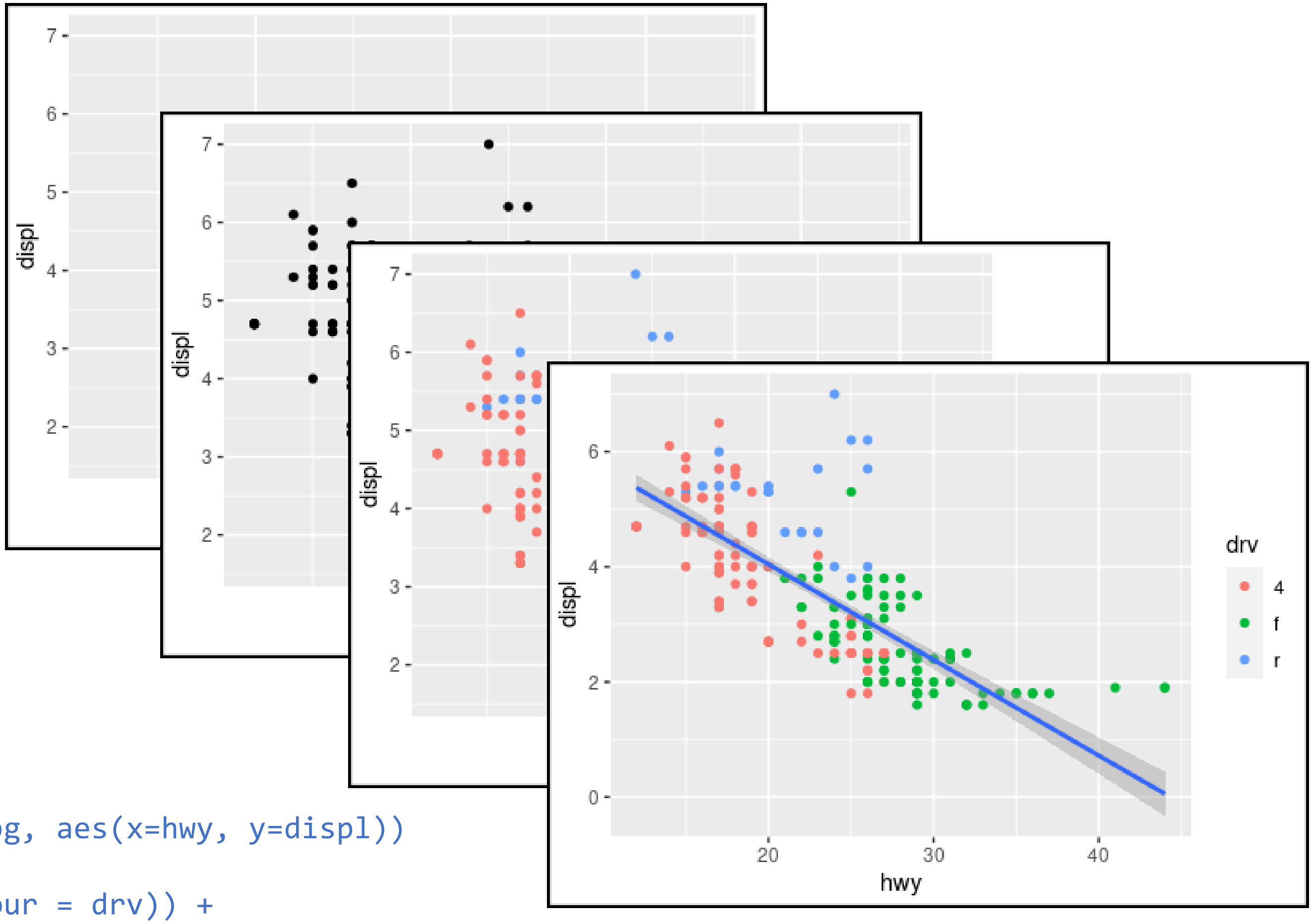

```
base = ggplot(data=mpg, aes(x=hwy, y=displ))
base +
  geom_point()
```

