

Advanced **data visualization** with **ggplot2**

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

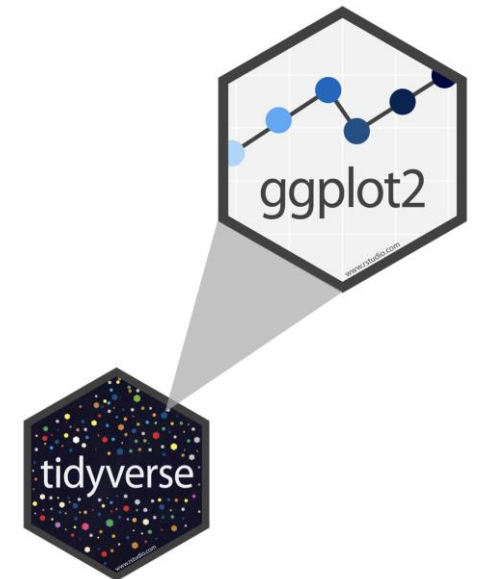
Notation of the slides

- 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



Day 2: **Building** a plot layer by layer

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

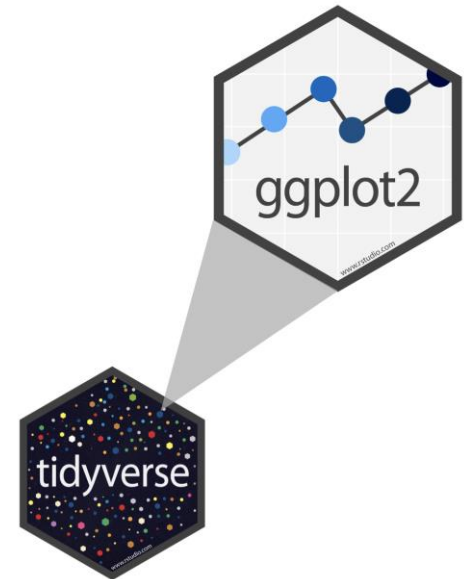
Overview

Time

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

Topics

- ☐ Layers
 - ☐ Geometry
 - ☐ Statistical transformation
 - ☐ Position adjustment
- ☐ Scale
- ☐ Facet
- ☐ Coordinate system



Summary – Day1

Key components of a ggplot2 object:

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

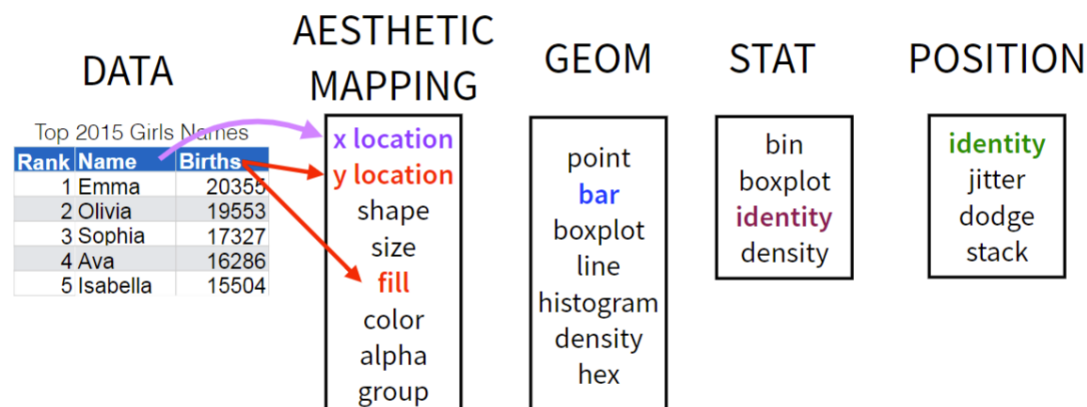
```
ggplot (data = <DATA>) +  
  <GEOM_FUNCTION> (mapping = aes(<MAPPINGS>),  
    stat = <STAT>, position = <POSITION>) +  
  <COORDINATE_FUNCTION> +  
  <FACET_FUNCTION> +  
  <SCALE_FUNCTION> +  
  <THEME_FUNCTION>
```

required

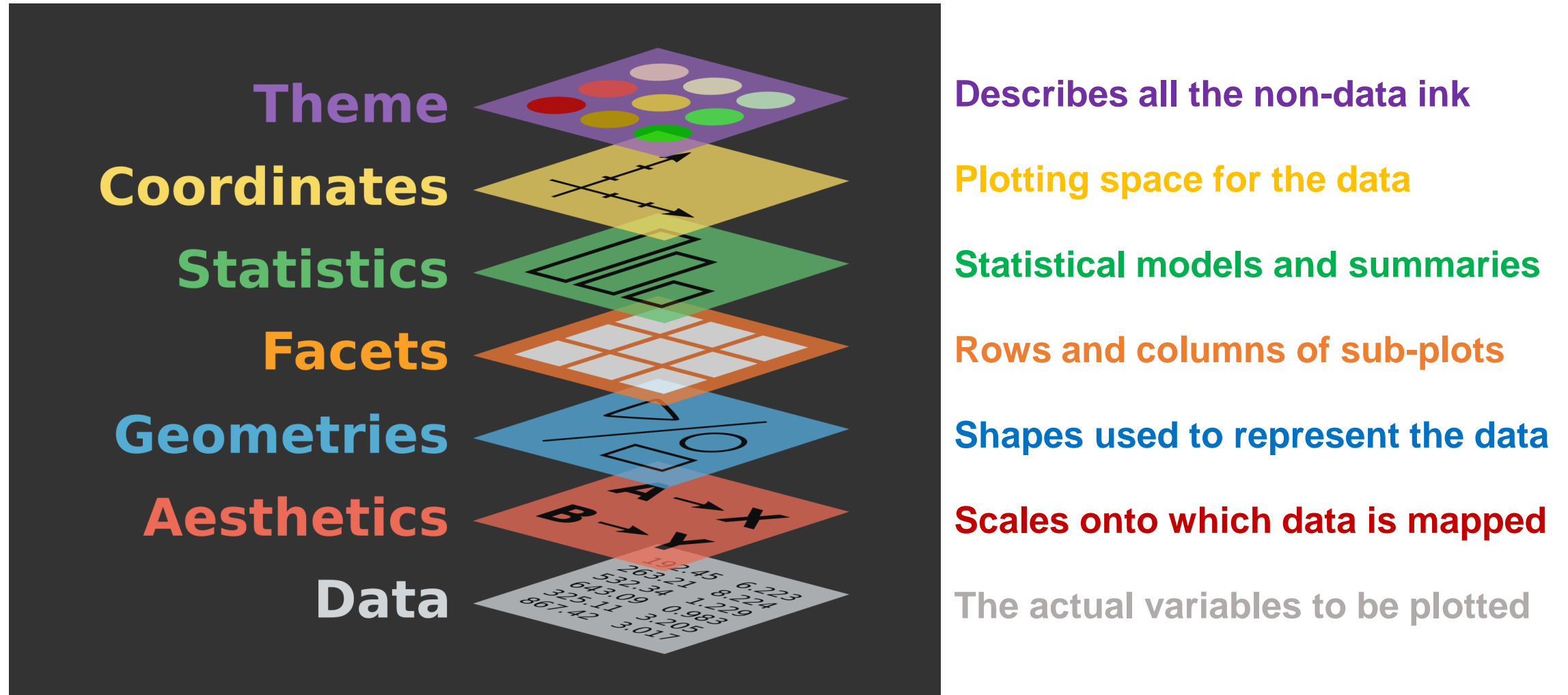
Not required, sensible defaults supplied

Syntax of ggplot2

Layers

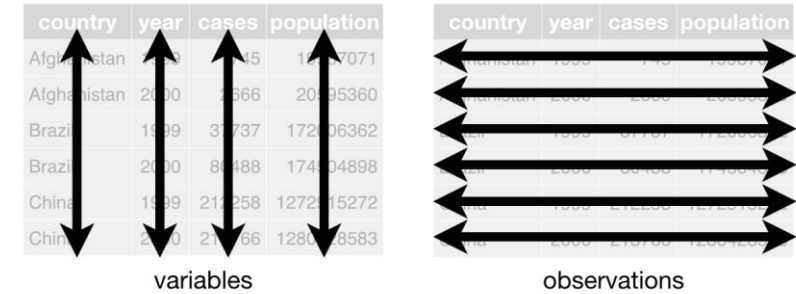


Summary – Day1



Summary – Day1

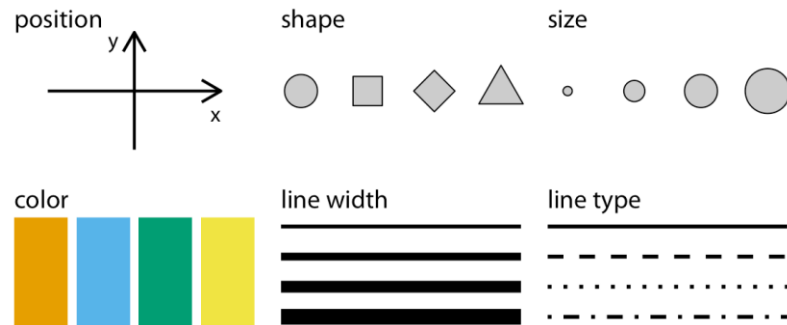
- Variable types, factors and data frame
- Data wrangling functions



