UCLA QCBio Collaboratory

# Advanced data visualization with ggplot2

Wenbin Guo
Bioinformatics IDP, UCLA
wbguo@ucla.edu
2024 Fall

# Notation of the slides

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

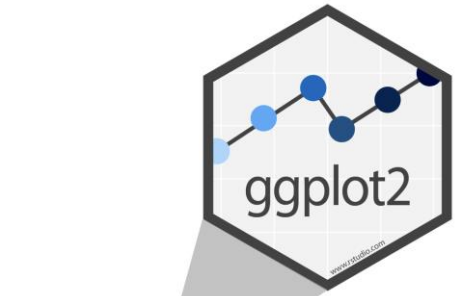
- Link is underlined

- Important terminology is in **bold** font

- Practice comes with

# 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


UCLA QCBio Collaboratory

# Day 3: Examples and useful packages

Wenbin Guo
Bioinformatics IDP, UCLA
wbguo@ucla.edu
2024 Fall

# Overview

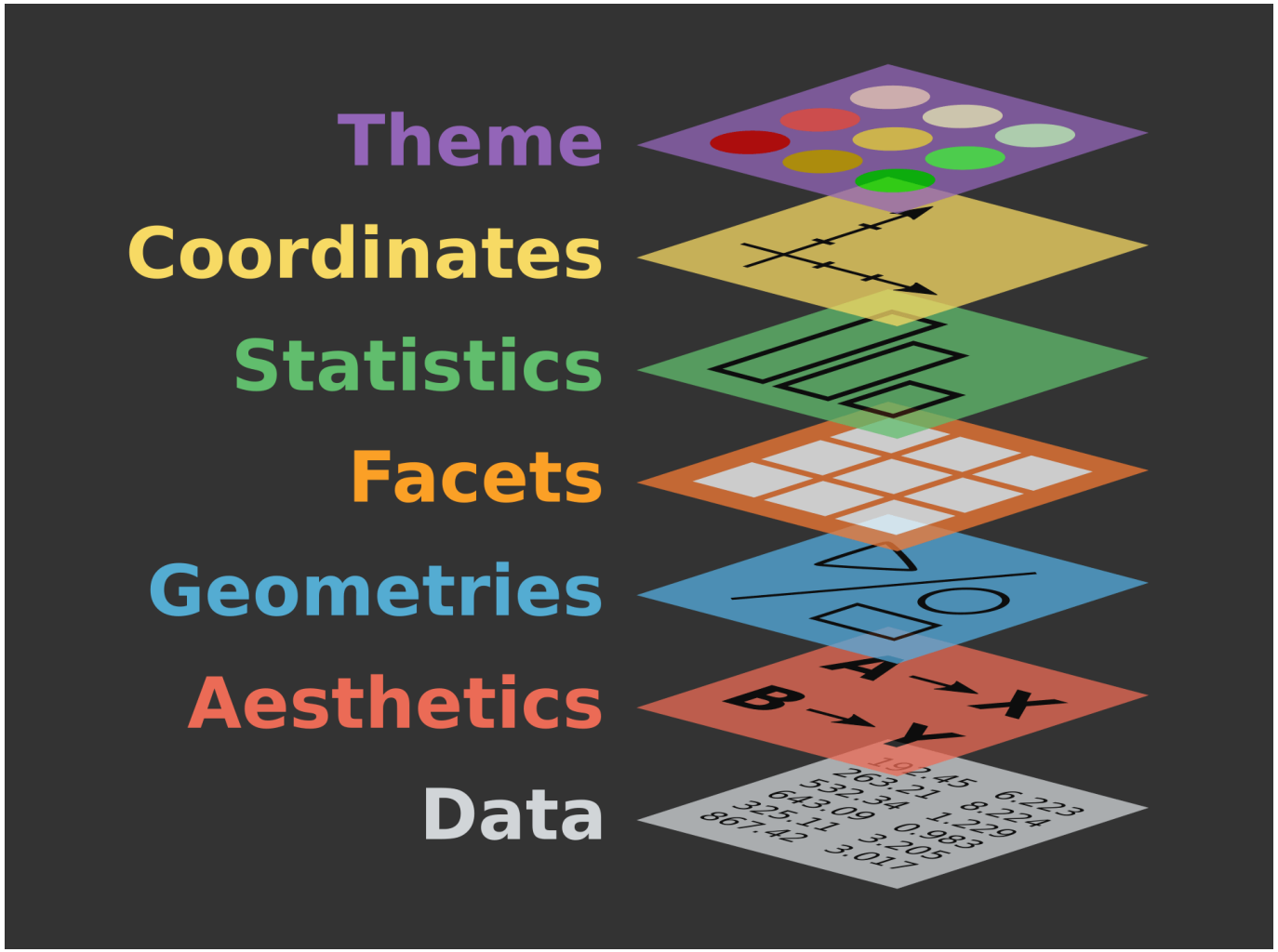
## Time

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

## Topics

- ❑ Legends
- ❑ Themes
- ❑ Principles of data visualization
- ❑ Useful packages

# Summary – Day1&2




**Describes all the non-data ink**

**Plotting space for the data**

**Statistical models and summaries**

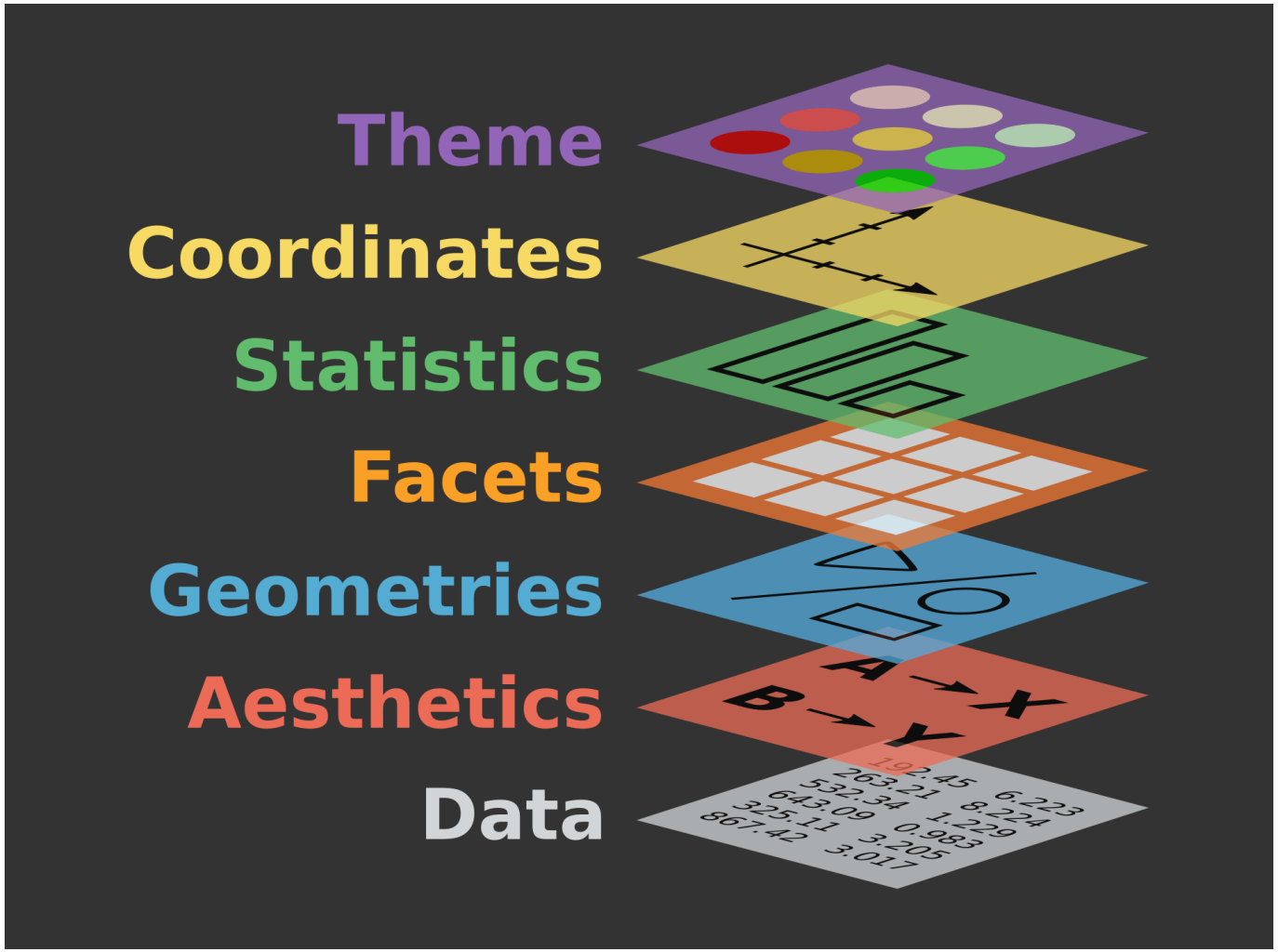
**Rows and columns of sub-plots**

**Shapes used to represent the data**

**Scales onto which data is mapped**

**The actual variables to be plotted**

# Summary – Day1&2




```
ggplot (data = <DATA>) +                                   ] required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat =<STAT>, position =<POSITION>) +                     ] Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

Syntax of ggplot2

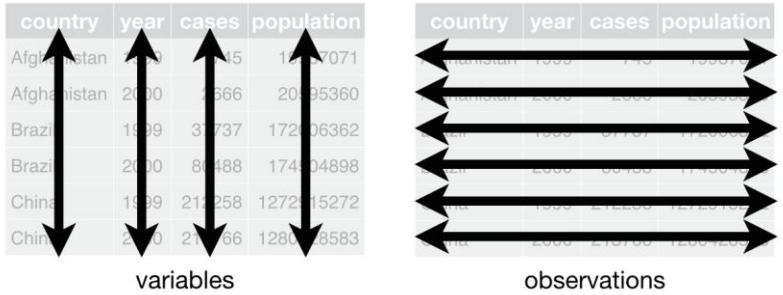
```
ggplot(df5, aes(date, cases, color = type)) +
  geom_point(size = 0.5) + geom_line(aes(y = cases7d)) +
  scale_x_date(date_breaks = "1 month", date_labels = "%d-%b") +
  scale_color_manual(values=c("darkorange2","firebrick","dodgerblue2")) +
  theme_classic(base_size = 24) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))
```

| Layer | Function |
|---|---|
| Data | `ggplot(data)` |
| Aesthetics | `aes()` |
| Layers | `geom_*()` and `stat_*()` |
| Scales | `scale_*()` |
| Coordinate System | `coord_*()` |
| Facets | `facet_*()` |
| Visual Themes | `theme()` and `theme_*()` |



Image from Internet

# Summary – Day1&2




## ❑ Data

- Variable types, factors and data frame
- Data wrangling functions

| Category | Function | Usage |
|---|---|---|
| Manipulate observations | `filter()` | keep rows that satisfy criterions |
| | `arrange()` | orders observations according to variables |
| | `bind_rows()` | bind any number of data frames by row |
| Manipulate variables | `select()` | keep variables using their names and types |
| | `mutate()` | create new variables |
| | `*_join()` | merge data frames by columns |
| Reshape data | `pivot_longer()` | convert data frame from wide to long |
| | `pivot_wider()` | convert data frame from long to wide |
| Summarize data | `group_by()` | group data frame by variable |
| | `summarize()` | summarize the grouped data frame |
| pipe | `%>%` | chain operations together |

# Summary – Day1&2

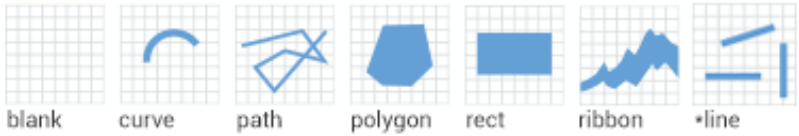
- ❑ Data
- ❑ Aesthetic mappings
  - Describes how variables are mapped to visual properties of geometric objects

➢ `aes(x = log(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 |

# Summary – Day1&2

- ❑ Data
- ❑ Aesthetic mappings
- ❑ Geometry




| Category | Function | Usage |
|---|---|---|
| Blank | `geom_blank()` | display nothing |
| Line | `geom_curve()` | draw a curved line |
| | `geom_segment()` | draw a line segment, specified by start and end position |
| | `geom_abline()` | draw a straight line, specified by slope and intercept |
| | `geom_path()` | connect observations in order of the data |
| | `geom_line()` | connect observation in order of the variables on x axis |
| Region | `geom_rect()` | draw rectangles |
| | `geom_polygon()` | draw filled polygons |
| | `geom_ribbon()` | draw ribbons, a path with vertical thickness |

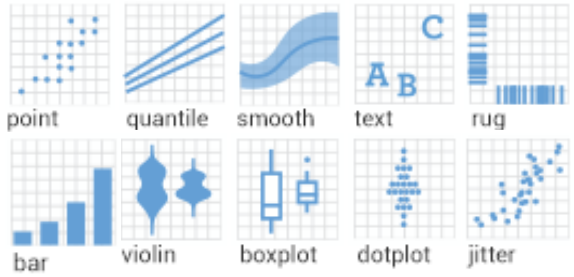
# Summary – Day1&2

- ❑ Data
- ❑ Aesthetic mappings
- ❑ Geometry


| Category | Function | Usage |
|---|---|---|
| Discrete | `geom_bar()` | display count distribution of discrete variable |
| Continuous | `geom_histogram()` | bin and count continuous variable, display with bars |
| | `geom_freqpoly()` | bin and count continuous variable, display with lines |
| | `geom_density()` | display smoothed density estimate |
| | `geom_dotplot()` | stack individual points into a dot plot |

# Summary – Day1&2

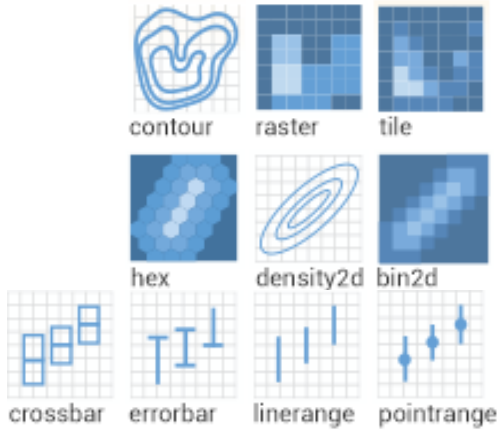
- ❑ Data
- ❑ Aesthetic mappings
- ❑ Geometry




| Category | Function | Usage |
|---|---|---|
| Both continuous | `geom_point()` | display scatterplot |
| | `geom_smooth()` | display smoothed line of best fit |
| | `geom_quantile()` | display smoothed quantile regression |
| | `geom_rug()` | display marginal distribution |
| 1 continuous + 1 discrete | `geom_bar(stat = "identity")` | display a bar chart of precomputed summaries |
| | `geom_boxplot()` | display box plots of different group |
| | `geom_violin()` | display density of values in each group |
| | `geom_jitter()` | randomly jitter overlapping points |

# Summary – Day1&2

- ❑ Data
- ❑ Aesthetic mappings
- ❑ Geometry




| Category | Function | Usage |
|---|---|---|
| 3 variables | `geom_contour()` | display contour |
| | `geom_tile()` | tile the plane with rectangles |
| | `geom_raster()` | fast version of `geom_tile()` for equal sized tiles |
| 2D distribution | `geom_bin2d()` | bin into rectangles and count |
| | `geom_hex()` | bin into hexagons and count |
| | `geom_density2d()` | display smoothed 2d density estimate |
| Uncertainty | `geom_crossbar()` | display vertical bar with center |
| | `geom_errorbar()` | display error bars |
| | `geom_linerange()` | display vertical line |
| | `geom_pointrange()` | displayvertical line with center |

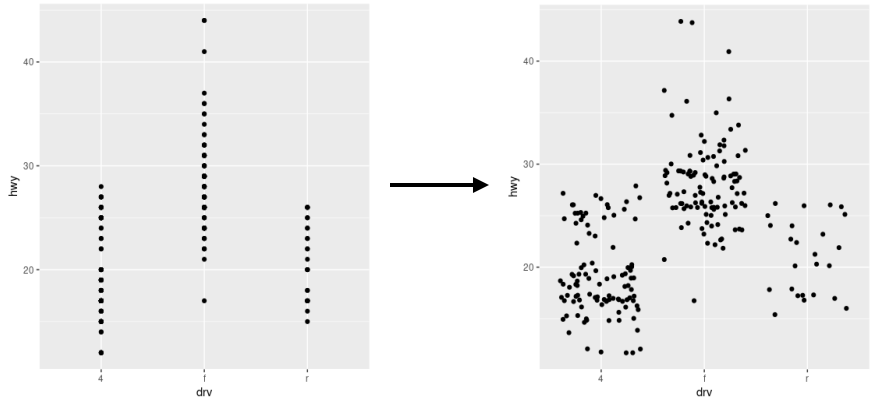
# Summary – Day1&2

- ❑ Data
- ❑ Aesthetic mappings
- ❑ Geometry
- ❑ Statistical transformation

| Function | Alternative form |
|---|---|
| `stat_bin()` | `geom_bar(stat="bin")` |
| `stat_bin2d()` | `geom_bin2d(stat="bin2d")` |
| `stat_binhex()` | `geom_hex(stat="binhex")` |
| `stat_contour()` | `geom_contour(stat="contour")` |
| `stat_smooth()` | `geom_smooth(stat="smooth")` |
| `stat_count()` | `geom_bar(stat="count")` |
| `stat_identity()` | `geom_point(stat="identity")` |

# Summary – Day1&2

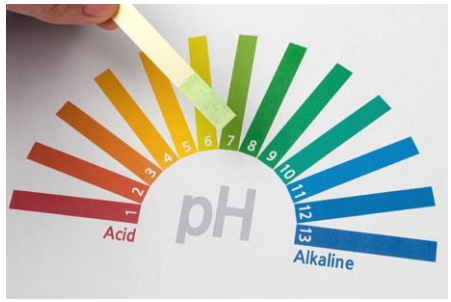
- ❑ Data
- ❑ Aesthetic mappings
- ❑ Geometry
- ❑ Statistical transformation
- ❑ Position adjustment




| Function | Usage |
|---|---|
| `position_nudge()` | move points by a fixed offset |
| `position_jitter()` | add a little random noise to every position |
| `position_jitterdodge()` | dodge points within groups, then add a little random noise |
| `position_stack()` | stack overlapping bars (or areas) on top of each other |
| `position_fill()` | stack overlapping bars, scaling so the top is always at 1 |
| `position_dodge()` | place overlapping bars (or boxplots) side-by-side |
| `position_identity()` | does nothing |

# Summary – Day1&2

- ❑ Data
- ❑ Aesthetic mappings
- ❑ Geometry
- ❑ Statistical transformation
- ❑ Position adjustment
- ❑ Scale




Syntax: `scale_<aesthetic>_<type>`

Aesthetics: `position`, `color`, `fill`, `size`, `shape`, `alpha`, `linetype`

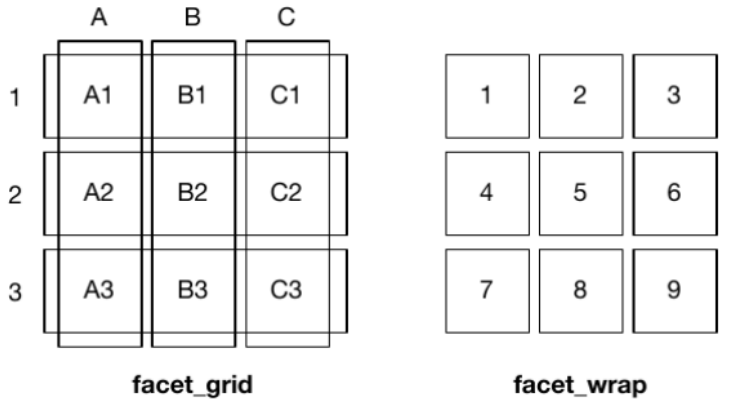
Type: `continuous`, `discrete`, …

| Function | Usage |
|---|---|
| `scale_x_continous()` | set x axis with name, limits, breaks, labels, expansion, transformation |
| `scale_fill_gradient()` | produces a 2 colour gradient for variable |
| `scale_fill_gradient2()` | produces a 3 colour gradient with specified midpoint |
| `scale_fill_gradientn()` | produces a n colour gradient |
| `scale_fill_brewer()` | produce a color palette for discrete variables |
| `scale_fill_manual()` | manually set color palette for discrete variables |

# Summary – Day1&2

- ❑ Data
- ❑ Aesthetic mappings
- ❑ Geometry
- ❑ Statistical transformation
- ❑ Position adjustment
- ❑ Scale
- ❑ Facet

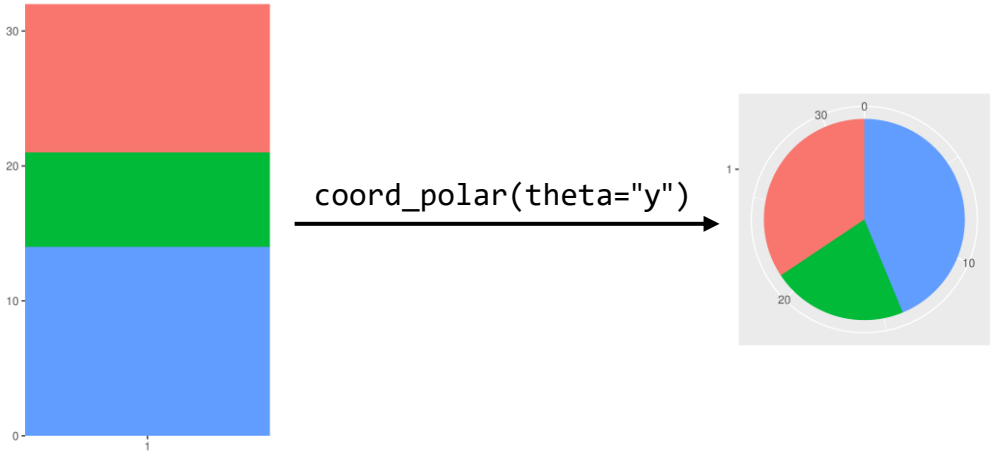
| Function | Usage |
|---|---|
| `facet_null()` | a single plot, the default |
| `facet_wrap()` | wrap a 1d ribbon of panels into 2d |
| `facet_grid()` | generate a 2d grid of panels defined by variables which form the rows and columns |



# Summary – Day1&2

- ❑ Data
- ❑ Aesthetic mappings
- ❑ Geometry
- ❑ Statistical transformation
- ❑ Position adjustment
- ❑ Scale
- ❑ Facet
- ❑ Coordinate systems

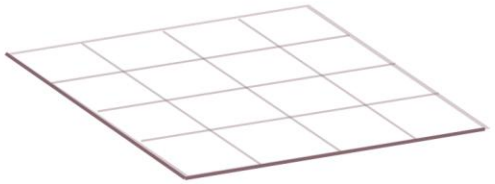
| Function | Usage |
|---|---|
| `coord_cartesian(xlim=c(a,b))` | zoom into a region |
| `coord_flip()` | flip the axis |
| `coord_fixed(ratio = 1)` | set x and y with equal scales |
| `coord_trans()` | apply transformation to axis |
| `coord_polar()` | Convert to polar coordinates |



# More customization

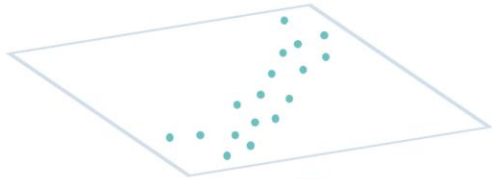

MAKING A GRAPH WITH GGPLOT2

**Customise the look of your plot with themes (pre-made or your own!):**
+ theme_bw()

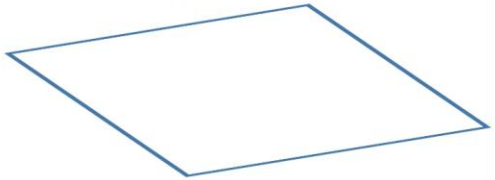


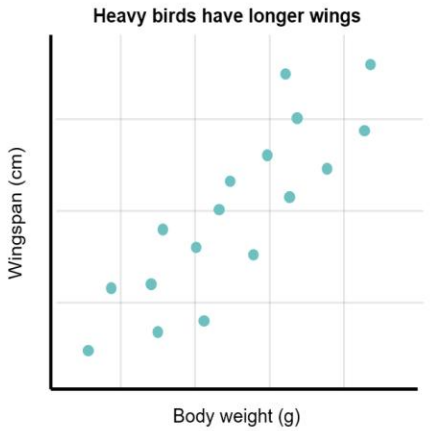
**Add labels and titles:**
+ labs(x = “Body weight (g)”, y = “Wingspan (cm)”,
title = “Heavy birds have longer wings”)


**Specify the type of graph and the variables to use:**
+ geom_point(aes(x = body.weight, y = wingspan))



**Plot the device containing your data:**
ggplot(data = birds)






https://codeahoy.com/learn/rtutorial/ch8/

# Title, Axis, legend



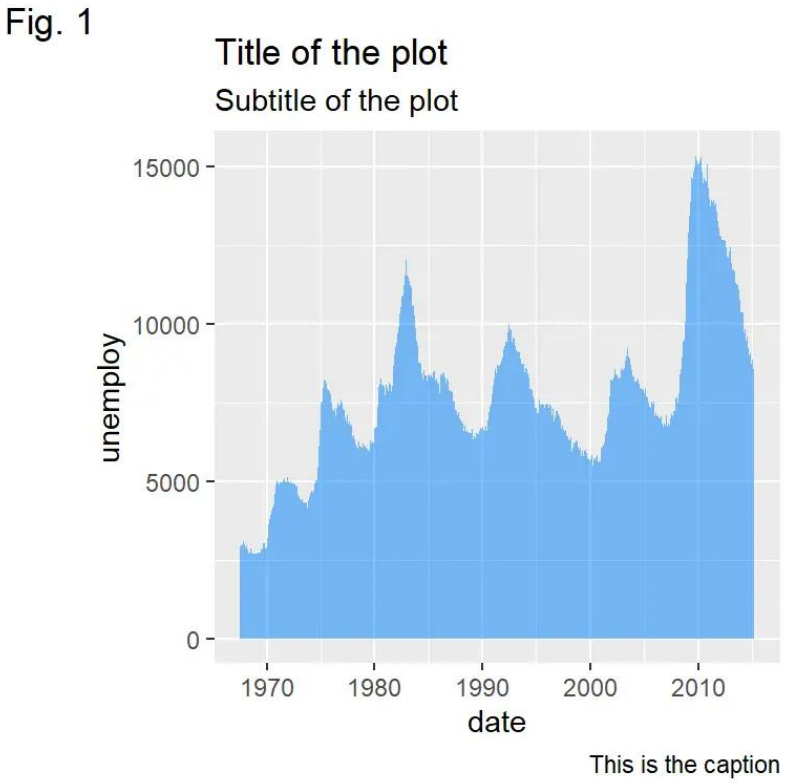

Image from Internet

# Plot title

Change the title:

- ```
  labs(title = "Title of the plot",
       subtitle = "Subtitle of the plot",
       caption = "This is the caption",
       tag = "Fig. 1")
  ```

- ```
  ggtitle("Title of the plot",
          subtitle = "Subtitle of the plot")
  ```



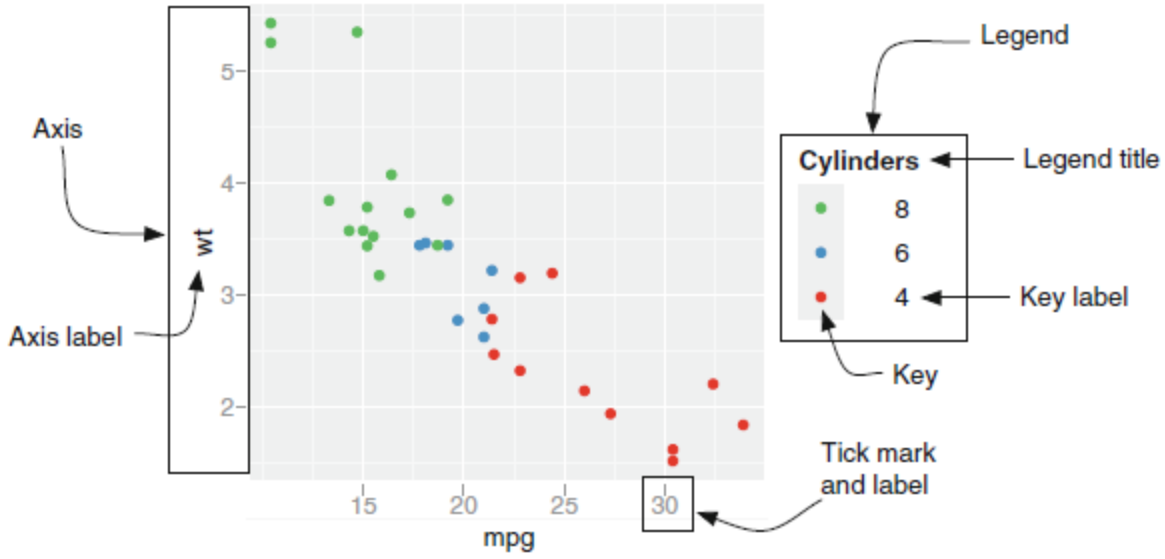
# Axis and legend components

Change axis label:

- `scale_x_continuous(name = "mpg")`
- `xlab("mpg")` or `ylab("wt")`
- `labs(x = "mpg", y = "wt")`

To remove axis label:

- `labs(x = "", y = "")`
- `labs(x = NULL, y = NULL)`



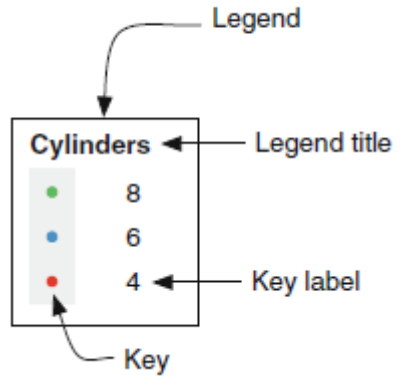

Question: what's the difference between these 2 removal approaches?

| Axis | Legend | Argument name |
|---|---|---|
| Label | Title | `name` |
| Ticks & grid line | Key | `breaks` |
| Tick label | Key label | `labels` |

# Legend

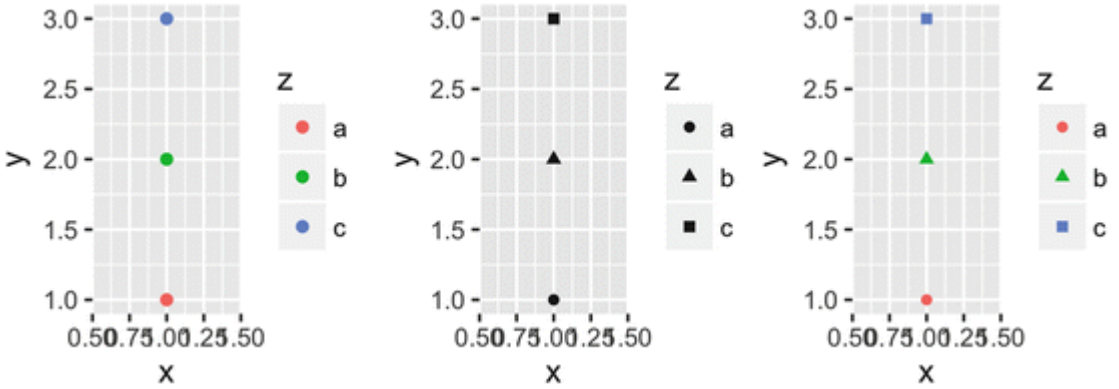
A legend can display multiple aesthetics (e.g. colour and shape) from multiple layers

- `show.legend = FALSE` to prevent layers appearing in legend
- `show.legend = TRUE` to force layers to appear




Legends will be combined when the same variable is mapped to different aesthetics

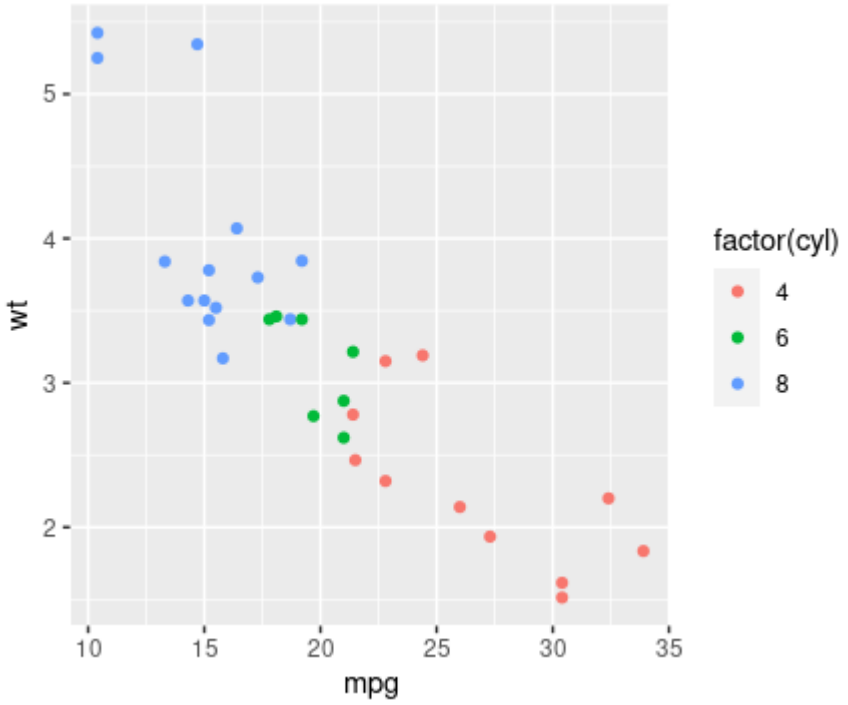
```
geom_point(aes(colour = z))
geom_point(aes(shape = z))
geom_point(aes(shape = z, colour = z))
```

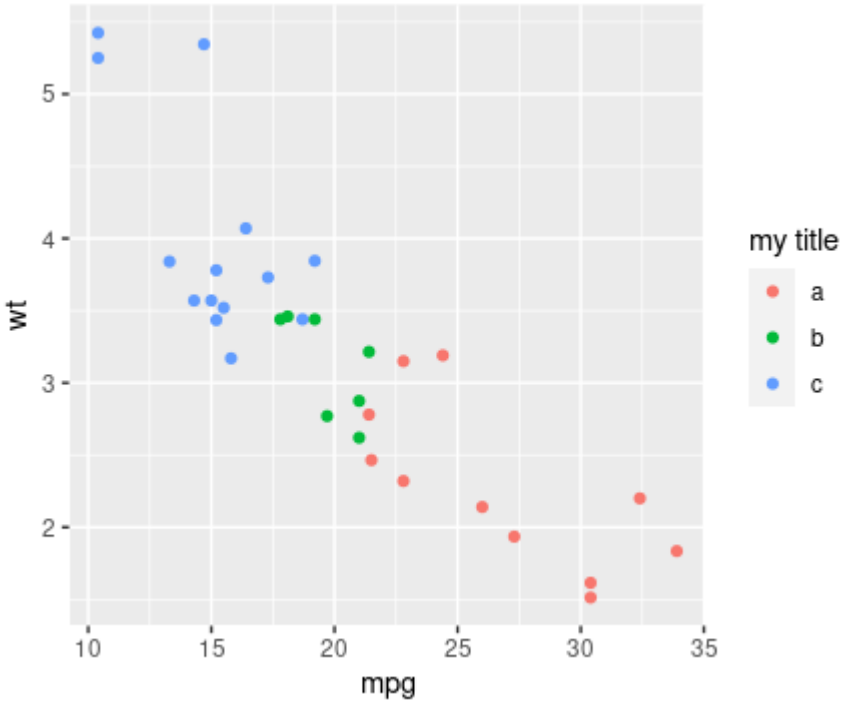


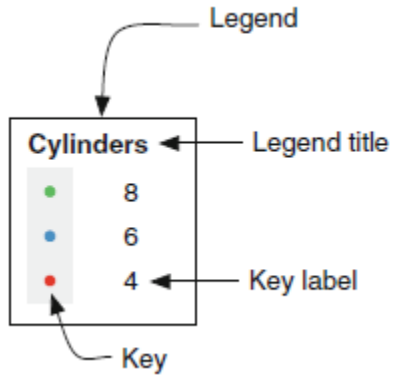
# Customize legend

Control legend title and label

- `scale_color_discrete(labels = c("a", "b", "c"), name = "my title")`



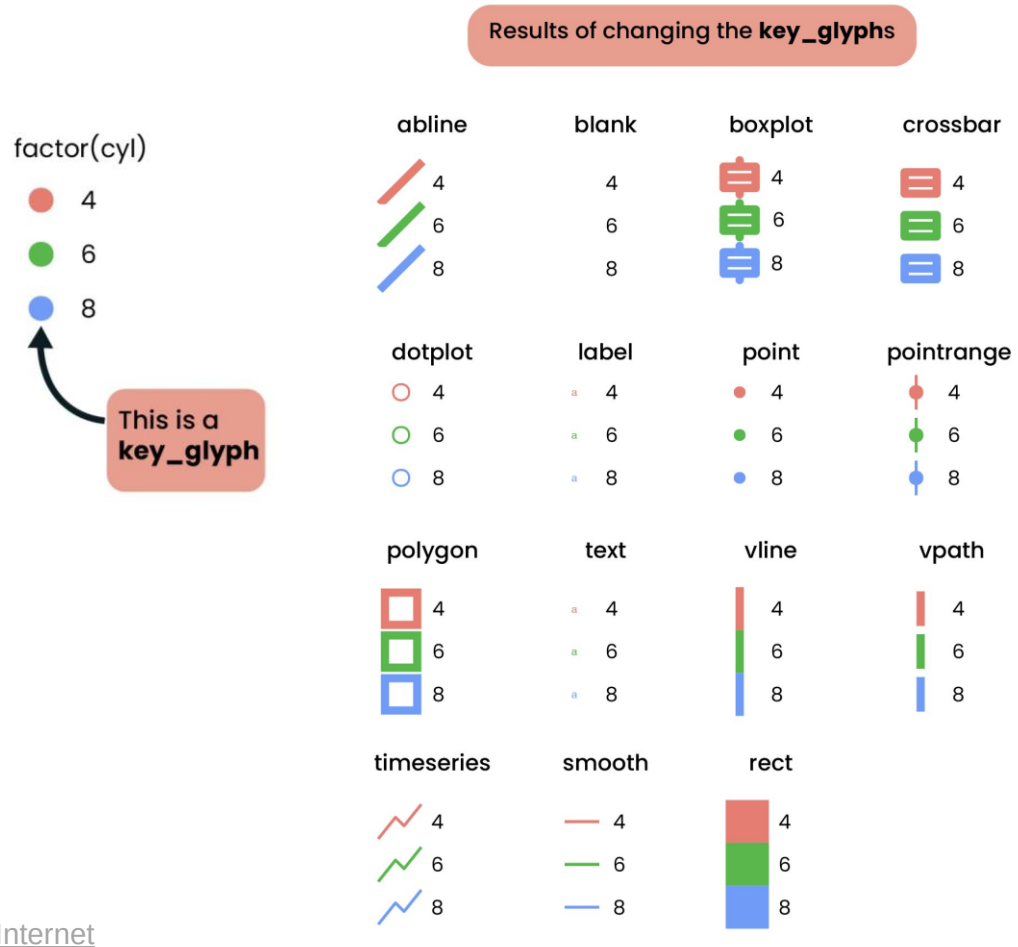



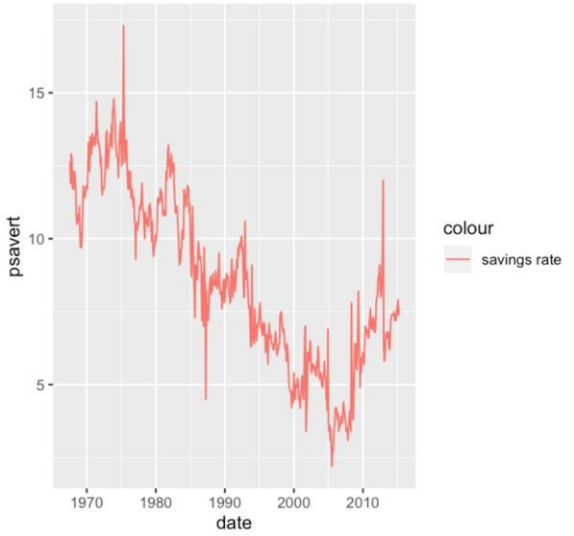
# Customize legend

## Control legend key glyph

- `geom_*(key_glyph = value)`




