

# Advanced data visualization with `ggplot2`

Wenbin Guo  
Bioinformatics IDP, UCLA  
[wbguo@ucla.edu](mailto:wbguo@ucla.edu)  
2024 Fall

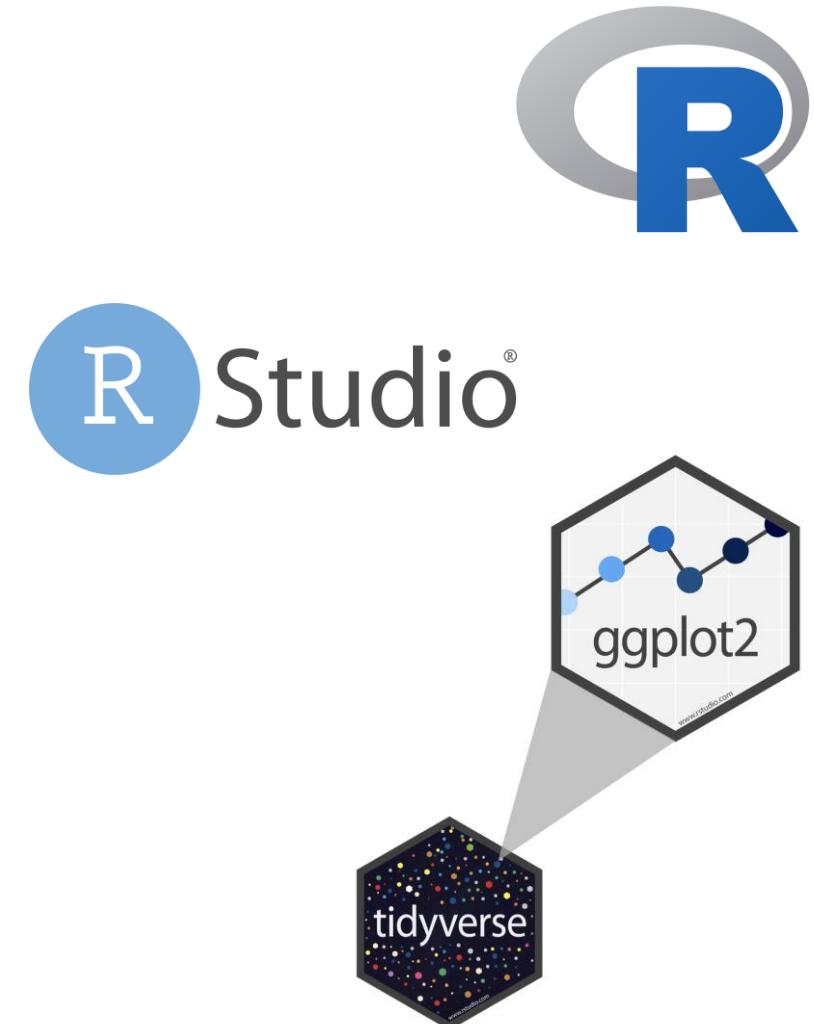
# Notation of the slides

- 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



# Day 3: Examples and useful packages

Wenbin Guo  
Bioinformatics IDP, UCLA  
[wbguo@ucla.edu](mailto:wbguo@ucla.edu)  
2024 Fall

# Overview

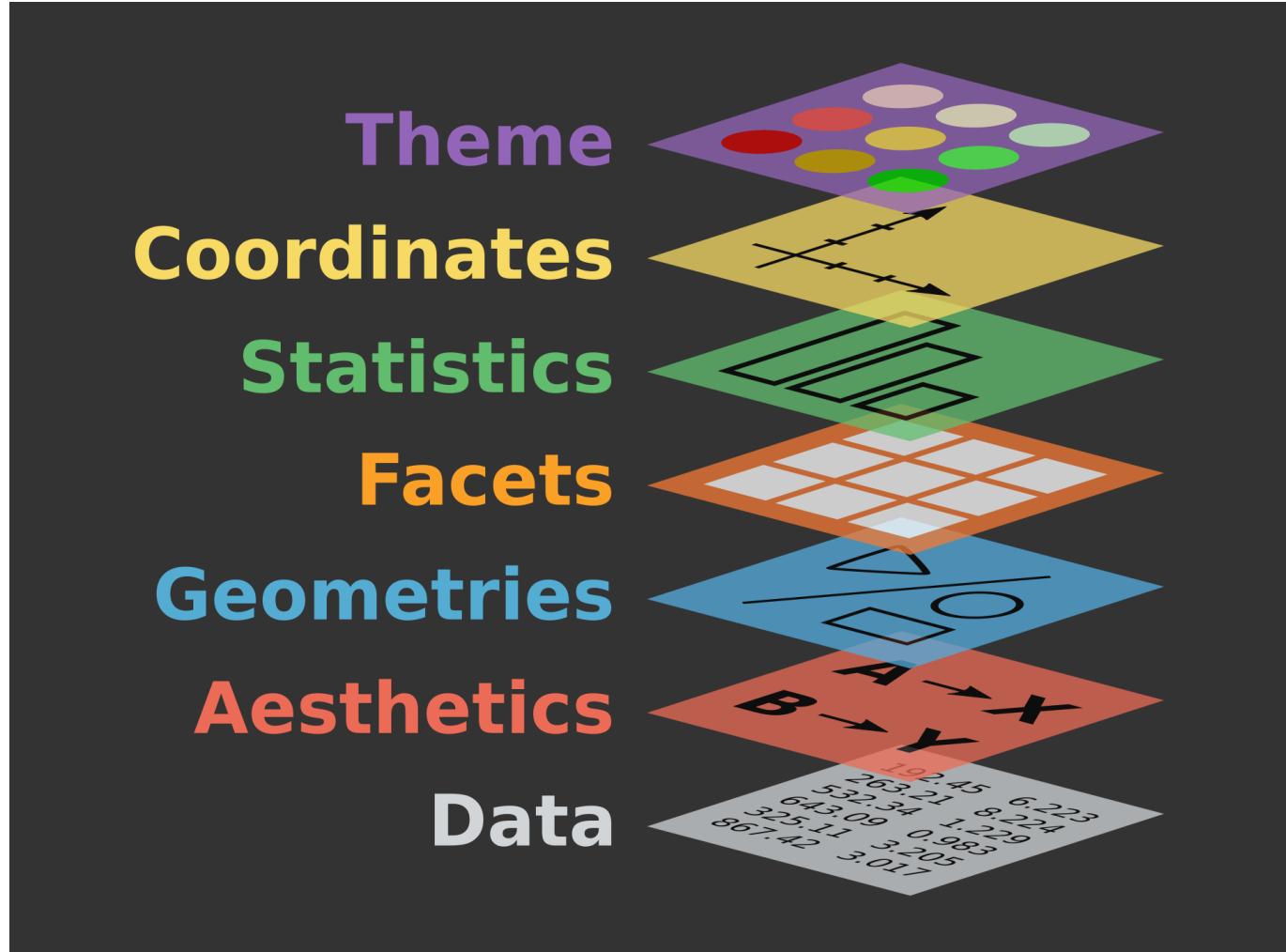
## Time

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

## Topics

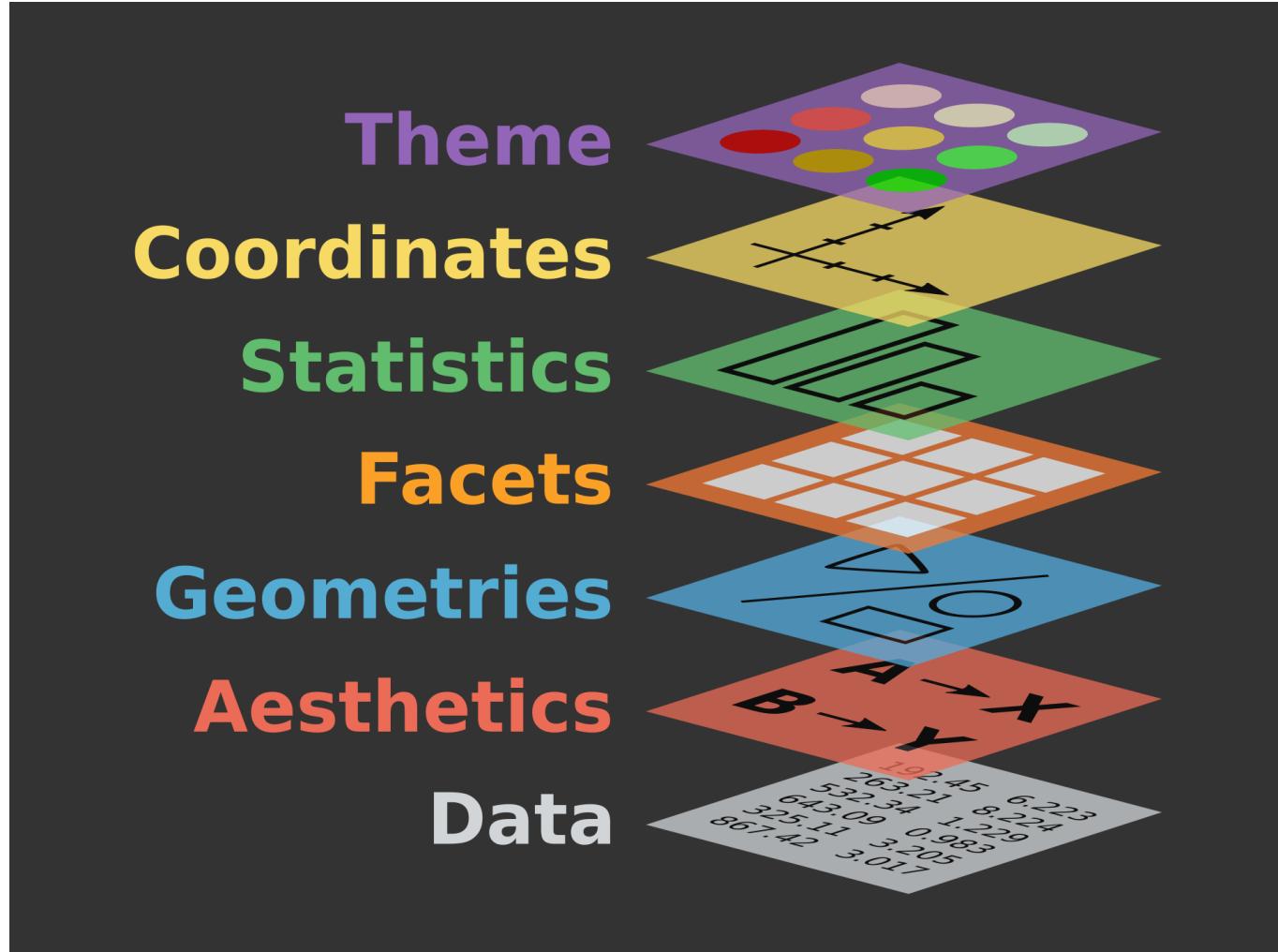
- ❑ Legends
- ❑ Themes
- ❑ Principles of data visualization
- ❑ Useful packages