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

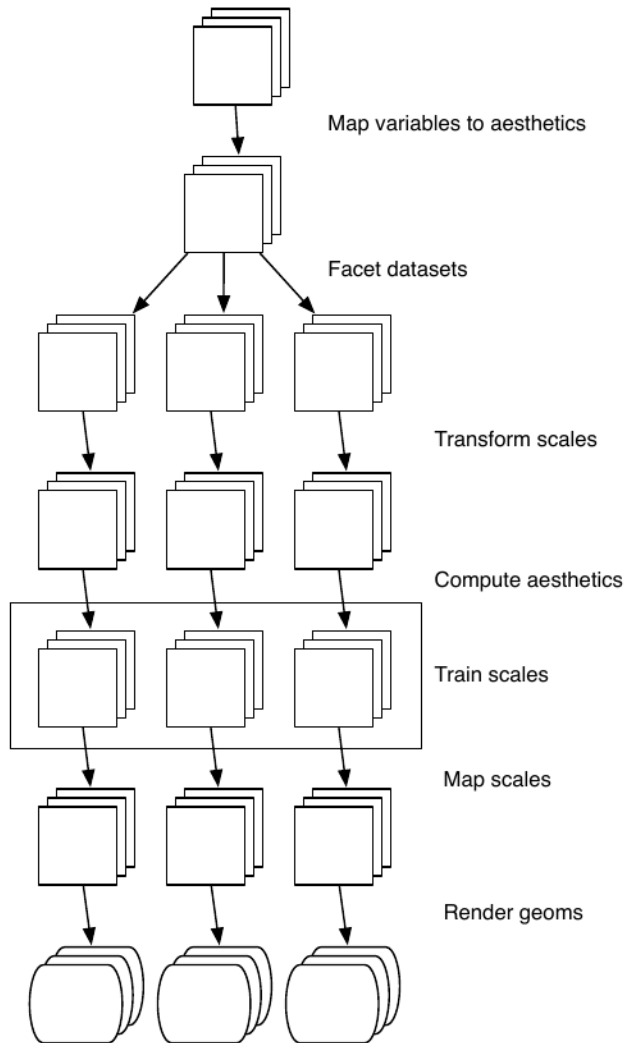
Summary – Day1

- Aesthetic mappings `aes()`
 - takes aesthetic-variable pairs (variable's type matters!)
 - describes how **variables** are mapped to **visual properties** of the **geometric objects**
- `aes(x = log(displ), y = hwy, colour = class)`

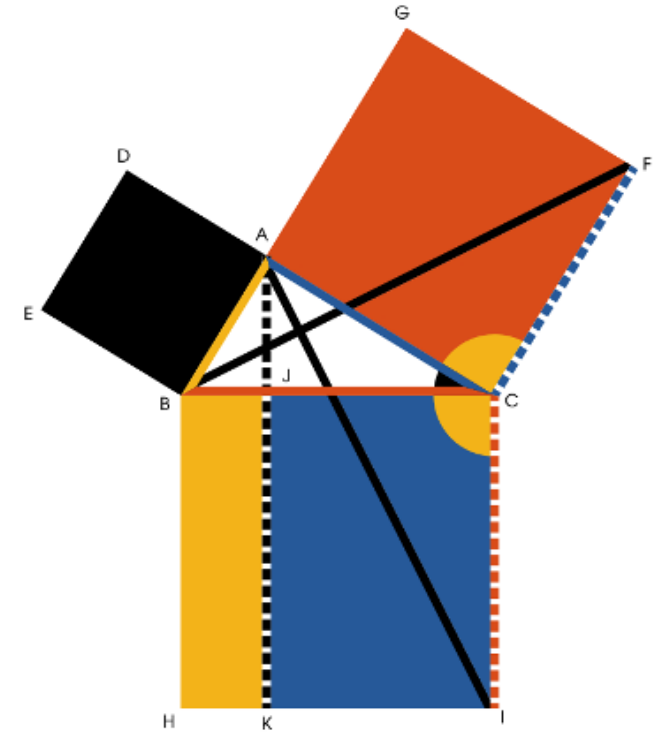


Aesthetic	Description
x	x-axis position
y	y-axis position
colour	Color of points or outlines of other shapes
fill	Fill color
size	size of the point or thickness of line
alpha	Transparency of the shape
linetype	Line type such a solid, dashed, dotted
labels	Text on the plot
shape	Shape of the geometry

Plot generation process



Geometry



Geometric objects

Geometric objects **geoms** perform the rendering of the layer, and control plot type

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



Let's do some practice!

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



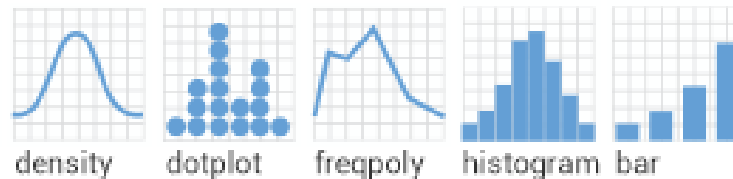
One variable

- 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()`: smoothed density estimate
 - `geom_dotplot()`: stack individual points into a dot plot

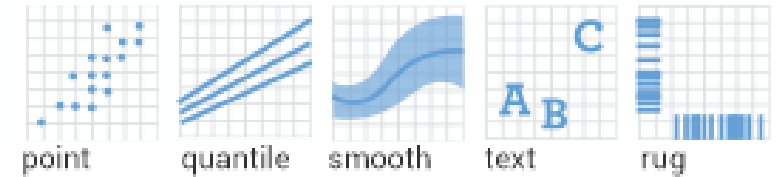


Two variables



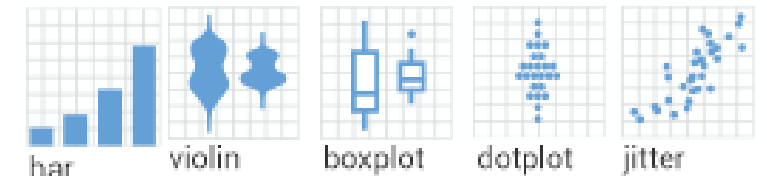
- Both continuous:

- `geom_point()`: scatterplot
- `geom_smooth()`: smoothed line of best fit
- `geom_quantile()`: smoothed quantile regression
- `geom_rug()`: marginal rug plots



- One continuous, one discrete:

- `geom_bar(stat = "identity")`: a bar chart of precomputed summaries
- `geom_boxplot()`: boxplots
- `geom_violin()`: show density of values in each group
- `geom_jitter()`: randomly jitter overlapping points



- Both discrete?

Two variables

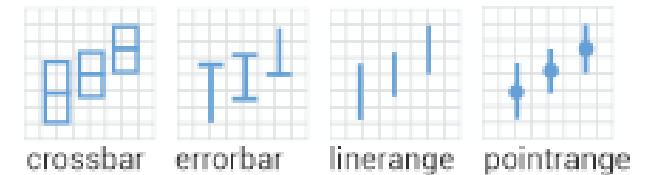
- Show 2D distribution:

- `geom_bin2d()`: bin into rectangles and count
- `geom_hex()`: bin into hexagons and count
- `geom_density2d()`: smoothed 2d density estimate



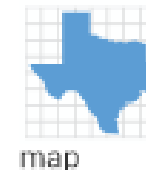
- Display uncertainty:

- `geom_crossbar()`: vertical bar with center
- `geom_errorbar()`: error bars
- `geom_linerange()`: vertical line
- `geom_pointrange()`: vertical line with center



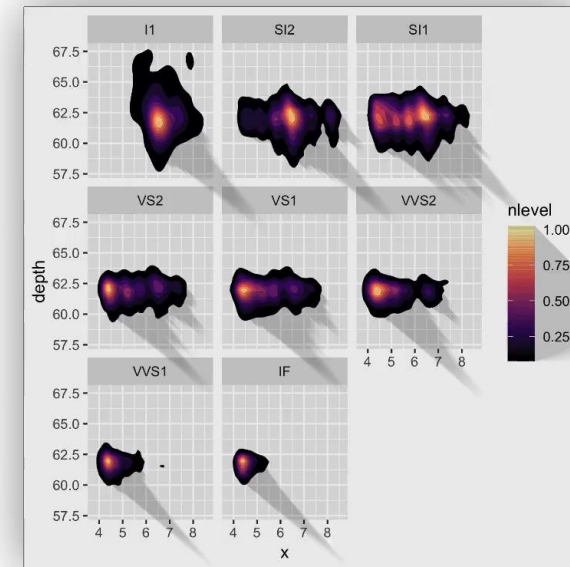
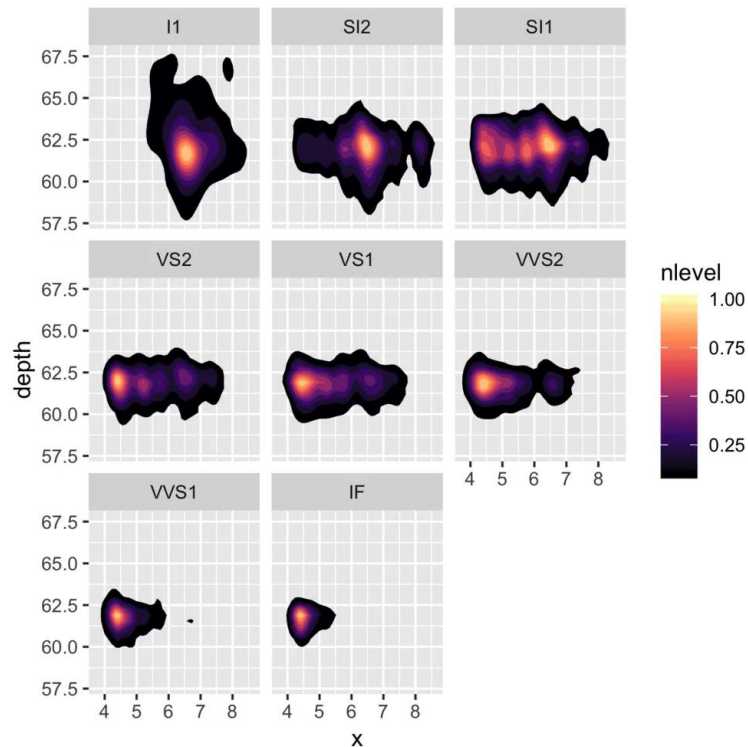
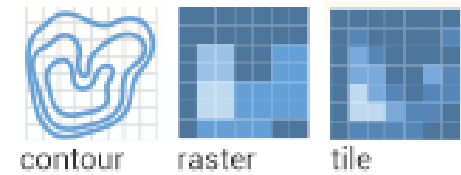
- Spatial:

- `geom_map()`: fast version of `geom_polygon()` for map data

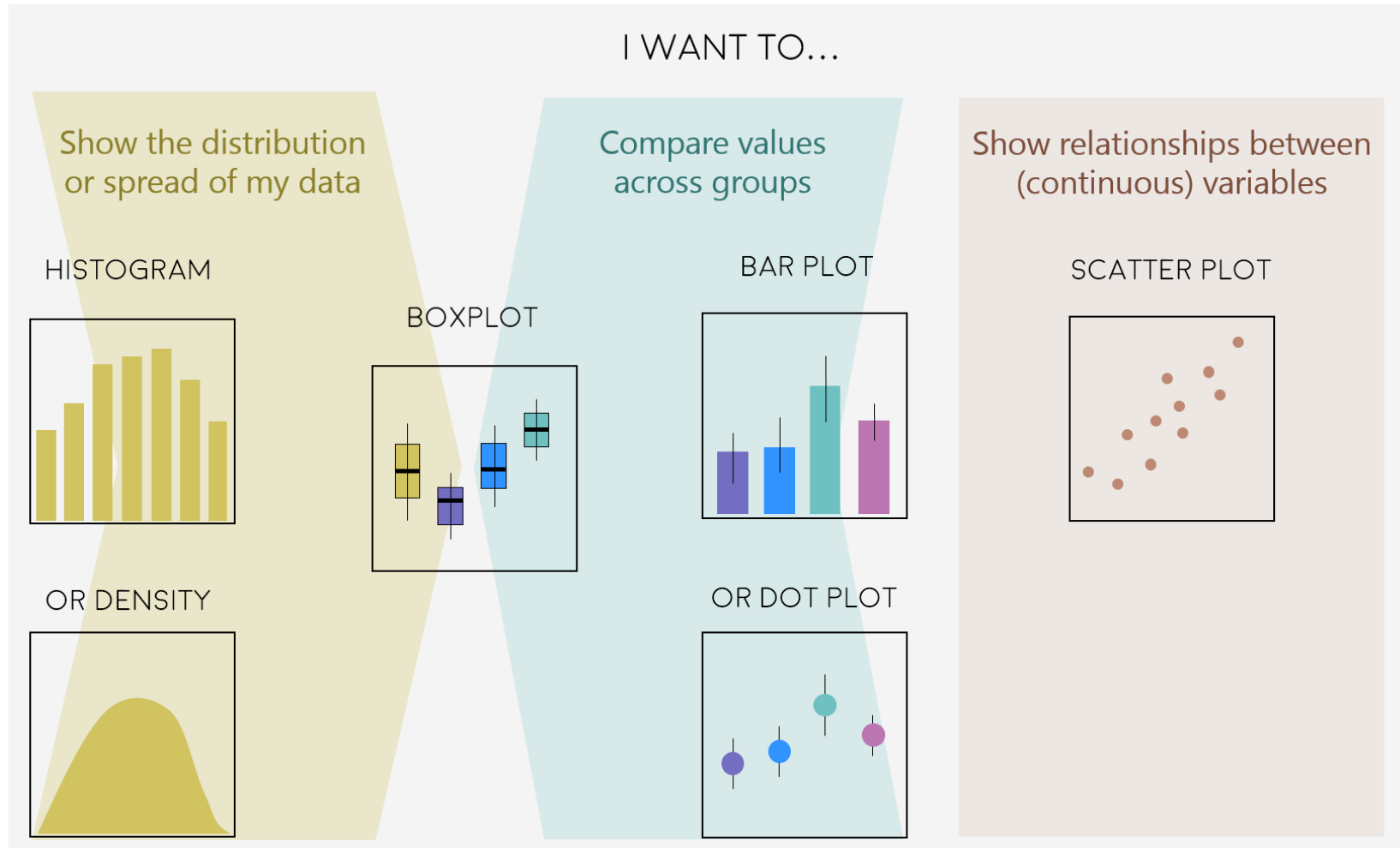


Three variables

- `geom_contour()`: contours
- `geom_tile()`: tile the plane with rectangles
- `geom_raster()`: fast version of `geom_tile()` for equal sized tiles

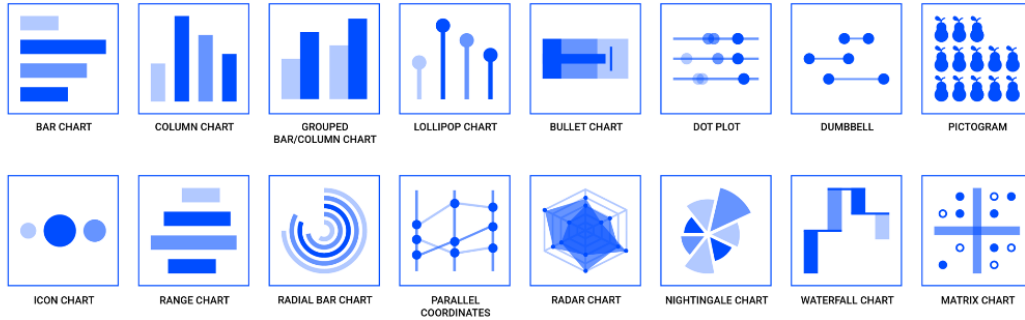


Choose geometry based on visualization goal

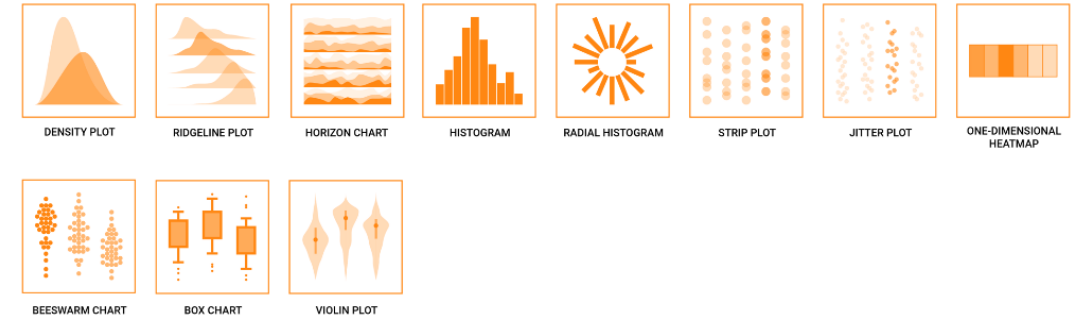


Choose geometry based on visualization goal

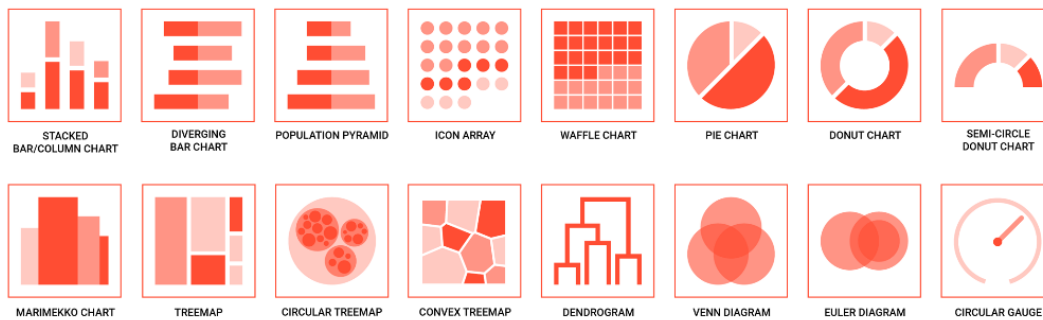
COMPARISON



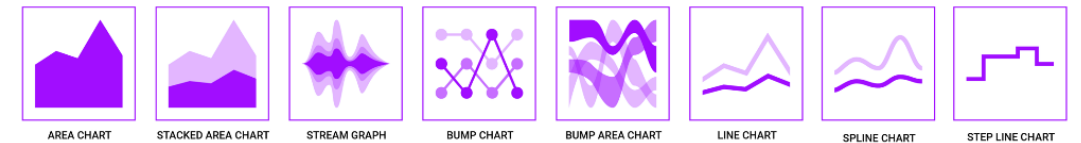
DISTRIBUTION



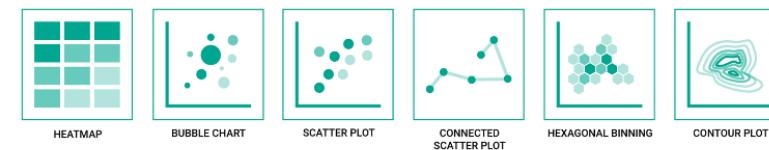
PART-TO-WHOLE & HIERARCHICAL

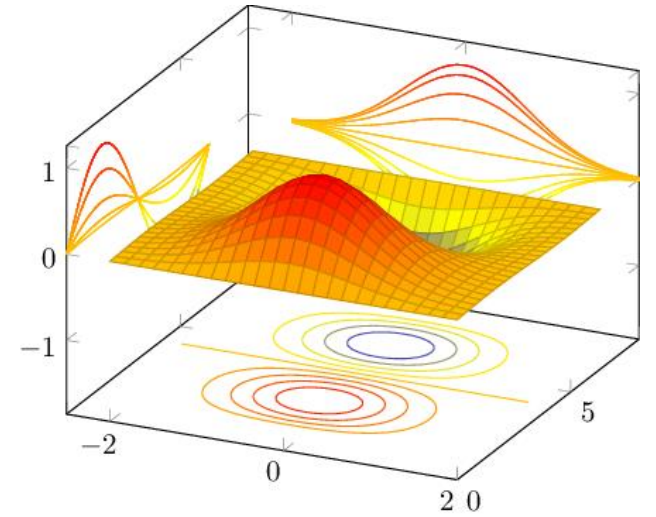


DATA OVER TIME (TEMPORAL)