# Building plots layer by layer (example)



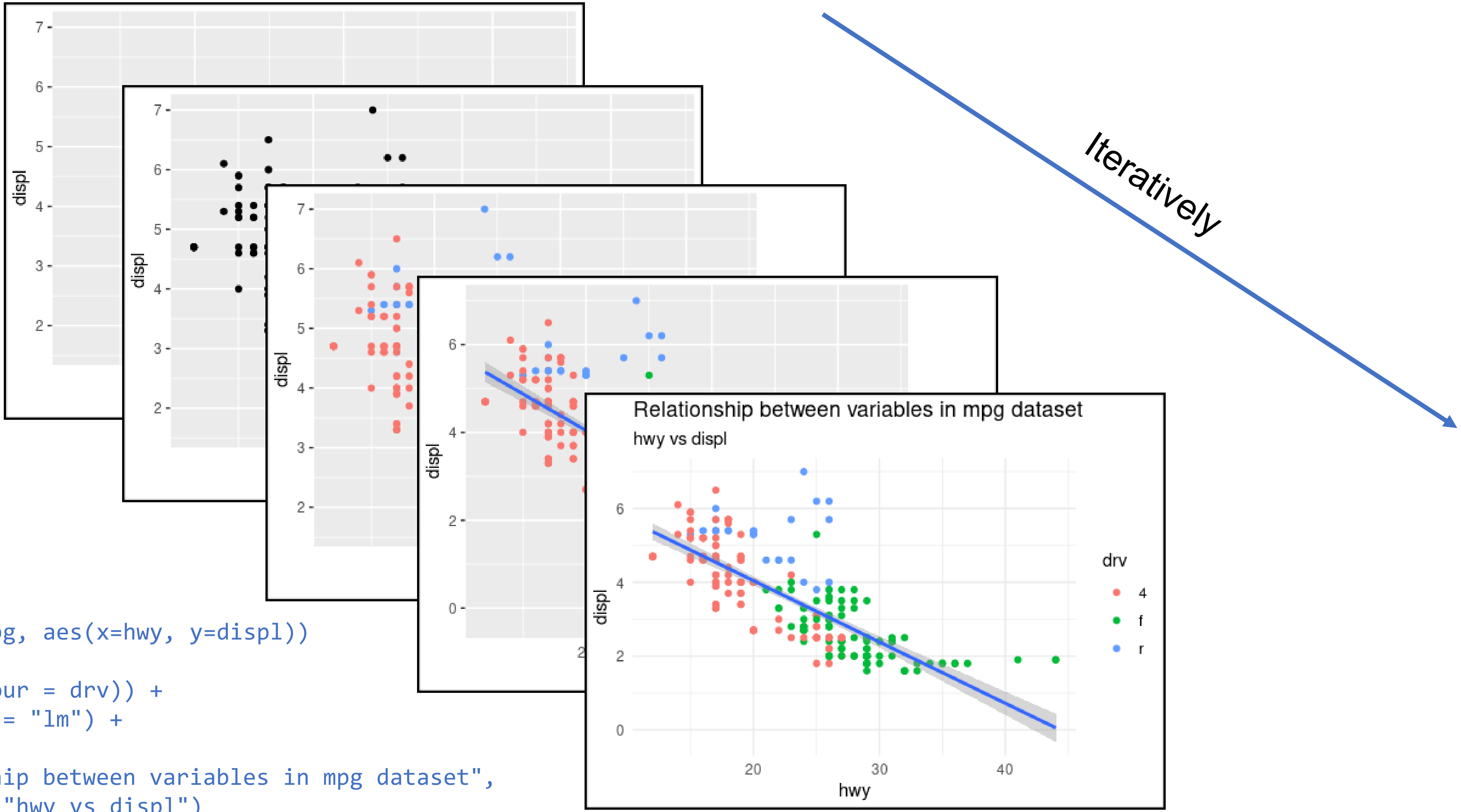

```
base = ggplot(data=mpg, aes(x=hwy, y=displ))
base +
  geom_point(aes(colour = drv))
```

# Building plots layer by layer (example)




```
base = ggplot(data=mpg, aes(x=hwy, y=displ))
base +
  geom_point(aes(colour = drv)) +
  geom_smooth(method = "lm")
```

# Building plots layer by layer (example)

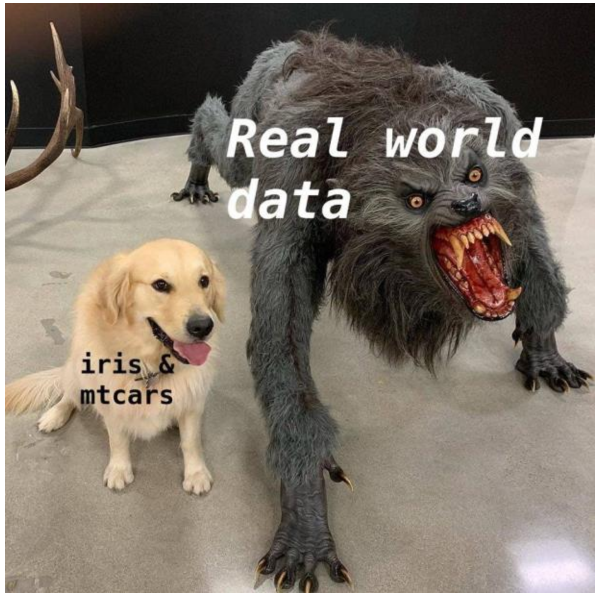



```
base = ggplot(data=mpg, aes(x=hwy, y=displ))
base +
  geom_point(aes(colour = drv)) +
  geom_smooth(method = "lm") +
  theme_minimal() +
  ggtitle("Relationship between variables in mpg dataset",
          subtitle = "hwy vs displ")
```

# Let’s do some practice!

- `git clone https://github.com/wbvguo/qcbio-DataViz_w_ggplot2.git`

# Data




"Once you have

- ❑ the right data,
- ❑ in the right format,
- ❑ aggregated in the right way,

the right visualization is often obvious"

# Variables

The container for storing values

- **Categorical variables**: take discrete values
  - ➢ `x = c("apple", "banana")` # nominal variables: without an order
  - ➢ `y = c("low", "medium", "high")`# ordinal variables: with an order
- **Continuous variables**: take any values within a range
  - ➢ `z = c(0.05, 1, -2)`

# Variables

The container for storing values

- **Categorical variables**: take discrete values
  - `x = c("apple", "banana")`  # nominal variables: without an order
  - `y = c("low", "medium", "high")`# ordinal variables: with an order
- **Continuous variables**: take any values within a range
  - `z = c(0.05, 1, -2)`