# 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>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),  
    stat = <STAT>, position = <POSITION>) +  
  <COORDINATE_FUNCTION> +  
  <FACET_FUNCTION> +  
  <SCALE_FUNCTION> +  
  <THEME_FUNCTION>
```

required  
Not required, sensible defaults supplied

## 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	<code>ggplot(data)</code>
Aesthetics	<code>aes()</code>
Layers	<code>geom_*</code> and <code>stat_*</code>
Scales	<code>scale_*</code>
Coordinate System	<code>coord_*</code>
Facets	<code>facet_*</code>
Visual Themes	<code>theme()</code> and <code>theme_*</code>

# 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

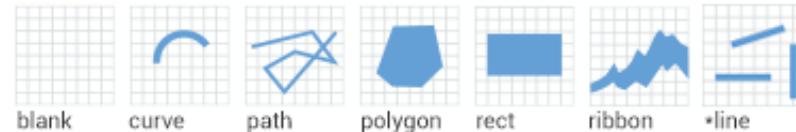
- Describes how **variables** are mapped to **visual properties** of geometric objects

➤ `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

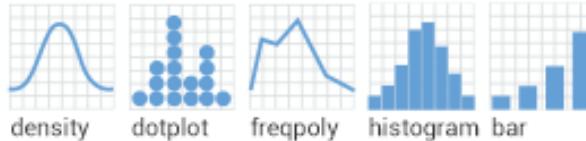
- Data
- Aesthetic mappings
- Geometry



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
	geom_rect()	draw rectangles
Region	geom_polygon()	draw filled polygons
	geom_ribbon()	draw ribbons, a path with vertical thickness

# Summary – Day1&2

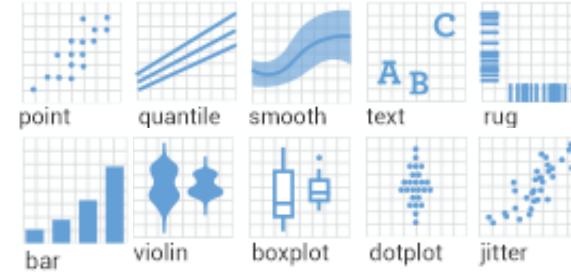
- Data
- Aesthetic mappings
- Geometry



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

# Summary – Day1&2

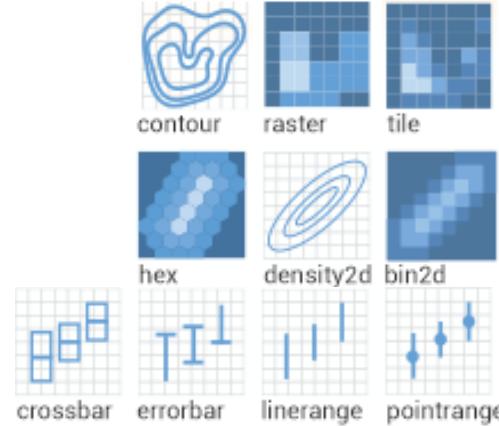
- Data
- Aesthetic mappings
- Geometry



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	<code>geom_contour()</code>	display contour
	<code>geom_tile()</code>	tile the plane with rectangles
	<code>geom_raster()</code>	fast version of <code>geom_tile()</code> for equal sized tiles
2D distribution	<code>geom_bin2d()</code>	bin into rectangles and count
	<code>geom_hex()</code>	bin into hexagons and count
	<code>geom_density2d()</code>	display smoothed 2d density estimate
Uncertainty	<code>geom_crossbar()</code>	display vertical bar with center
	<code>geom_errorbar()</code>	display error bars
	<code>geom_linerange()</code>	display vertical line
	<code>geom_pointrange()</code>	display vertical line with center

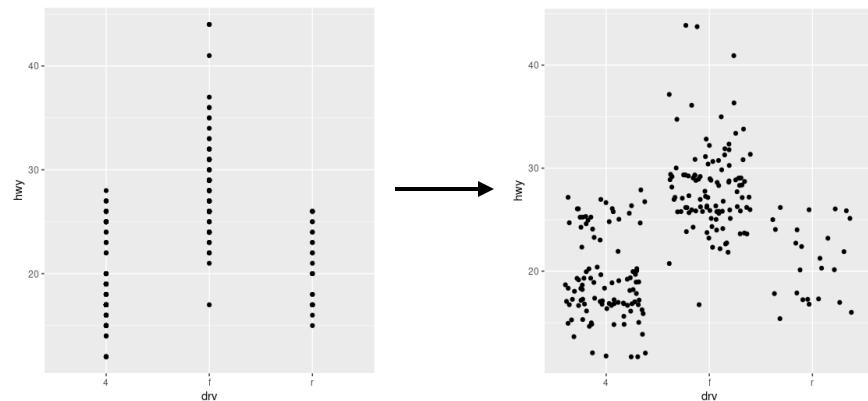
# Summary – Day1&2

- Data
- Aesthetic mappings
- Geometry
- Statistical transformation

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

# Summary – Day1&2

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



Function	Usage
<code>position_nudge()</code>	move points by a fixed offset
<code>position_jitter()</code>	add a little random noise to every position
<code>position_jitterdodge()</code>	dodge points within groups, then add a little random noise
<code>position_stack()</code>	stack overlapping bars (or areas) on top of each other
<code>position_fill()</code>	stack overlapping bars, scaling so the top is always at 1
<code>position_dodge()</code>	place overlapping bars (or boxplots) side-by-side
<code>position_identity()</code>	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

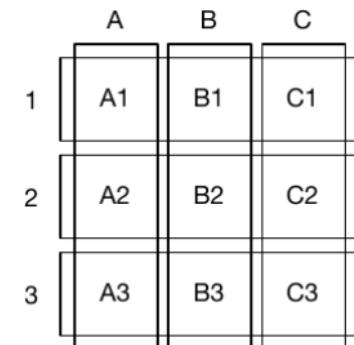
Type: continuous, discrete, ...

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

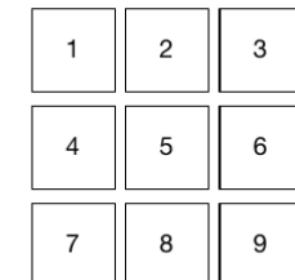
# Summary – Day1&2

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

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



`facet_grid`

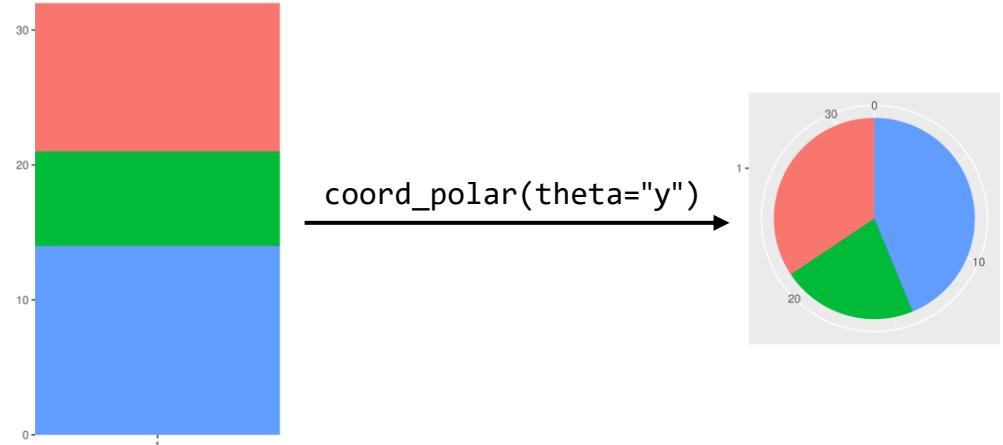


`facet_wrap`

# Summary – Day1&2

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

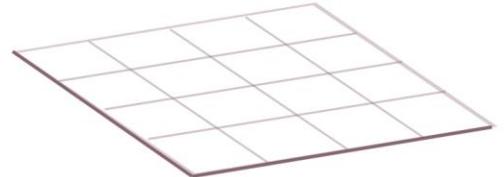
Function	Usage
<code>coord_cartesian(xlim=c(a,b))</code>	zoom into a region
<code>coord_flip()</code>	flip the axis
<code>coord_fixed(ratio = 1)</code>	set x and y with equal scales
<code>coord_trans()</code>	apply transformation to axis
<code>coord_polar()</code>	Convert to polar coordinates



# More customization



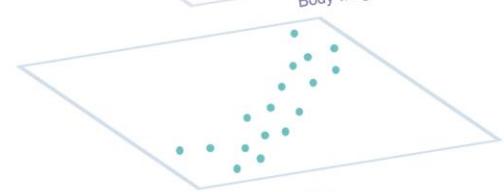
Customise the look of your plot with themes  
(pre-made or your own!);  
+ theme\_bw()



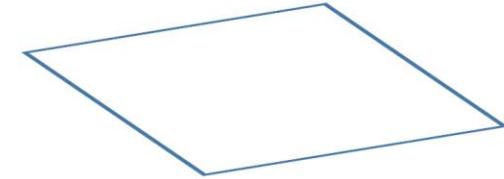
Add labels and titles:  
+ labs(x = "Body weight (g)", y = "Wingspan (cm)",  
title = "Heavy birds have longer wings")



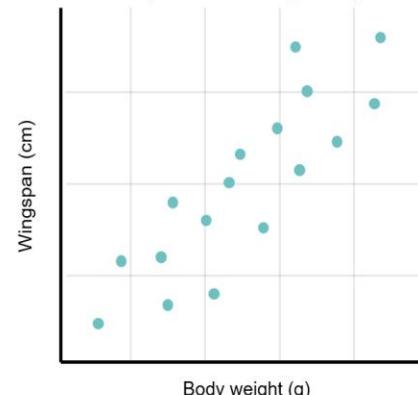
Specify the type of graph and the variables to use:  
+ geom\_point(aes(x = body.weight, y = wingspan))



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



Heavy birds have longer wings



# Los Angeles Transit Map

Diagram by Sean Sirota AKA u/thedogpill

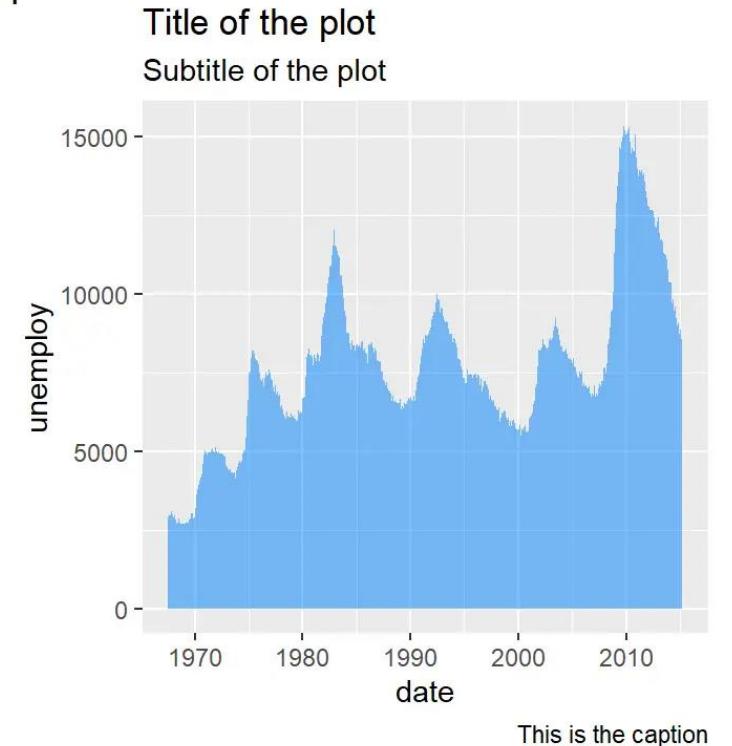


# Plot title

Change the title:

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

Fig. 1



# Axis and legend components

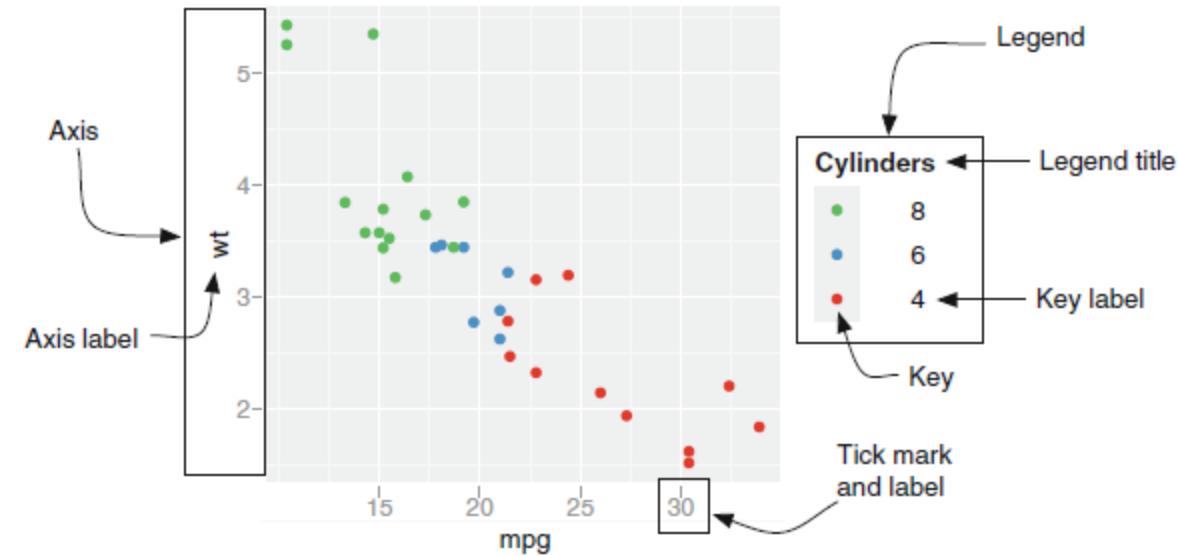
Change axis label:

- `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?

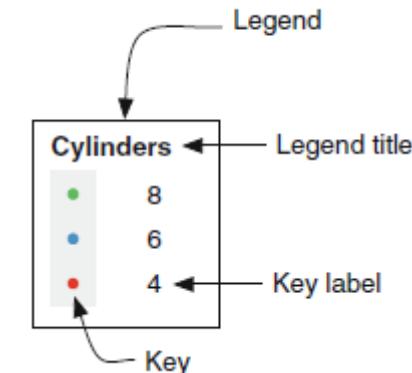


Axis	Legend	Argument name
Label	Title	<code>name</code>
Ticks & grid line	Key	<code>breaks</code>
Tick label	Key label	<code>labels</code>

# Legend

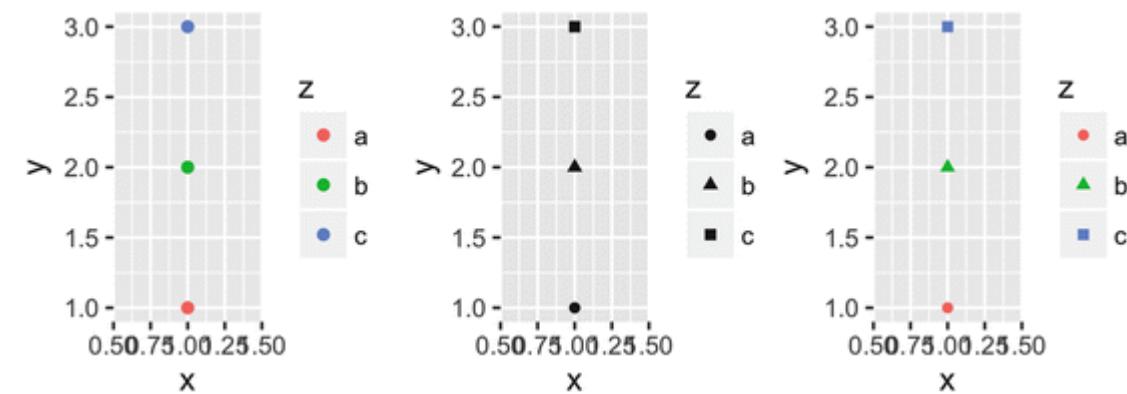
A legend can display multiple aesthetics (e.g. colour and shape) from multiple layers

- `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

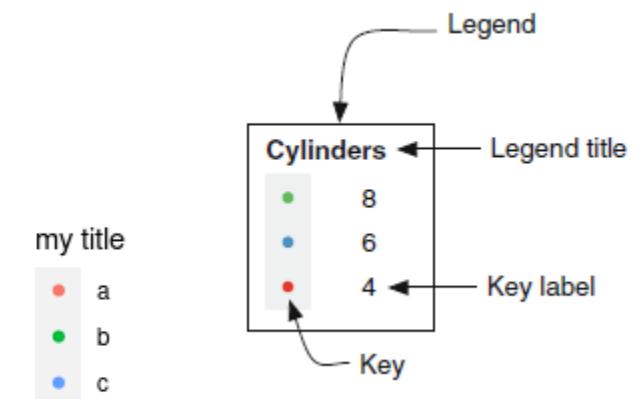
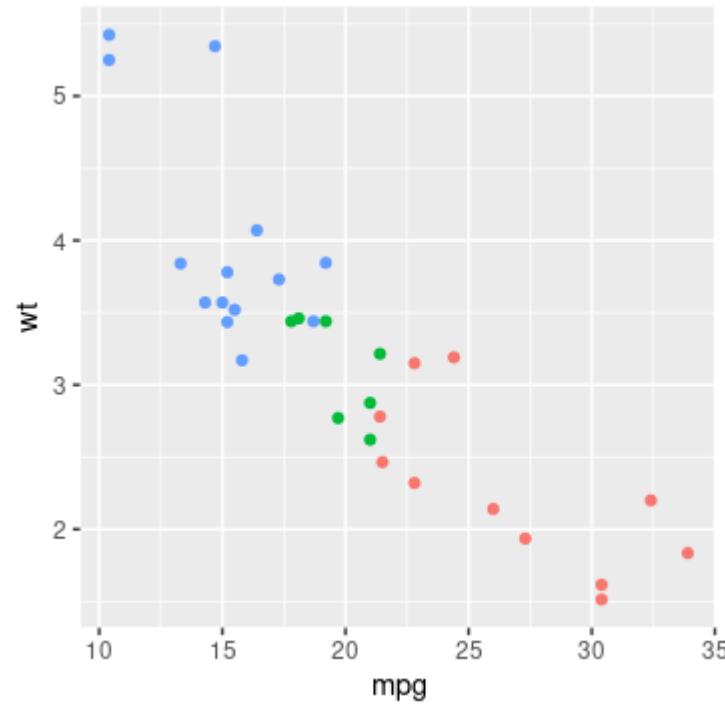
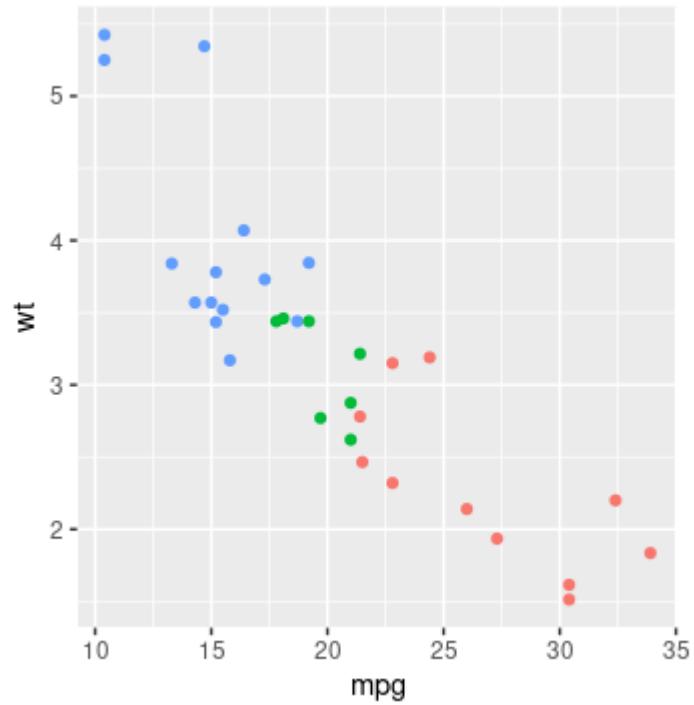
```
geom_point(aes(colour = z))
geom_point(aes(shape = z))
geom_point(aes(shape = z, colour = z))
```



# Customize legend

Control legend title and label

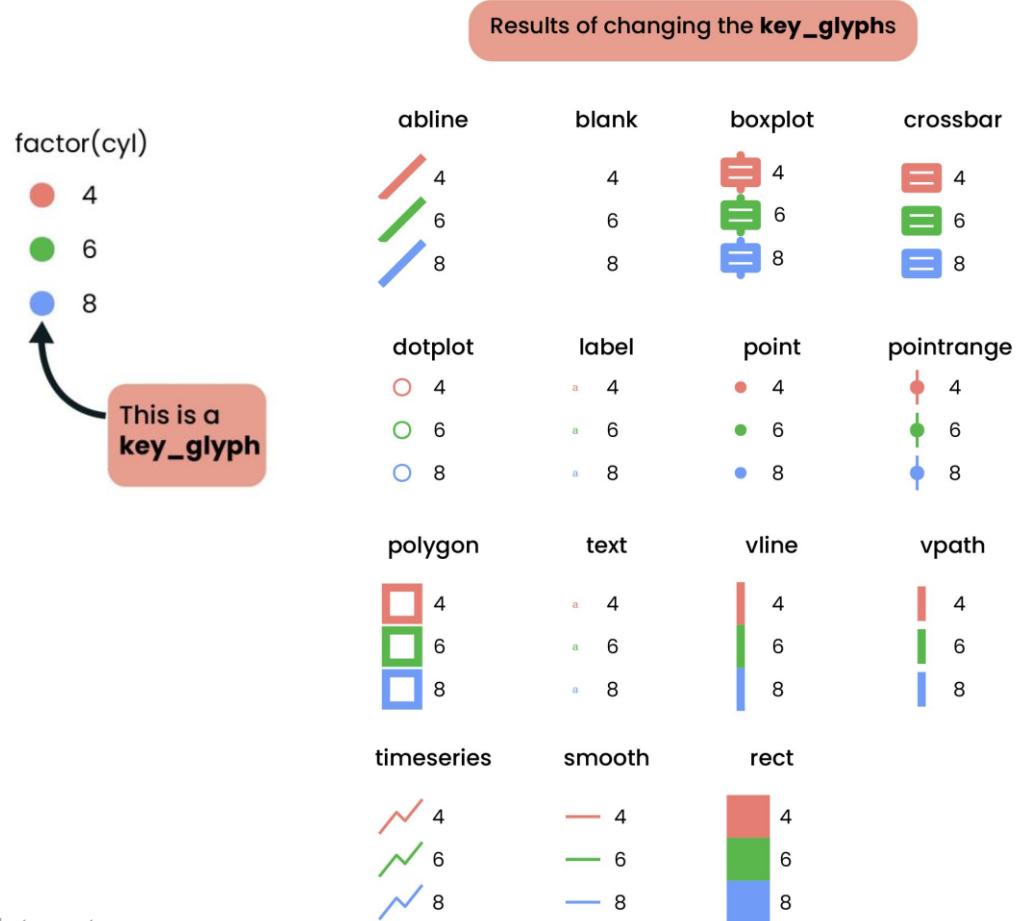
➤ `scale_color_discrete(labels = c("a", "b", "c"), name = "my title")`



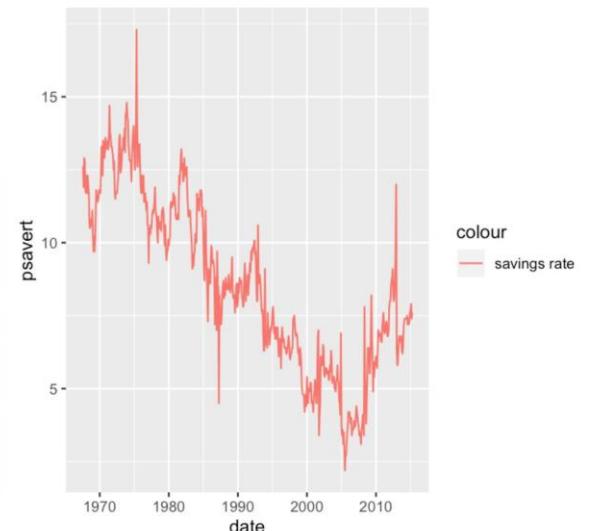
# Customize legend

## Control legend key glyph

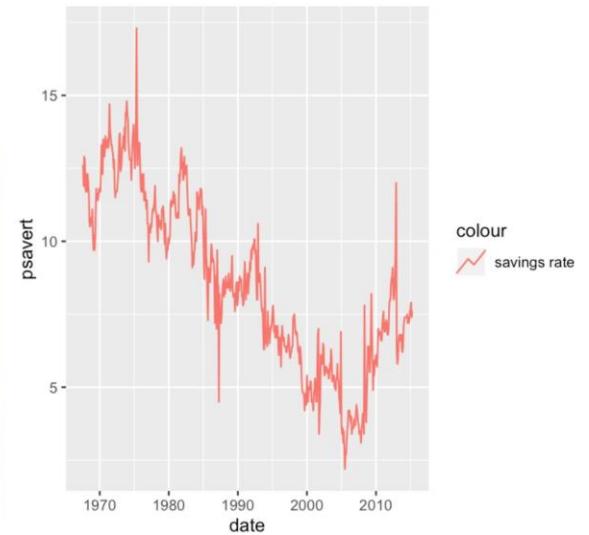
➤ `geom_*(key_glyph = value)`



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



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



# Fine control on legend

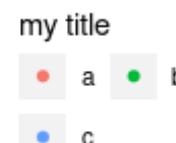
`guide_legend()` offers additional control over the fine details of the legend

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))`

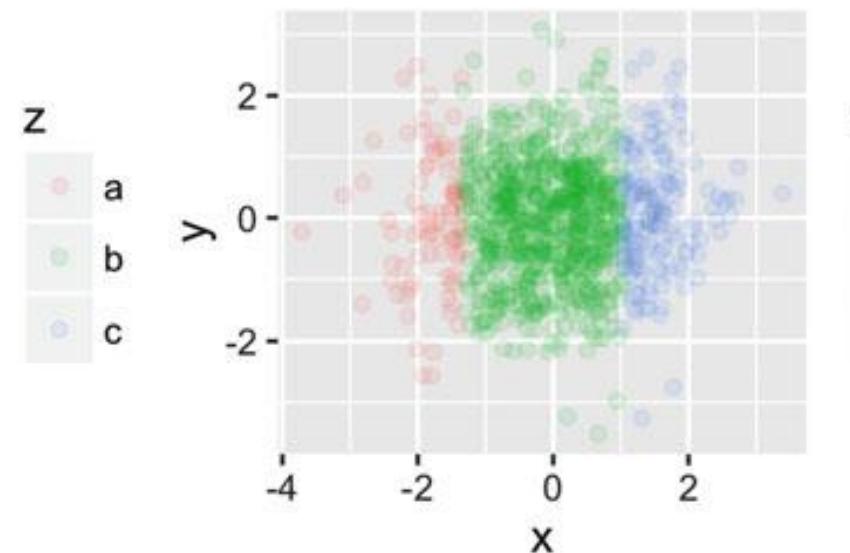
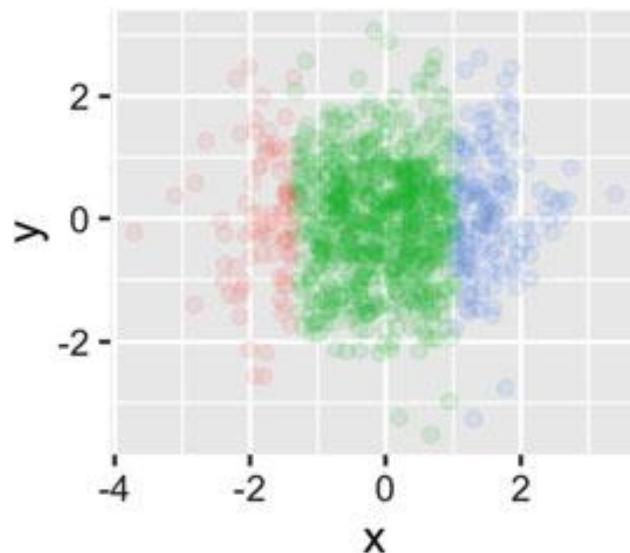


# Fine control on legend

`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

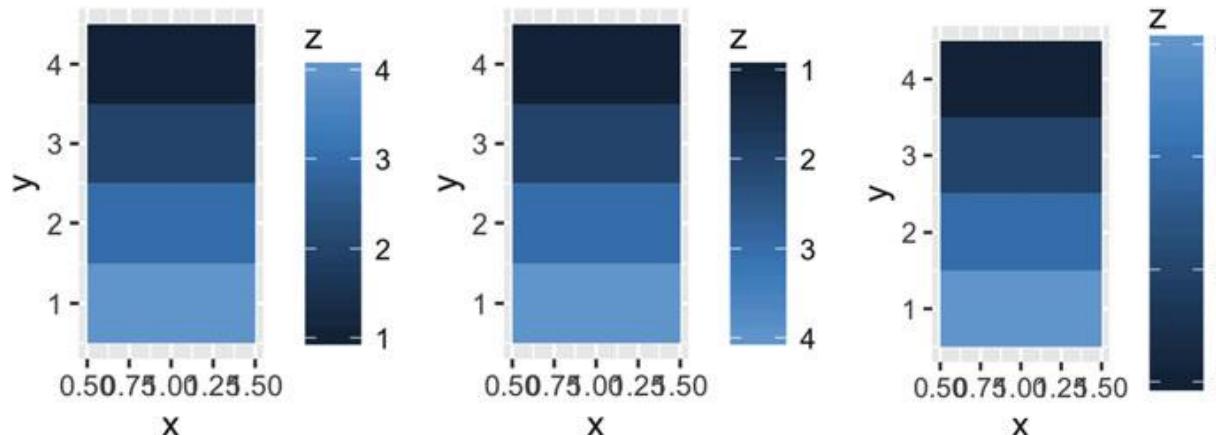
`guide_colorbar()` offers control on the continuous colour scales

Important parameters:

- `barwidth` and `barheight` controls the 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")))`

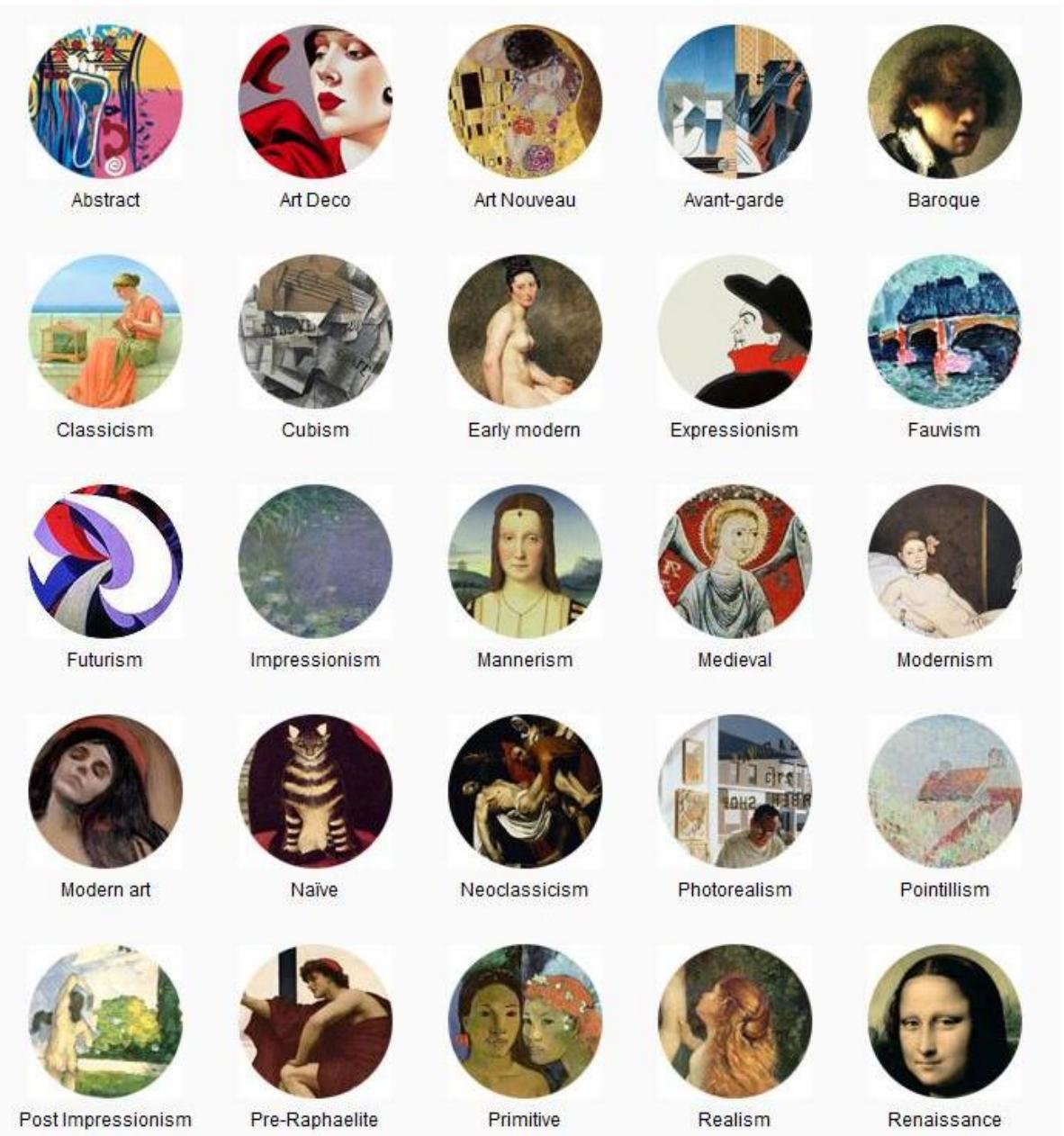


# Let's do some practice!

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



# Theme



# Fine control on the non-data elements

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

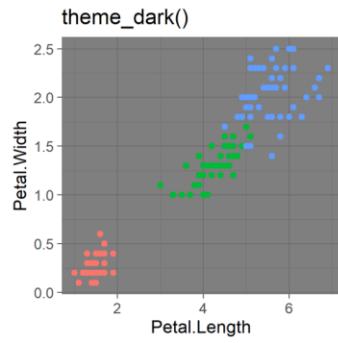
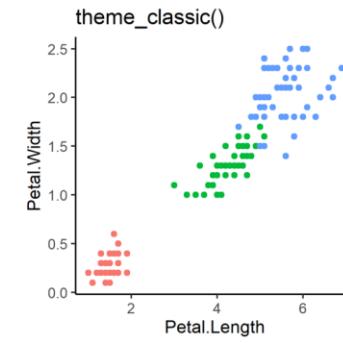
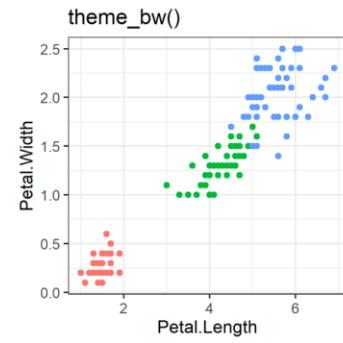
There are 4 main components in the theming system

- **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

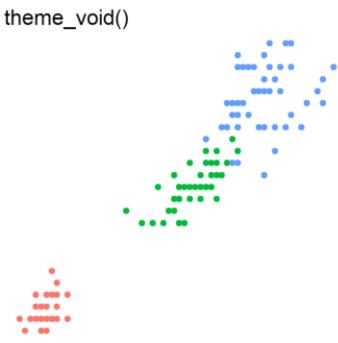
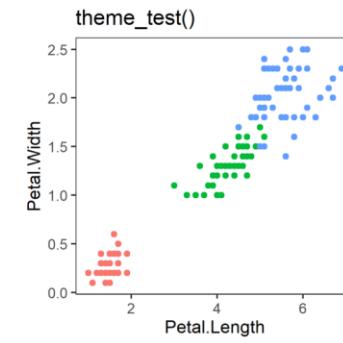
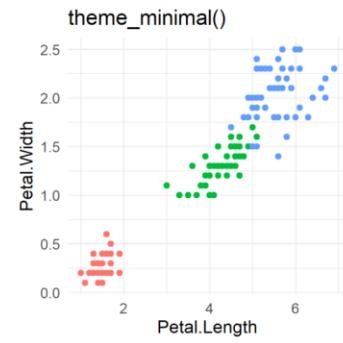
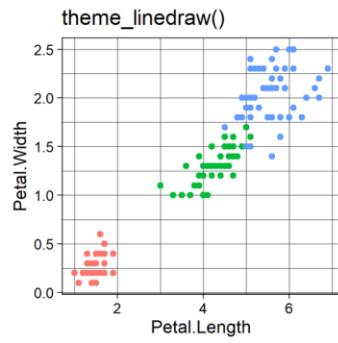
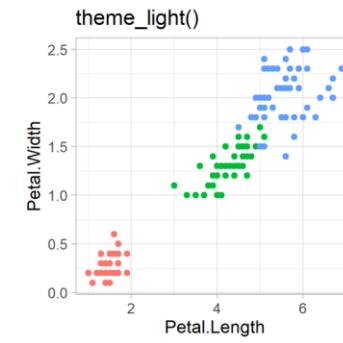
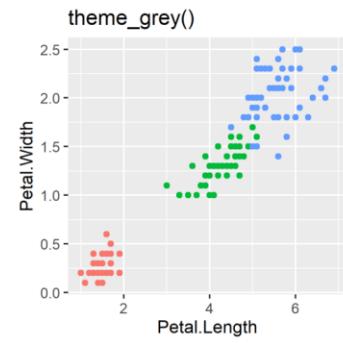
Applying theme for a plot at a time

- `plot + theme_bw()`



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

- `theme_set(theme_bw())`



# Theme elements

## ggplot2 Theme Elements