CORRELATION





statistical transformation

statistical transformation

A statistical transformation (**stat**), transforms the data, typically by summarizing the input data in some manner

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>



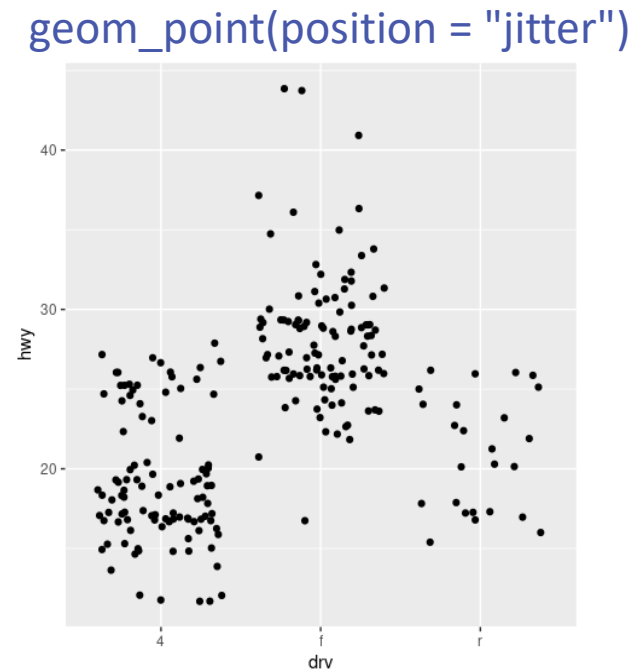
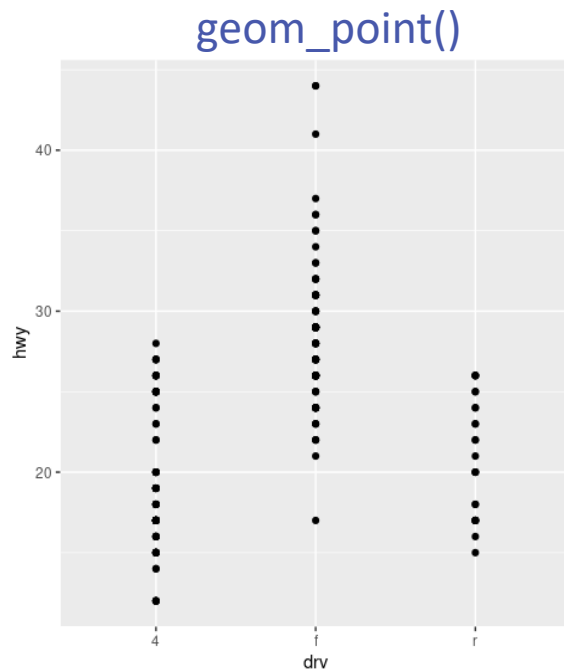
Position adjustments

Position adjustments

Position adjustments apply minor tweaks to the elements' position

For points:

- `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



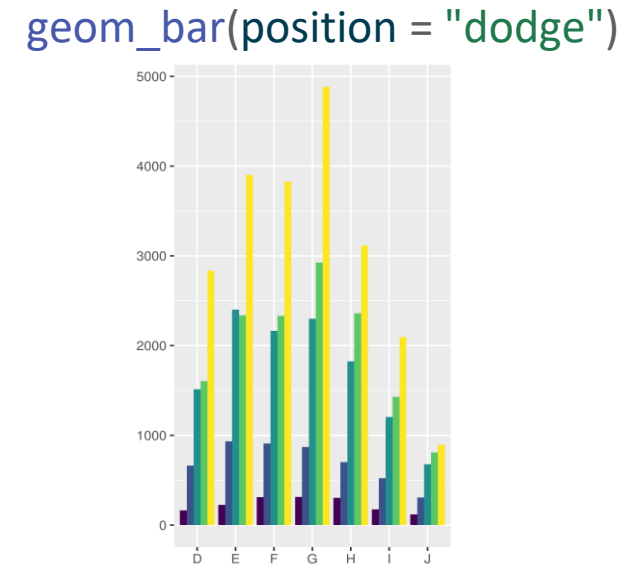
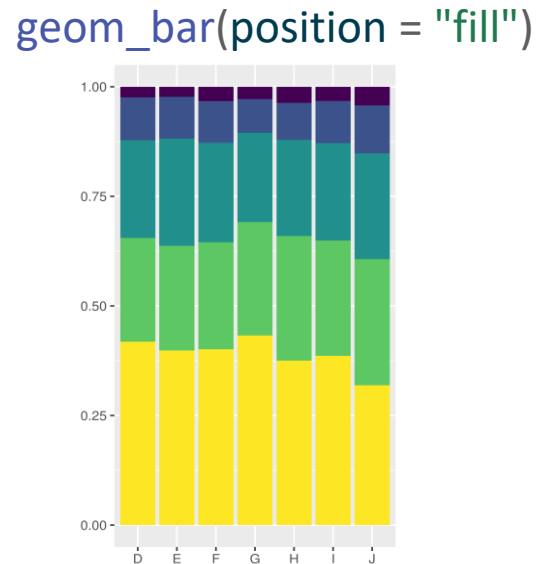
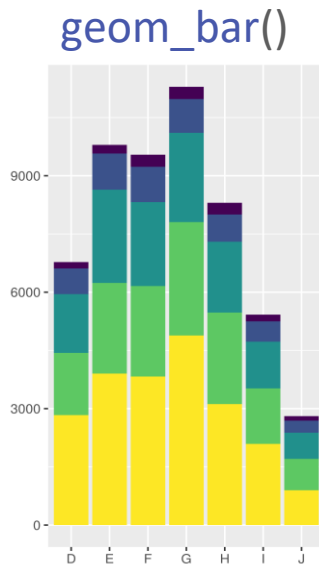
Position adjustments



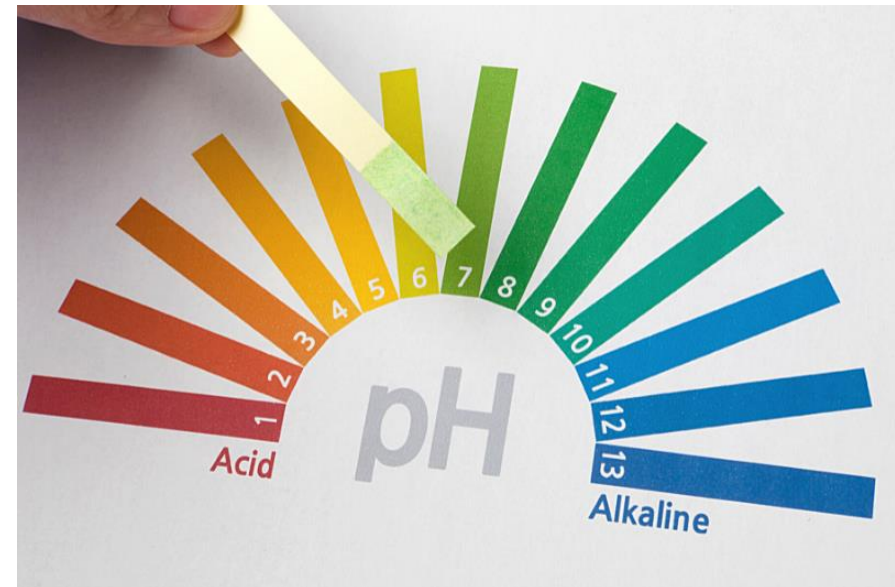
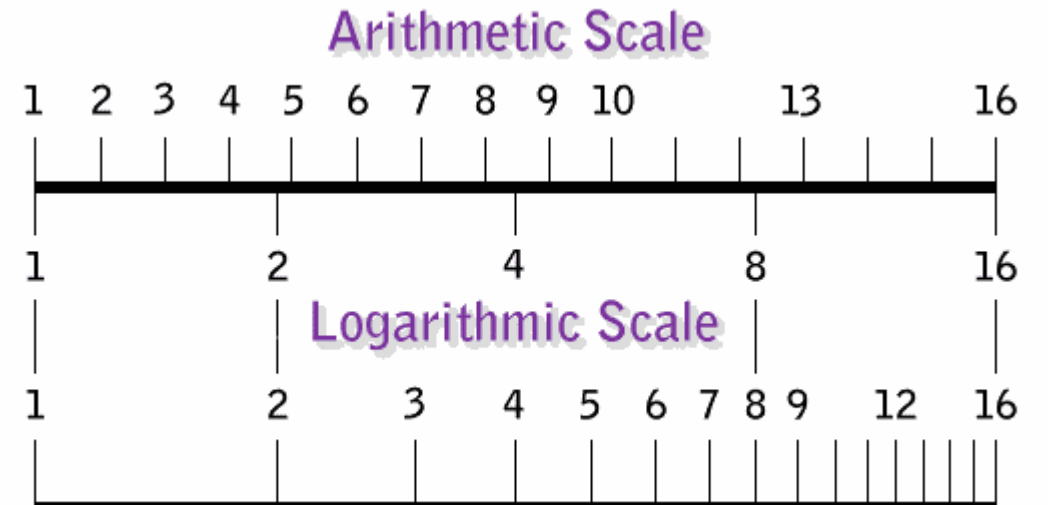
Position adjustments apply minor tweaks to the elements' position