**Factors**: takes the categorical variable and stores data in levels

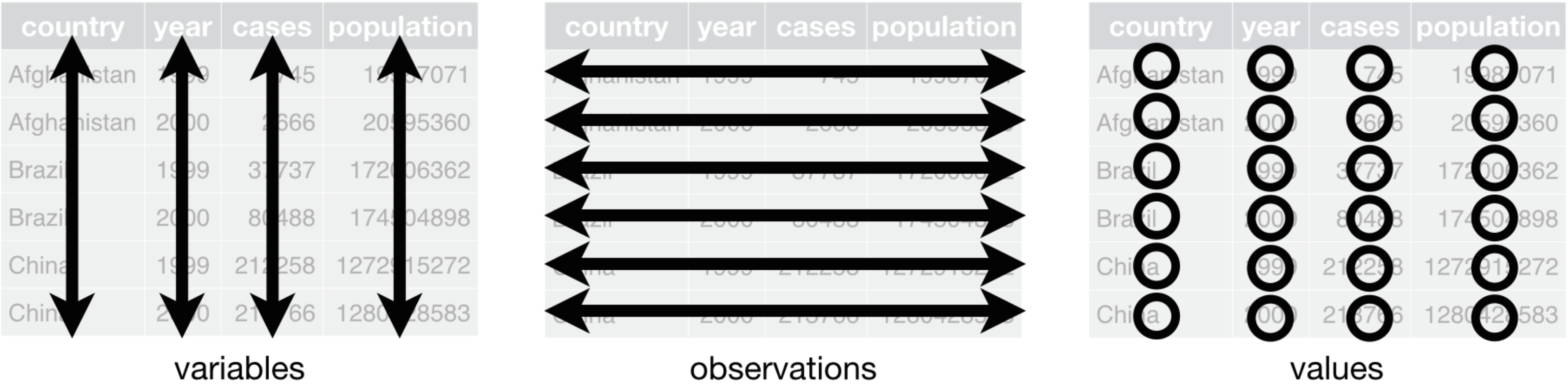
- Use function `factor()` to convert categorical (nominal) and ordered categorical (ordinal) variables to factors

Exercise: compare the following 2 lines' results, what do you find?

- `y1 = factor(y)`
- `y2 = factor(y, levels = c("low", "medium", "high"))`

# Data frame

A generic data object that are used to store tabular data




Use function `data.frame()` to create a data frame

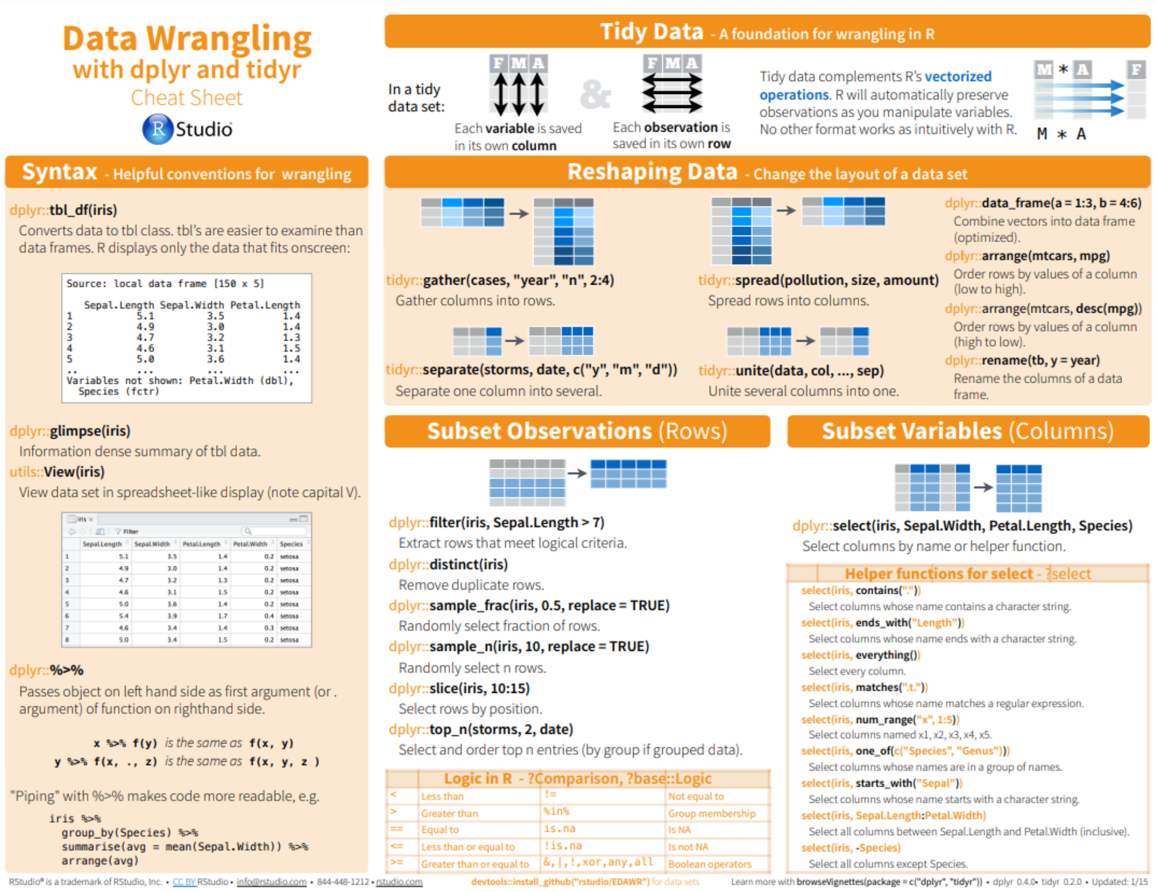
➢ `df = data.frame(x = c(1,2,3), y = c("low", "medium", "high"))`



https://ismayc.github.io/moderndiver-book/4-tidy.html

# Data wrangling functions

- Manipulate observations (rows)
  - `filter()`
  - `arrange()`
  - `bind_rows()`
- Manipulate variables (columns)
  - `select()`
  - `mutate()`
  - `left_join(), right_join() …`
- Reshape the data
  - `pivot_longer(), pivot_wider()`
- Group and summarize
  - `group_by()`
  - `summarize()`



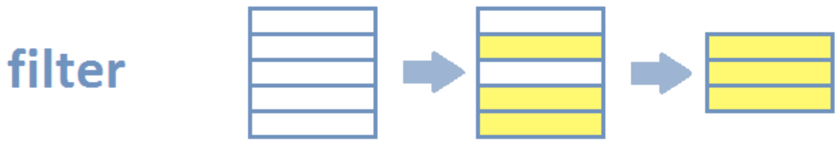

For more information, check the [cheatsheet]()

# Manipulate observations

## `filter()`: keep rows that satisfy certain conditions

- The first argument is a data frame
- The second and subsequent arguments must be logical vectors

## Create logical vectors:




### ❑ Comparison operators

- `x == y`: x and y are equal.
- `x != y`: x and y are not equal.
- `x %in% c("a", "b", "c")`: x is one of the values in the right hand side.
- `x > y`, `x >= y`, `x < y`, `x <= y`: greater than, greater than or equal to, less than, less than or equal to.