```
theme(element_name = element_function())
```

- element\_text()
- element\_line()
- element\_rect()
- element\_blank()

## Plot elements:

plot.background  
element\_rect()  
  
plot.title  
element\_text()  
  
plot.margin  
margin()

## Facetting elements:

strip.background  
element\_rect()  
  
panel.spacing  
unit()  
  
strip.text  
element\_text()

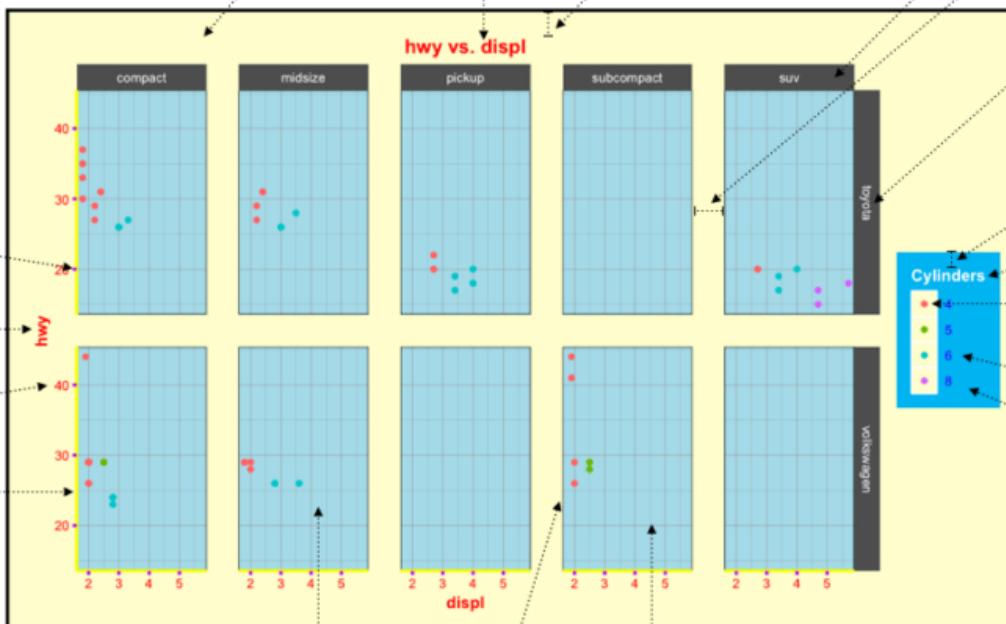
## Axis elements:

axis.ticks  
element\_line()

axis.title  
element\_text()

axis.text  
element\_text()

axis.line  
element\_line()



panel.background  
element\_rect()

panel.border  
element\_rect(fill = NA)

panel.grid  
element\_line()

## Panel 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:

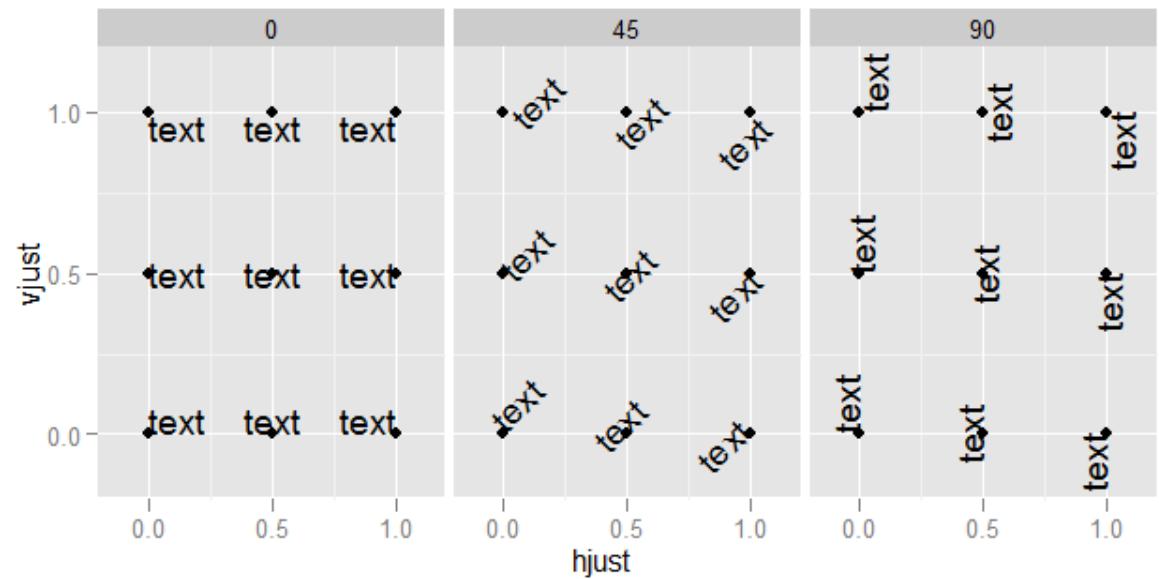
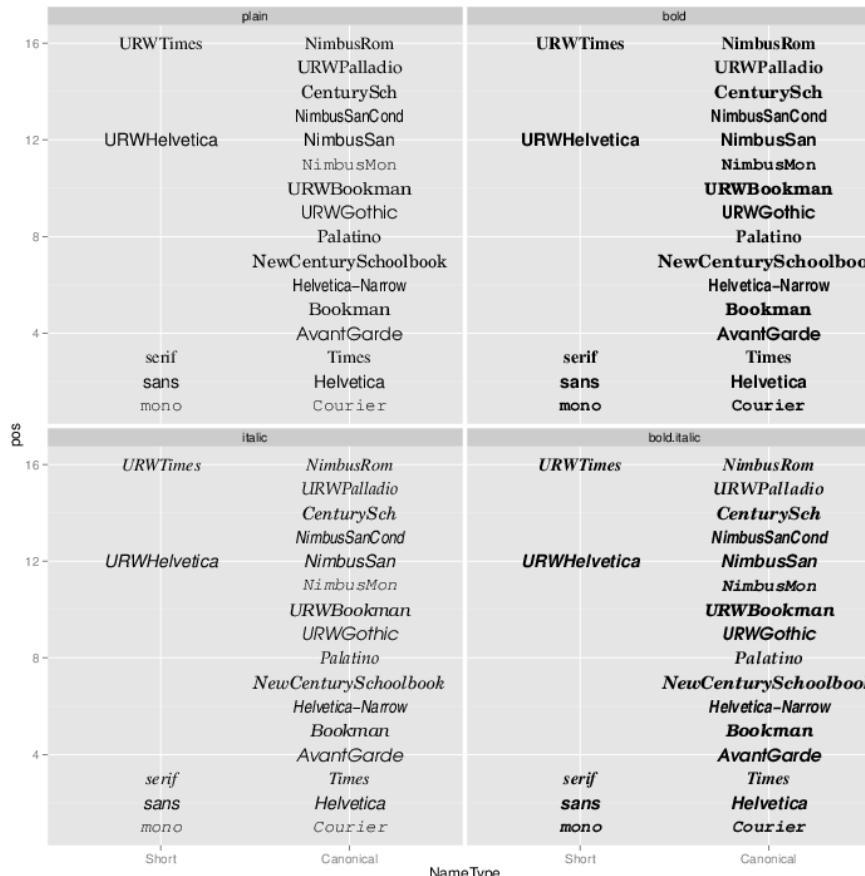
- `element_text()`
- `element_line()`
- `element_rect()`
- `element_blank()`

# 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)

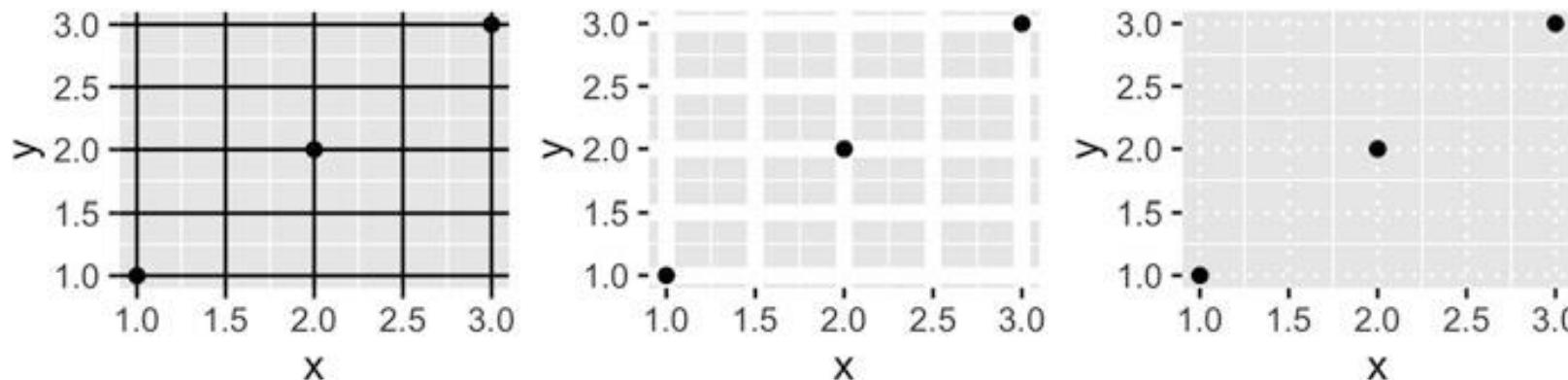
```
➤ theme(plot.title = element_text(margin = margin(t=1, b=1, r=1, l=1)))
```

# 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"))
```