```
ggplot(economics) +
  aes(date,
      psavert,
      color = "savings rate") +
  geom_line()
```




```
ggplot(economics) +
  aes(date,
      psavert,
      color = "savings rate") +
  geom_line(key_glyph = "timeseries")
```

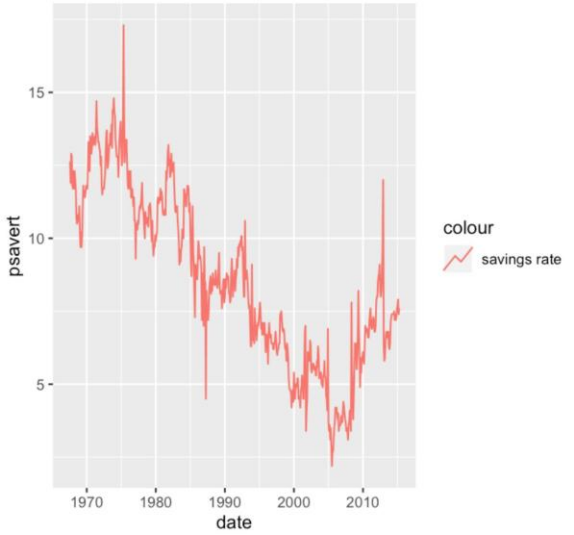


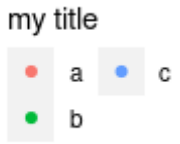

Image from Internet

# Fine control on legend

`guide_legend()` offers additional control over the fine details of the legend

Control the legend keys' layout:

- `nrow` or `ncol` specifies the dimensions of the table.
- `byrow` controls how the table is filled
  - `guides(color = guide_legend(ncol = 2))`
  - `guides(color = guide_legend(ncol = 2, byrow = TRUE))`



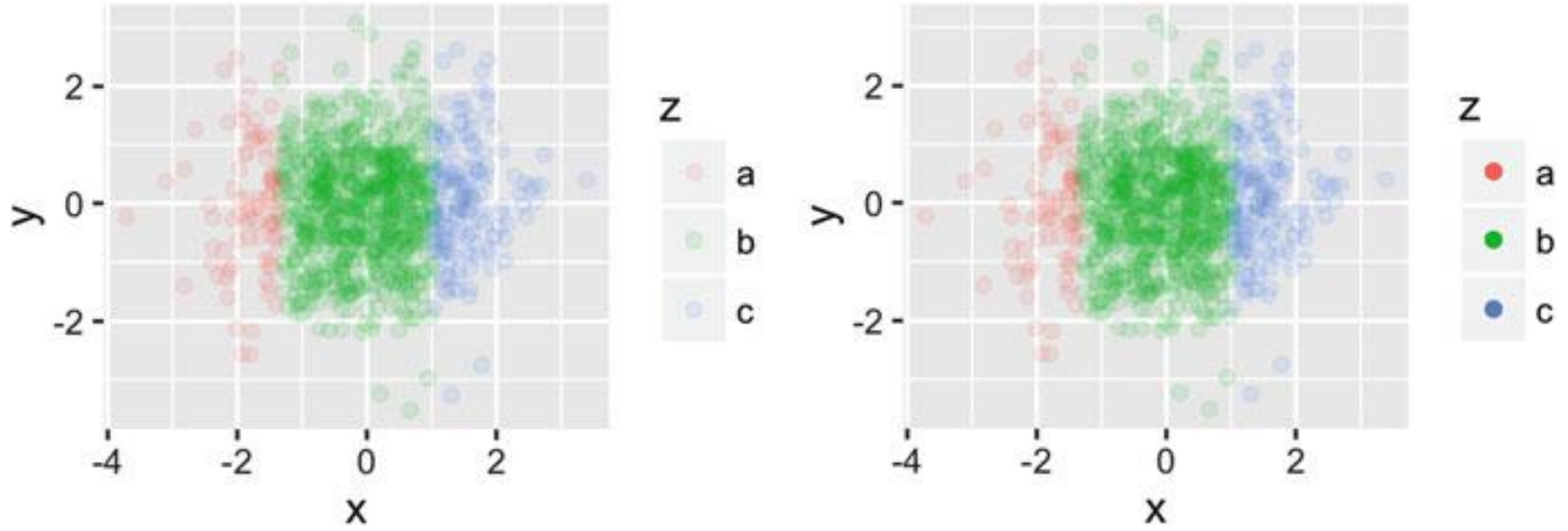
- `reverse:` reverse the order of keys
  - `guides(color = guide_legend(ncol = 2, byrow = TRUE, reverse=TRUE))`

# Fine control on legend

`guide_legend()` offers additional control over the fine details of the legend

Use `override.aes` to override some aesthetic settings (the geoms in the legend will display differently to the geoms in the plot)

- `guides(colour = guide_legend(override.aes = list(alpha = 1)))`

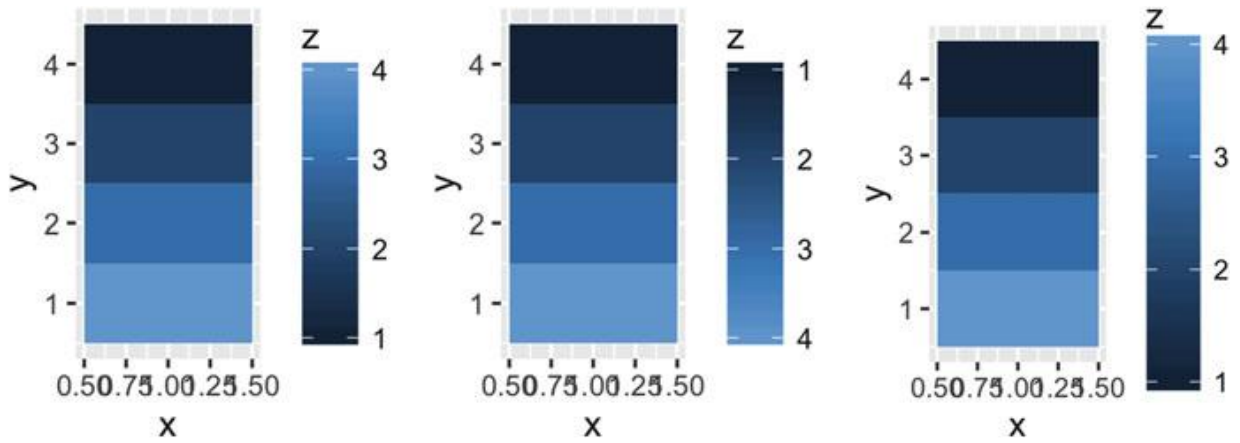


# Fine control on legend

`guide_colorbar()` offers control on the continuous colour scales

Important parameters:

- `barwidth` and `barheight` controls the the size of the bar
- `nbin` controls the number of slices
- `reverse` flips the colour bar to put the lowest values at the top
  - `guides(fill = guide_colorbar(reverse = TRUE))`
  - `guides(fill = guide_colorbar(barheight = unit(4, "cm")))`



# Let’s do some practice!

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

# Theme

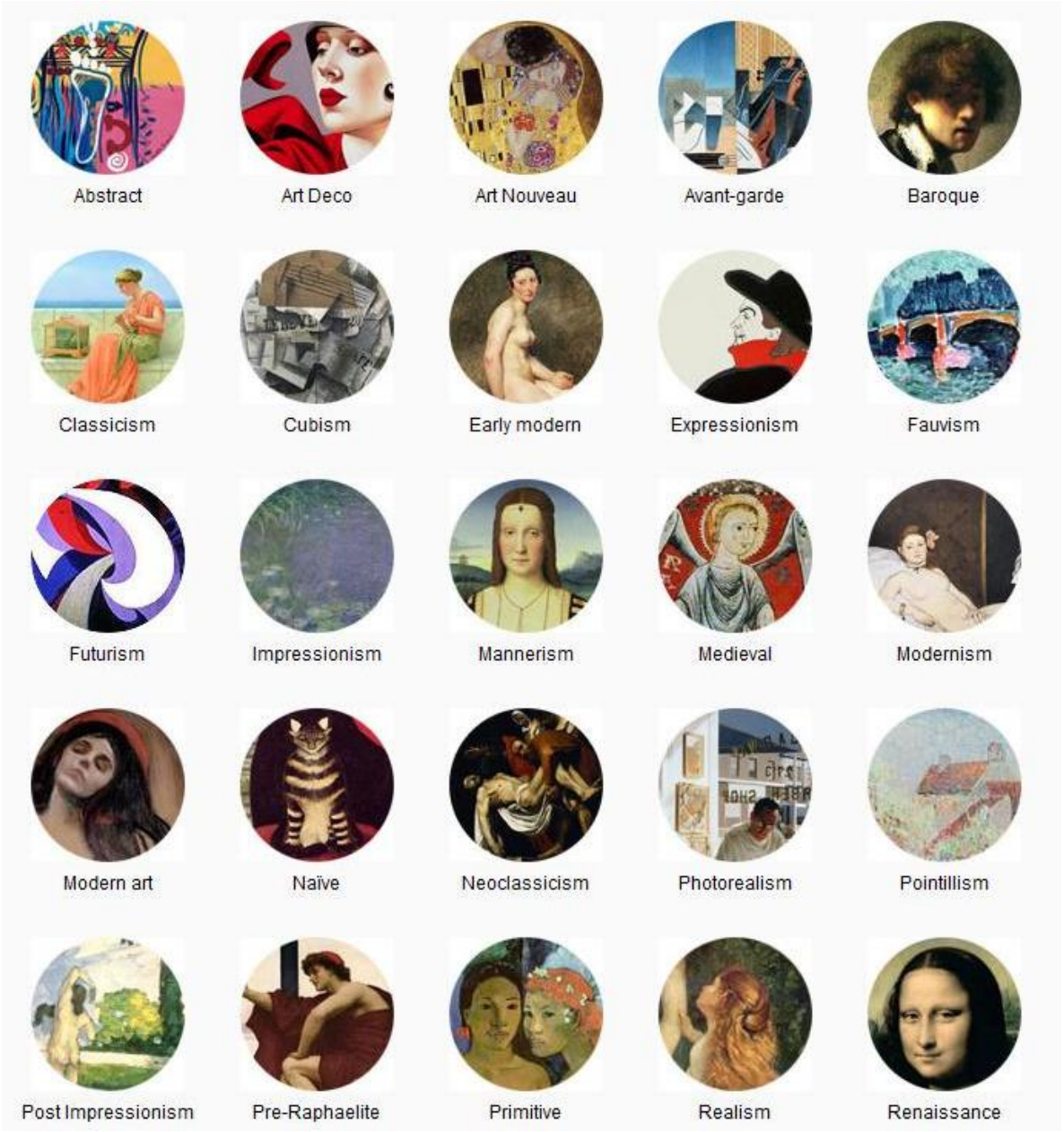





Image from Internet

# Fine control on the non-data elements

After the figure has been created, we can edit the rendering details using the theming system

There are 4 main components in the theming system

- **Theme elements** specify the non-data elements that you can control
- Each element is associated with an **element function**, which describes the visual properties
- **The theme() function** allows you to override the default theme elements by calling theme functions
- **Complete themes** set all of the theme elements to values designed to work together harmoniously

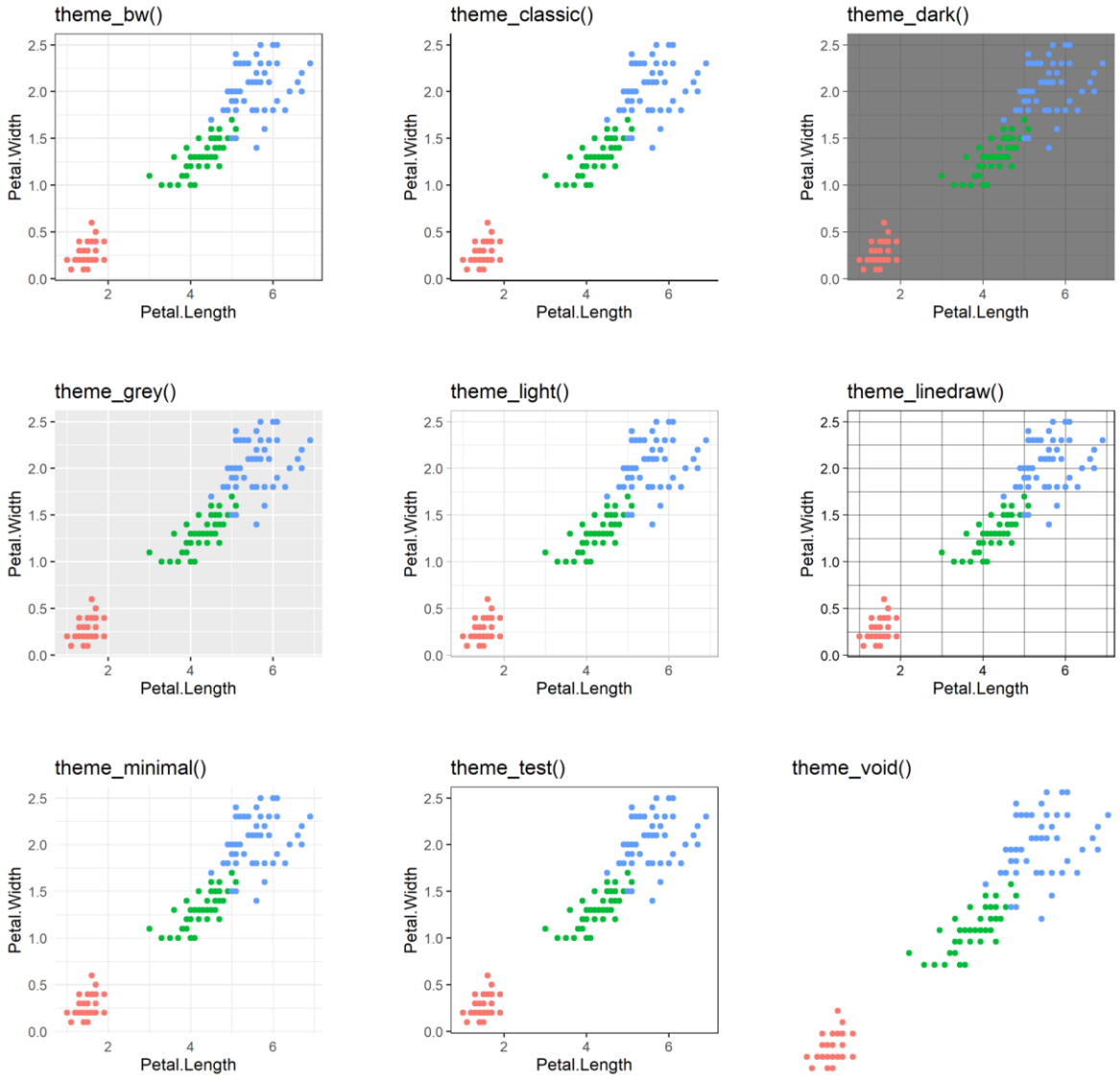
# Built-in themes in ggplot2

Applying theme for a plot at a time

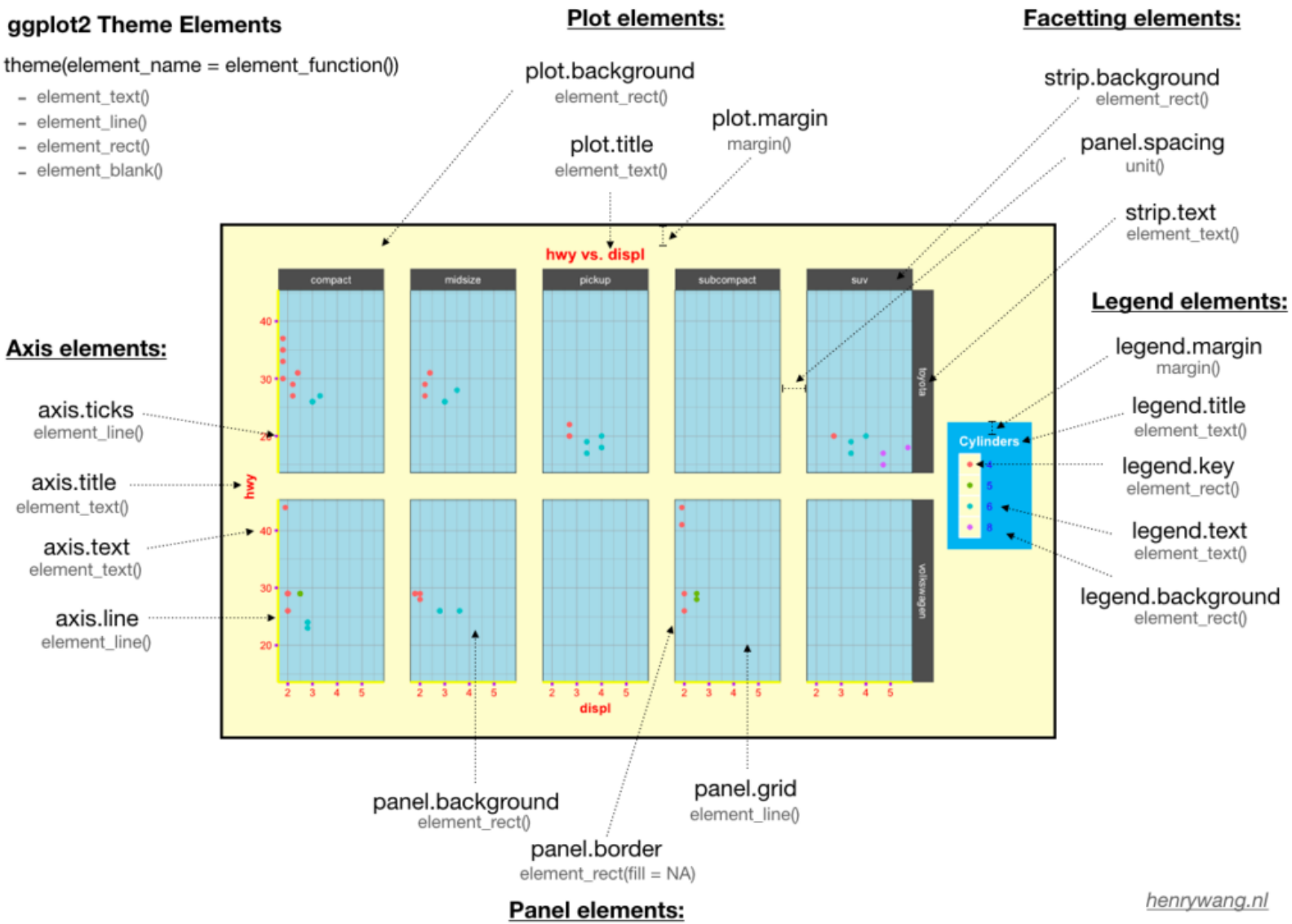
- `plot + theme_bw()`

Use `theme_set()` to change the default theme for all plots

- `theme_set(theme_bw())`



# Theme elements




henrywang.nl

