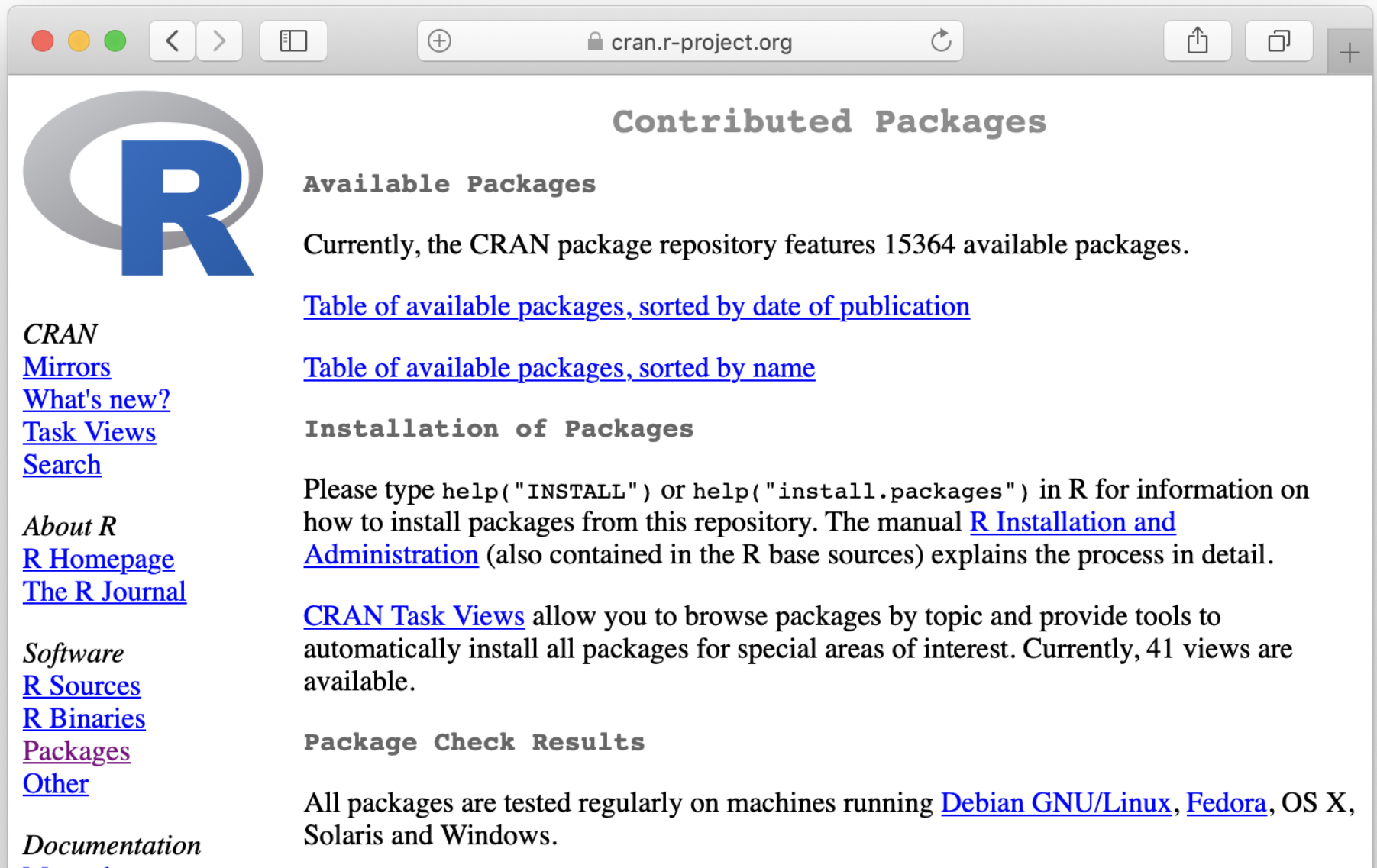


Introduction to R for Biologists

Day 2 – Data visualization with ggplot2

Extending R through packages:
There's a package for everything

R packages are available on CRAN (Comprehensive R Archive Network)

A screenshot of a web browser displaying the CRAN (Comprehensive R Archive Network) website. The browser's address bar shows 'cran.r-project.org'. The website features the R logo on the left and a main content area with various links and information. The browser window includes standard macOS window controls (red, yellow, green buttons) and navigation icons (back, forward, home, search, share, print, and a plus sign for more options).

Contributed Packages

Available Packages

Currently, the CRAN package repository features 15364 available packages.

[Table of available packages, sorted by date of publication](#)

[Table of available packages, sorted by name](#)

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual [R Installation and Administration](#) (also contained in the R base sources) explains the process in detail.

[CRAN Task Views](#) allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 41 views are available.

Package Check Results

All packages are tested regularly on machines running [Debian GNU/Linux](#), [Fedora](#), OS X, Solaris and Windows.

CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

You can install packages using `install.packages()` in RStudio

Console ~/

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> `install.packages("ggplot2")`

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
0	0	0	0	0	--:--:--	--:--:--	0 38 1932k
38	751k	0	0 1529k	0 0:00:01	0:00:01	1527k100	1932k 100 1932k
0	0	2918k	0	--:--:--	--:--:--	--:--:--	2918k

The downloaded binary packages are in
/var/folders/q8/wptgtbdn1pz0cfgrz39gq00m0000gn/T//RtmpvQgw1u/downloaded_packages

> |

ggplot2: A grammar of graphics

Traditional plotting: You **are** a painter

- Manually place individual graphical elements

ggplot2: You **employ** a painter

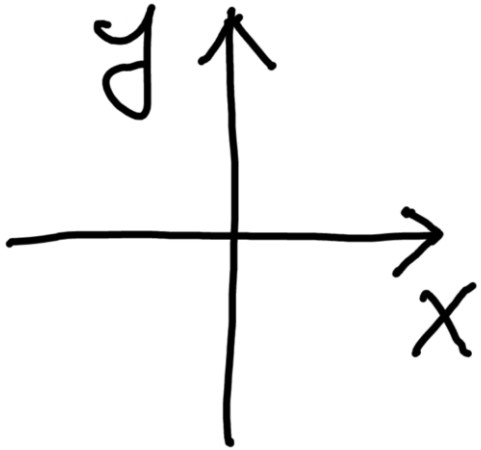
- Describe conceptually how data should be visualized