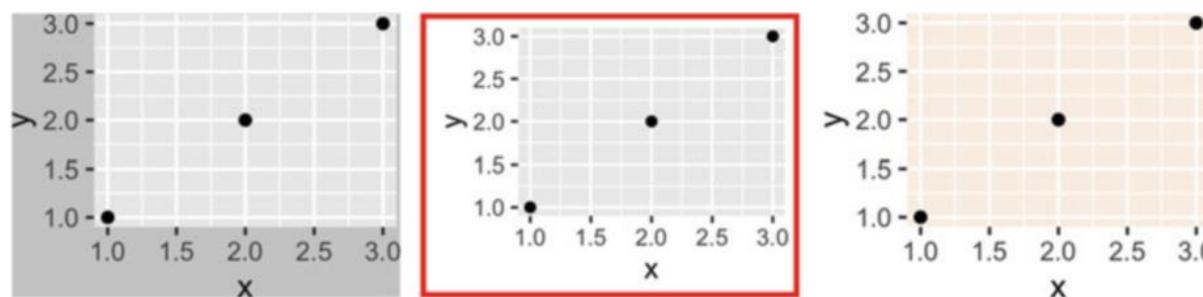


# Modifying theme components

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

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

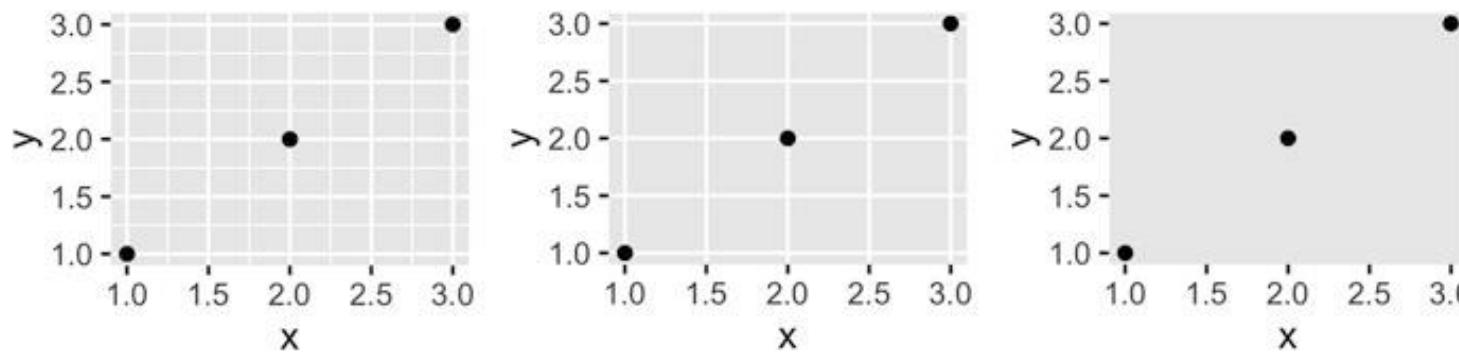
```
➤ theme(plot.background = element_rect(fill = "grey80", colour = NA))  
➤ theme(plot.background = element_rect(colour = "red", size = 2))  
➤ theme(panel.background = element_rect(fill = "linen"))
```



# 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

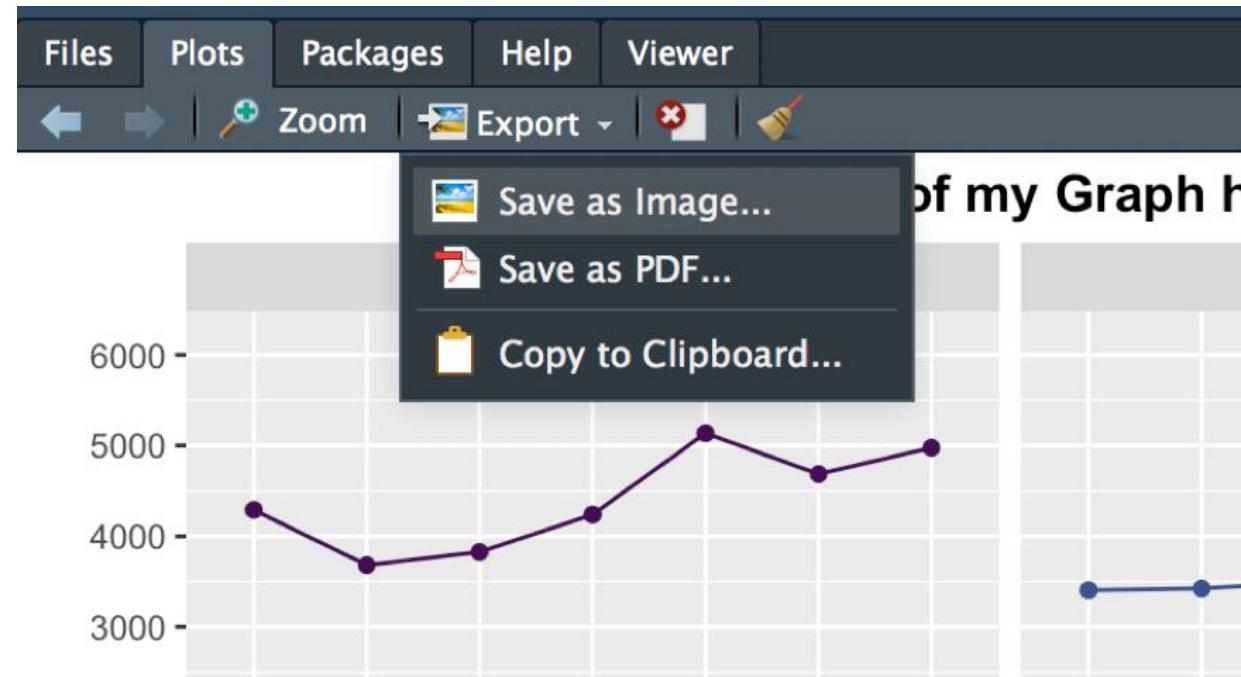
`theme_update()`: modify theme elements for all future plots

- 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



# Figure file format

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

- draw a line from  $(x_1, y_1)$  to  $(x_2, y_2)$
- draw a circle at  $(x_3, x_4)$  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

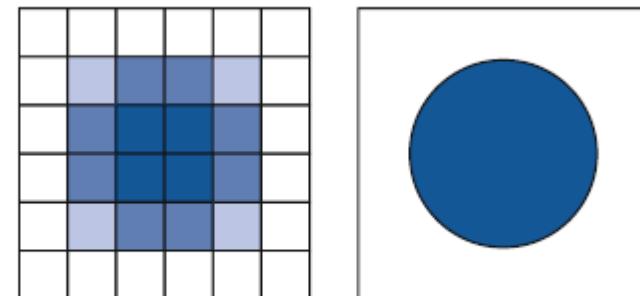
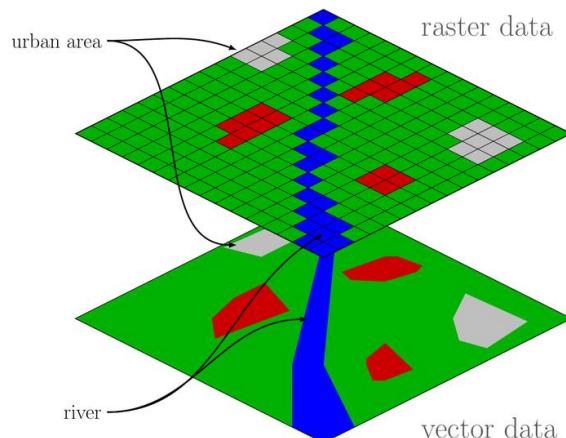