Derived from *"ggplot2: Elegant Graphics for Data Analysis"*

# Modifying theme components

Syntax:

- `plot + theme(element.name = element function())`

There are four basic types of built-in element functions:

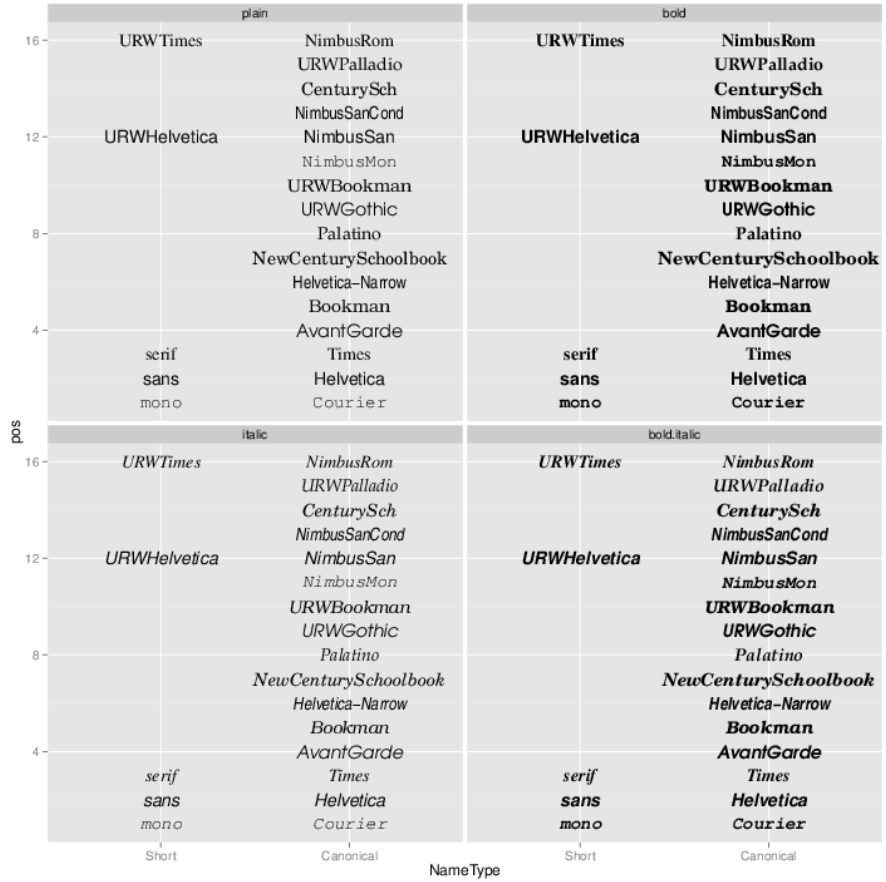
- `element_text()`
- `element_line()`
- `element_rect()`
- `element_blank()`

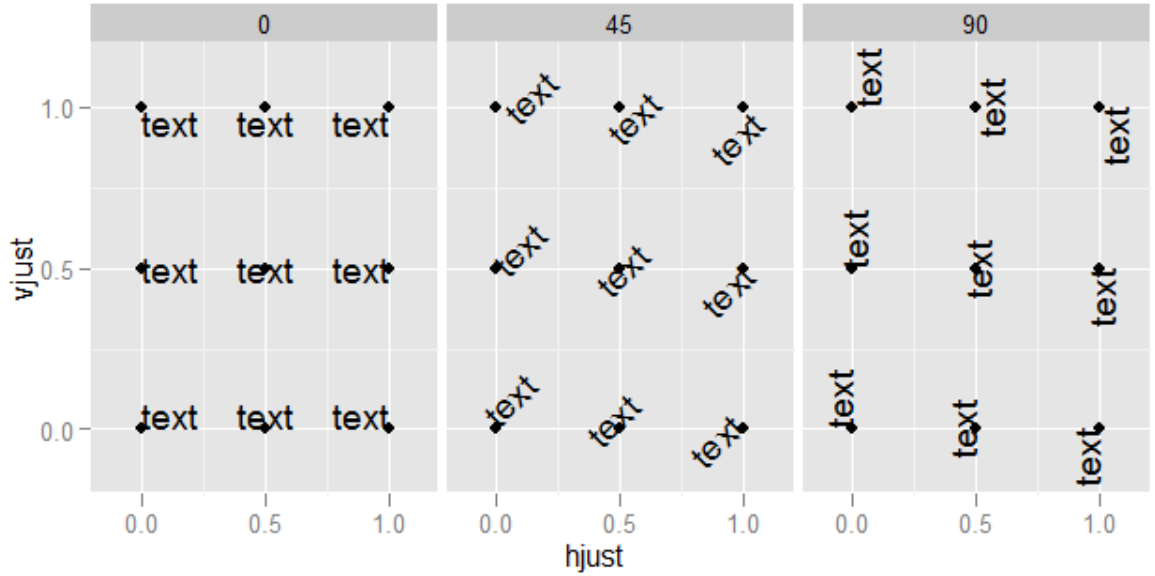
# Modifying theme components

`element_text()`: draws labels and headings

- Arguments: font family, face, colour, size, hjust, vjust, angle (in degrees)

➢ `theme(plot.title = element_text(face = "bold", colour = "red“, size = 2))`




- 0 means left-justified
- 1 means right-justified

# Modifying theme components

`element_text()`: draws labels and headings

- Arguments: font family, face, colour, size, hjust, vjust, angle (in degrees)

➢ `theme(plot.title = element_text(face = "bold", colour = "red“, size = 2))`

Control the margins around the text:

- Use margin argument and margin() function (specify space to add to top, bottom, right, left)

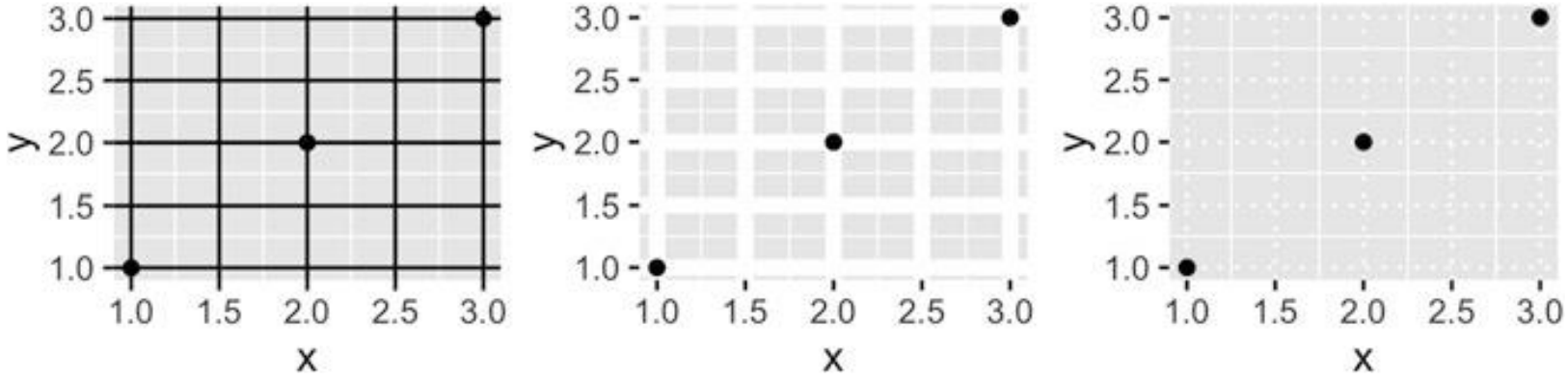
➢ `theme(plot.title = element_text(margin = margin(t=1, b=1, r=1, l=1)))`

# Modifying theme components

`element_line()`: draws lines

- Arguments: colour, size and linetype

- `theme(panel.grid.major = element_line(colour = "black"))`
- `theme(panel.grid.major = element_line(size = 2))`
- `theme(panel.grid.major = element_line(linetype = "dotted"))`

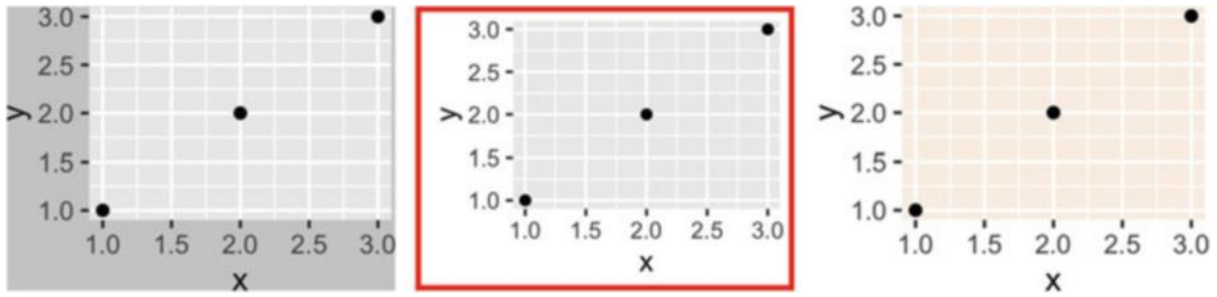


# Modifying theme components

`element_rect()`: draws rectangles, mostly used for backgrounds

- Arguments: fill, colour and border colour, size and linetype

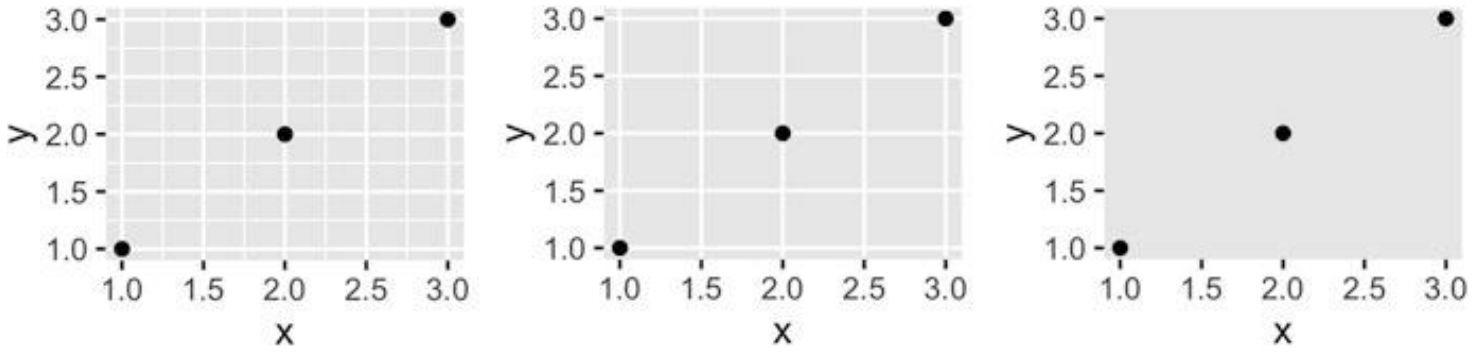
- `theme(plot.background = element_rect(fill = "grey80", colour = NA))`
- `theme(plot.background = element_rect(colour = "red", size = 2))`
- `theme(panel.background = element_rect(fill = "linen"))`



# Modifying theme components

`element_blank()`: draws nothing

- `theme(panel.grid.minor = element_blank())`
- `theme(panel.grid.major = element_blank())`




Set the element's color or fill as NA vs use element_blank():

- Set element as NA: `theme(axis.title = element_text(colour = "NA"))` will create invisible elements, space will be reserved.
- Use element_blank(): `theme(axis.title = element_blank())` will not draw anything, no space is allocated for that element

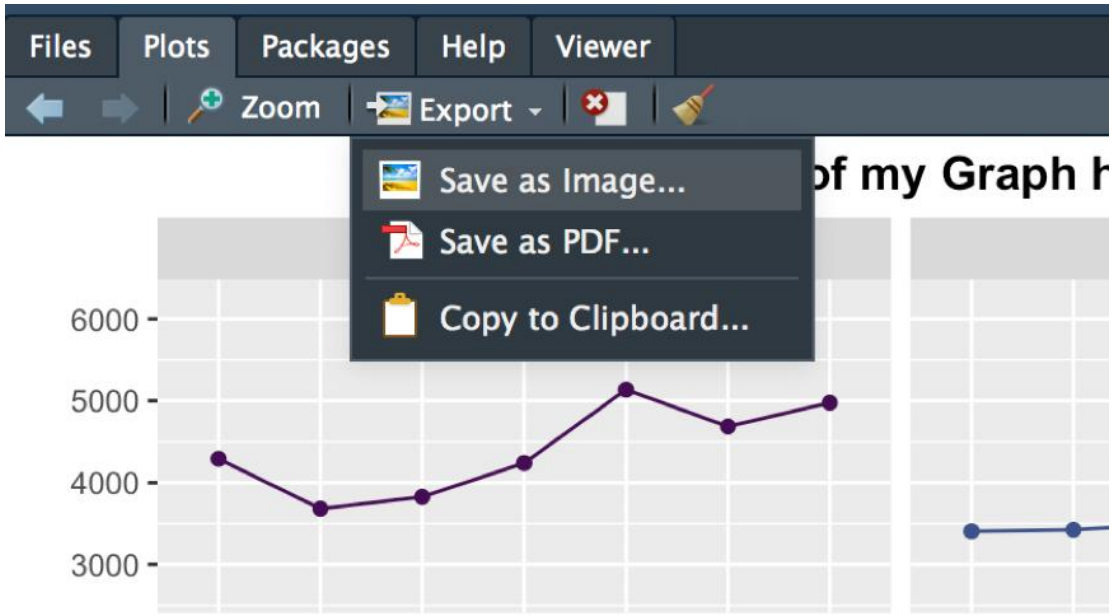
# Useful functions for modifying theme elements

`theme_update()`: modify theme elements for all future plots

- It returns the previous theme settings, so you can easily restore the original settings once you're done