Most confusing key concept: aesthetic mapping

Maps data values to visual elements of the plot

A few examples of aesthetics

position



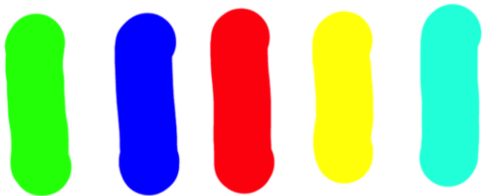
shape



size



color



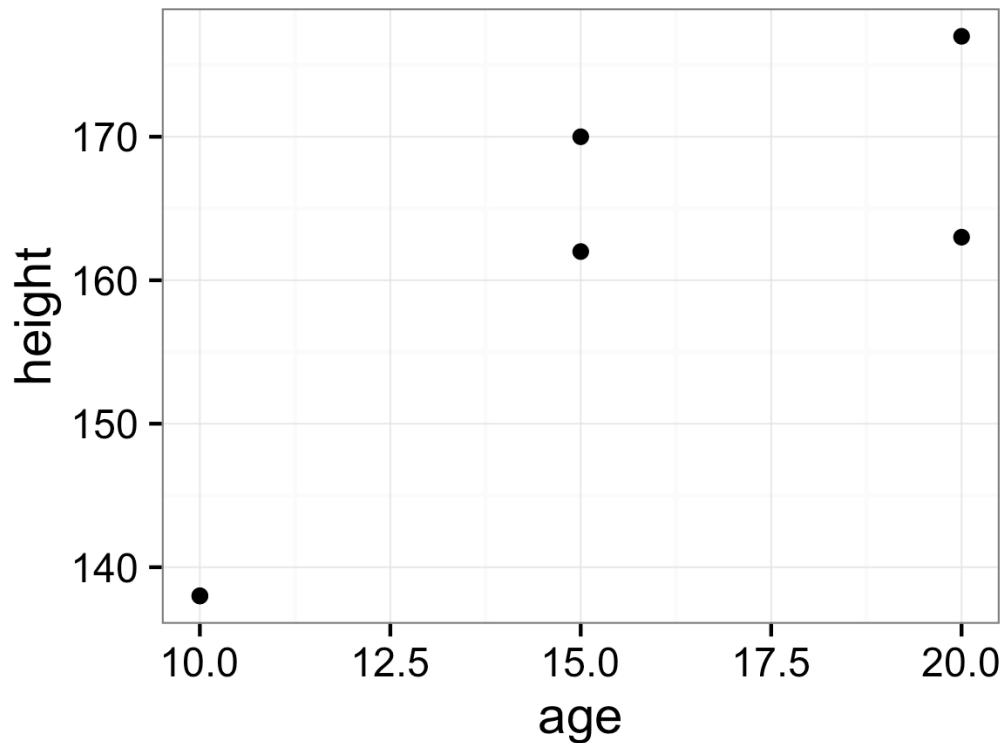
Let's go over a simple example: mean height and weight of boys/girls ages 10-20

age (yrs)	height (cm)	weight (kg)	sex
10	138	32	M
15	170	56	M
20	177	71	M
10	138	33	F
15	162	52	F
20	163	53	F

Data from: <http://www.cdc.gov/growthcharts/>

Map age to x, height to y, visualize using points

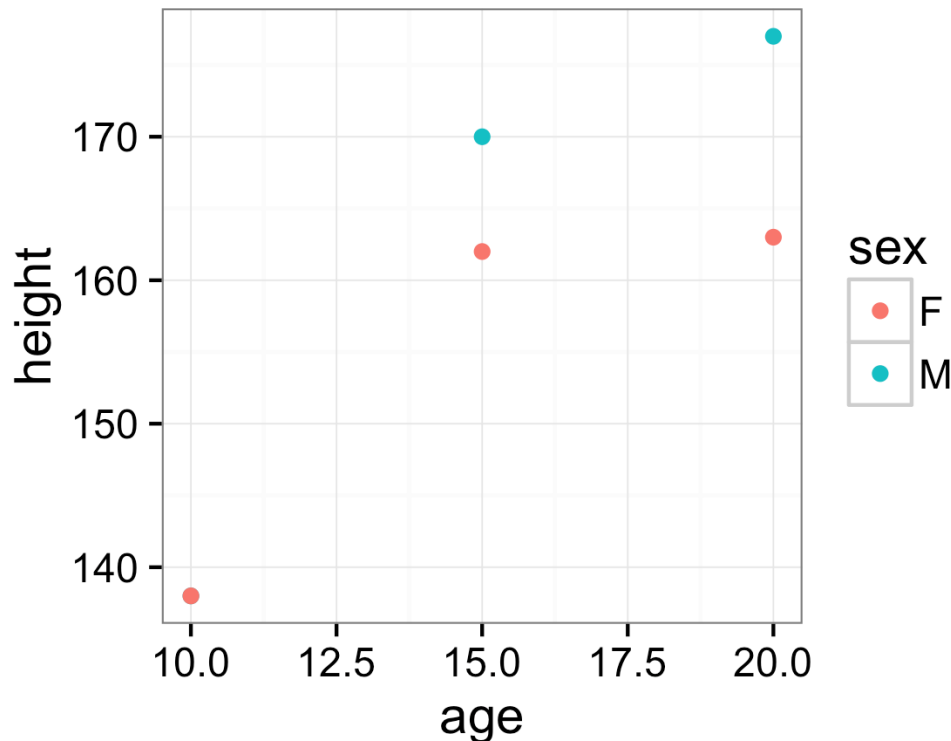
```
ggplot(data, aes(x=age, y=height)) +  
  geom_point()
```