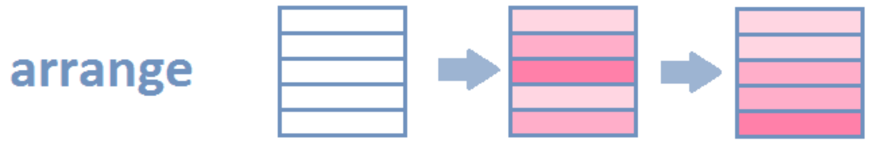
### ❑ Logical operators

- `!x` (pronounced "not x"), flips `TRUE` and `FALSE` so it keeps all the values where x is `FALSE`.
- `x & y`: `TRUE` if both x and y are `TRUE`.
- `x | y`: `TRUE` if either x or y (or both) are `TRUE`.
- `xor(x, y)`: `TRUE` if either x or y are `TRUE`, but not both (exclusive or).

# Manipulate observations

`arrange()`: orders observations according to variables

- The first argument is a data frame
- The second and subsequent arguments are variables or function of variables
- `.by_group`: If `TRUE`, will sort first by grouping variable. Applies to grouped data frames only



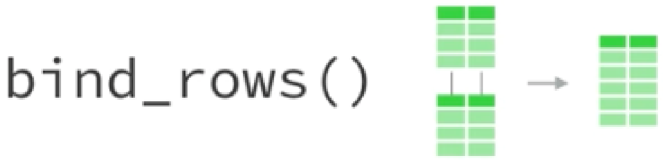

Note:

- the default sorting order is ascending
- use `desc()` to sort a variable in descending order

# Manipulate observations

`bind_rows()`: Bind any number of data frames by row

- The first and subsequent arguments are data frames to combine
- Columns are matched by name, and missing columns will be filled with `NA`

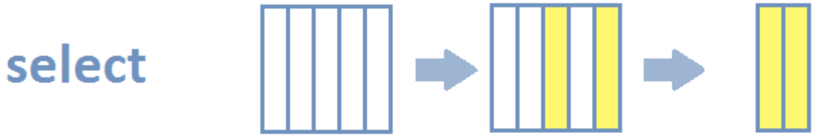



Exercise: let’s do some practice

# Manipulate variables

## `select()`: keep or drop variables using their names and types

- The first argument is a data frame
- The second and subsequent arguments are unquoted expressions separated by comma




### Useful functions

- `all_of()`: Matches variable names in a character vector
- `starts_with()/ends_with()`: Starts/ends with a substring
- `where()`: Applies a function to all variables and selects those for which the function returns TRUE

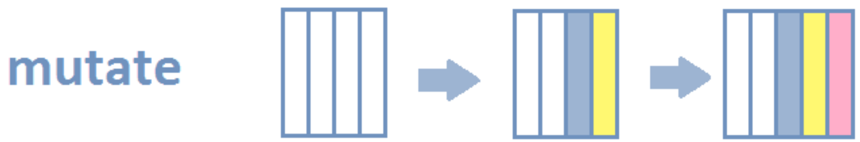
### Useful Operators

- `!`: take the complement of a set of variables
- `&` or `|`: select the intersection or union of two sets of variables
- `c()` : combine selections

# Manipulate variables

## `mutate()`: create new variables

- The first argument is a data frame.
- The second and subsequent arguments are name-value pairs (named expression that generate the new variables)

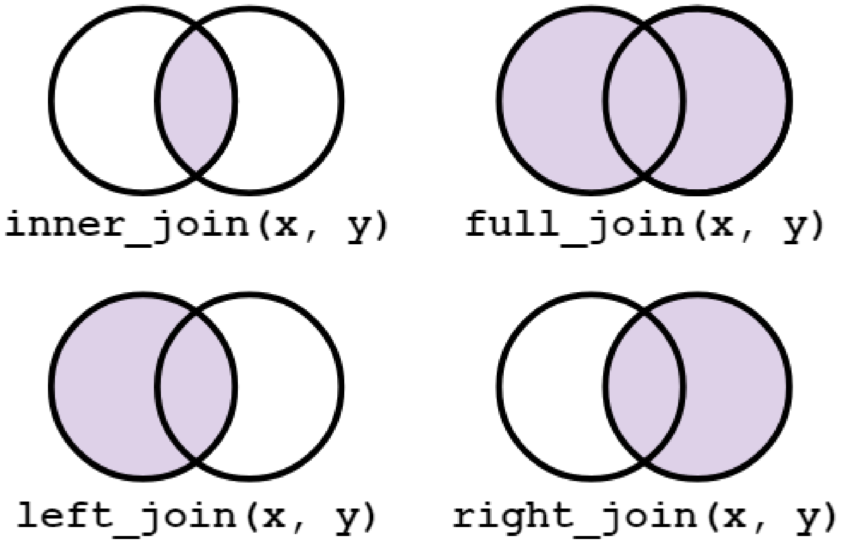



## The values can be

- Vector of length 1
- Vector of the same length as whole data frame or current group (for grouped data frame)
- `NULL` to remove the column