```
old_theme <- theme_update(
  plot.background = element_rect(fill = "lightblue3", colour = NA),
  panel.background = element_rect(fill = "lightblue", colour = NA),
  axis.text = element_text(colour = "linen"),
  axis.title = element_text(colour = "linen")
)
base
theme_set(old_theme)
base
```

# Save plots




"Everything not saved will be lost" – Nintendo "Quite screen" message

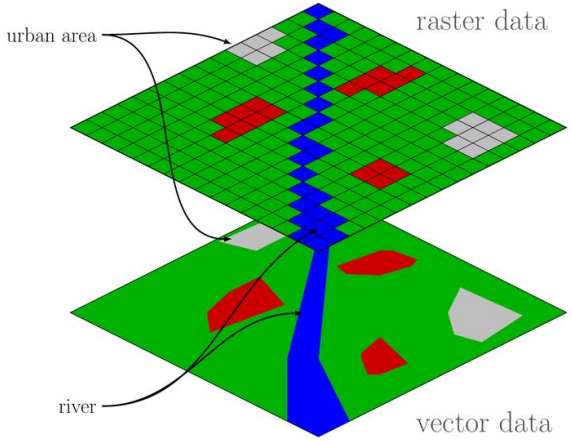
# Figure file format

**Vector graphics** describe a plot as sequence of operations:

- draw a line from ($x1$, $y1$) to ($x2$, $y2$)
- draw a circle at ($x3$, $x4$) with radius $r$

This means that they are effectively ‘infinitely’ zoomable; there is no loss of detail. Widely-used vector graphic formats are `pdf` and `svg`

**Raster graphics** store plot as an array of pixel colours and have a fixed optimal viewing size. Widely-used raster graphic format are `png` and `jpg`



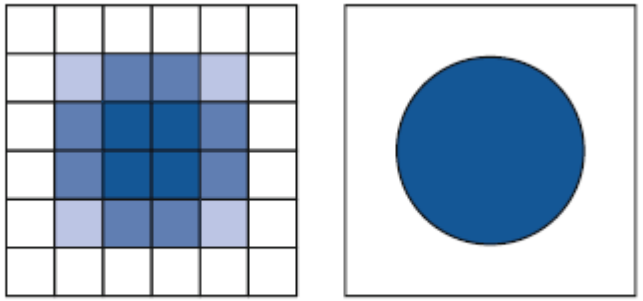



**Fig. 8.1** The schematic difference between raster (*left*) and vector (*right*) graphics

# Raster vs vector, which one to choose?

Usually the vector format is preferred, unless there is a compelling reason not to

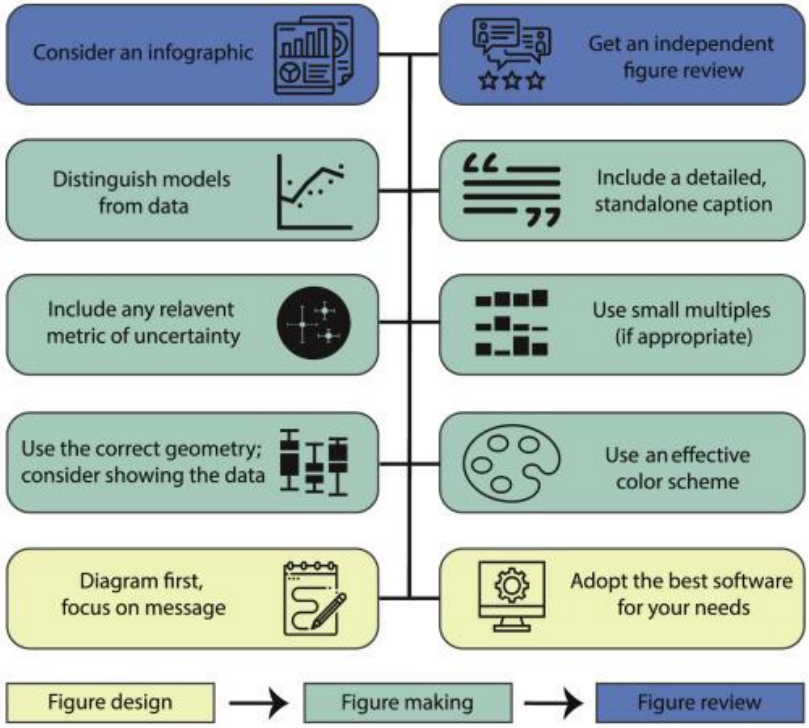
- Format restriction: e.g. if `png` is the only accepted format
- Speed and size consideration: if there are thousands of graphical objects (i.e. points) in the figure. A vector version will be large and slow to render

# Save plots to file

- `pdf()`/`png()`: the standard R approach where you open a graphics device, generate the plot, then close the device
  - `pdf("output.pdf", width = 6, height = 6)`
  - `ggplot(mpg, aes(displ, cty)) + geom_point()`
  - `dev.off()`

- `ggsave()`: save the ggplot plot to a file, it can automatically select the graphics device based on the file extension
  - `plot = ggplot(mpg, aes(displ, cty)) + geom_point()`
  - `ggsave("output.pdf", plot)`

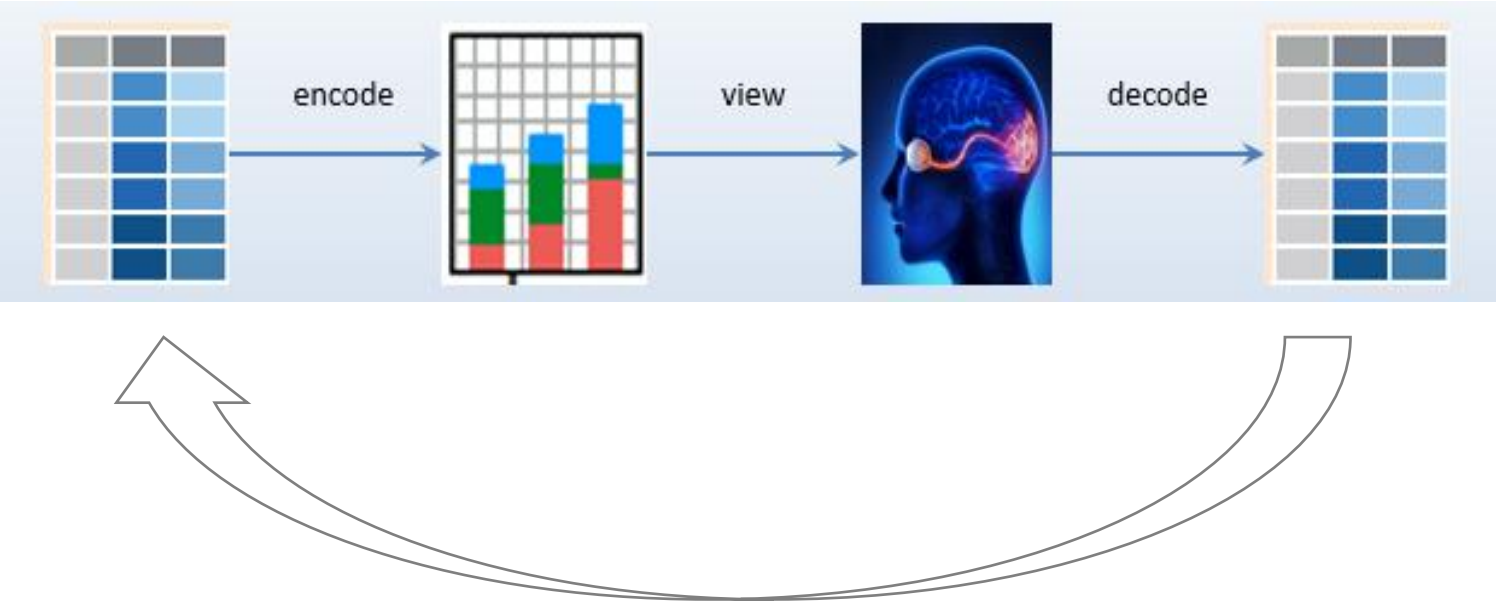
  Parameters
  - width and height: control the output size, specified in inches. If left blank, they'll use the size of the on-screen graphics device.
  - dpi: control the resolution of plots for raster graphics
  - plot: ggplot object. If omitted, it will save the last plot