Let's color the points by sex

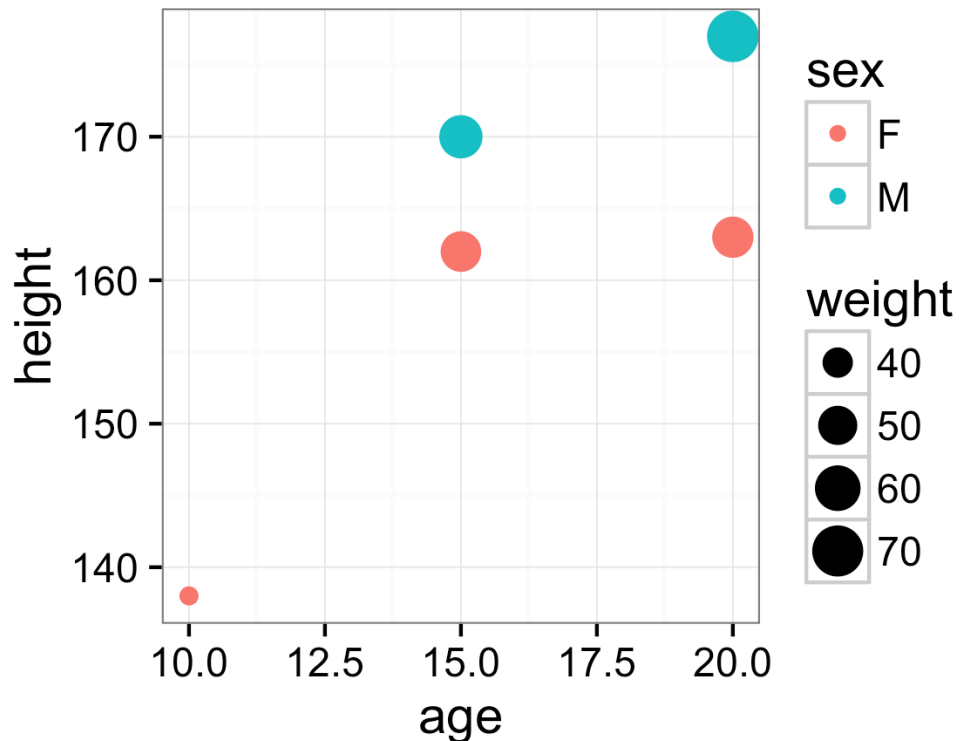
```
ggplot(data, aes(x=age, y=height,  
                  color=sex)) + geom_point()
```

★ NOTE: “color”
aesthetic is for
coloring points
& lines;
“fill” aesthetic is
for coloring bars
& distributions



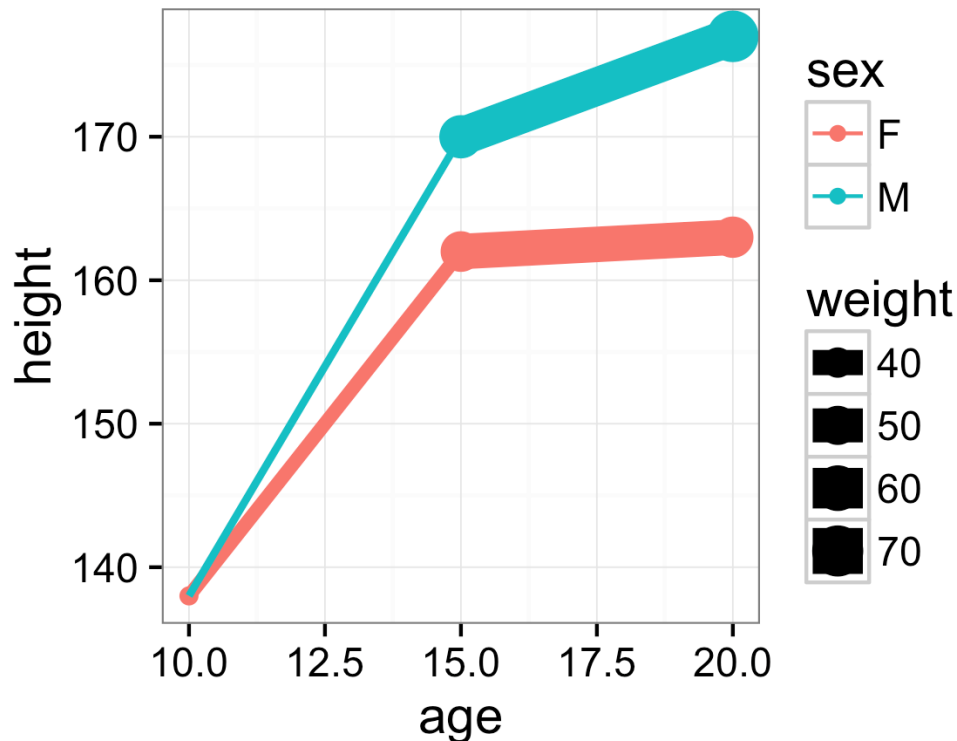
And change point size by weight

```
ggplot(data, aes(x=age, y=height,  
  color=sex, size=weight)) + geom_point()
```



And connect the points with lines

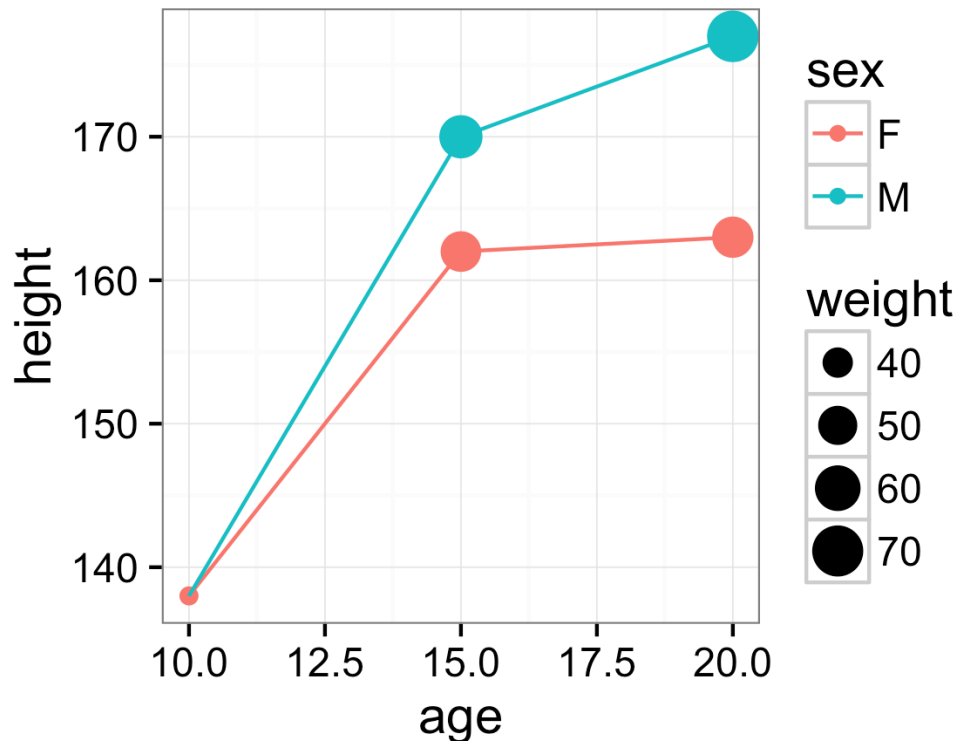
```
ggplot(data, aes(x=age, y=height,  
  color=sex, size=weight)) +  
  geom_point() + geom_line()
```