# Manipulate variables

`*_join()`:



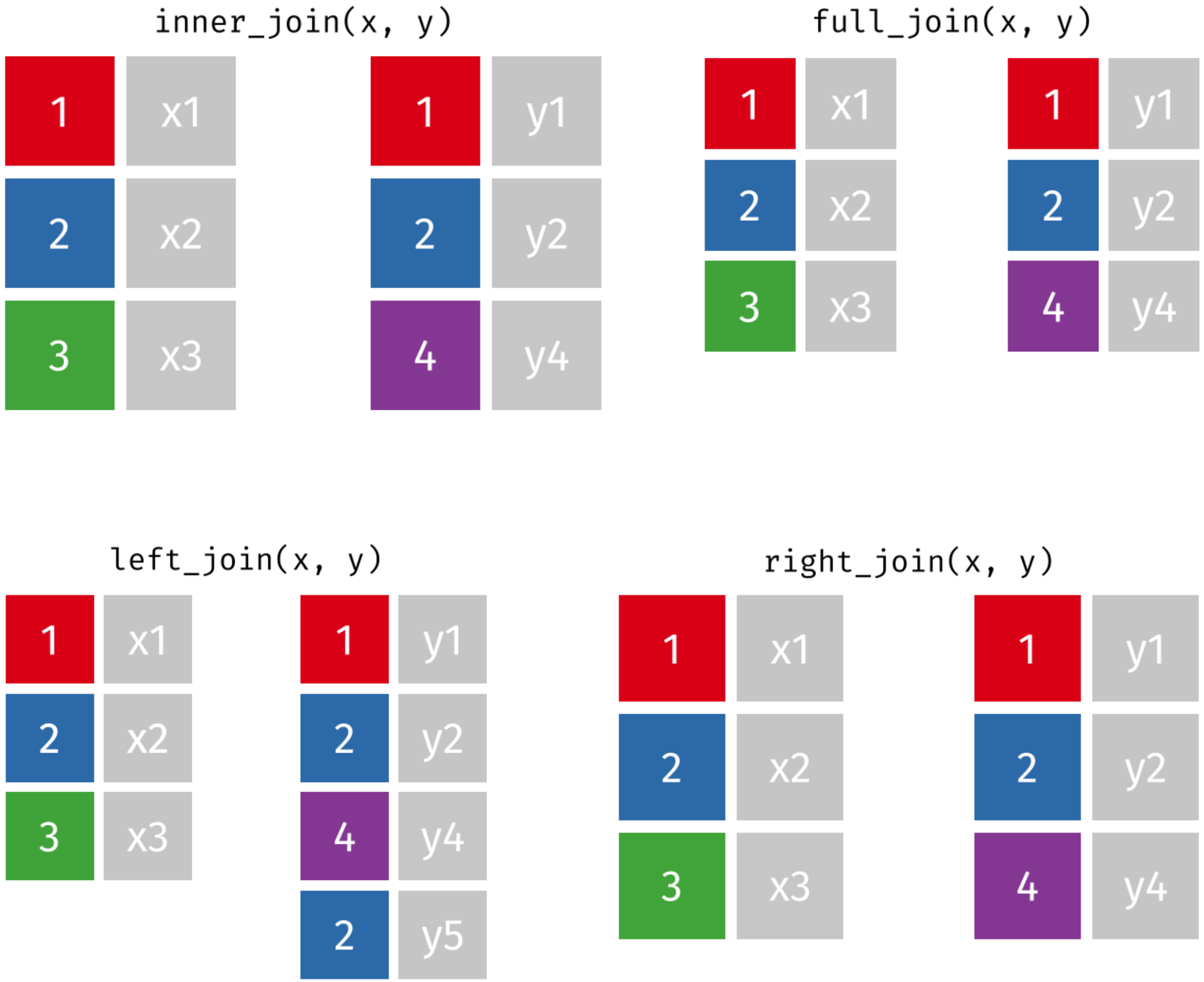
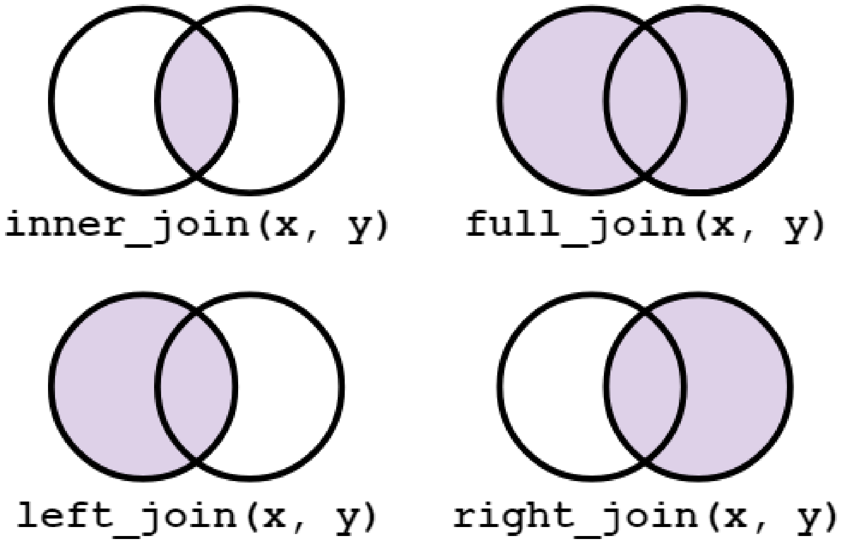