Fig. 8.1 The schematic difference between raster (*left*) and vector (*right*) graphics

# 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



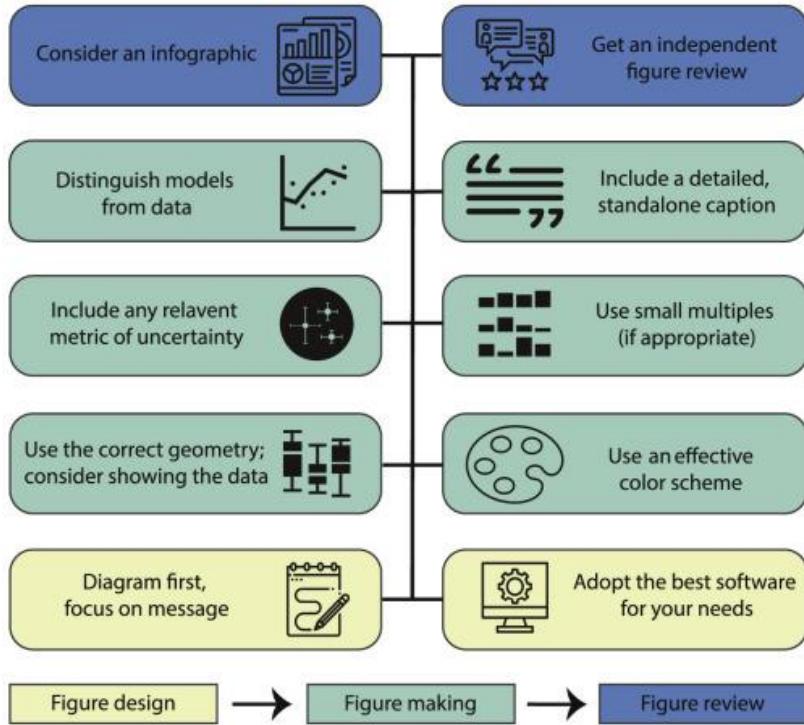
# Save plots to file

- `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

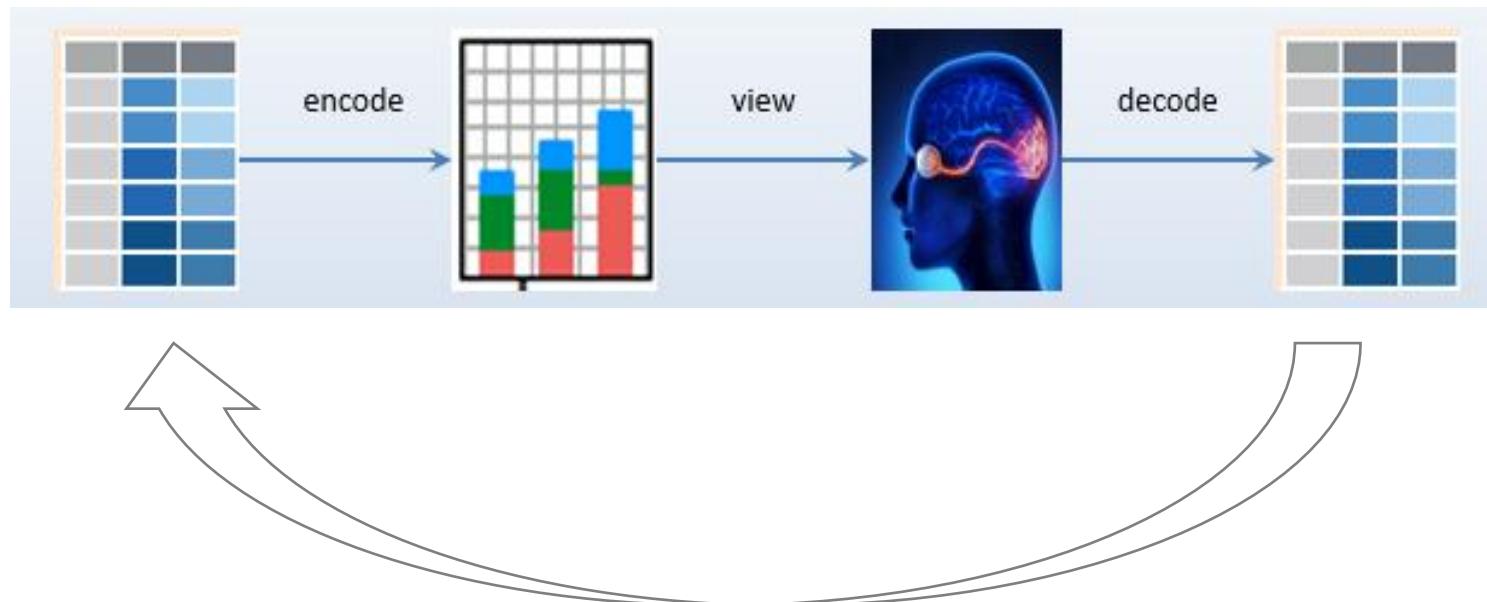
- `width` and `height`: control the output size, specified in inches. If left blank, they'll use the size of the on-screen graphics device.
- `dpi`: control the resolution of plots for raster graphics
- `plot`: ggplot object. If omitted, it will save the last plot

# Principles of visualization



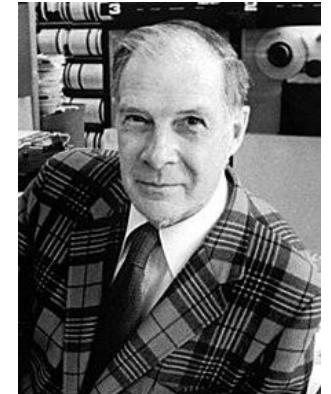
# Effective data visualization

- 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?

# Think about the message you want to convey



Richard Hamming  
(1915-1998)

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

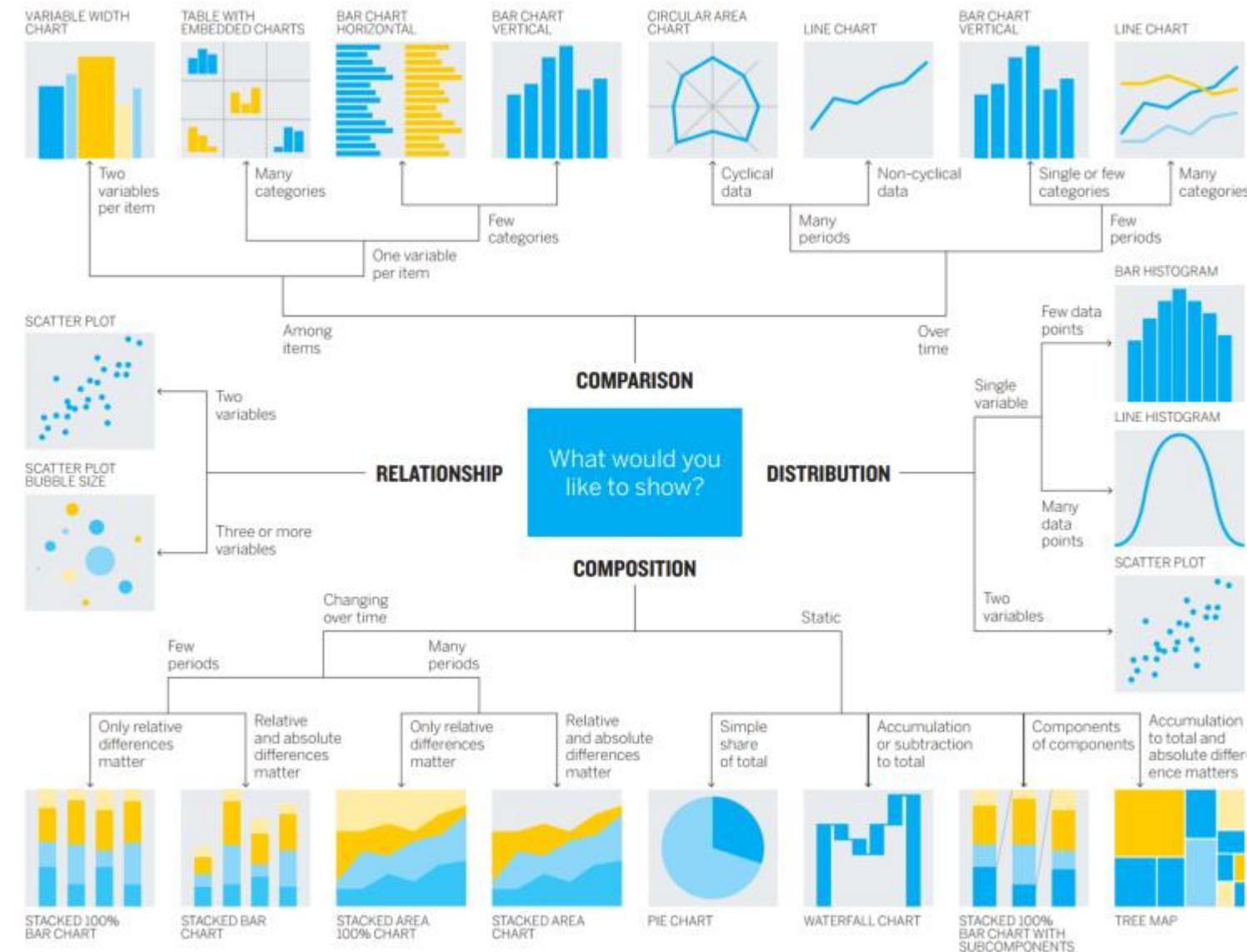
— *Numerical Methods for Scientists and Engineers* (1962)

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

— *UCLA QCBio Collaboratory workshop... (2024)\**

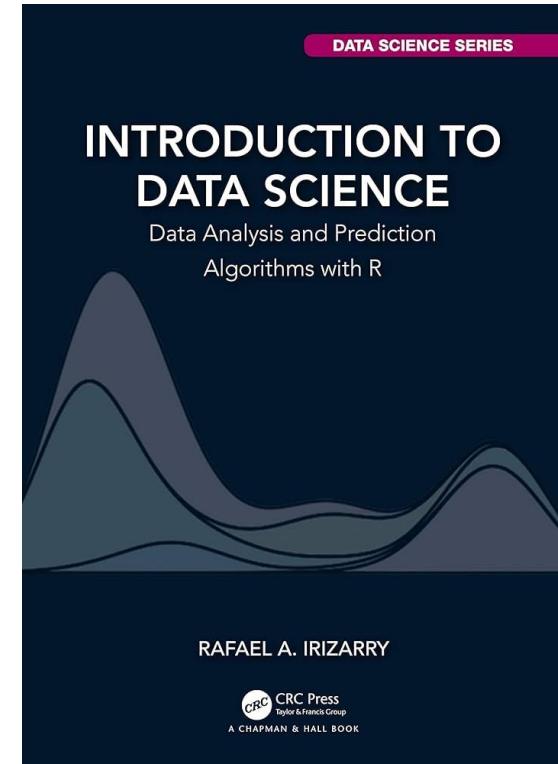


# Decide on the right type of plot



# Principles for data visualization

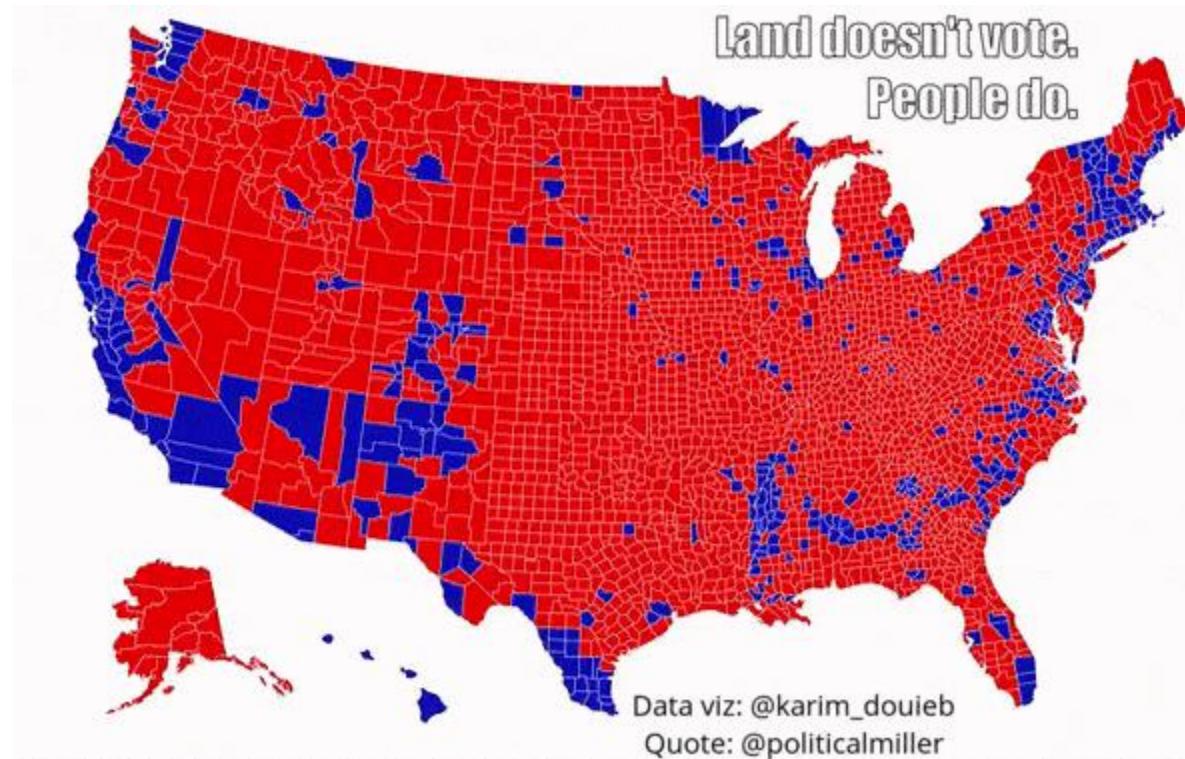
- Use appropriate representations
- Add necessary annotation
- Don't mislead the reader
- Order categories by a meaningful value
- Show the data
- Reduce image processing
- Make comparisons easy
- Be mindful about the color
- Avoid pseudo-three-dimensional plots



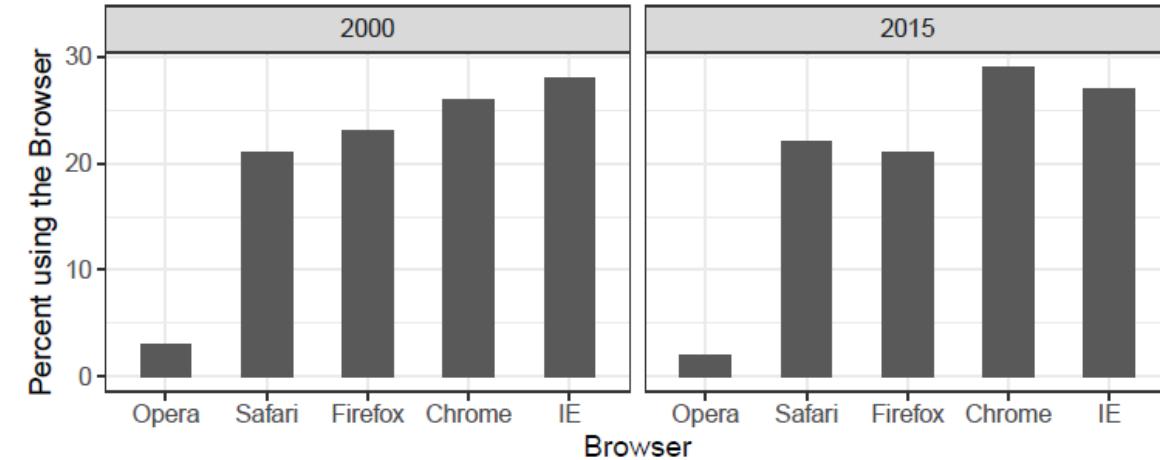
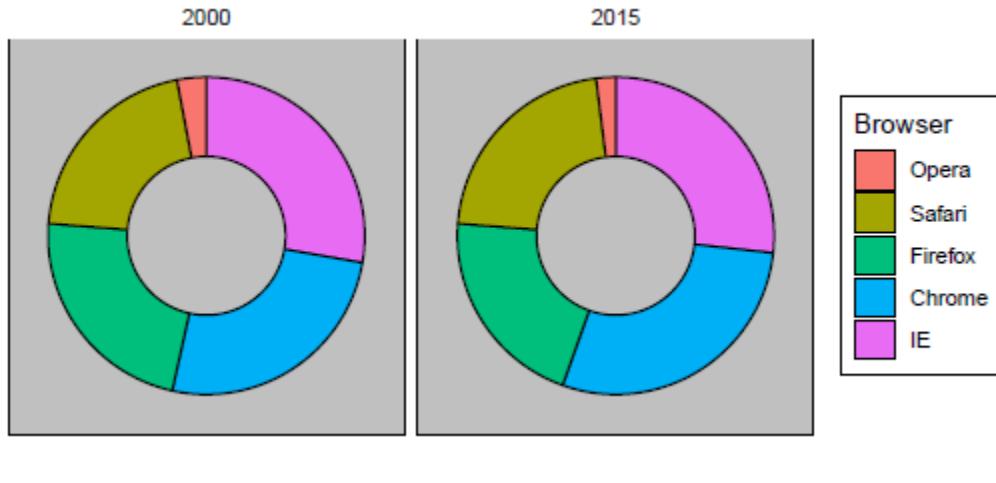
[link](#)