Oops!

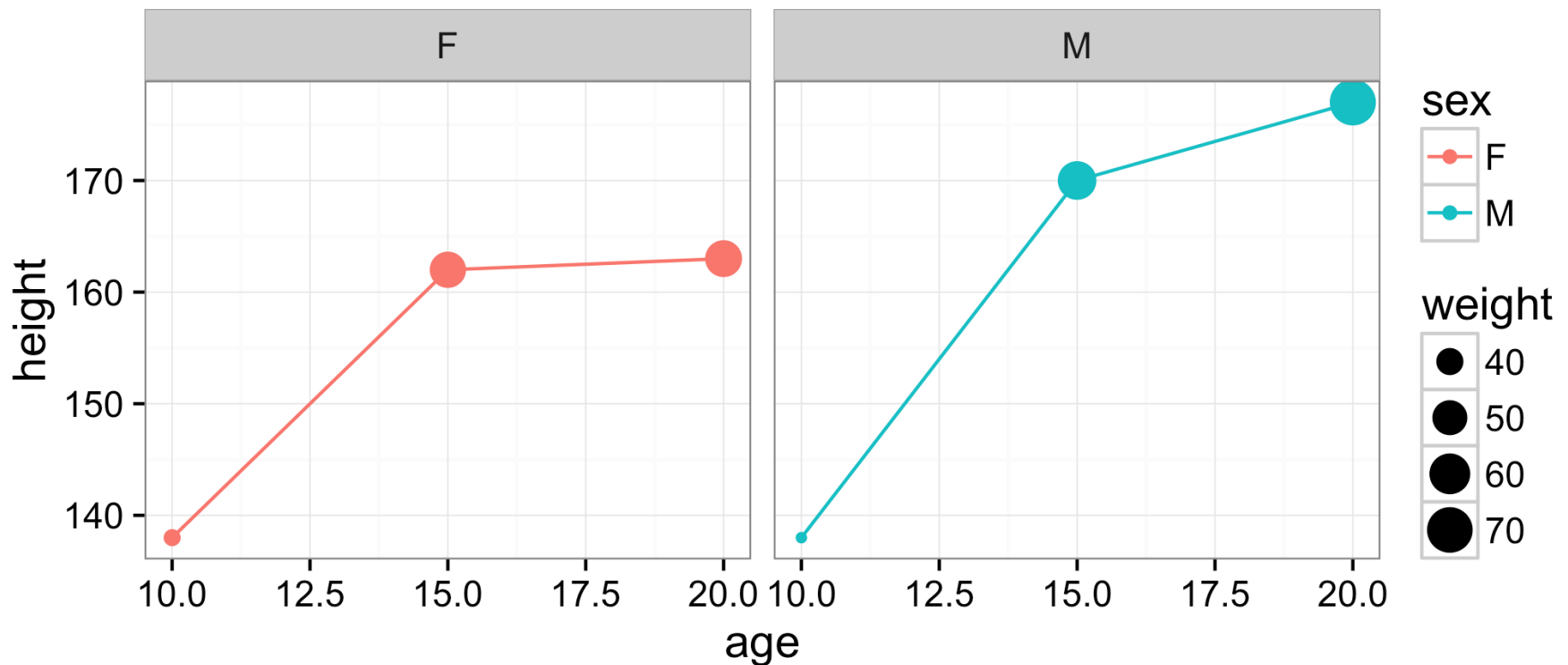
The weight-to-size mapping should only be applied to points

```
ggplot(data, aes(x=age, y=height,  
  color=sex)) + geom_point(aes(size=weight)) +  
  geom_line()
```



We can also make side-by-side plots (called facets)

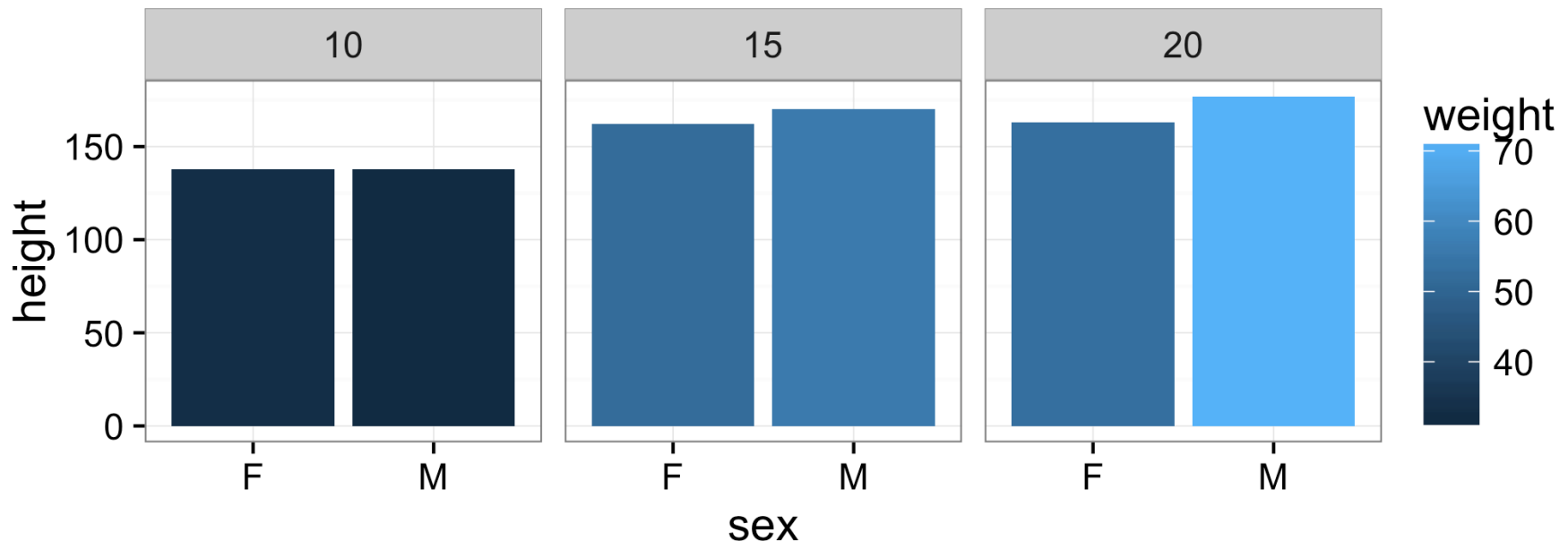
```
ggplot(data, aes(x=age, y=height,  
  color=sex)) + geom_point(aes(size=weight)) +  
  geom_line() + facet_wrap(~sex)
```