For bars:

- `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



Scales



Scale settings

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

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

Type: continuous, discrete

Position scales

syntax: `scale_<axis>_<type>`

axis: x, y

type: discrete, continuous

arguments: name, limits, expansion, breaks, labels

Position scales

Name:

- `scale_x_continuous(names = "x")`

Limits:

- `scale_x_continuous(limits=c(1,7))`

Expansion:

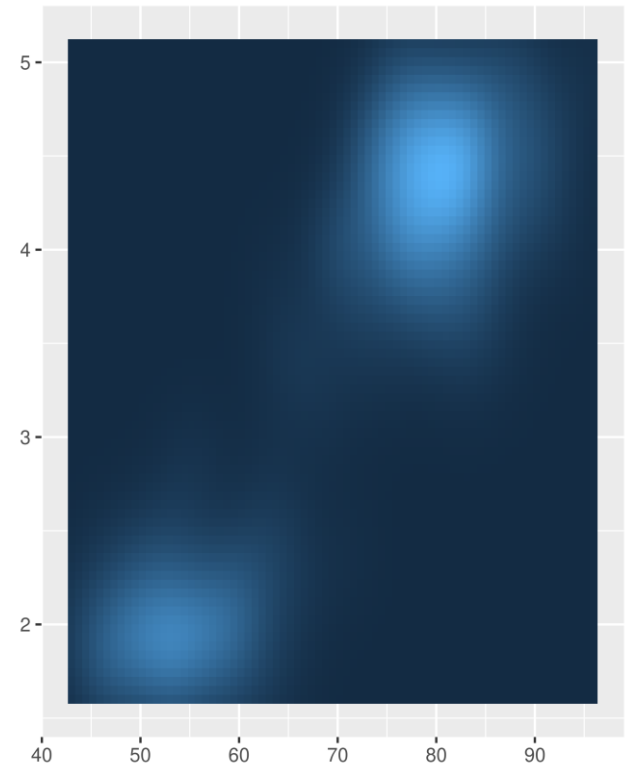
- `scale_x_continuous(expand = expansion(0))`
- `scale_x_continuous(expand = expansion(mult = c(.05, .2)))`
- `scale_x_continuous(expand = c(.05, 0, .2, 0))`

Breaks:

- `scale_x_continuous(breaks = NULL)`
- `scale_x_continuous(breaks = c(50, 75, 100))`

Labels:

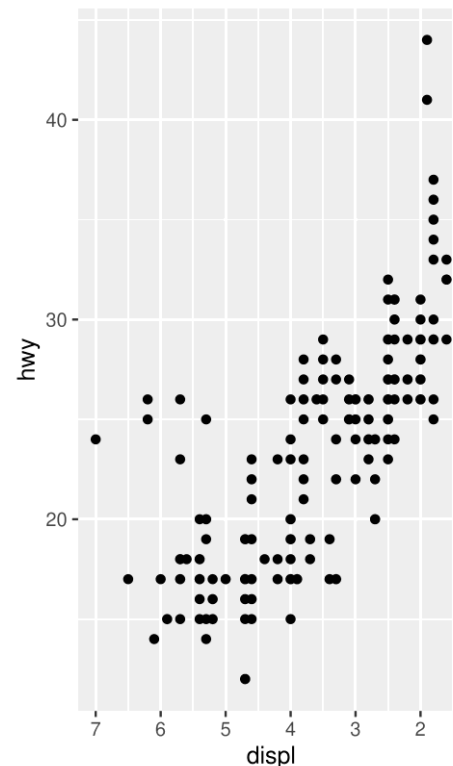
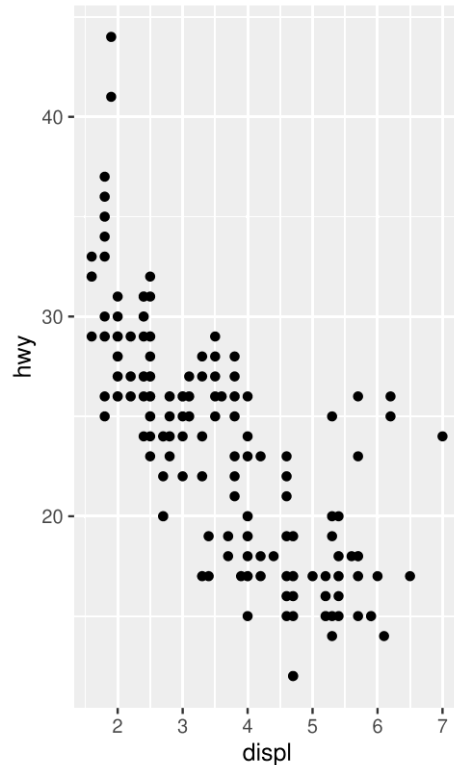
- `scale_x_continuous(labels = c(1/2, 3/4, 1))`



Position scales

Transformation:

- `scale_x_reverse()`
- `scale_x_log10()`
- `scale_x_continuous(trans = "log10")`



Name	Transformer	Function $f(x)$	Inverse $f^{-1}(x)$
"asn"	<code>scales::asn_trans()</code>	$\tanh^{-1}(x)$	$\tanh(y)$
"exp"	<code>scales::exp_trans()</code>	e^x	$\log(y)$
"identity"	<code>scales::identity_trans()</code>	x	y
"log"	<code>scales::log_trans()</code>	$\log(x)$	e^y
"log10"	<code>scales::log10_trans()</code>	$\log_{10}(x)$	10^y
"log2"	<code>scales::log2_trans()</code>	$\log_2(x)$	2^y
"logit"	<code>scales::logit_trans()</code>	$\log\left(\frac{x}{1-x}\right)$	$\frac{1}{1+e(y)}$
"probit"	<code>scales::probit_trans()</code>	$\Phi(x)$	$\Phi^{-1}(y)$
"reciprocal"	<code>scales::reciprocal_trans()</code>	x^{-1}	y^{-1}
"reverse"	<code>scales::reverse_trans()</code>	$-x$	$-y$
"sqrt"	<code>scales::scale_x_sqrt()</code>	$x^{1/2}$	y^2

More transformations

Color scales

syntax: `scale_<aes>_<type>`

aes: fill, color

type: discrete, continuous, gradient, manual...

use fill as an example

Color scales (continuous)

Set color palettes

- `scale_fill_gradient()`: produces a 2 colour gradient
 - `scale_fill_gradient(low = "grey", high = "brown")`
- `scale_fill_gradient2()`: produces a 3 colour gradient with specified midpoint
 - `scale_fill_gradient2(low = "grey", mid = "white", high = "brown", midpoint = .02)`
- `scale_fill_gradientn()`: produces an n-colour gradient