# Use appropriate representations

- Different representations have different focuses



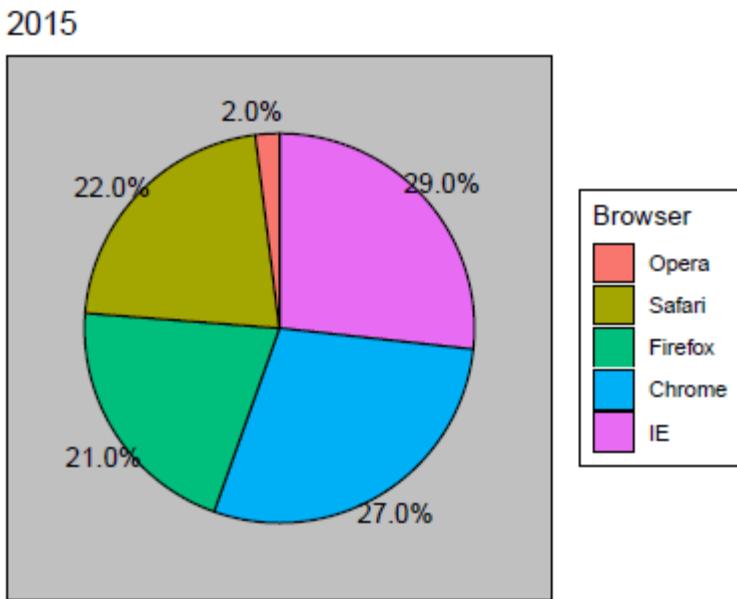
# Use appropriate representations



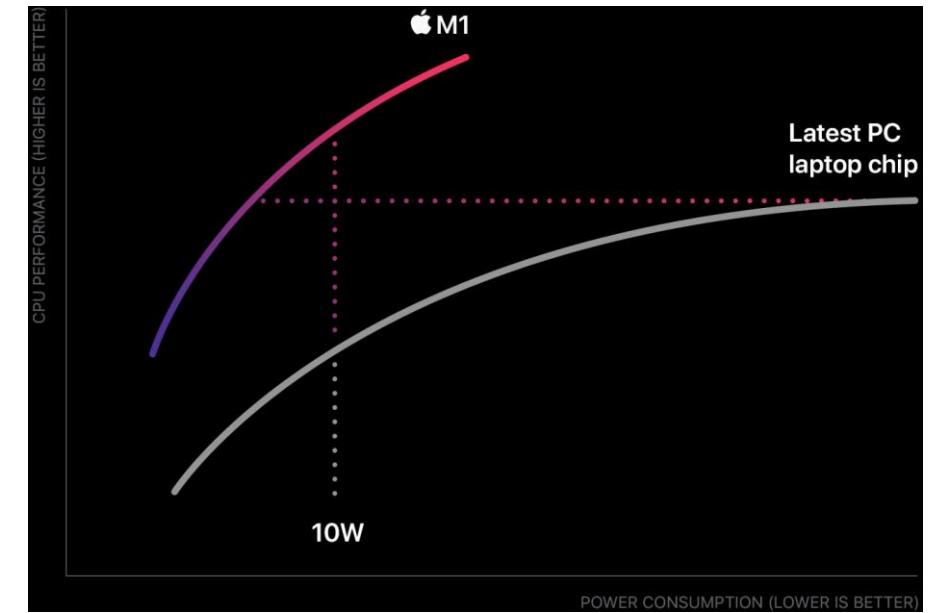
Our eyes are good at judging linear measures and bad at judging relative areas

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



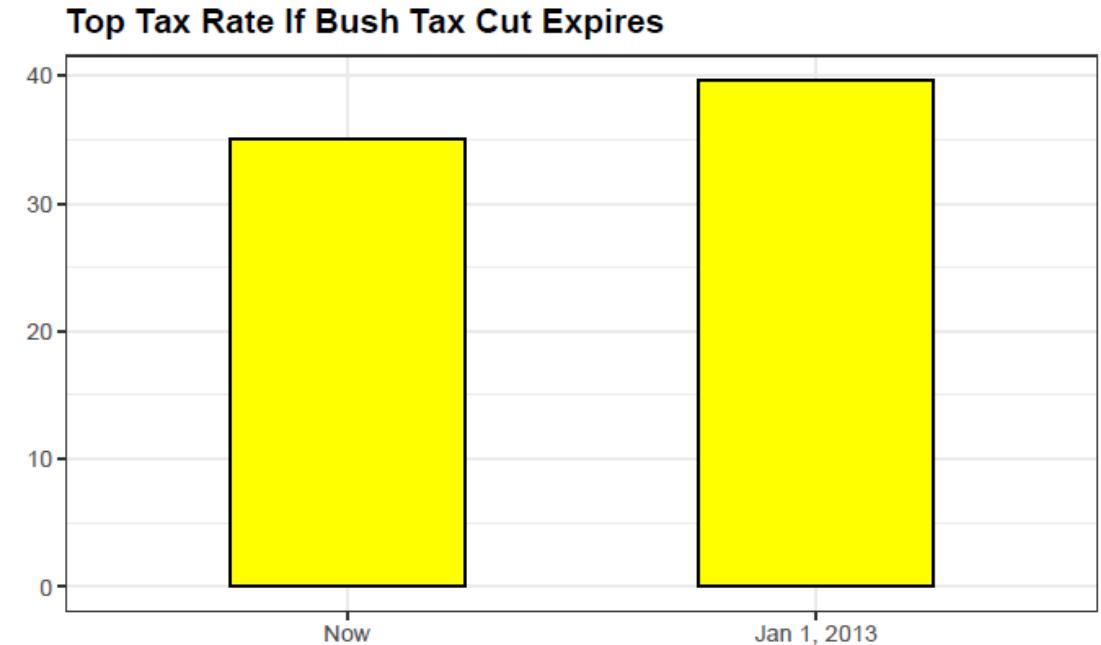
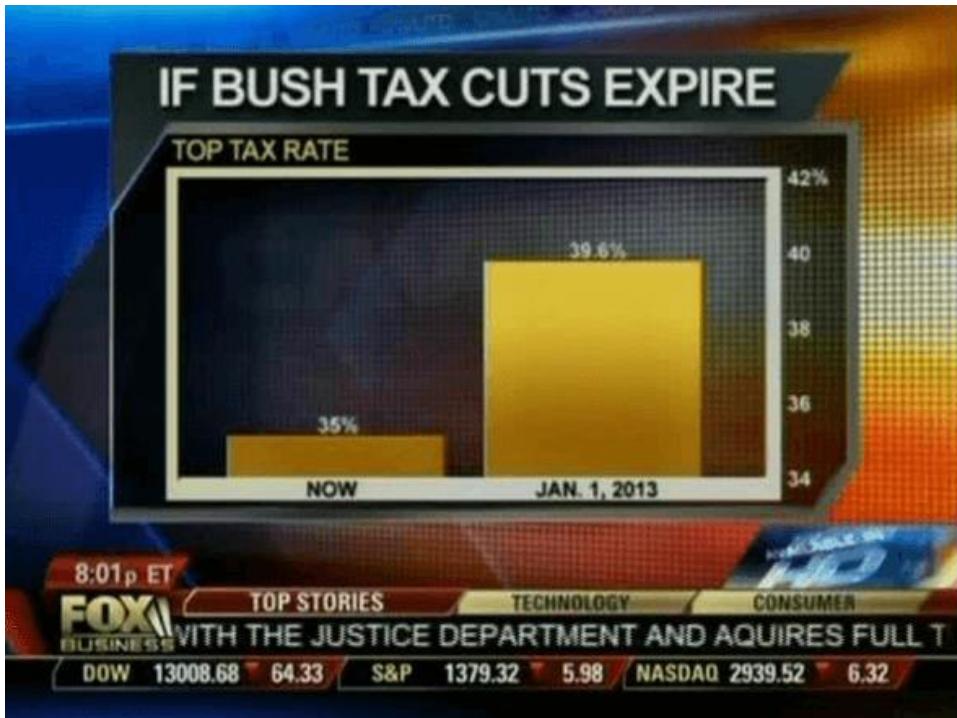
Example 1



Example 2

Add necessary annotation to help better understand the plots

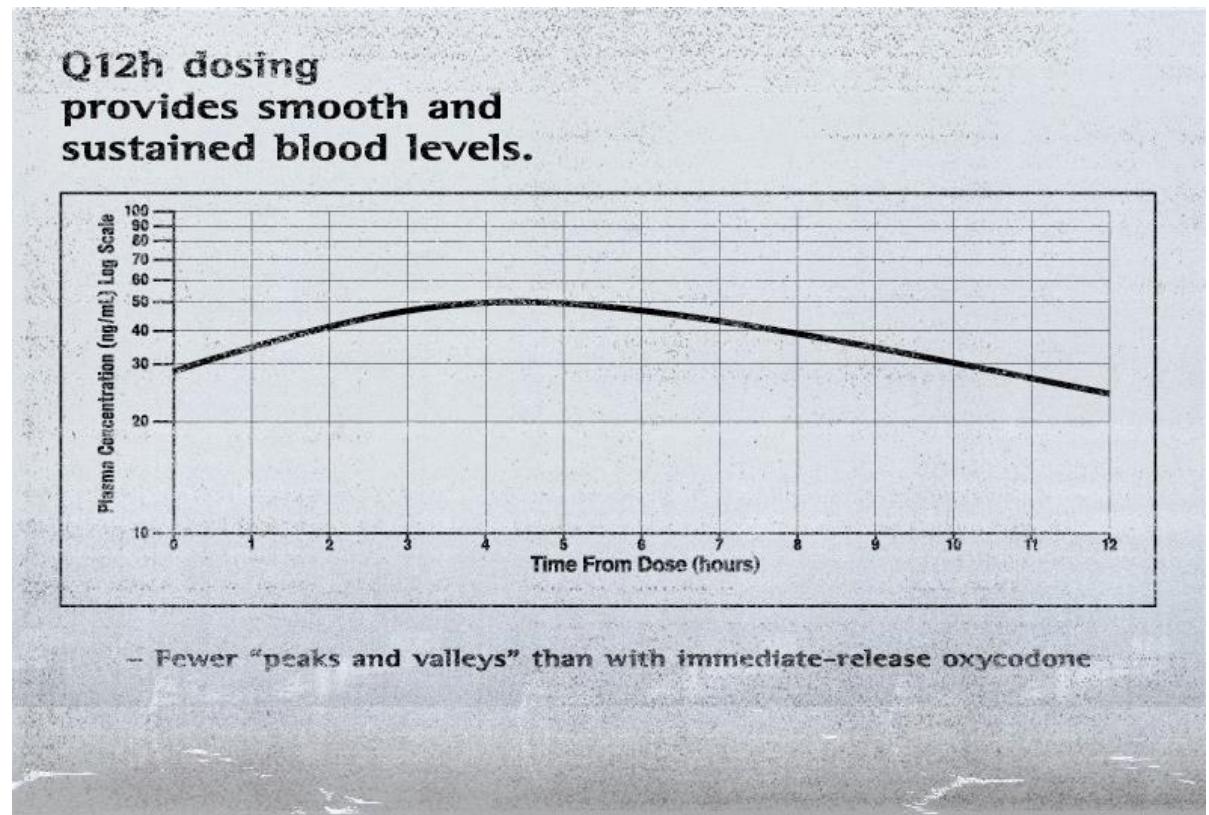
# Don't mislead the reader



$$(39.6\%-35\%)/35\% = 13\% \text{ increase}$$

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

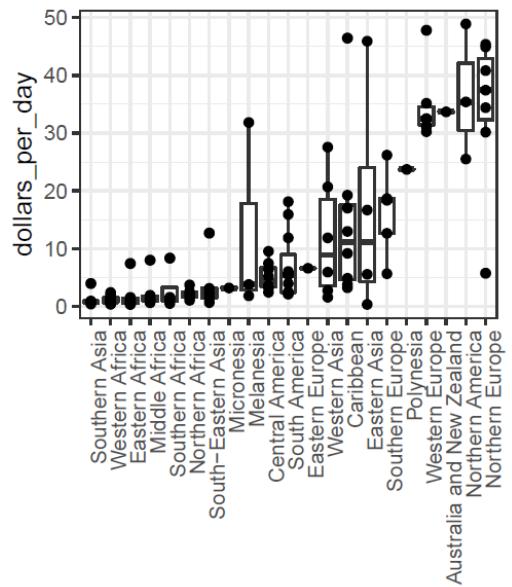
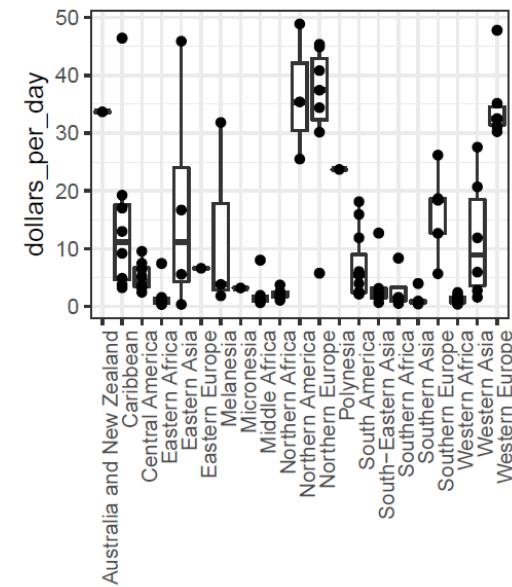
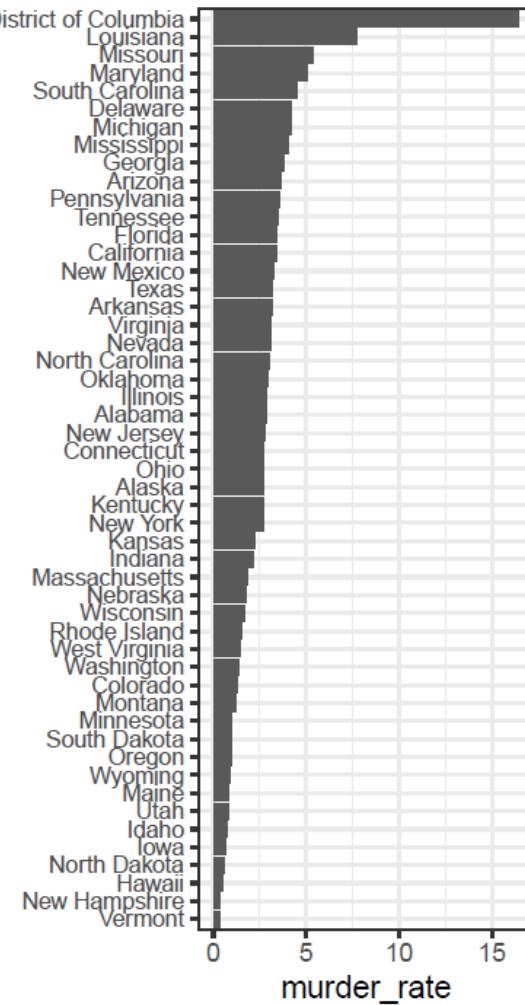
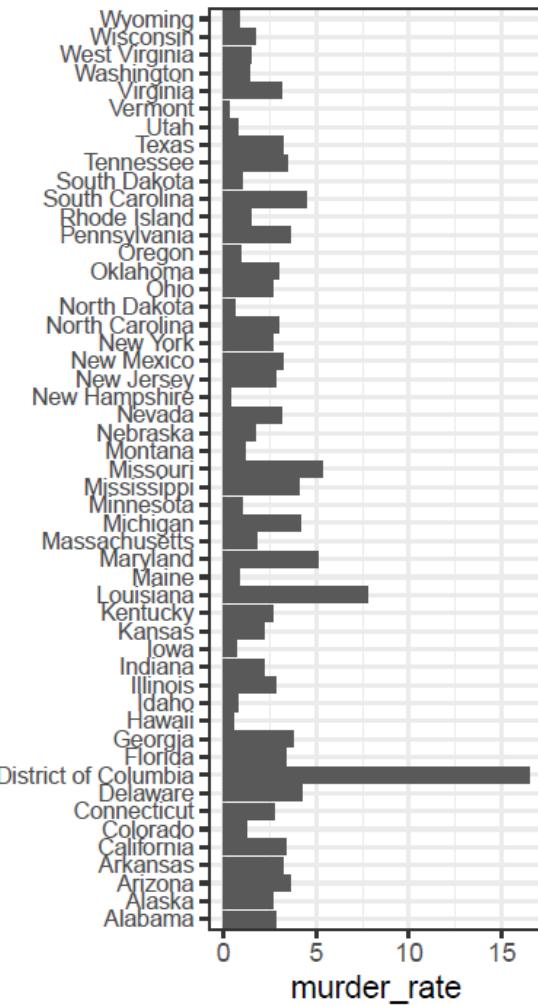
# Don't mislead the reader



Seeing is not believing, be skeptical !!