Now let's facet by age, color by weight, and use bars (columns) to plot height

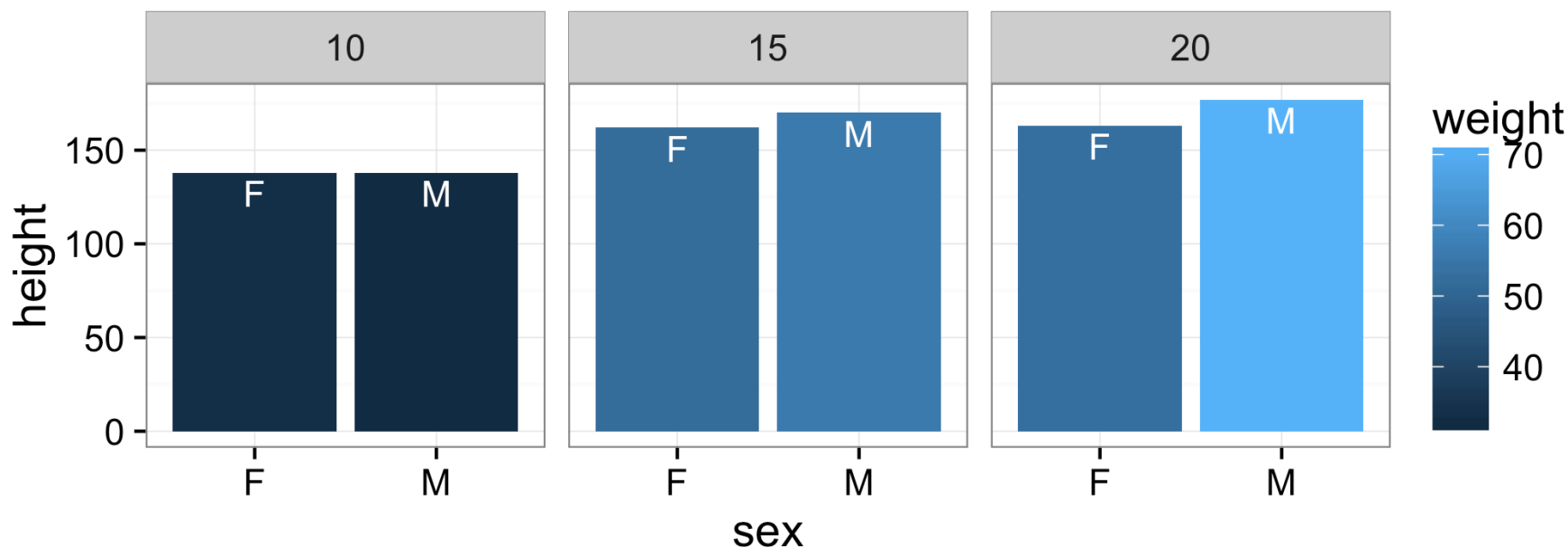
```
ggplot(data, aes(x=sex, y=height, fill=weight)) +  
  geom_col() + facet_wrap(~age)
```

NOTE: "fill" ★
aesthetic is for
coloring bars &
distributions



Let's plot the sex also at the top of the bar

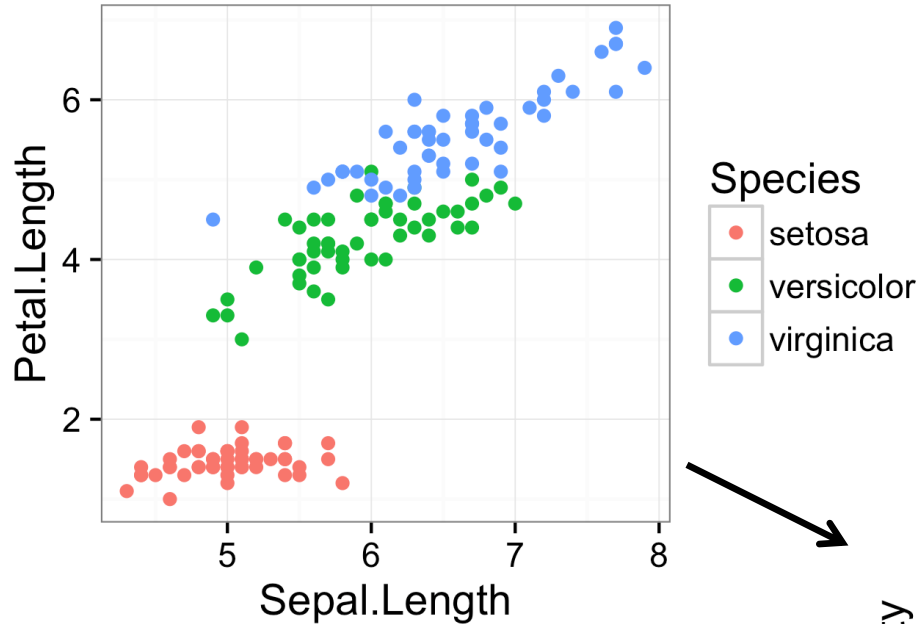
```
ggplot(data, aes(x=sex, y=height, fill=weight)) +  
  geom_col() +  
  geom_text(aes(label=sex), vjust=1.3, color='white') +  
  facet_wrap(~age)
```