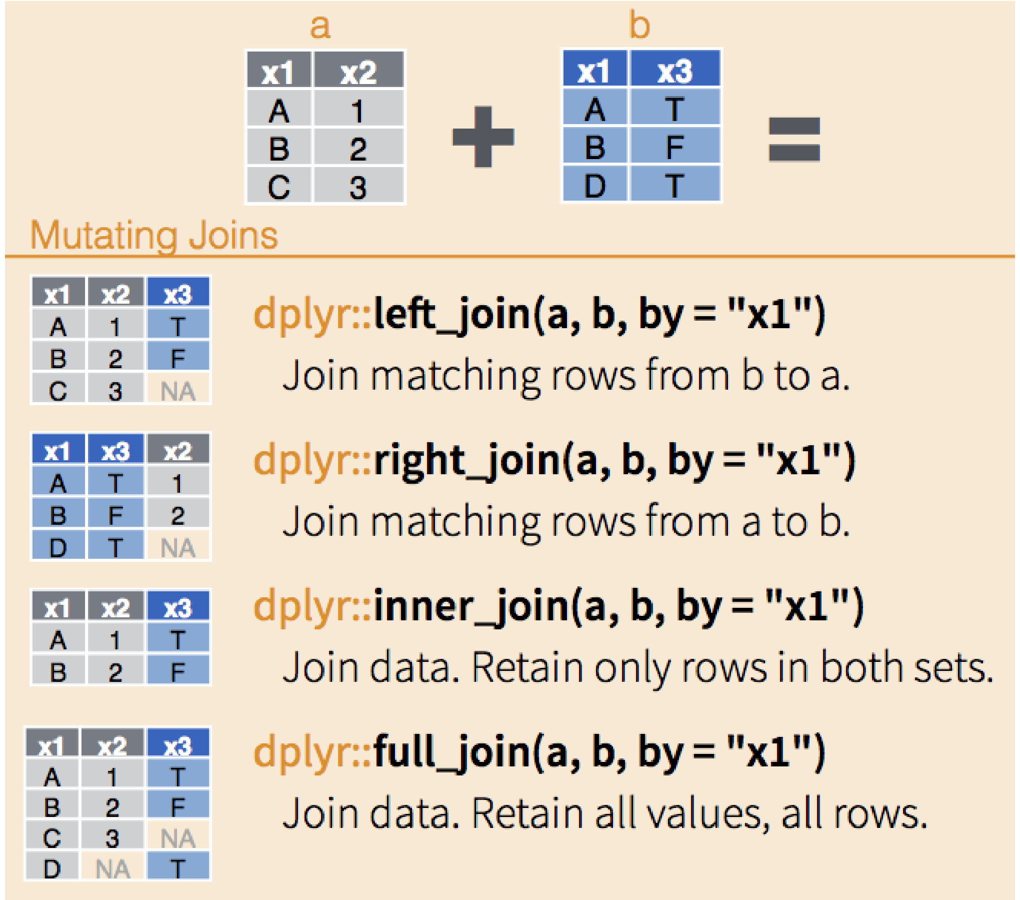



Image from internet

# Manipulate variables

*_join():




Exercise: let's do some practice

Image from internet

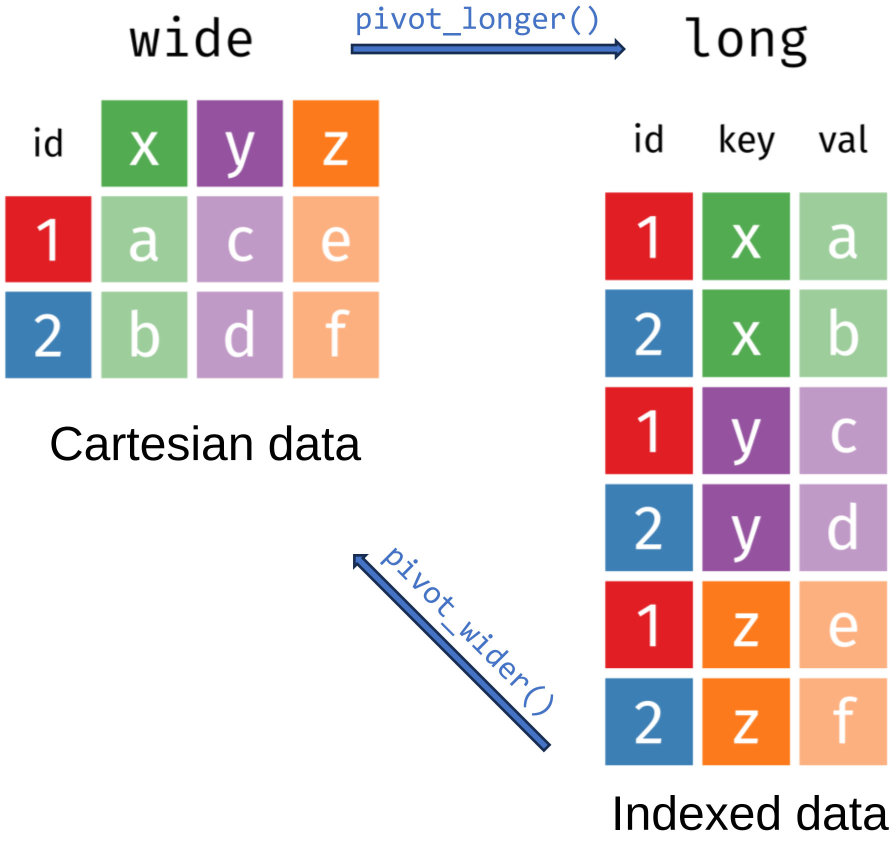
# Reshape data frame

## `pivot_longer()`: pivot into long format

- The first argument is a data frame
- The second argument is columns to pivot into longer format
- `names_to:` new column name for column names
- `values_to:` new column name for cell values