# Order categories by a meaningful value

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

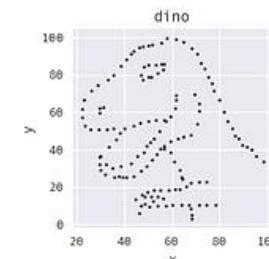
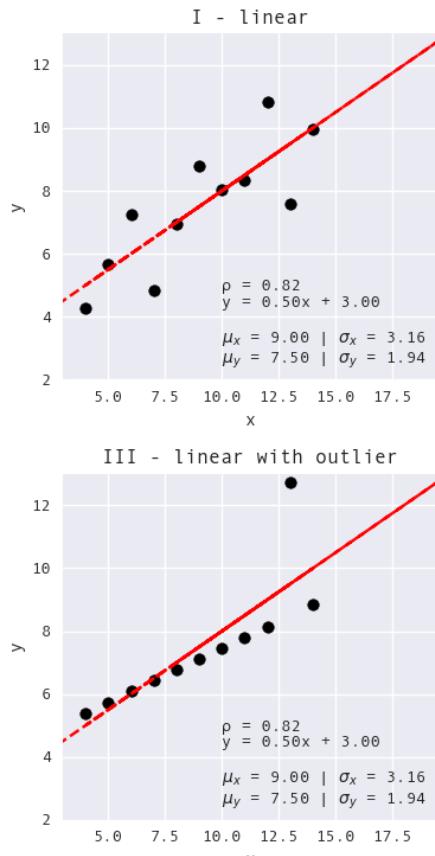


Chaotic

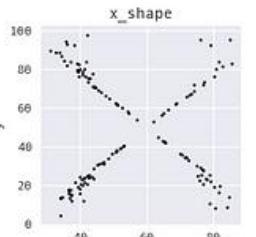
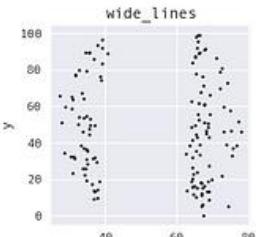
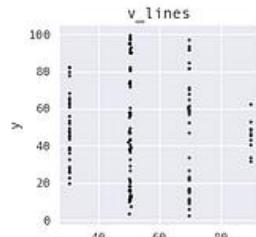
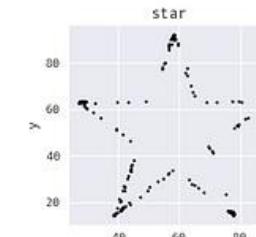
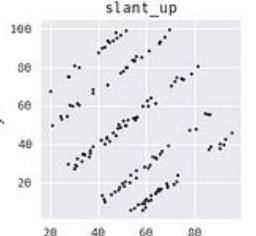
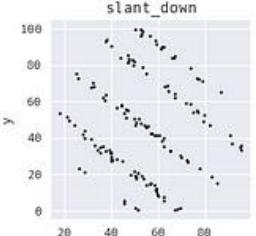
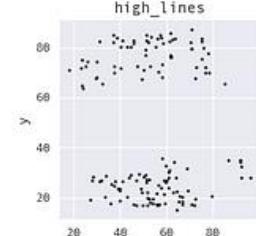
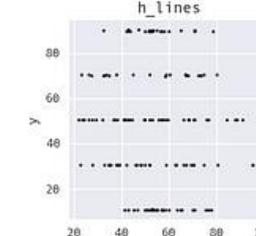
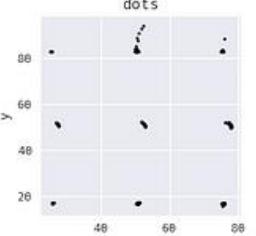
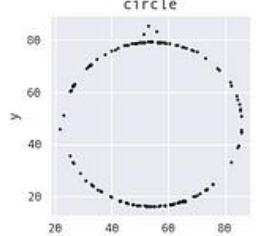
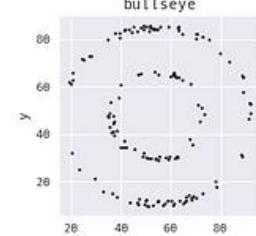
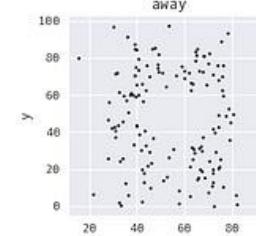
Organized

# Show the data

Anscombe's Quartet

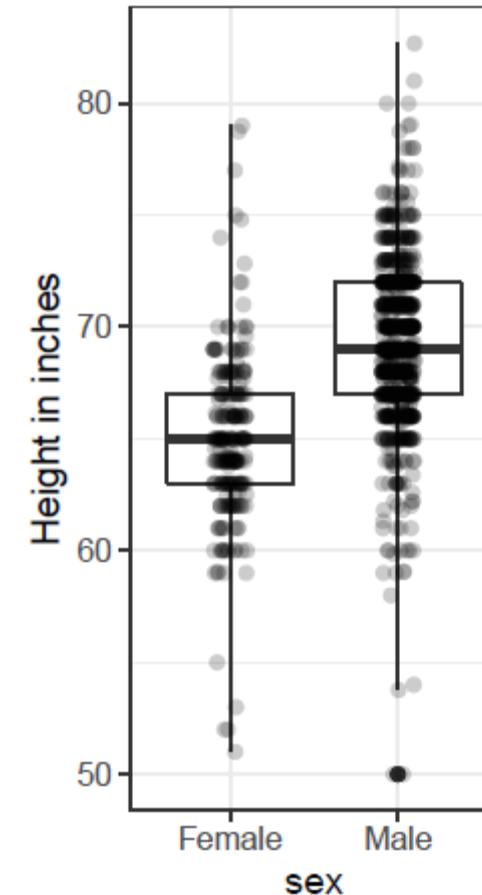
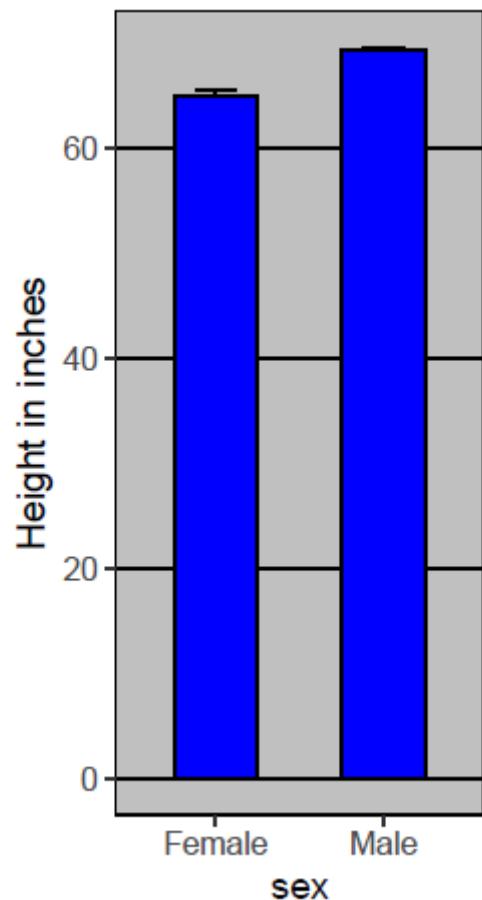


$\rho_p = -0.06$   
 $\mu_x = 54.26$   
 $\sigma_x = 16.71$   
 $\mu_y = 47.83$   
 $\sigma_y = 26.84$



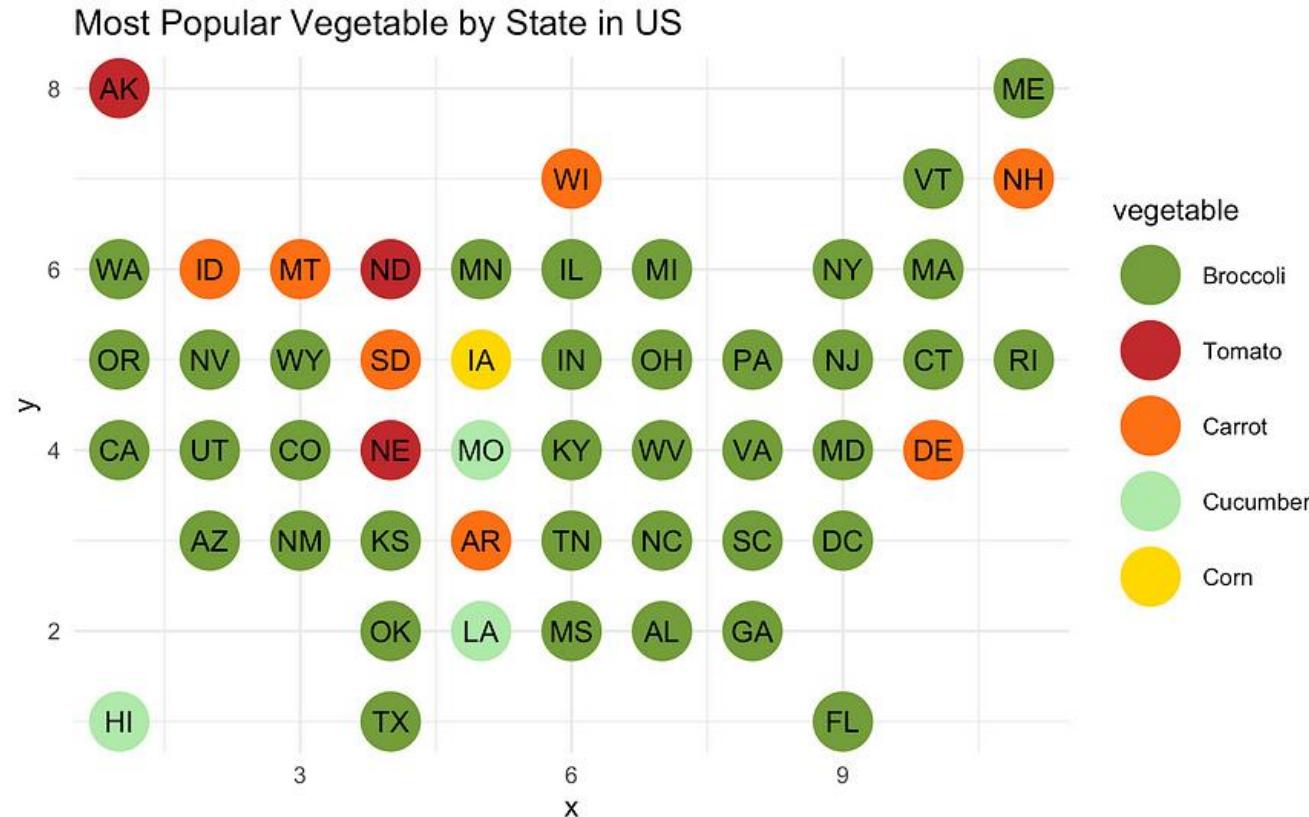
Don't purely rely on summary statistics, it's important to show the data

# Show the data



Visualizing the distribution is much more informative than summary statistics alone

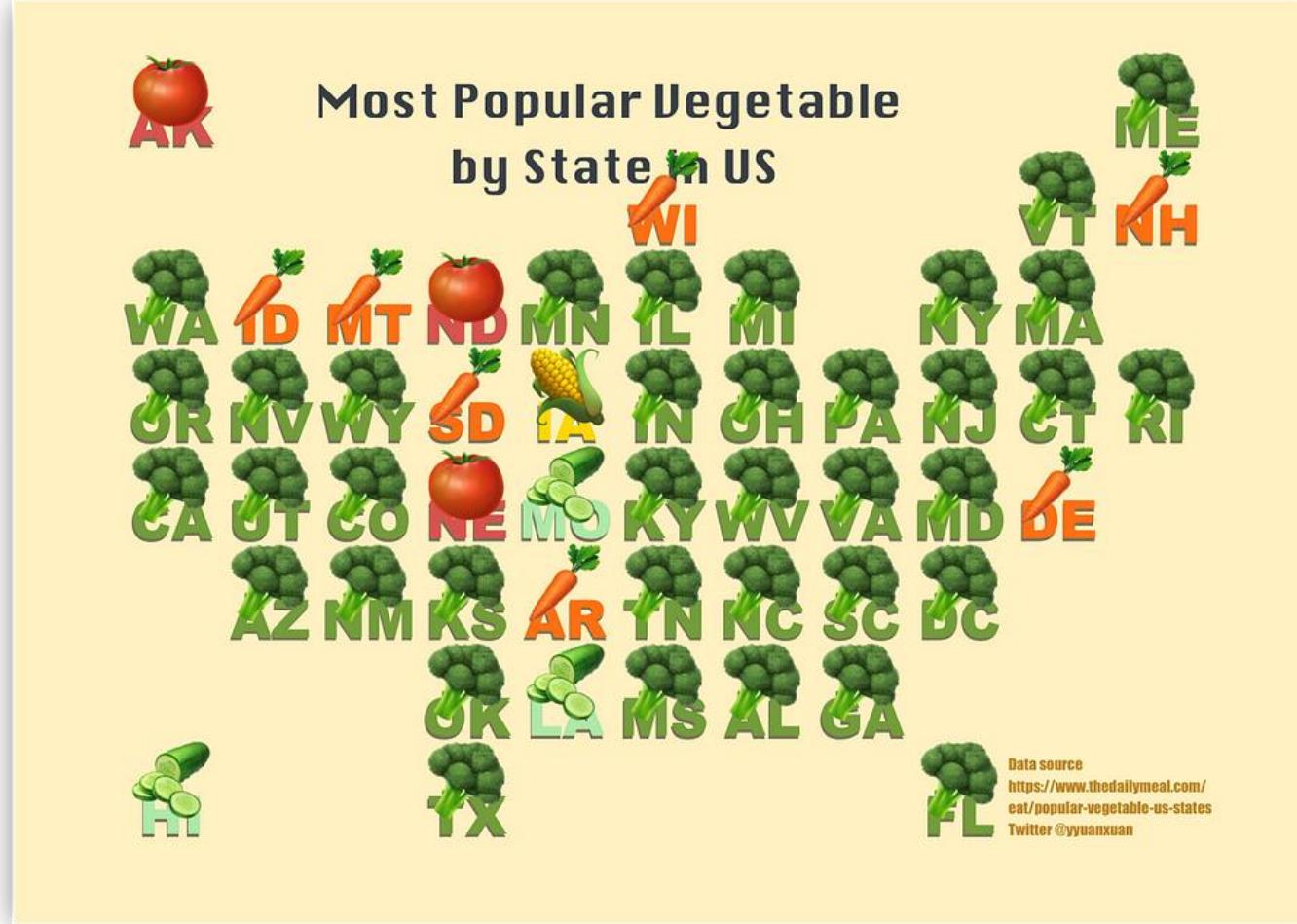
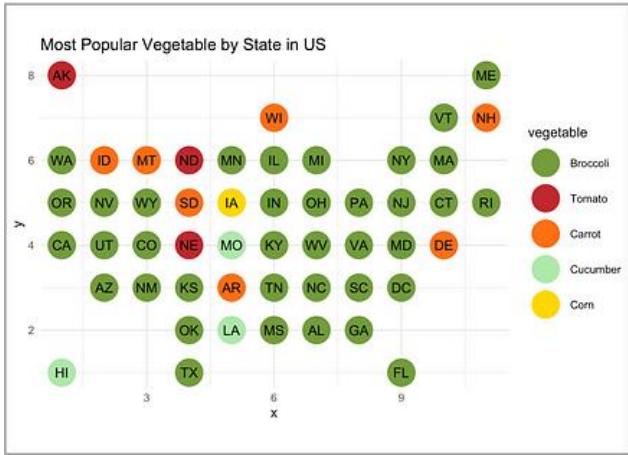
# Reduce image processing