# Principles of visualization

# Effective data visualization

- A game of perception and cognition
  - Perception: Processing signals coming in – what you see
  - Cognition: How you understand and interpret what you see

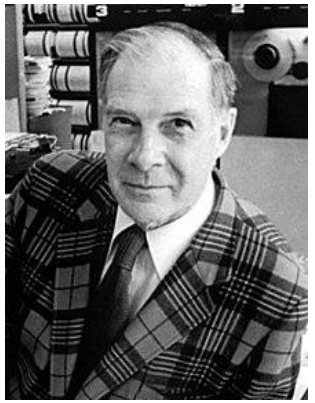



Minimize the reconstruction loss?



Image from Internet

# Think about the message you want to convey



Richard Hamming
(1915-1998)

*"The purpose of (scientific) computing is insight, not numbers"*

—— *Numerical Methods for Scientists and Engineers* (1962)

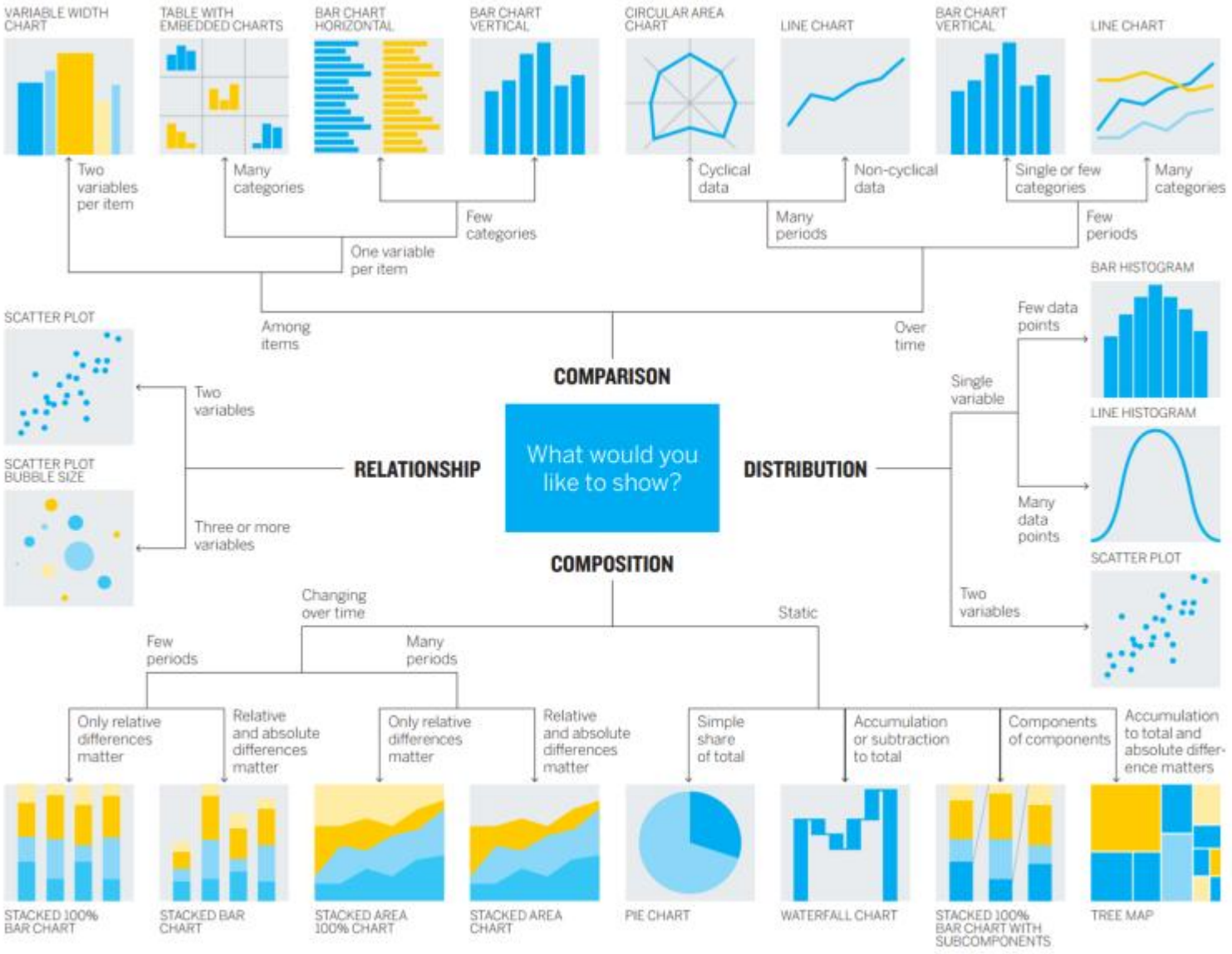
The purpose of plotting is message, not .jpeg/.png/.svg/.pdf…

—— *UCLA QCBio Collaboratory workshop…* (2024)*



*: just kidding, we did not invent this quote, it's more like a common sense

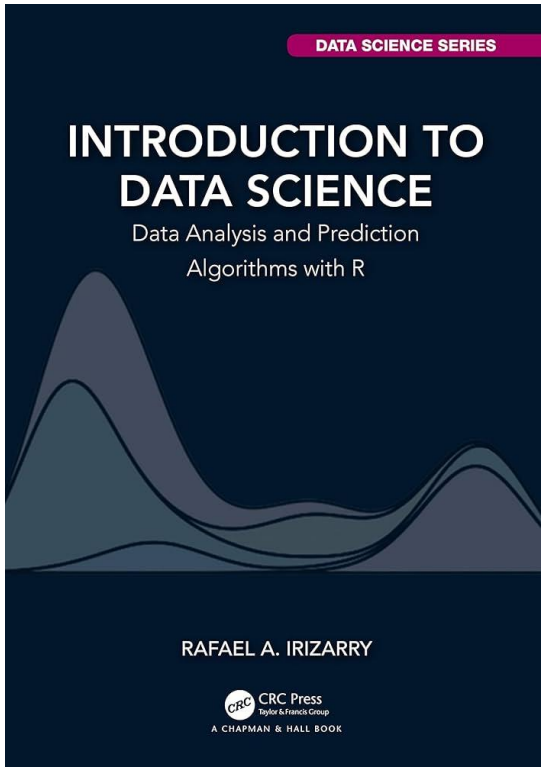
# Decide on the right type of plot




For more information, please refer to the R Graph Gallery

# Principles for data visualization

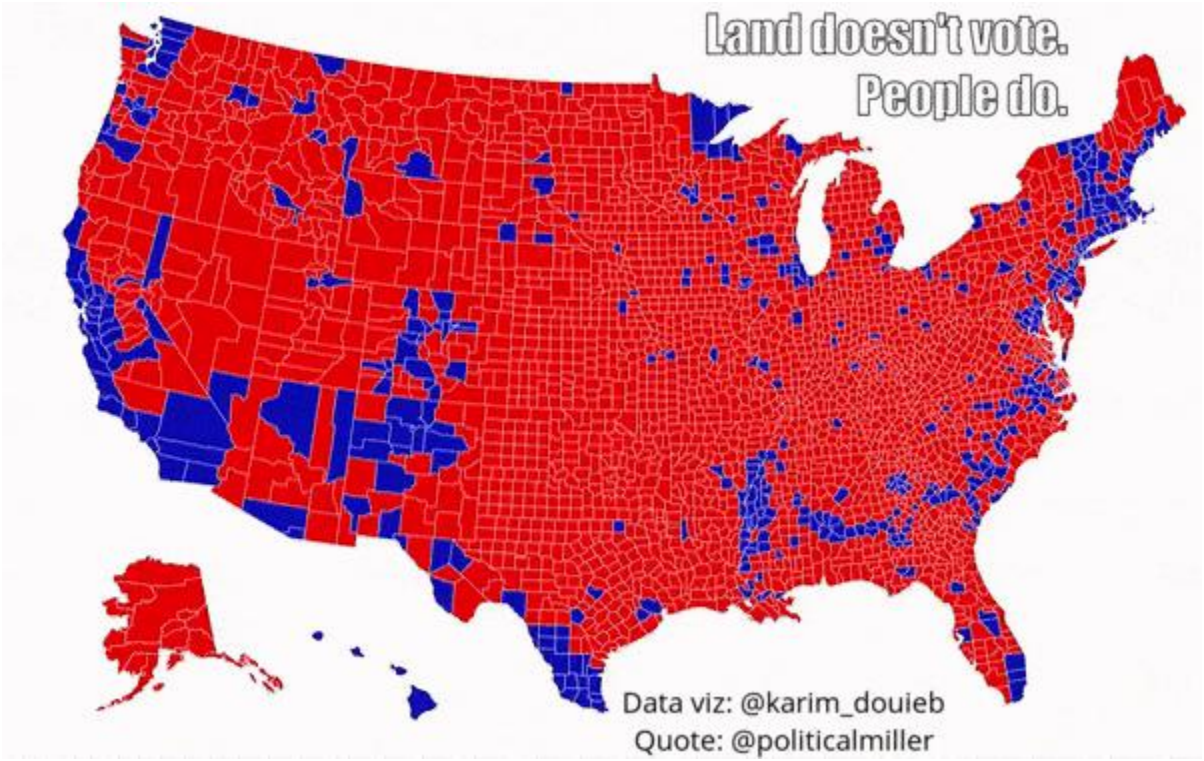
- Use appropriate representations
- Add necessary annotation
- Don’t mislead the reader
- Order categories by a meaningful value
- Show the data
- Reduce image processing
- Make comparisons easy
- Be mindful about the color
- Avoid pseudo-three-dimensional plots



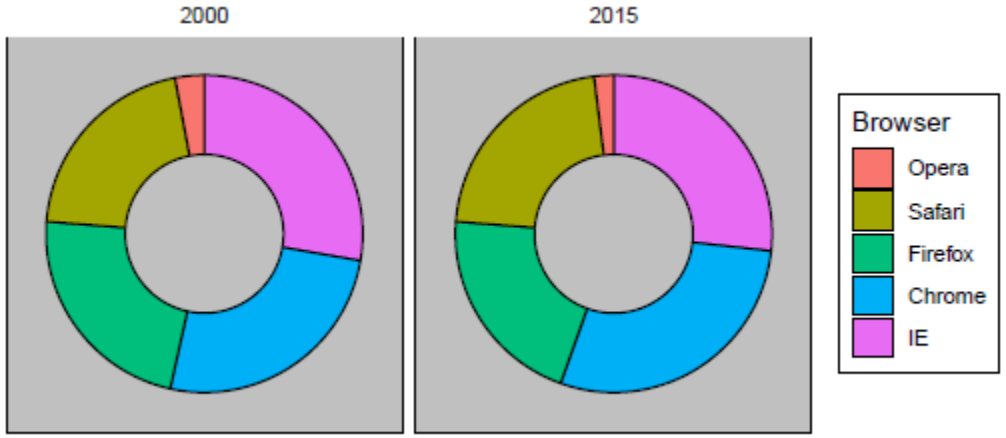

link

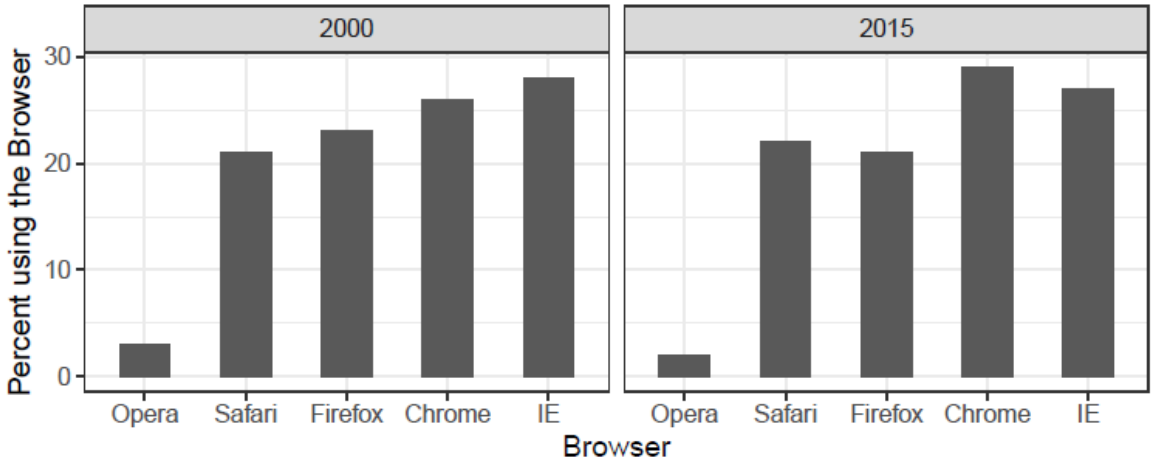
# Use appropriate representations

- Different representations have different focuses



# Use appropriate representations

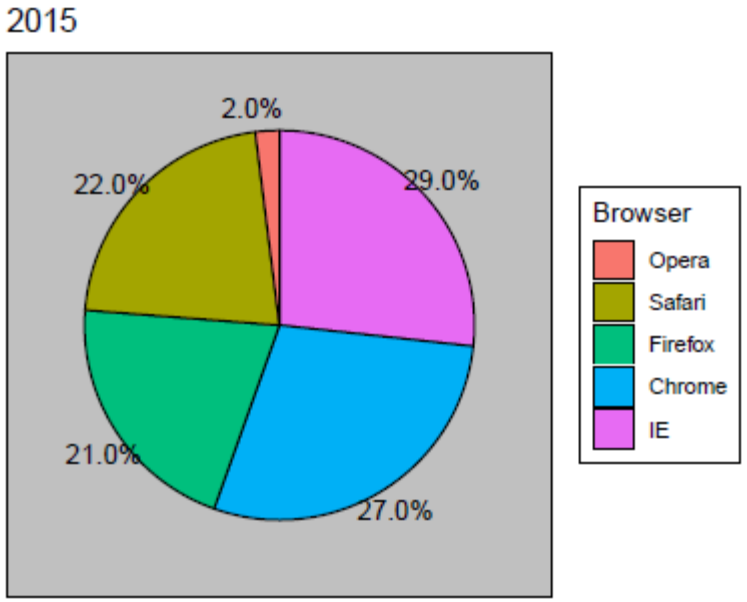


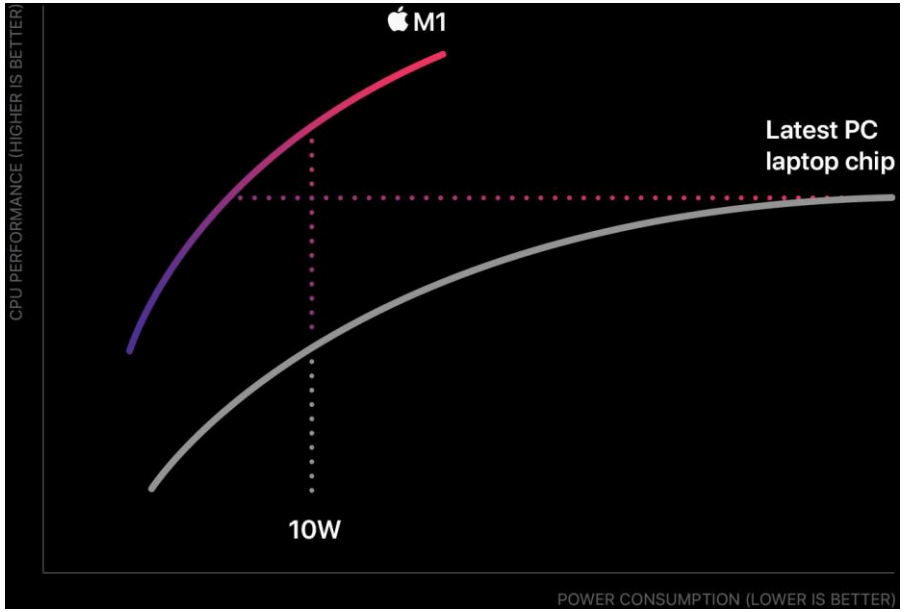

Our eyes are good at judging linear measures and bad at judging relative areas

In general, when displaying quantities, position and length are preferred over angles and/or area. Brightness and color are even harder to quantify than angles.

# Add necessary annotation



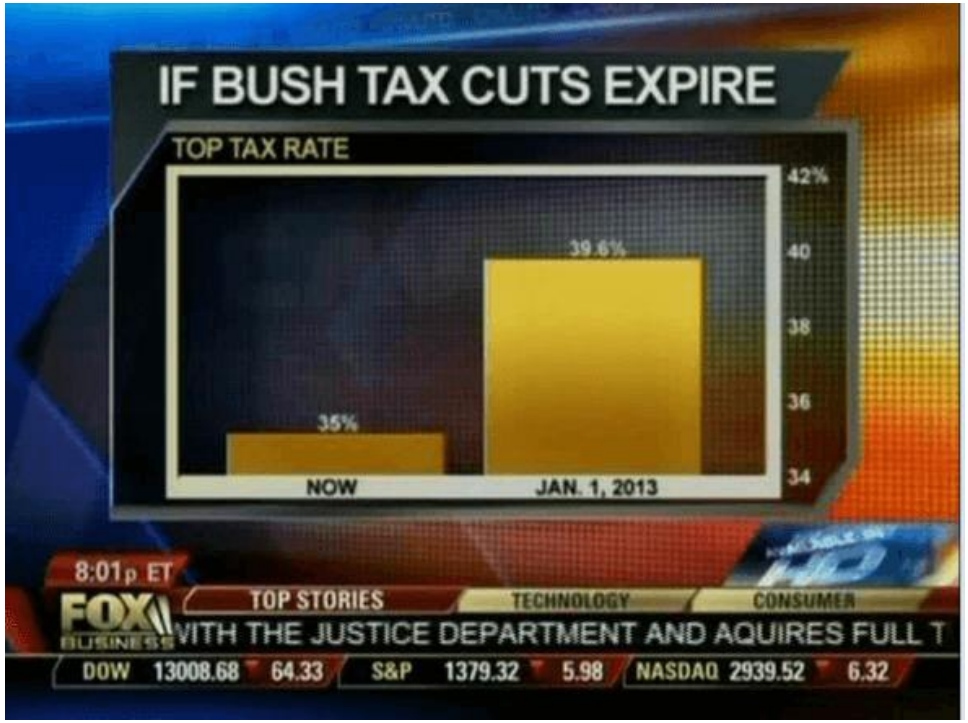

Example 1

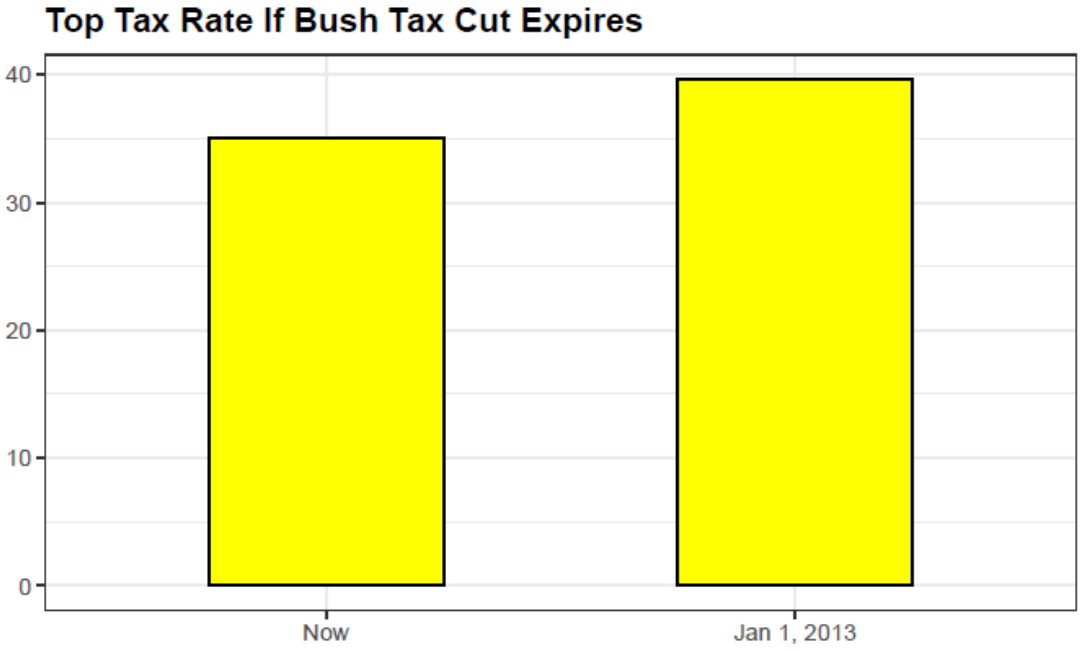



Example 2

Add necessary annotation to help better understand the plots

# Don’t mislead the reader

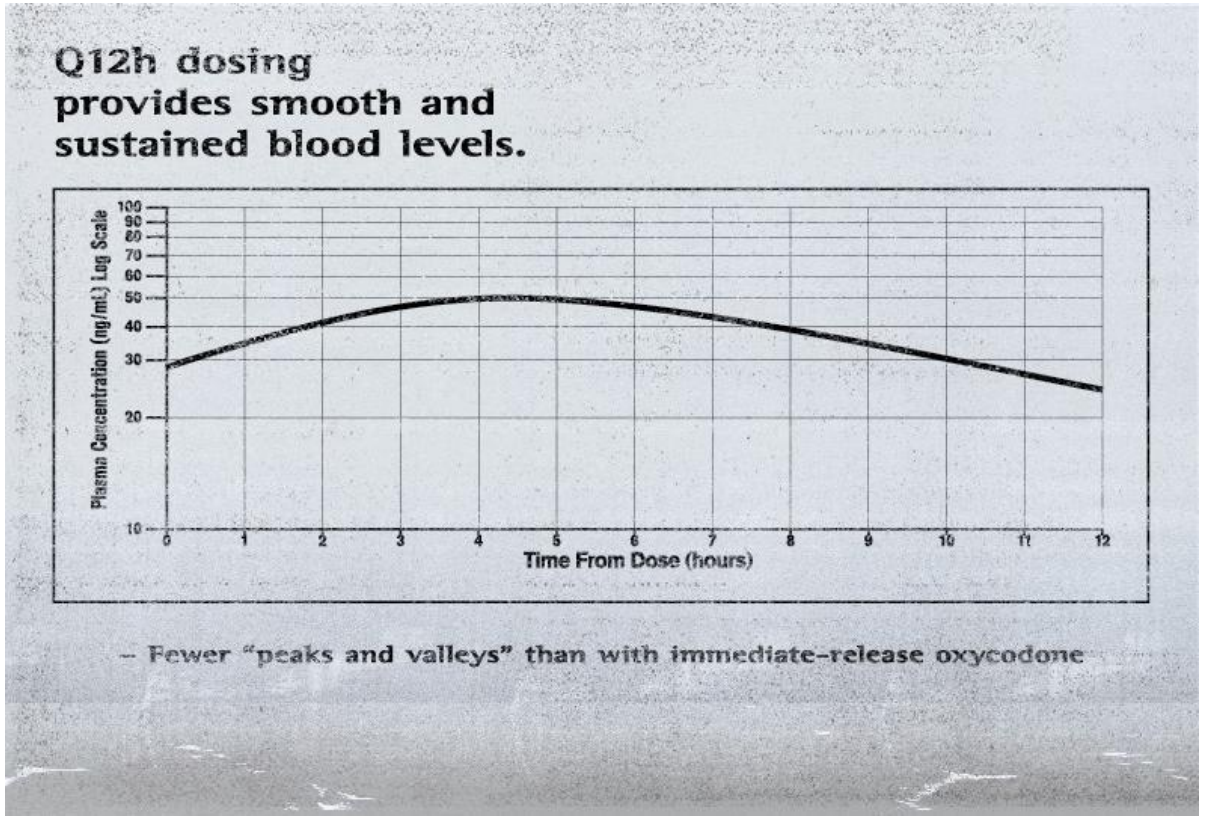



(39.6%-35%)/35% = 13% increase

When using barplots, it is misinformative not to start the bars at 0