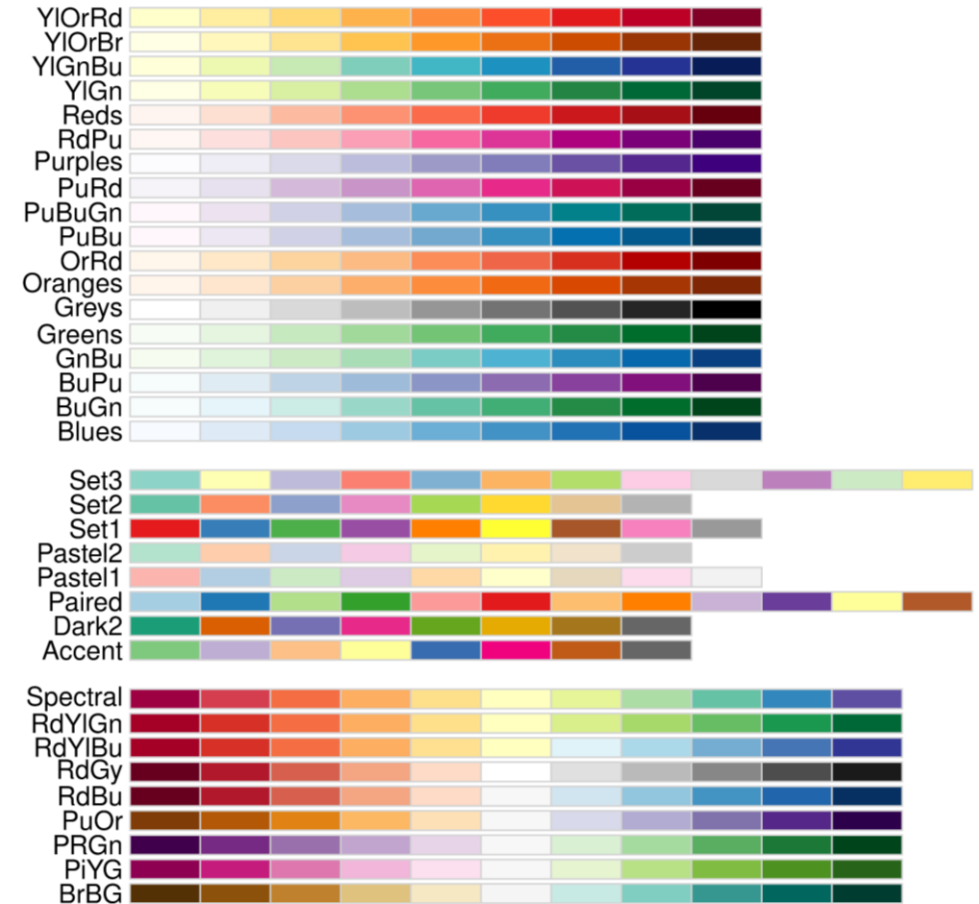
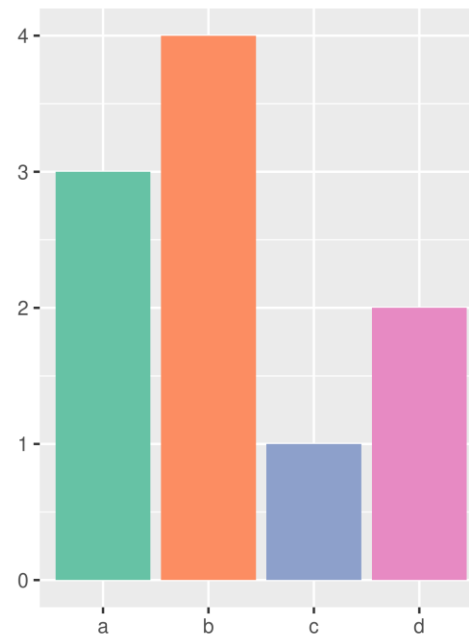
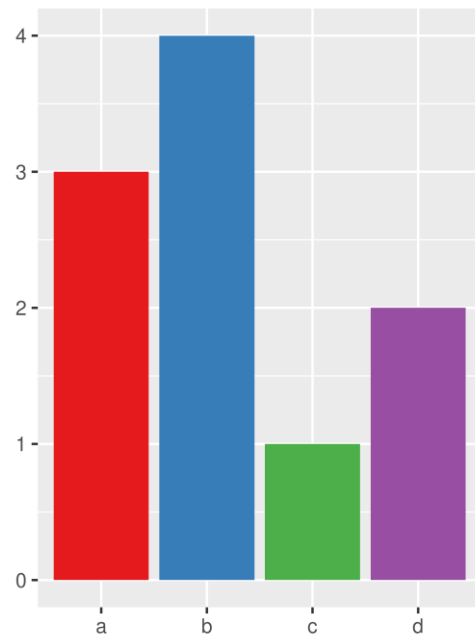
Set color for missing values

- `scale_fill_gradient(na.value = NA)`
- `scale_fill_gradient(na.value = "red")`

Color scales (discrete)

Brewer scales

- `RColorBrewer::display.brewer.all()`
- `scale_fill_brewer(palette="Set1")`
- `scale_fill_brewer(palette="Set2")`



Color scales (discrete)



Manually set color

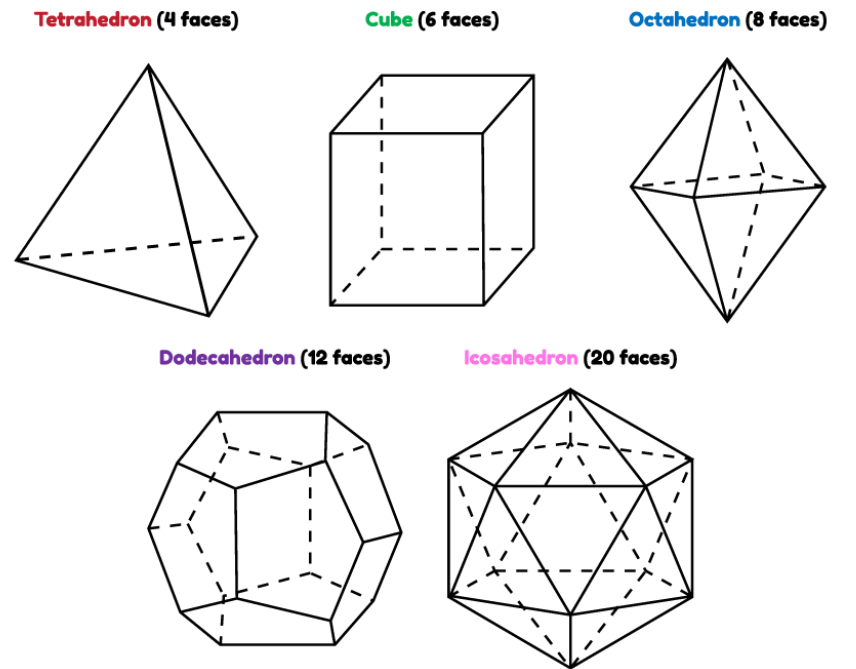
Use a vector

```
➤ scale_fill_manual(values = c("grey", "black", "grey", "grey"))
```

Use a named vector

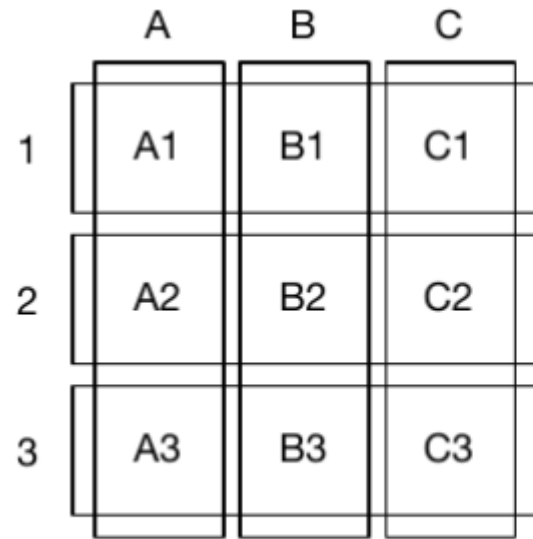
```
➤ scale_fill_manual(values = c( "d" = "grey", "c" = "grey",  
                                "b" = "black", "a" = "grey" ))
```

Facet



3 types of facet

- `facet_null()`: a single plot, the default.
- `facet_wrap()`: “wraps” a 1d ribbon of panels into 2d.
- `facet_grid()`: produces a 2d grid of panels defined by variables which form the rows and columns.



facet_grid



facet_wrap

Facet wrap

`facet_wrap()` makes a long ribbon of panels (generated by any number of variables) and wraps it into 2d

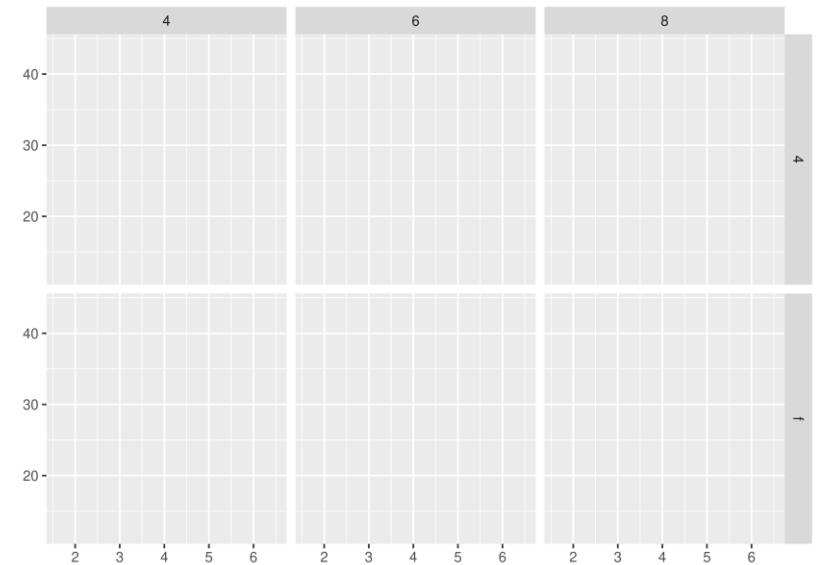
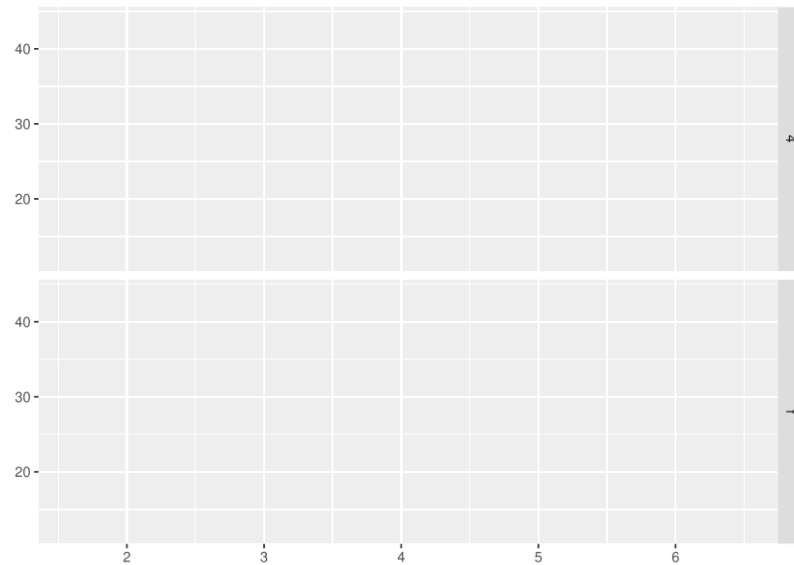
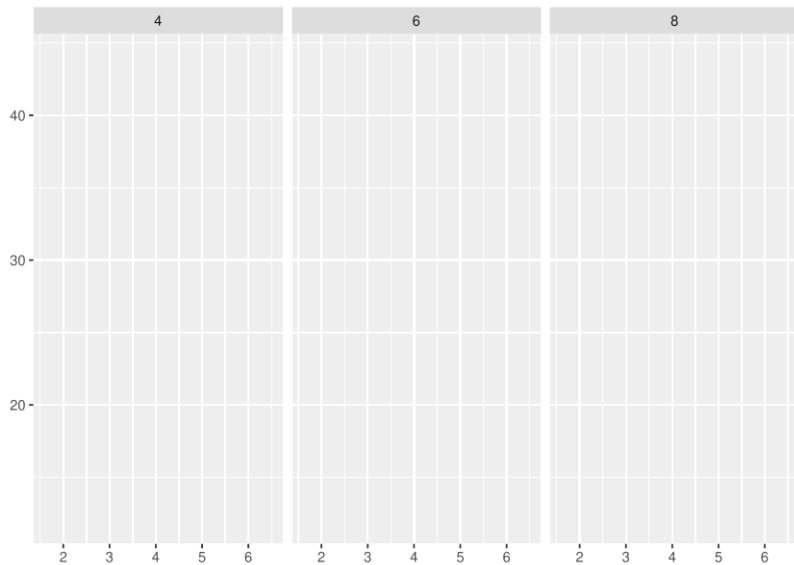
➤ `facet_wrap(~a, ...)`