## `pivot_wider()`: pivot into wide format

- The first argument is a data frame
- `names_from:` column to get the names of output column
- `values_from:` column to get the cell values



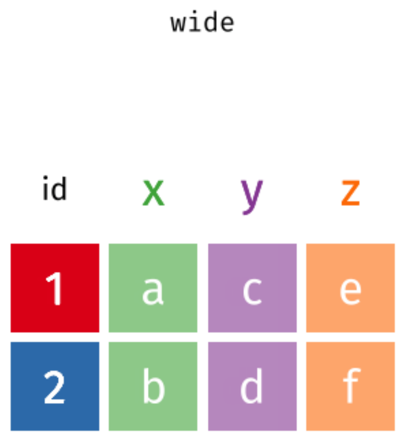



https://bookdown.org/Maxine/r4ds/pivoting.html

# Reshape data frame

`pivot_longer()`: pivot into long format

- The first argument is a data frame
- The second argument is columns to pivot into longer format
- `names_to:` new column name for column names
- `values_to:` new column name for cell values




`pivot_wider()`: pivot into wide format

- The first argument is a data frame
- `names_from:` column to get the names of output column
- `values_from:` column to get the cell values

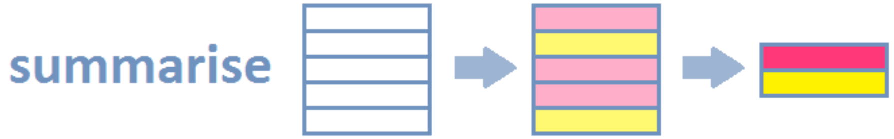


Image from internet

# Group and summarize

## `group_by()`: Define the grouping variables

- The first argument is a data frame
- The second and subsequent arguments are variables used for grouping




## `summarise()`/`summarize()`:

- The first argument is a data frame
- The second and subsequent arguments are name-value pairs for summary function
  - Counts: `n()`, `n_distinct(x)`.
  - Middle: `mean(x)`, `median(x)`.
  - Spread: `sd(x)`, `mad(x)`, `IQR(x)`.
  - Extremes: `quartile(x)`, `min(x)`, `max(x)`.
  - Positions: `first(x)`, `last(x)`, `nth(x, 2)`.

## `ungroup()`: takes a data frame and removes the grouping

# Chain the functions together using pipe (%>%)

```r
# By using intermediate values
cut_depth <- group_by(diamonds, cut, depth)
cut_depth <- summarise(cut_depth, n = n())
cut_depth <- filter(cut_depth, depth > 55, depth < 70)
cut_depth <- mutate(cut_depth, prop = n / sum(n))
```

```r
# By "composing" functions
mutate(
  filter(
    summarise(
      group_by(
        diamonds,
        cut,
        depth
      ),
      n = n()
    ),
    depth > 55,
    depth < 70
  ),
  prop = n / sum(n)
)
```

```r
cut_depth <- diamonds %>%
  group_by(cut, depth) %>%
  summarise(n = n()) %>%
  filter(depth > 55, depth < 70) %>%
  mutate(prop = n / sum(n))
```

Question: Which one do you think is the most elegant?

# Chain the functions together using pipe (%>%)

```r
# By using intermediate values
cut_depth <- group_by(diamonds, cut, depth)
cut_depth <- summarise(cut_depth, n = n())
cut_depth <- filter(cut_depth, depth > 55, depth < 70)
cut_depth <- mutate(cut_depth, prop = n / sum(n))
```

```r
# By "composing" functions
mutate(
  filter(
    summarise(
      group_by(
        diamonds,
        cut,
        depth
      ),
      n = n()
    ),
    depth > 55,
    depth < 70
  ),
  prop = n / sum(n)
)
```

```r
cut_depth <- diamonds %>%
  group_by(cut, depth) %>%
  summarise(n = n()) %>%
  filter(depth > 55, depth < 70) %>%
  mutate(prop = n / sum(n))
```

`%>%` works by taking the object on the left hand side (LHS) and using it as the first argument to the function on the right hand side (RHS)

- `f(x,y)  <=>  x %>% f(y)`

**Exercise:** rewrite `g(f(x, y), z)` using pipe

# Aesthetics

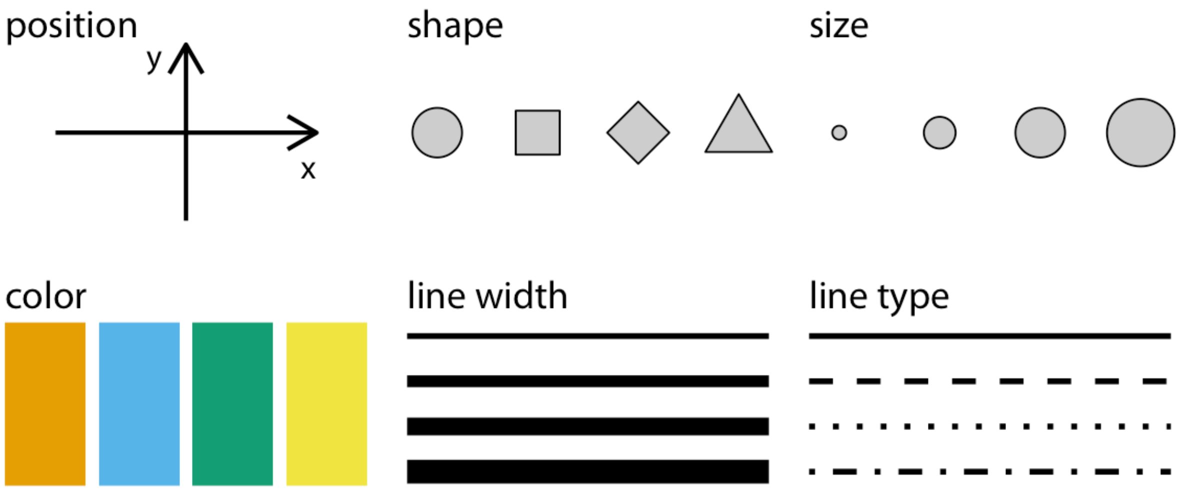



Image from book *Fundamentals of Data Visualization*

# Aesthetic mappings

Aesthetic mappings `aes()` describe how variables are mapped to visual properties or aesthetics.  It takes aesthetic-variable pairs