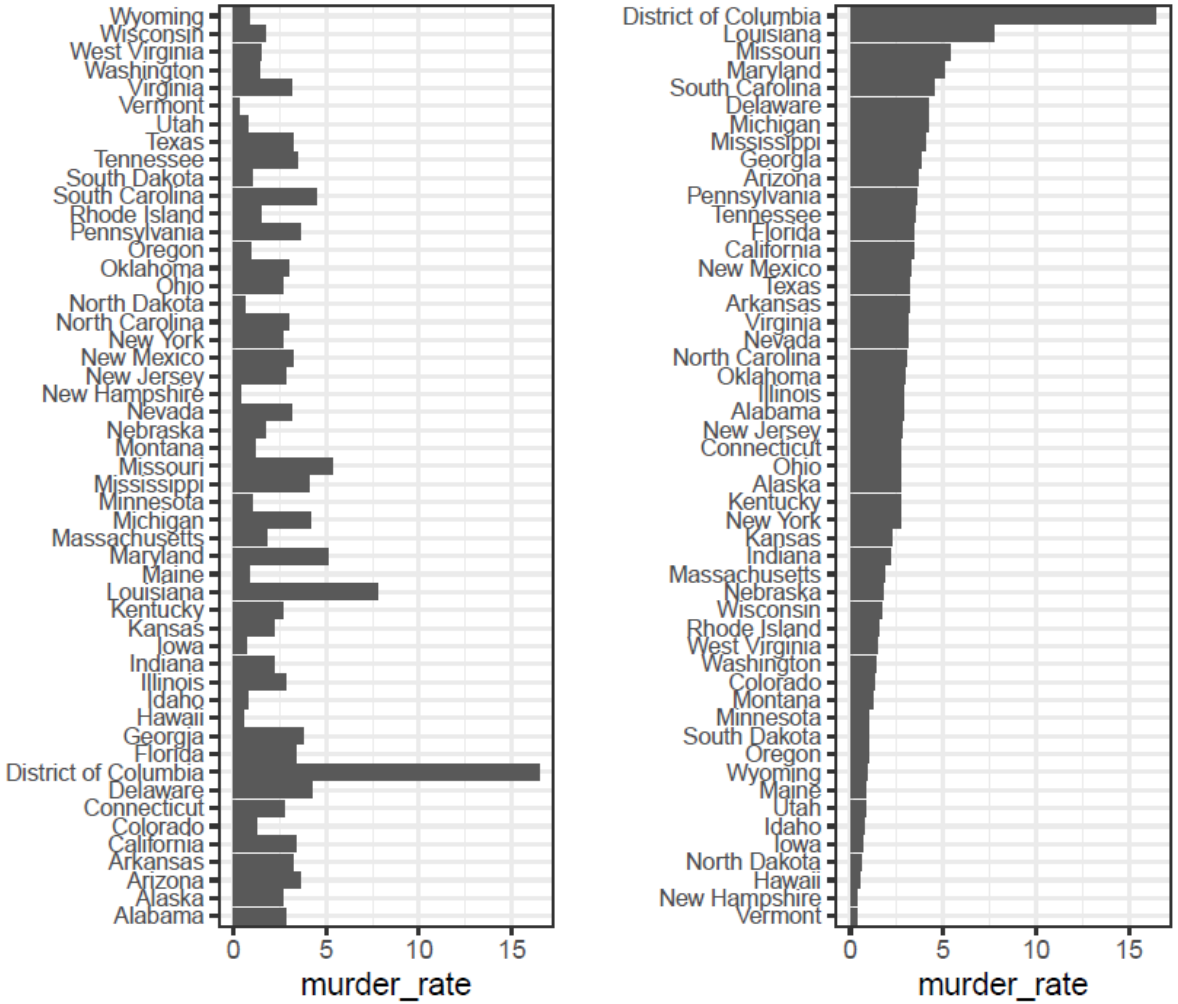
# Don’t mislead the reader

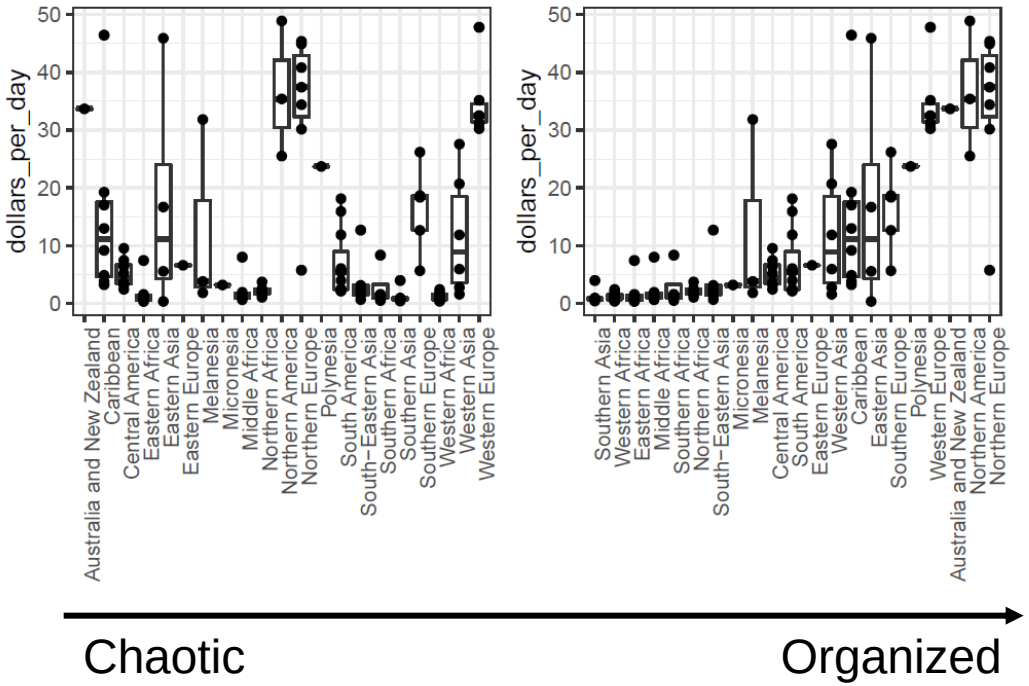




Seeing is not believing, be skeptical !!

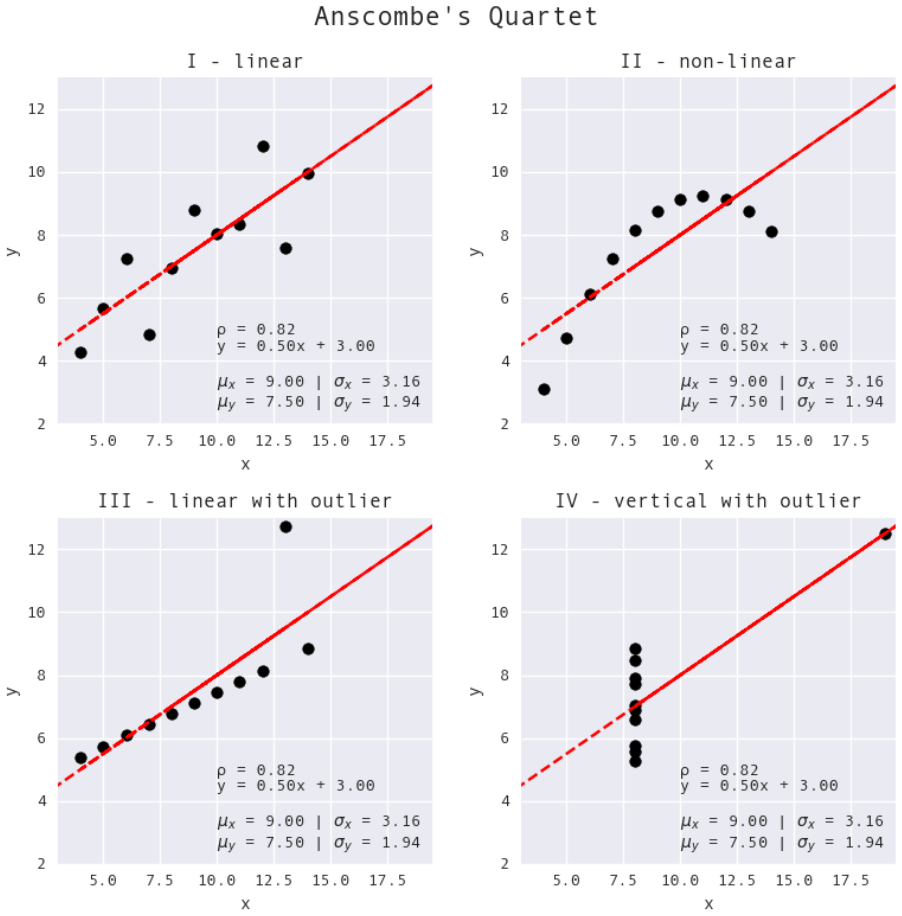
# Order categories by a meaningful value

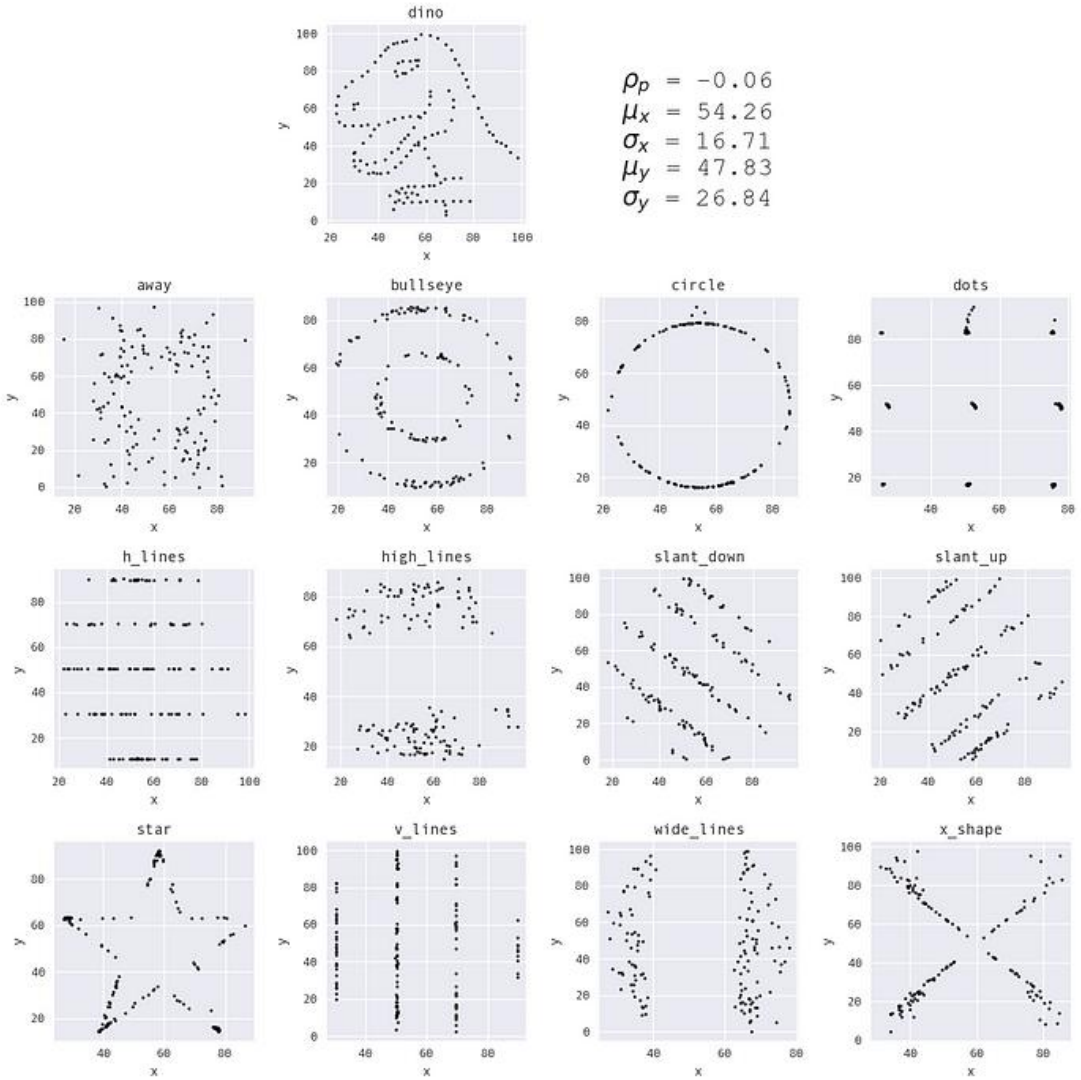
the default **ggplot2** behavior is to order the categories alphabetically



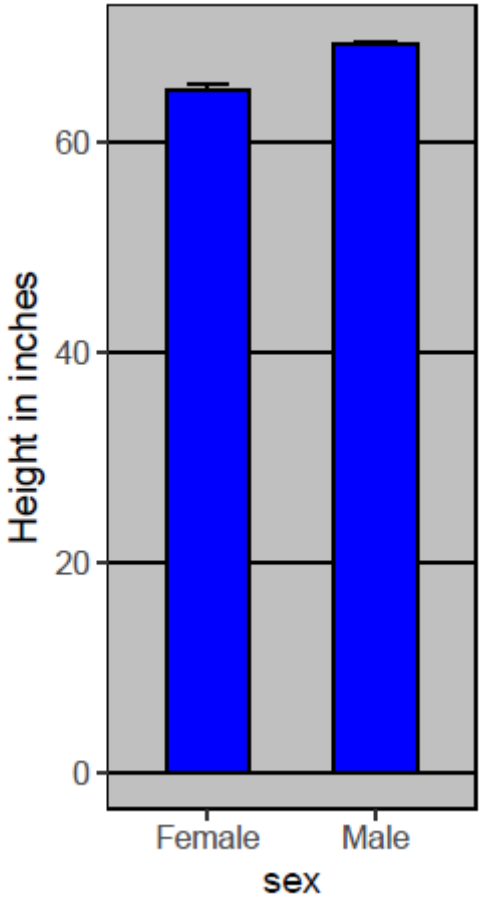
# Show the data

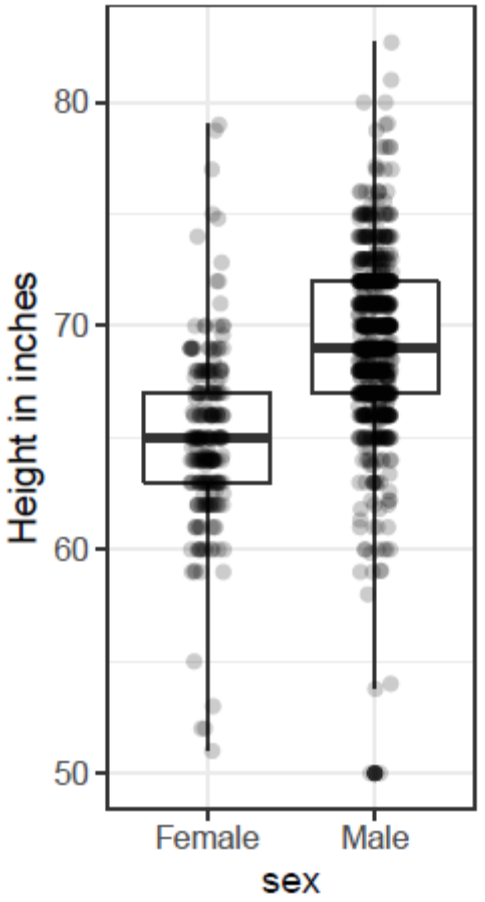



Don’t purely rely on summary statistics, it’s important to show the data

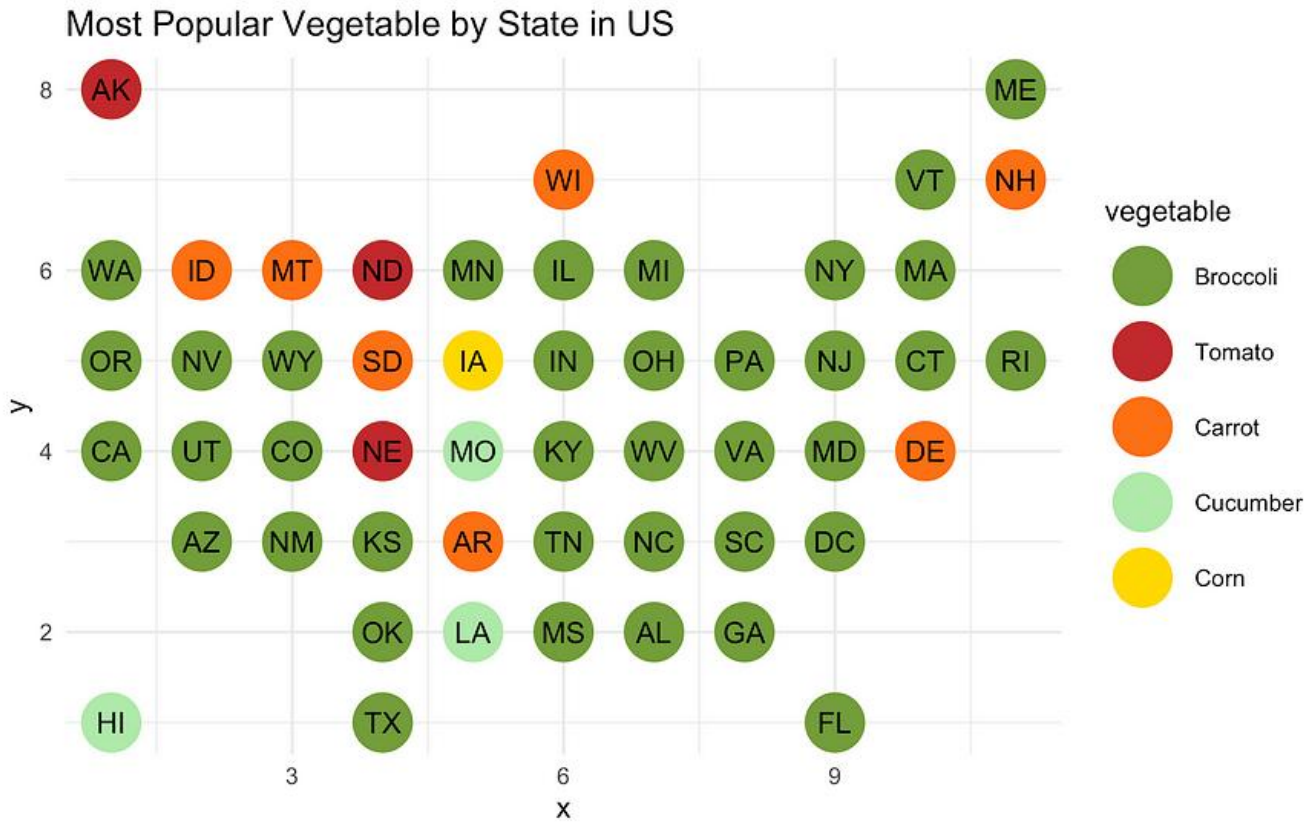
# Show the data




Visualizing the distribution is much more informative than summary statistics alone

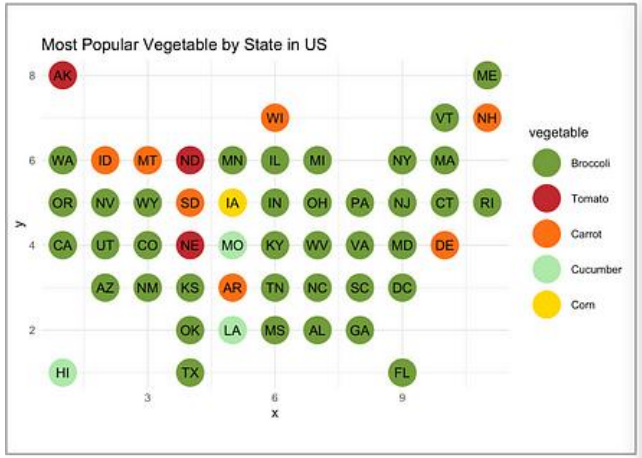
# Reduce image processing

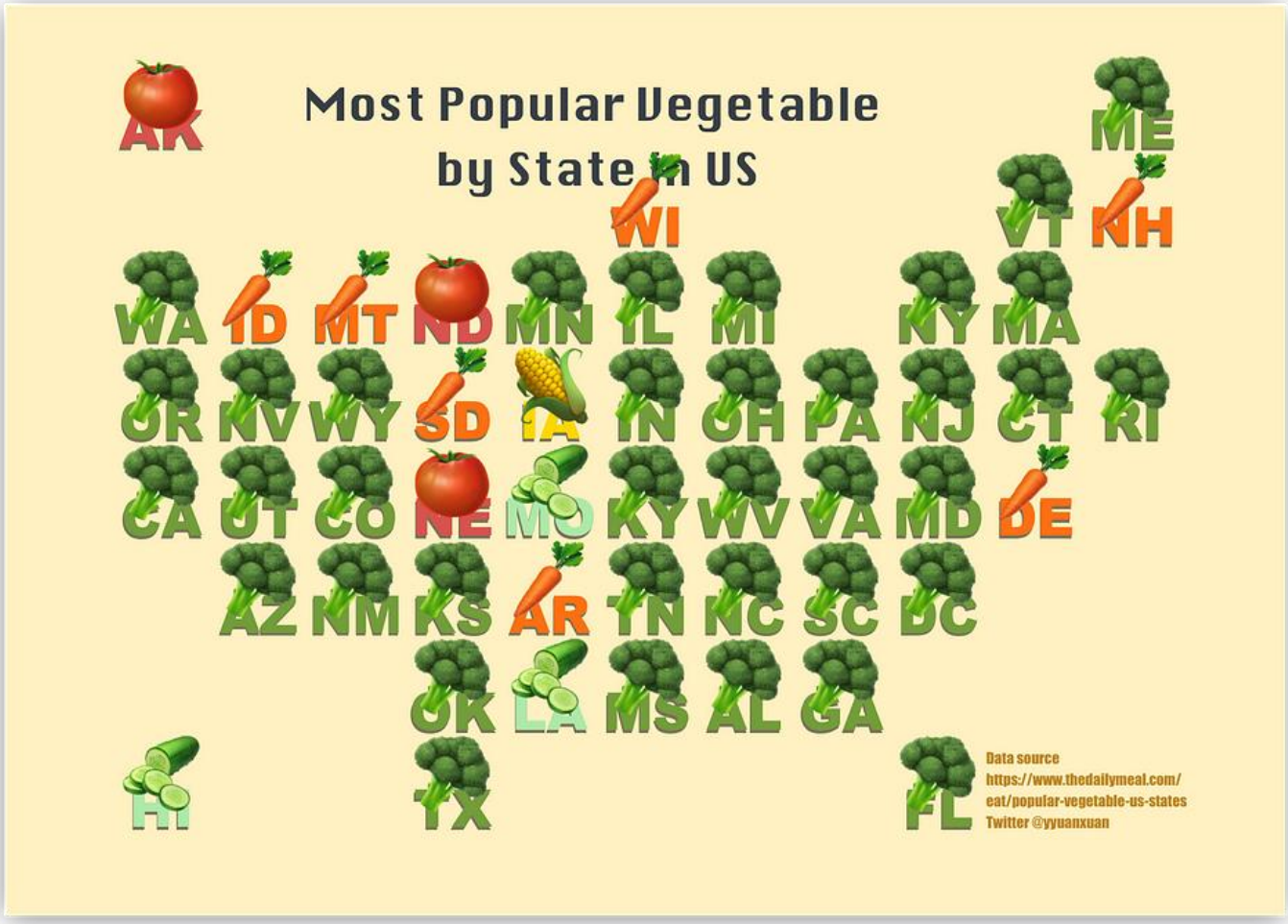



Find the most popular vegetable of a specific state:

- Locate state -> get color -> get vegetable name

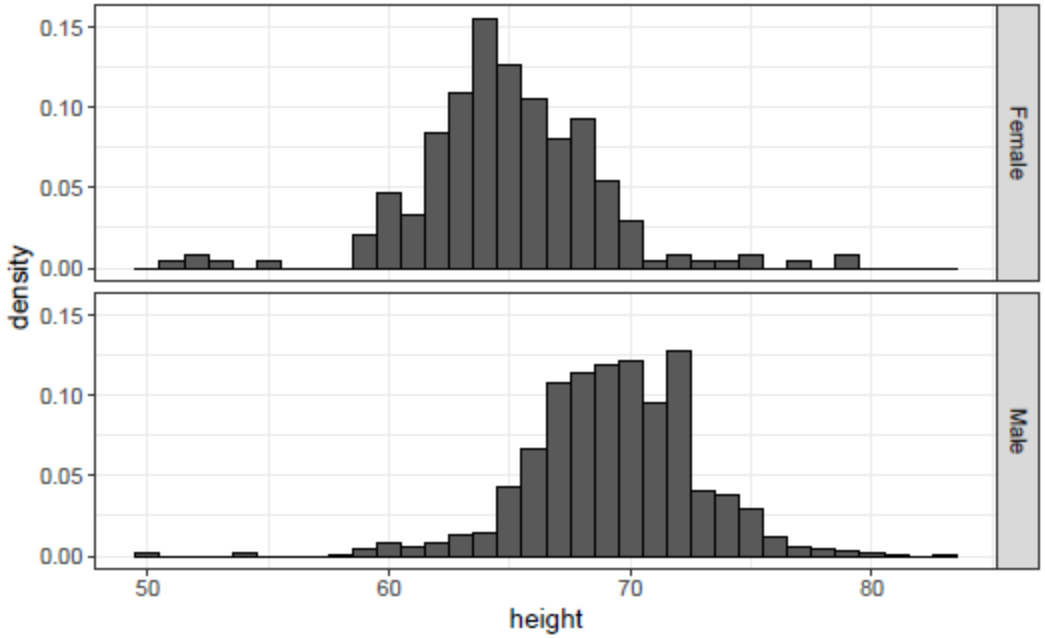
# Reduce image processing



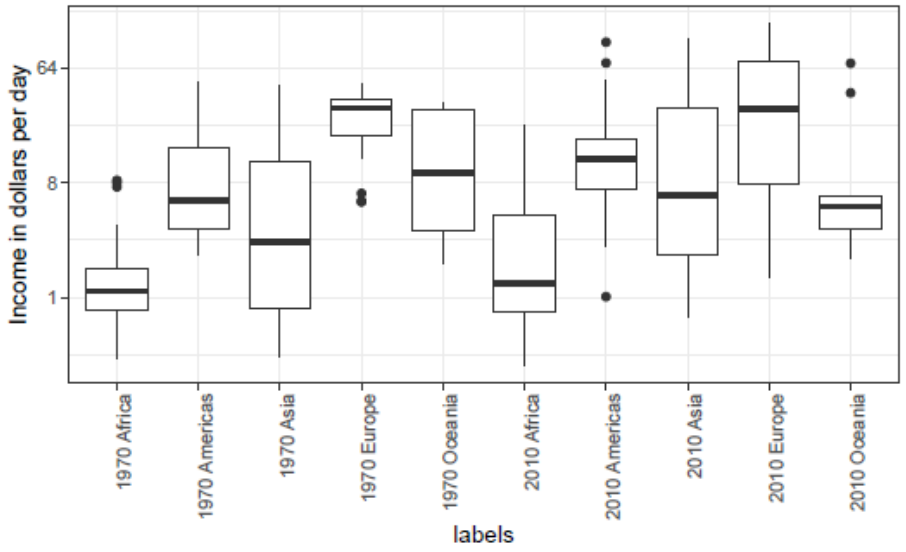
# Make comparisons easy

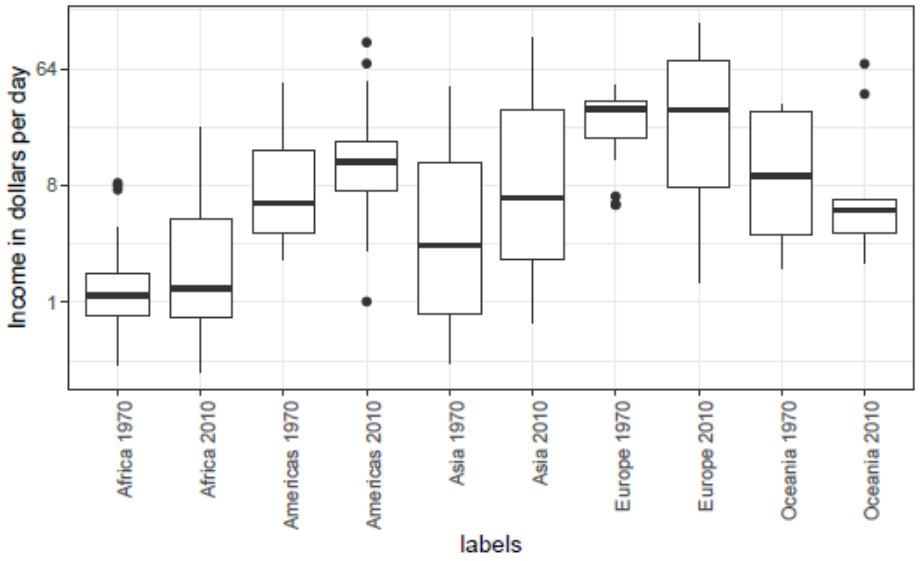
- An important principle here is to **keep the axes the same** when comparing data across two plots.
- **Align plots vertically to see horizontal changes and horizontally to see vertical changes**



# Make comparisons easy

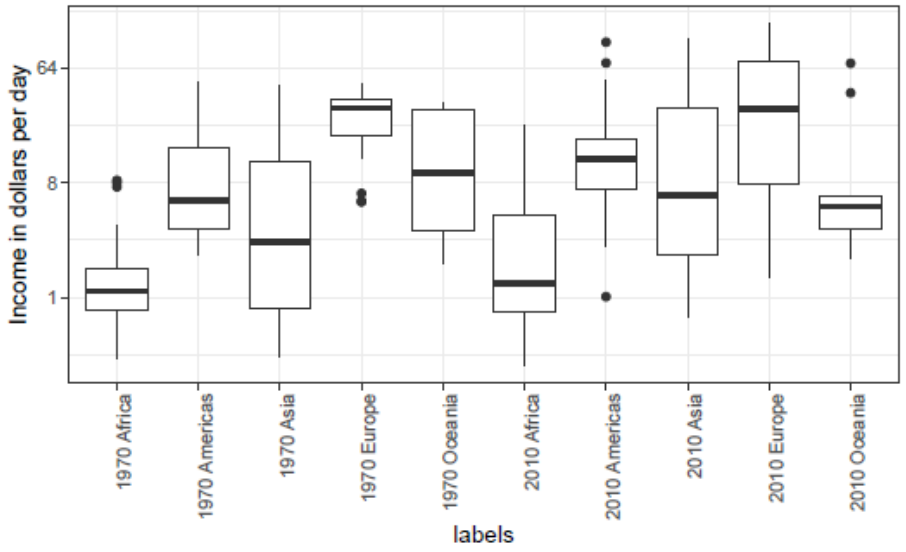
- An important principle here is to **keep the axes the same** when comparing data across two plots.
- **Align plots vertically to see horizontal changes and horizontally to see vertical changes**
- **Visual cues to be compared should be adjacent**

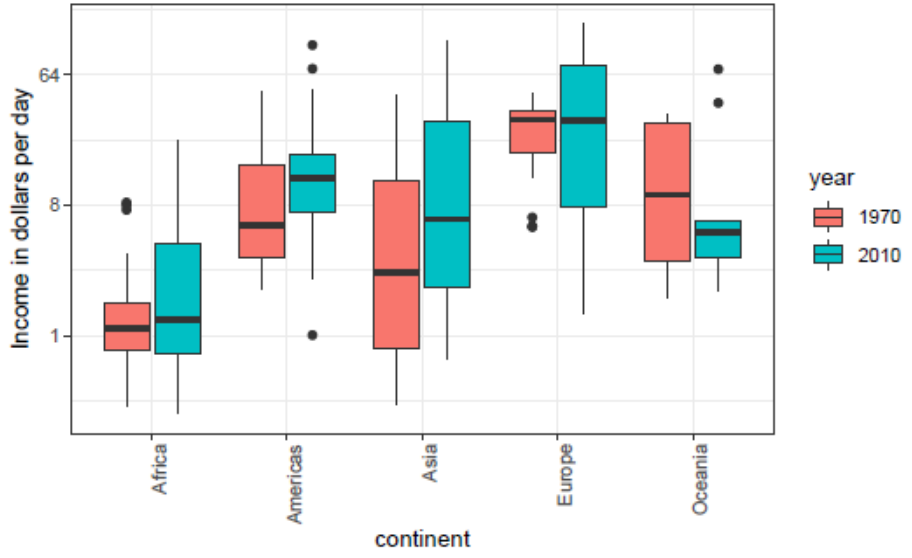


# Make comparisons easy

- An important principle here is to **keep the axes the same** when comparing data across two plots.
- **Align plots vertically to see horizontal changes and horizontally to see vertical changes**
- **Visual cues to be compared should be adjacent**
- **Use color**



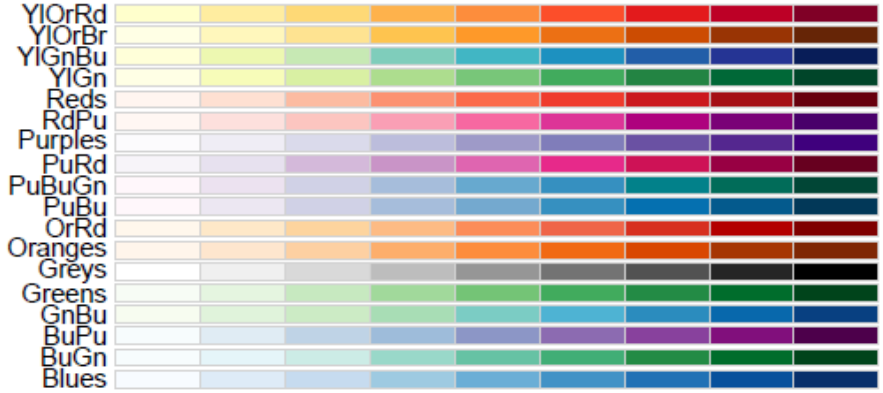



# Be mindful about color

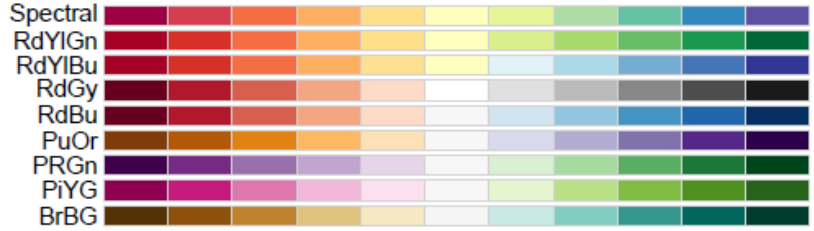
Select the color that suits your message best

- Sequential colors are suited for data that goes from high to low, with high values more clearly distinguished from low values