Parameters

- `ncol, nrow`: controls how many rows/columns in the 2d panel
- `dir`: controls the direction of wrap: horizontal or vertical.

Facet grid

- `facet_grid()` lays out plots in a 2d grid
 - `facet_grid(. ~ a)`: spreads the values of a across the columns
 - `facet_grid(b ~ .)`: spreads the values of b down the rows
 - `facet_grid(b ~ a)`: b ~ a spreads a across columns and b down rows



Controlling scales

control whether the position scales are the same in all panels (fixed) or allowed to vary between panels (free) with the `scales` parameter:

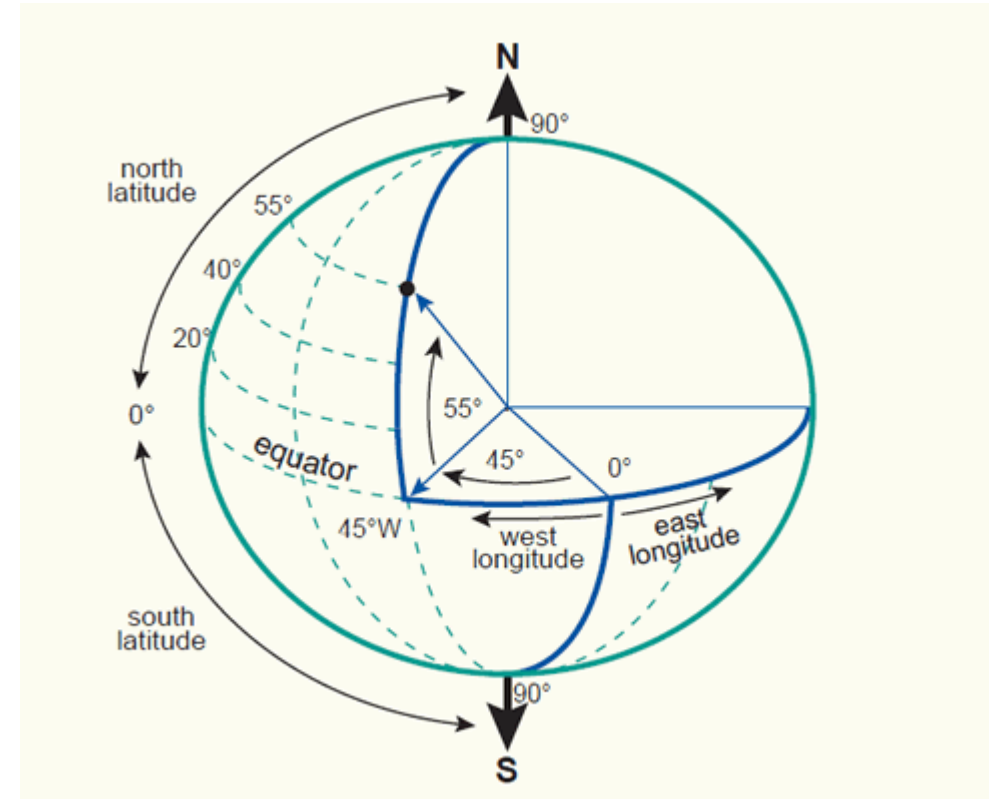
- `scales = "fixed"`: x and y scales are fixed across all panels.
- `scales = "free_x"`: the x scale is free, and the y scale is fixed.
- `scales = "free_y"`: the y scale is free, and the x scale is fixed.
- `scales = "free"`: x and y scales vary across panels.



Things to consider when using facet

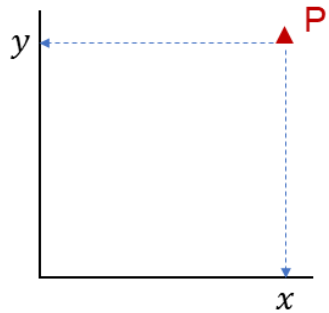
- When multiple data frames are used, missing faceting variables are treated like they exist in every data frame
- If we want to facet on a continuous variable, discretization is needed
 - Divide data into n bins with same length `cut_interval(x, n)`
 - Divide data into bins of width `cut_width(x, width)`
 - Divide data into n bins with same number of points `cut_number(x, n = 10)`
- Grouping vs. faceting
 - When using aesthetics to differentiate groups, the groups are close together and may overlap, but small differences are easier to see
 - With faceting, each group is quite far apart in its own panel, which is beneficial to avoid overlapping, but it does make small differences harder to see

Coordinates



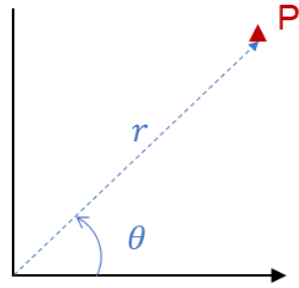
Coordinate system: linear vs nonlinear

Cartesian coordinates



$$P = (x, y)$$

Polar coordinates

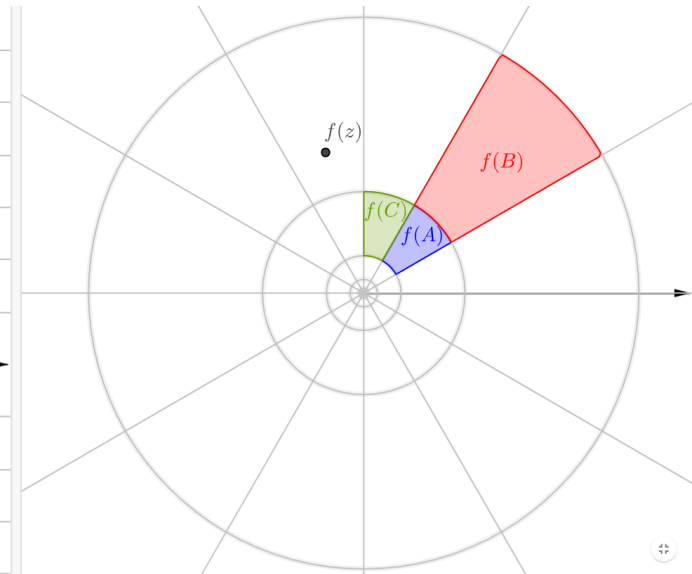
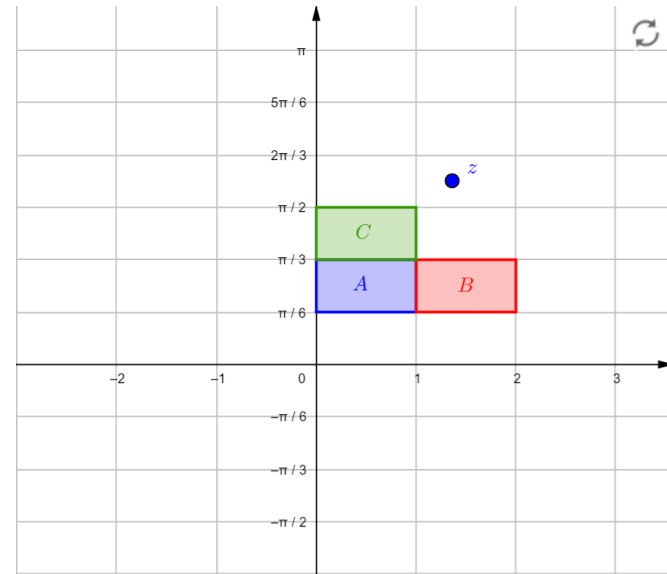