Find the most popular vegetable of a specific state:

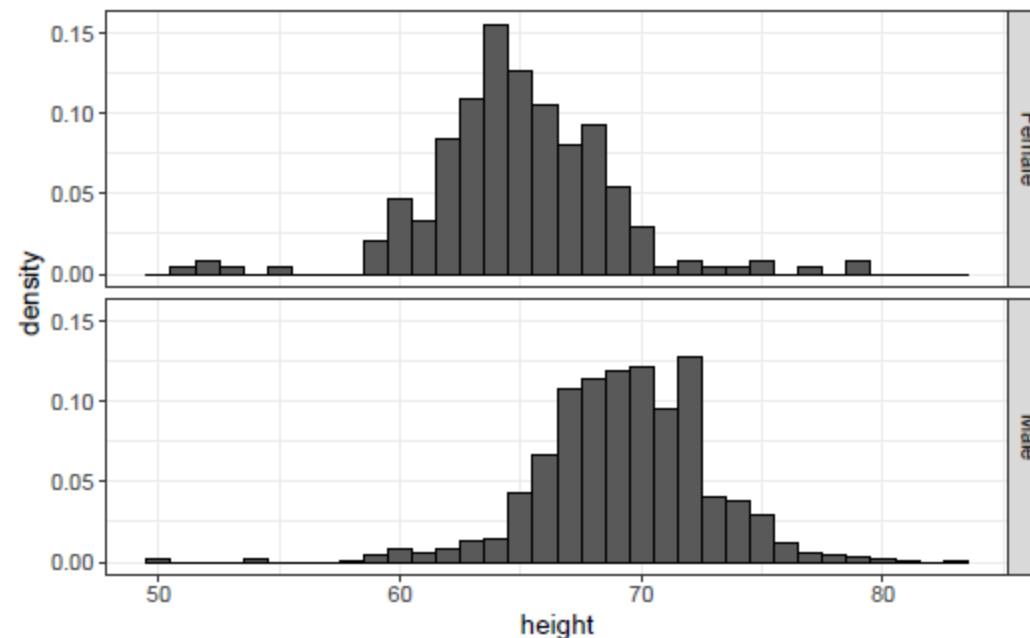
- Locate state -> get color -> get vegetable name

# Reduce image processing



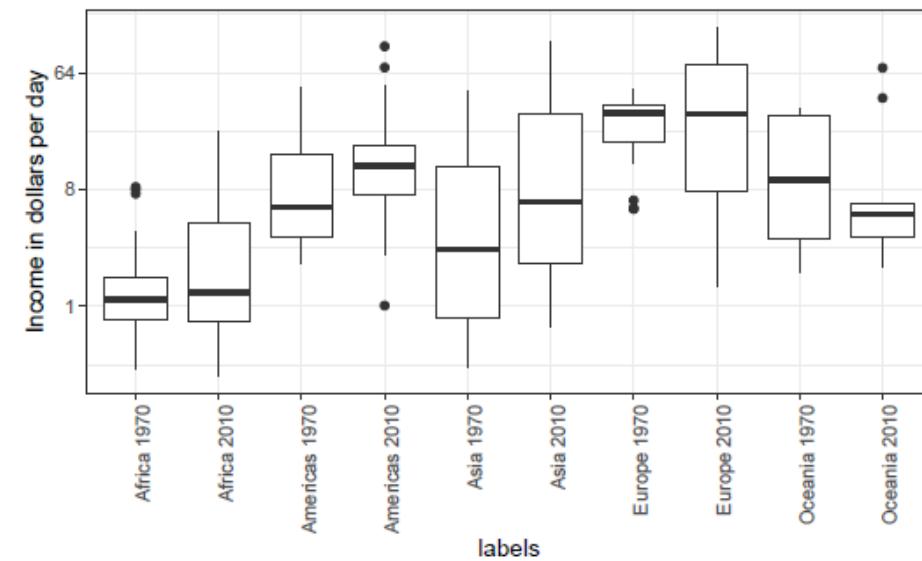
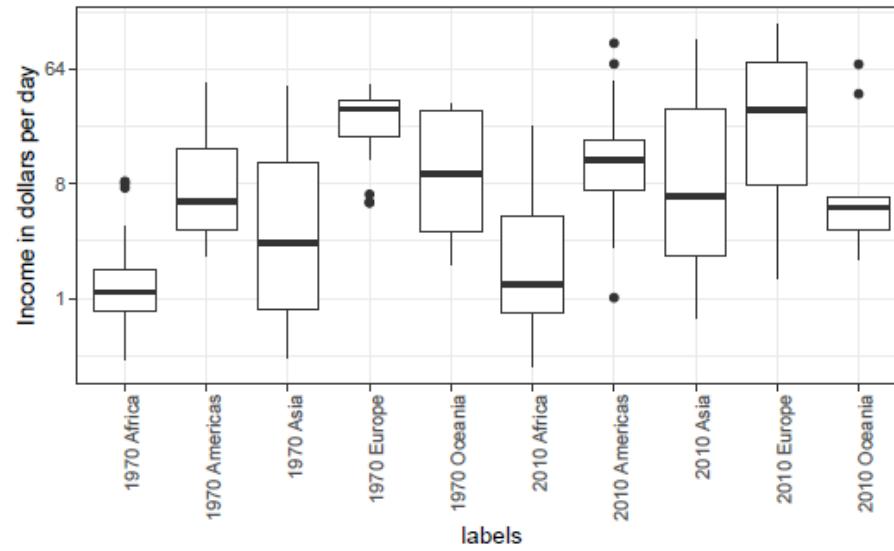
# Make comparisons easy

- 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**



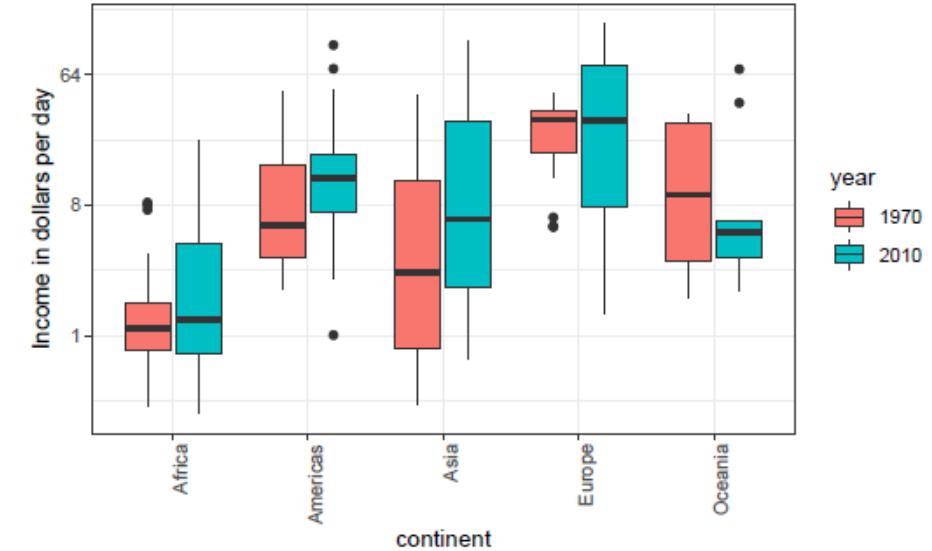
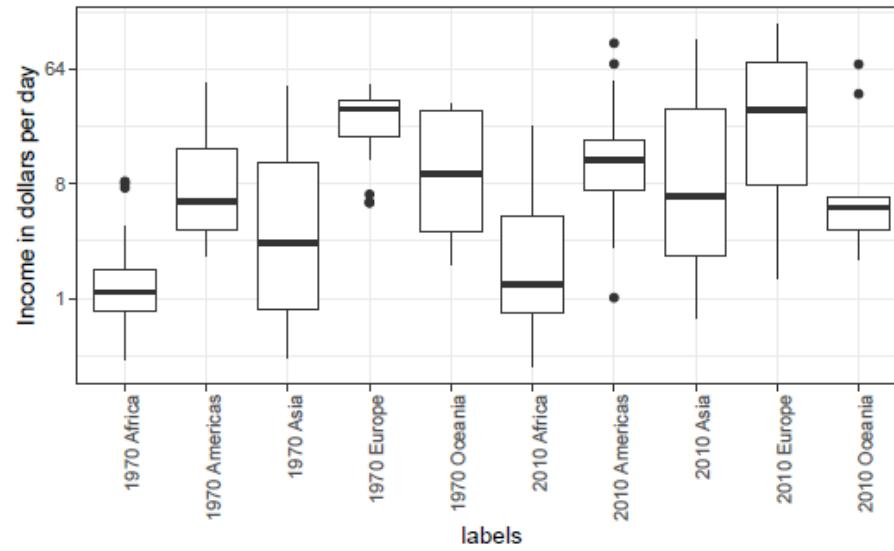
# Make comparisons easy

- 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



# Be mindful about color

## Color theory

PURPLE	BLUE	GREEN	YELLOW	ORANGE	RED
<b>Association:</b> Wisdom, wealth, royalty, power, luxury, magic	<b>Association:</b> Depth, stability, wisdom, trust, confidence	<b>Association:</b> Growth, health, harmony, safety nature.	<b>Association:</b> Energy, happy, warming, attention	<b>Association:</b> Enthusiasm, heat, success creativity	<b>Association:</b> Passion, energy, strength, love, power, determination
<b>Mood:</b> Powerful, calming, strength	<b>Mood:</b> Calming	<b>Mood:</b> Calm, refreshed	<b>Mood:</b> Aggravation, joy	<b>Mood:</b> Warmth, excitement	<b>Mood:</b> Intensity, angry, excitement

Which button makes you more inclined to buy a product?

**BUY NOW!**

**BUY NOW!**

# Be mindful about color

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

Use colorblind-friendly palette:

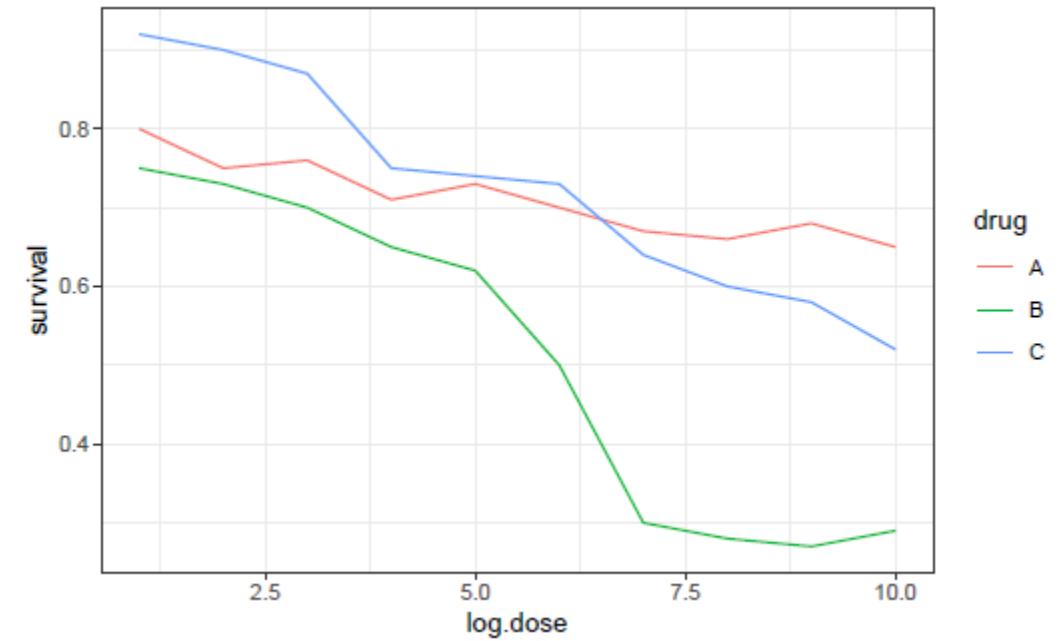
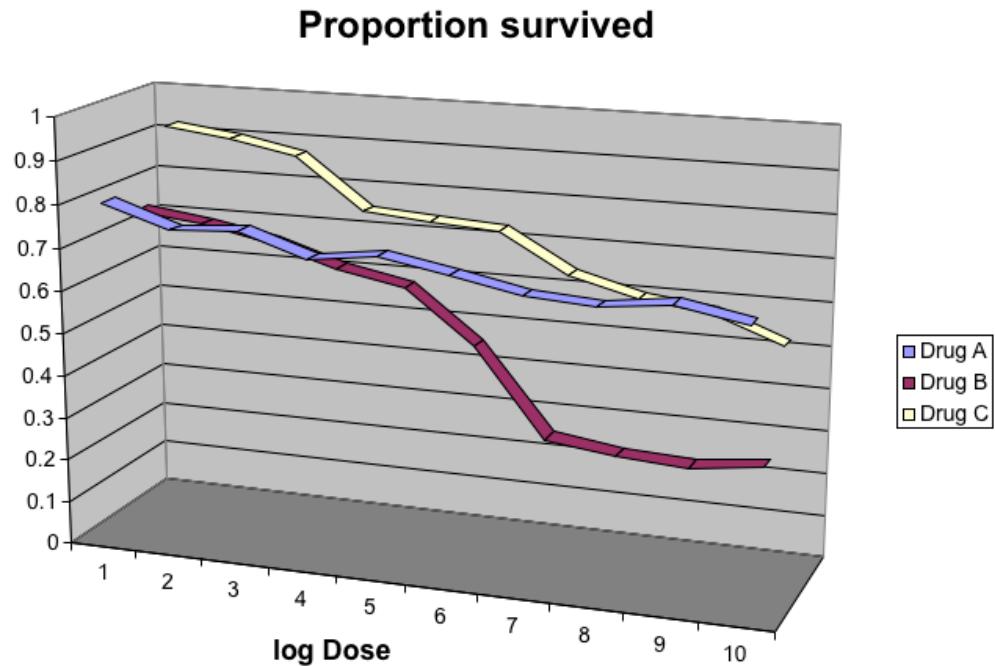
```
> c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
```



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



# 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/>



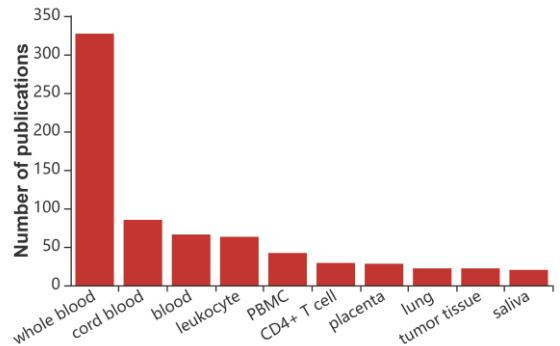
Google



# Ask ChatGPT to help you make the plot

## Example figure

TOP 10 tissues



customize

Ask ChatGPT

## 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

please write ggplot2 code to generate the example figure

ChatGPT

The screenshot shows a ChatGPT interface. The user asks for ggplot2 code to generate the example figure. ChatGPT generates a bar chart titled "TOP 10 tissues" with the same data as the example figure. Below the chart, ChatGPT provides an R code template. A message box at the bottom says "Message ChatGPT..." and "ChatGPT can make mistakes. Consider checking important information."

TOP 10 tissues

Number of publications

Tissues

I have generated a similar bar chart using Matplotlib in Python, which is a different library from ggplot2. The ggplot2 library is specific to R, not Python. However, if you need a ggplot2 code example in R, here it is:

R

```
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)
```

Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.



Good luck and have fun!

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