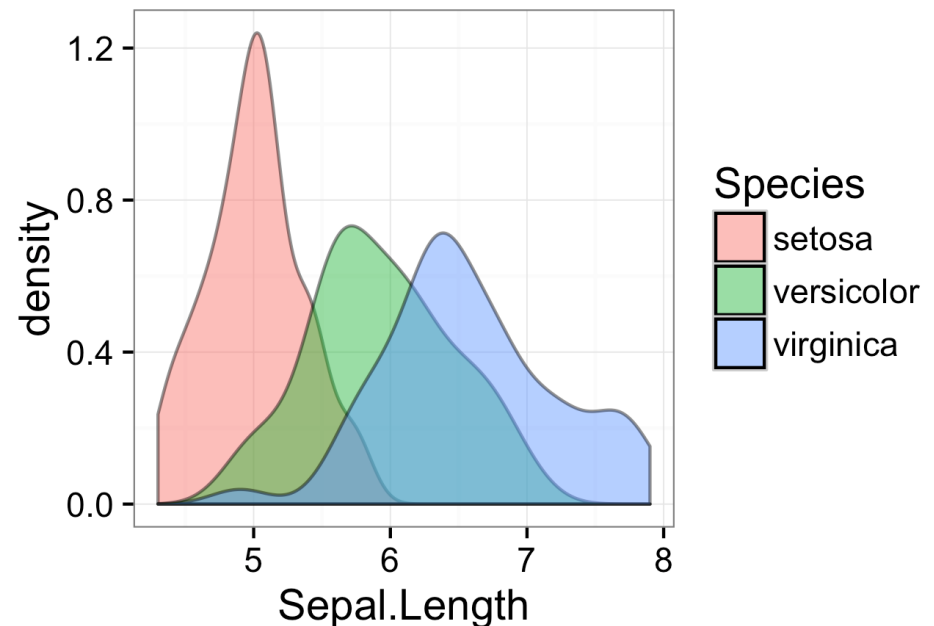
All the geoms with all their options are described on the ggplot2 web page

<https://ggplot2.tidyverse.org/reference/>

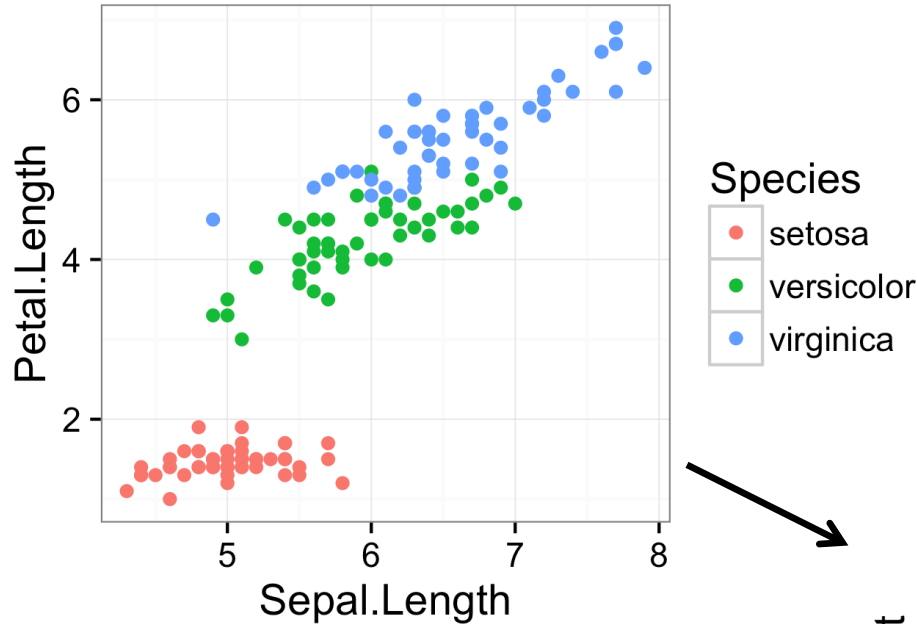
We often need to do statistical transformations before plotting