$$P = (\theta, r)$$

Convert Polar to Cartesian

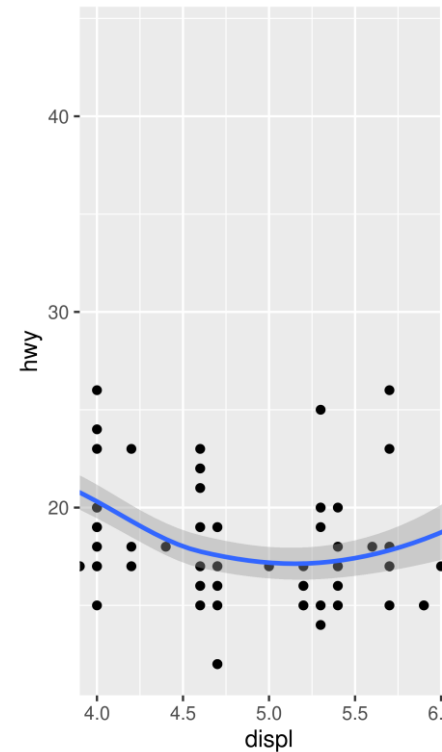
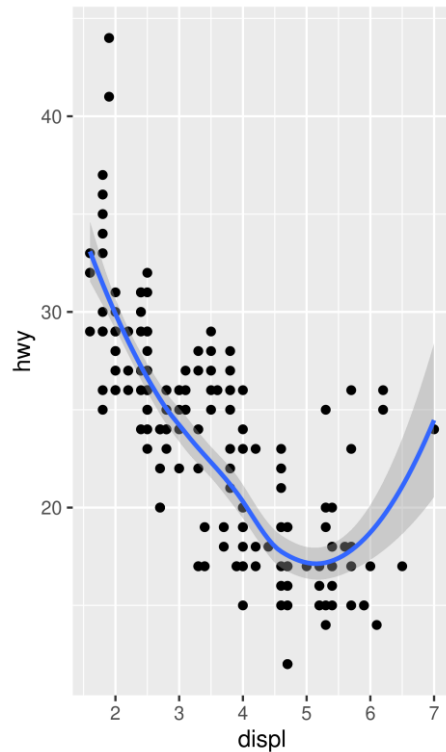
$$x = r \cos \theta$$

$$y = r \sin \theta$$



Linear coordinate systems

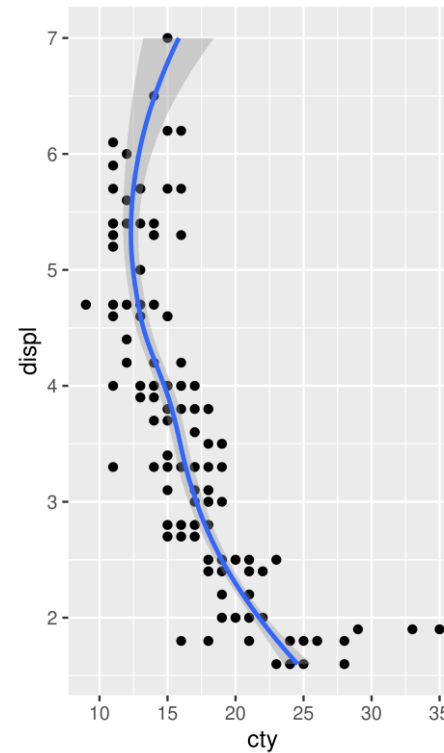
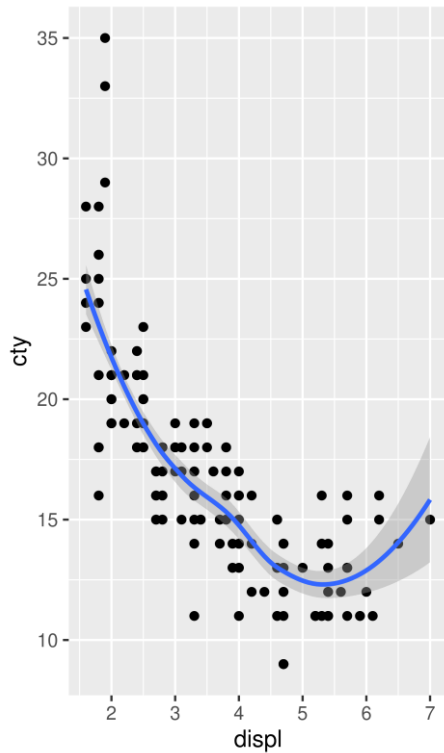
- Zooming into a plot with `coord_cartesian()`
 - `coord_cartesian(xlim = c(4, 6))`



Compare it to `scale_x_continuous(limits = c(4, 6))`, what do you find?

Linear coordinate systems

- Flipping the axes with `coord_flip()`
 - `coord_flip()`



Compared to directly switch the x and y, what do you find?

Linear coordinate systems

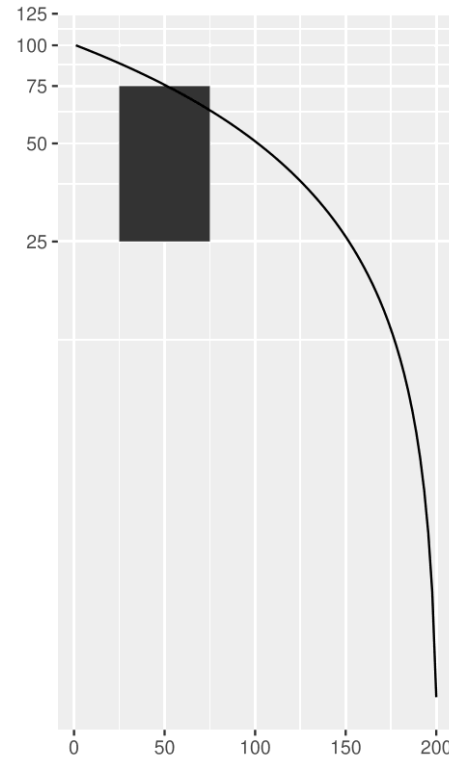
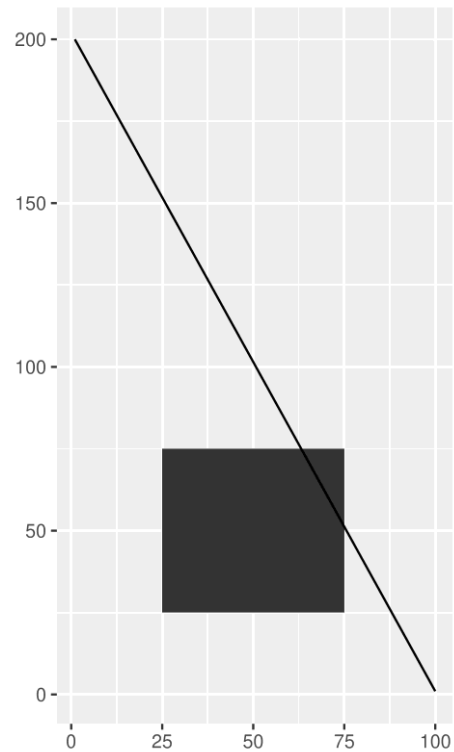
- Equal scales with `coord_fixed()`

➤ `coord_fixed(ratio = 1)`

Non-linear coordinate systems

- Transformations with `coord_trans()`

- `coord_trans(x = log10, y = log10)`

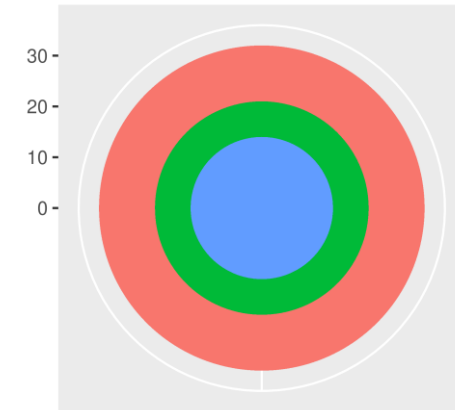
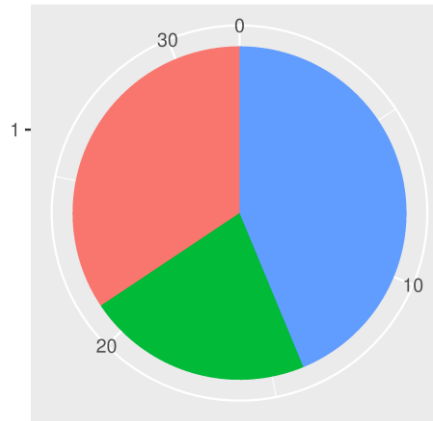
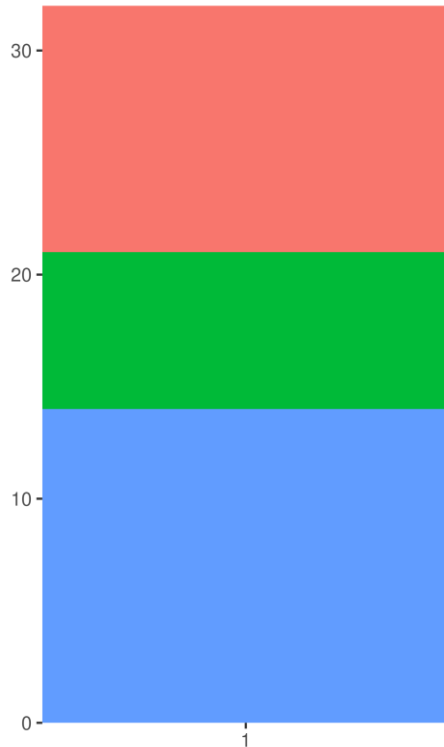


Non-linear coordinate systems

- Polar coordinates with `coord_polar()`

➤ `coord_polar(theta = "x")`

The `theta` argument determines which position variable is mapped to angle (by default, `x`) and which to radius



Valentine's special

Use what we learnt today to generate a heart curve

1. Generate the data
2. Plot the heart curve
3. Save into an image