```
library(RColorBrewer)
```




- Diverging colors are used to represent values that diverge from a center. We put equal emphasis on both ends of the data range: higher than the center and lower than the center



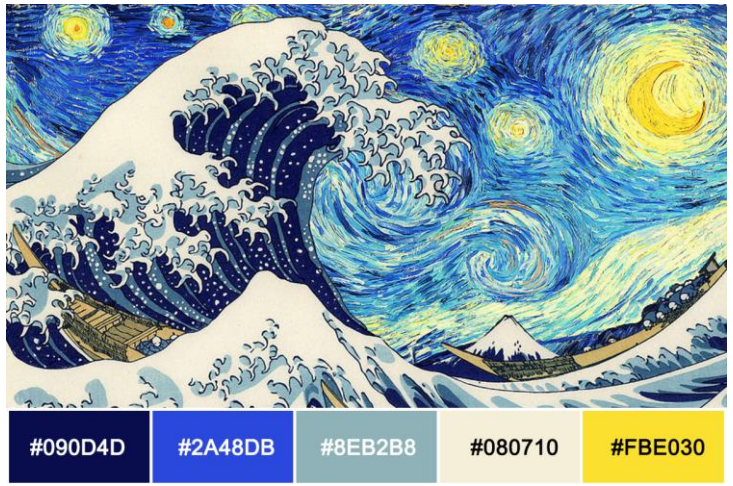
# Wes Anderson palettes



check this package: https://github.com/karthik/wesanderson

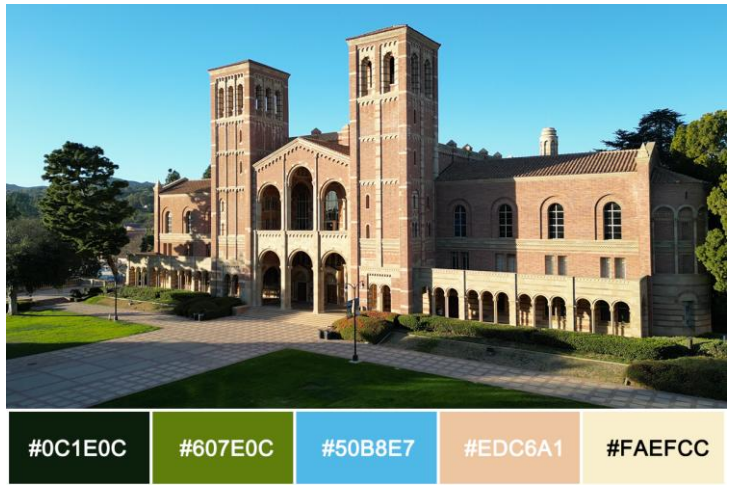
# Color from daily life (movies, artwork, etc.)


La La Land (2016)




Starry night & The Great Wave


The Truman Show (1998)




UCLA Royce Hall

# Be mindful about color

Color theory


Which button makes you more inclined to buy a product?


# Be mindful about color

Be mindful about your audience (~4.5% people are color blind in the world)

Use colorblind-friendly palette:

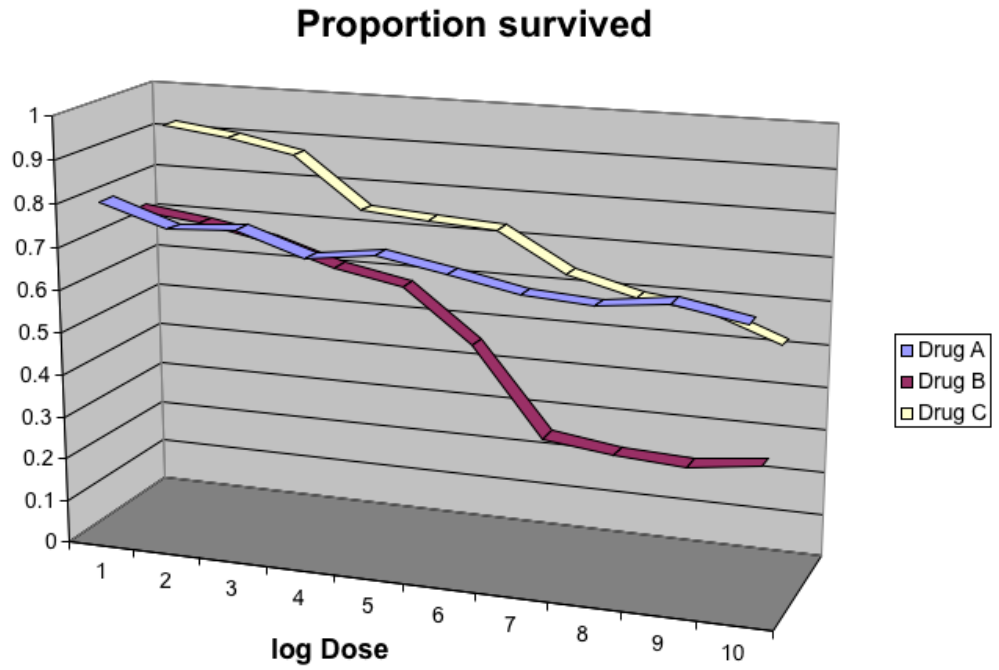
- `c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")`

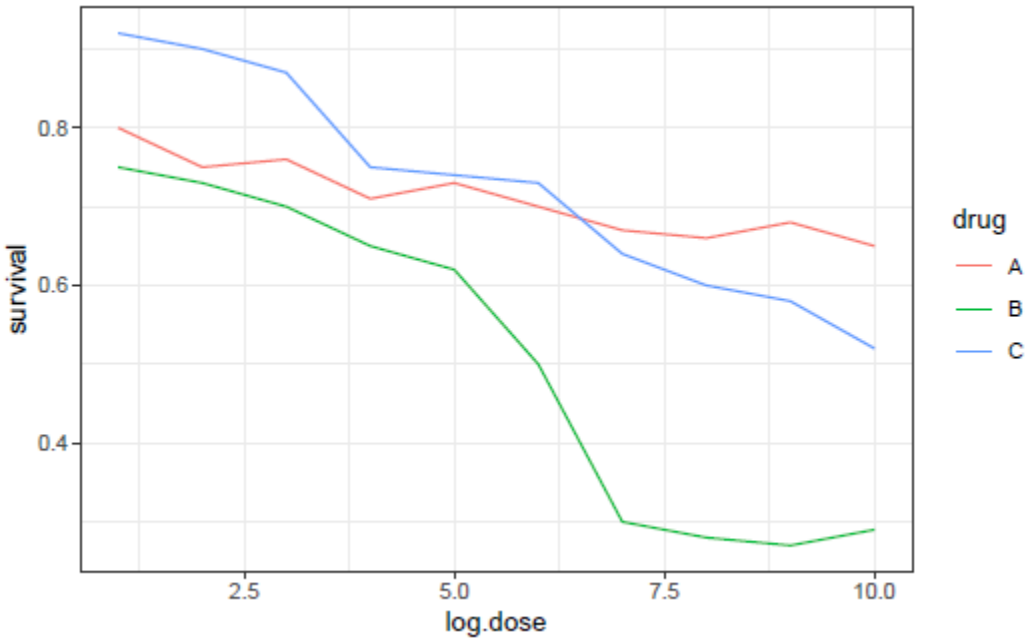
- `c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")`



http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/#a-colorblind-friendly-palette

# Avoid pseudo-three-dimensional plots



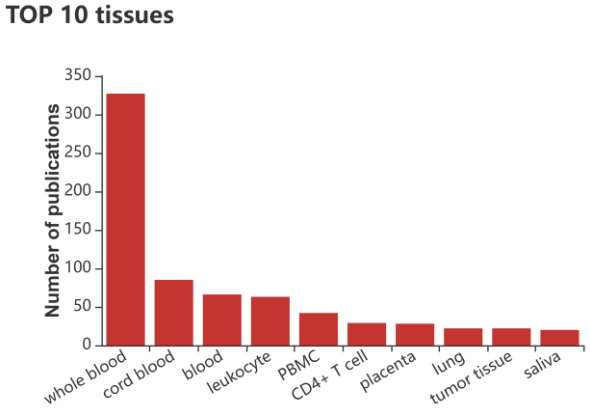

Humans are not good at seeing in three dimensions (which explains why it is hard to parallel park)

# Where to get help?

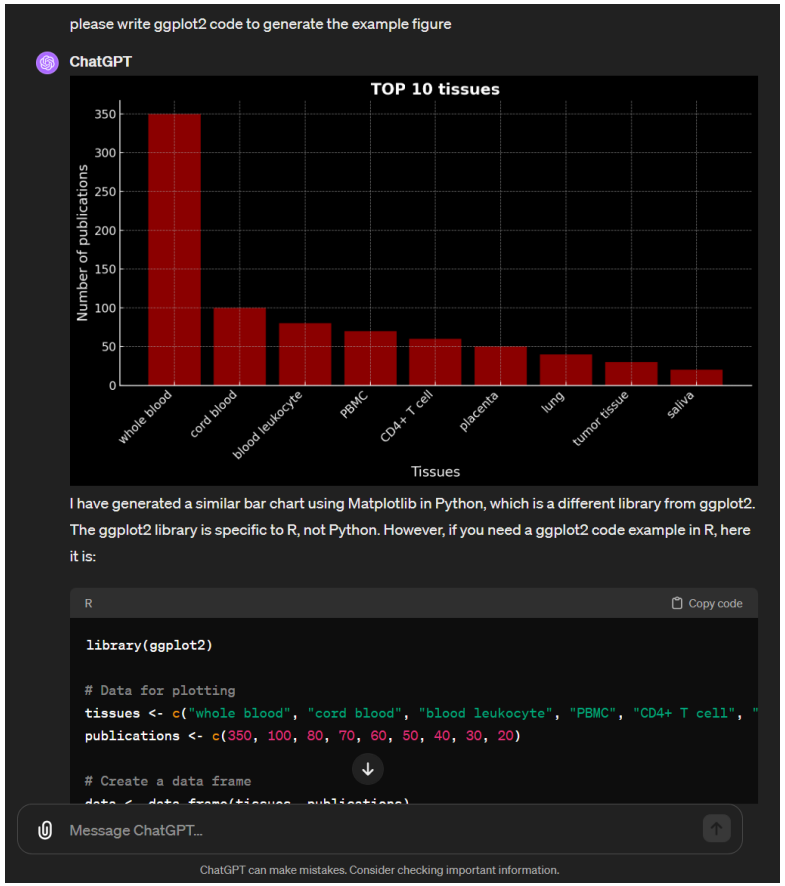
- https://community.rstudio.com/tag/ggplot2
- https://r-graph-gallery.com/
- https://www.google.com
- https://stackoverflow.com
- https://chat.openai.com/

# Ask ChatGPT to help you make the plot

**Example figure**




**Ask ChatGPT**




**customize**

**Code template by ChatGPT**

```
library(ggplot2)

# Data for plotting
tissues <- c("whole blood", "cord blood", "blood leukocyte", "PBMC",
             "CD4+ T cell", "placenta", "lung", "tumor tissue", "saliva")
publications <- c(350, 100, 80, 70, 60, 50, 40, 30, 20)

# Create a data frame
data <- data.frame(tissues, publications)

# Create a ggplot2 bar chart
ggplot(data, aes(x=reorder(tissues, -publications), y=publications, fill=tissues)) +
  geom_bar(stat="identity") +
  theme_minimal() +
  theme(
    text = element_text(colour = "white"),
    plot.background = element_rect(fill = "black", color = NA),
    panel.background = element_rect(fill = "black", color = NA),
    axis.title = element_text(color = "white"),
    axis.text = element_text(color = "white"),
    legend.position = "none"
  ) +
  labs(
    title = "TOP 10 tissues",
    x = "Tissues",
    y = "Number of publications"
  ) +
  scale_fill_manual(values=rep("darkred", length(tissues)))
```

**Return the code**


# Good luck and have fun!

(although the picture is **not** a good example of effective data visualization)



Image created by ChatGPT