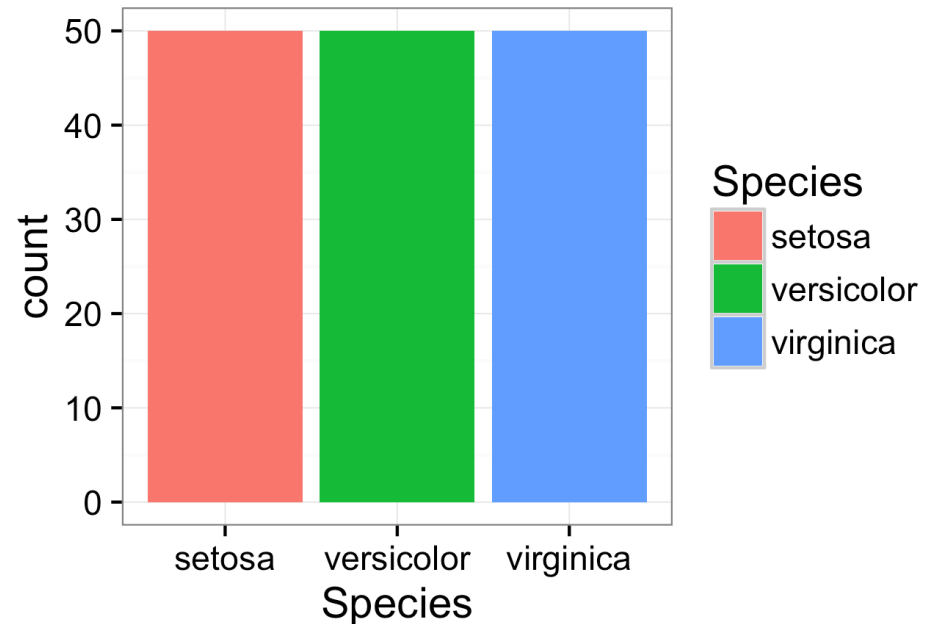
density of
data points



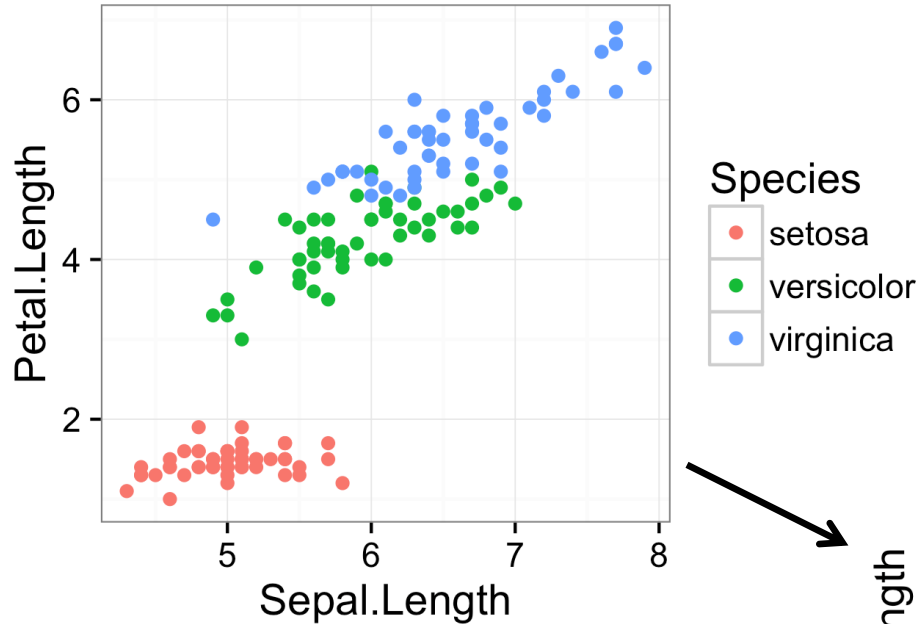
We often need to do statistical transformations before plotting