- `aes(x = displ, y = hwy, colour = class)`

| Aesthetic | Description |
|---|---|
| x | x-axis position |
| y | y-axis position |
| colour | Color of points or outlines of other shapes |
| fill | Fill color |
| size | size of the point or thickness of line |
| alpha | Transparency of the shape |
| linetype | Line type such a solid, dashed, dotted |
| labels | Text on the plot |
| shape | Shape of the geometry |



Image from internet

# Aesthetic mappings

Aesthetic mappings `aes()` describe how variables are mapped to visual properties or aesthetics.  It takes aesthetic-variable pairs

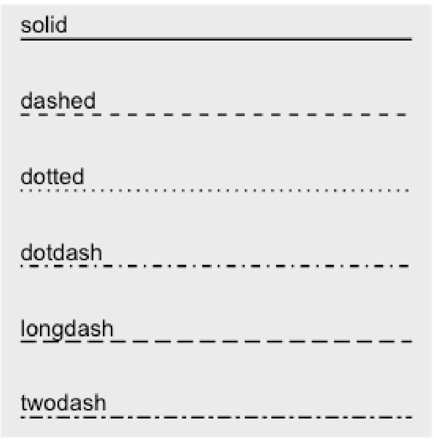
➢ `aes(x = displ, y = hwy, colour = class)`

| Aesthetic | Description |
|---|---|
| x | x-axis position |
| y | y-axis position |
| colour | Color of points or outlines of other shapes |
| fill | Fill color |
| size | size of the point or thickness of line |
| alpha | Transparency of the shape |
| linetype | Line type such a solid, dashed, dotted |
| labels | Text on the plot |
| shape | Shape of the geometry |

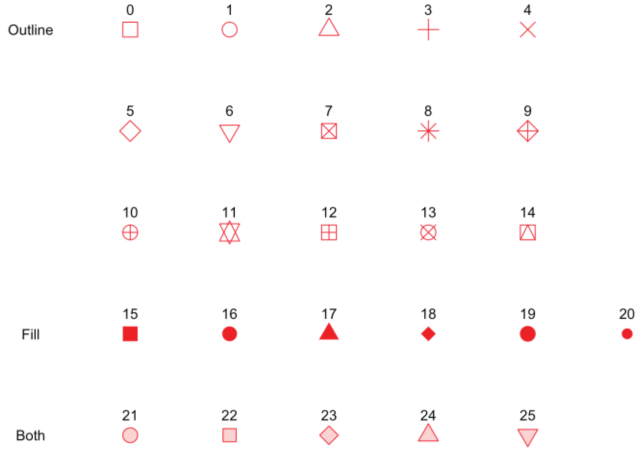
Check available options

➢ `vignette("ggplot2-specs")`

**linetype**




**shape**



# Aesthetic mappings and setting

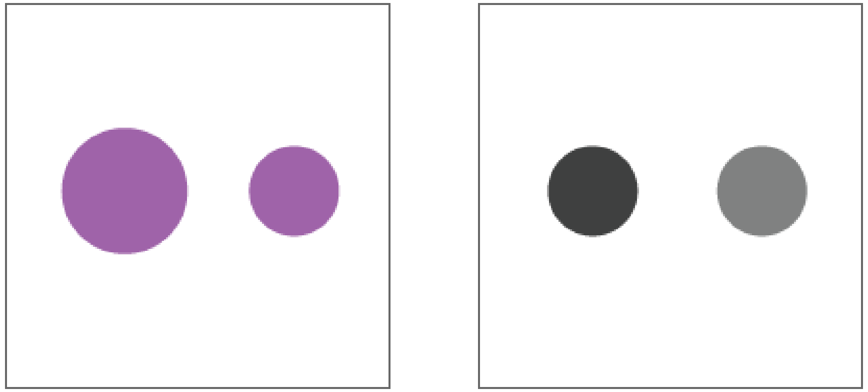
- The function supports some simple transformation (e.g. log(x) )
  - ➢ `aes(x = log(displ), y = hwy, colour = class)`

- Within each layer, you can add, override, or remove mappings
  - ➢ `base = ggplot(mpg, aes(displ, hwy, colour =class))`
    - Add: `base + geom_point(aes(shape  = drv))`
    - Override: `base + geom_point(aes(colour = drv))`
    - Remove: `base + geom_point(aes(colour = NULL))`

- Aesthetics mapping vs setting
  - Map an aesthetic to a variable when the appearance is governed by a variable
  - Set the aesthetic atrribute to a single value in the layer parameters
    - ➢ `ggplot(mpg) + geom_point(aes(displ, hwy, colour = "blue"))`
    - ➢ `ggplot(mpg) + geom_point(aes(displ, hwy), colour = "blue")`

# Things to consider when using `aes()`

- Choose visual aesthetics based on the type of data variables
  - colour and shape work well with categorical variables
  - size works well for continuous variables (bubble plot)



Size works better for quantitative variables than lightness

- Don't make the plot too busy, less is more
  - It's difficult to see the simultaneous relationships among colour, shape and size

# Where to get help?

- https://community.rstudio.com/tag/ggplot2
- https://www.google.com
- https://stackoverflow.com
- https://chat.openai.com/