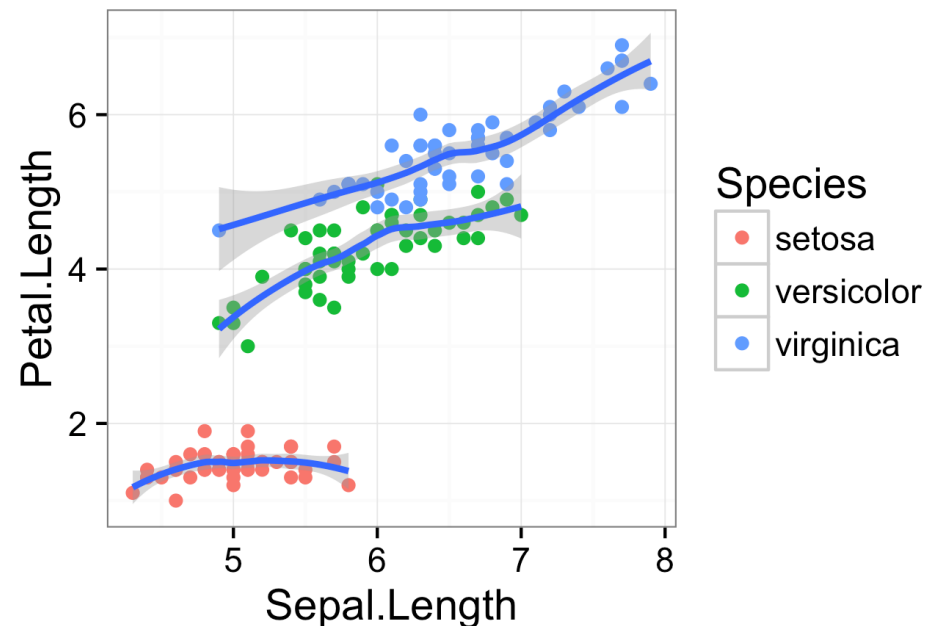
count of number
of different types



We often need to do statistical transformations before plotting



statistical smoothing/
trend lines



In ggplot2, these transformations are done with stats

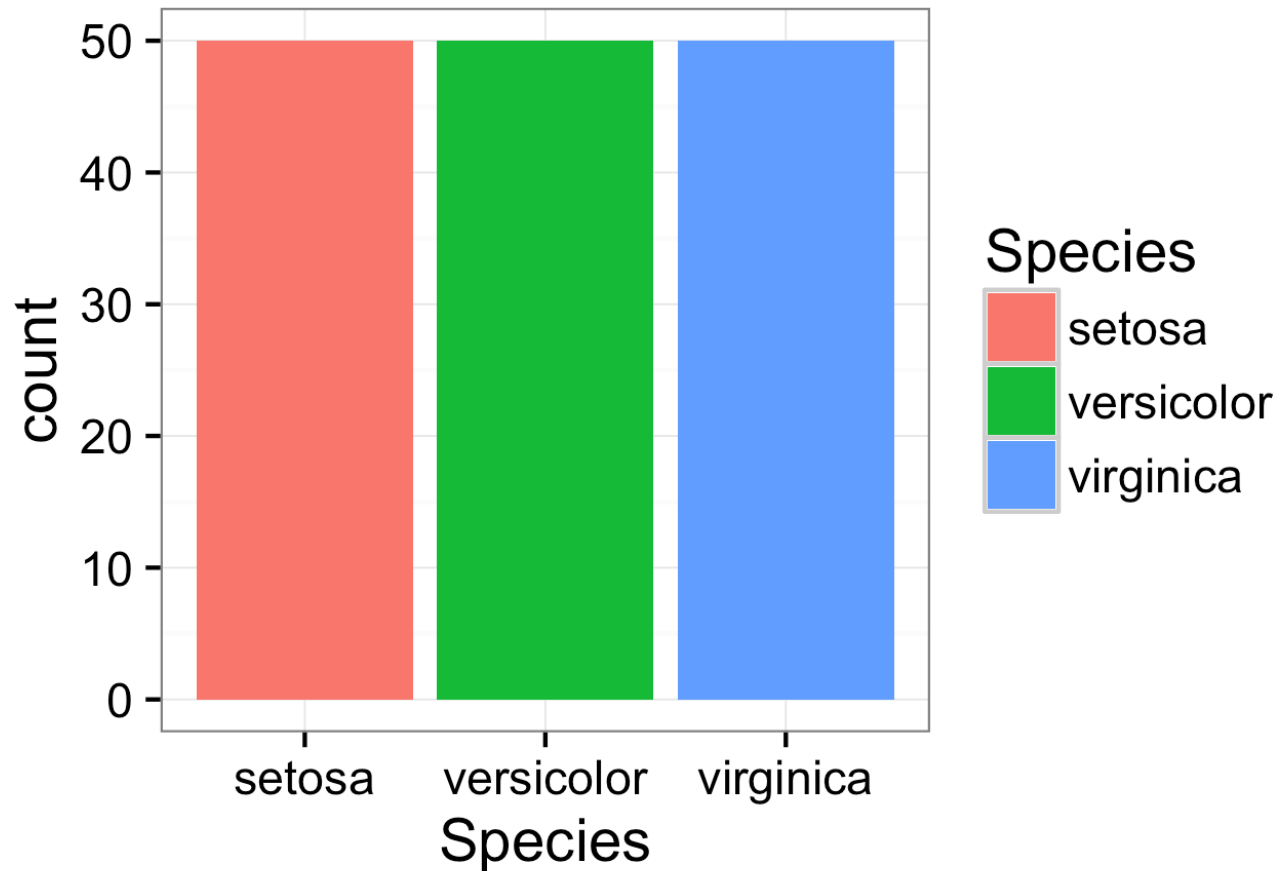
- `stat_ecdf`
Empirical Cumulative Density Function
- `stat_ellipse`
Plot data ellipses.
- `stat_function`
Superimpose a function.
- `stat_identity`
Identity statistic.
- `stat_qq` (`geom_qq`)
Calculation for quantile-quantile plot.
- `stat_summary_2d` (`stat_summary_2d`, `stat_summary_hex`)
Bin and summarise in 2d (rectangle & hexagons)
- `stat_unique`
Remove duplicates.



$$f(x) = x$$

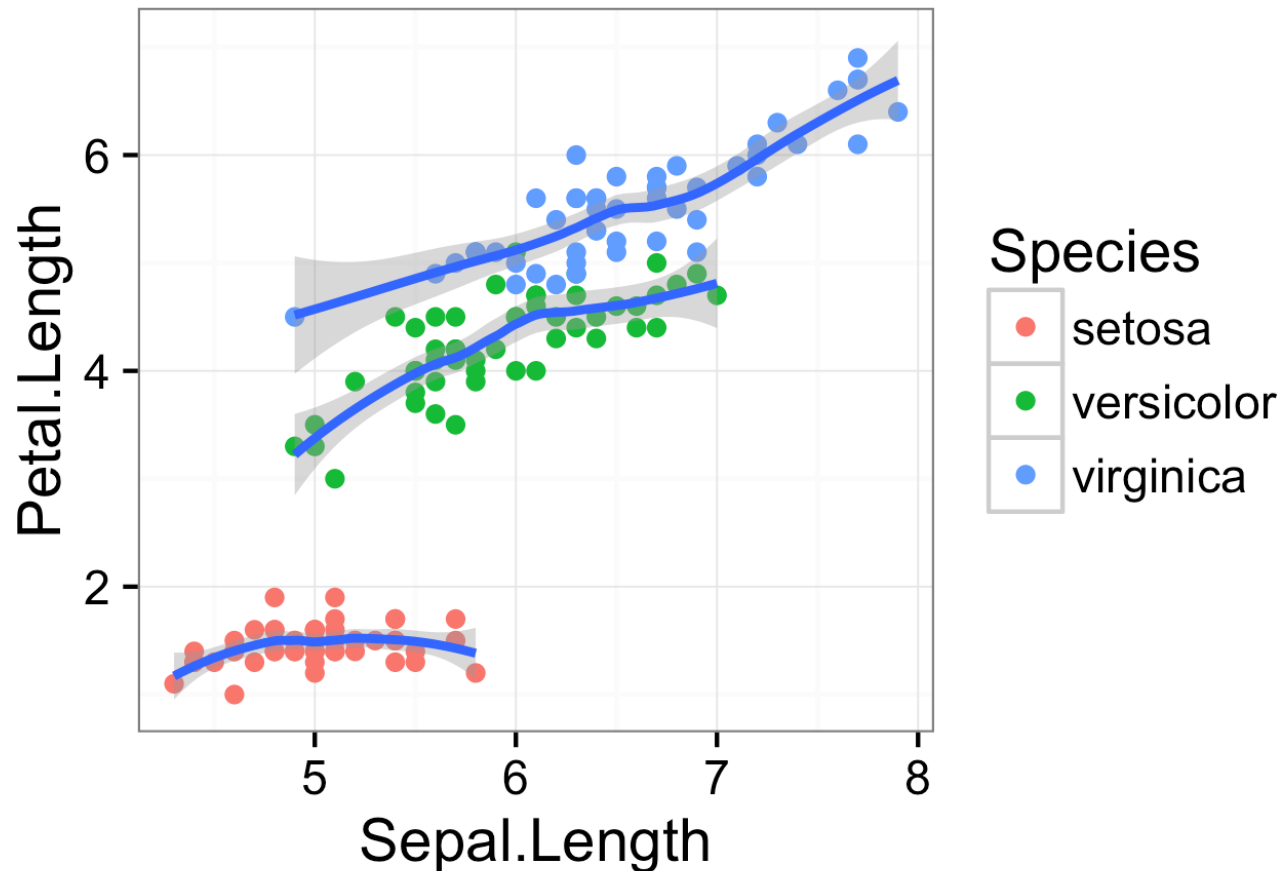
In most cases we just need to call the appropriate geom and it calls a stat

```
ggplot(iris, aes(x=Species, fill=Species)) +  
  geom_bar()
```



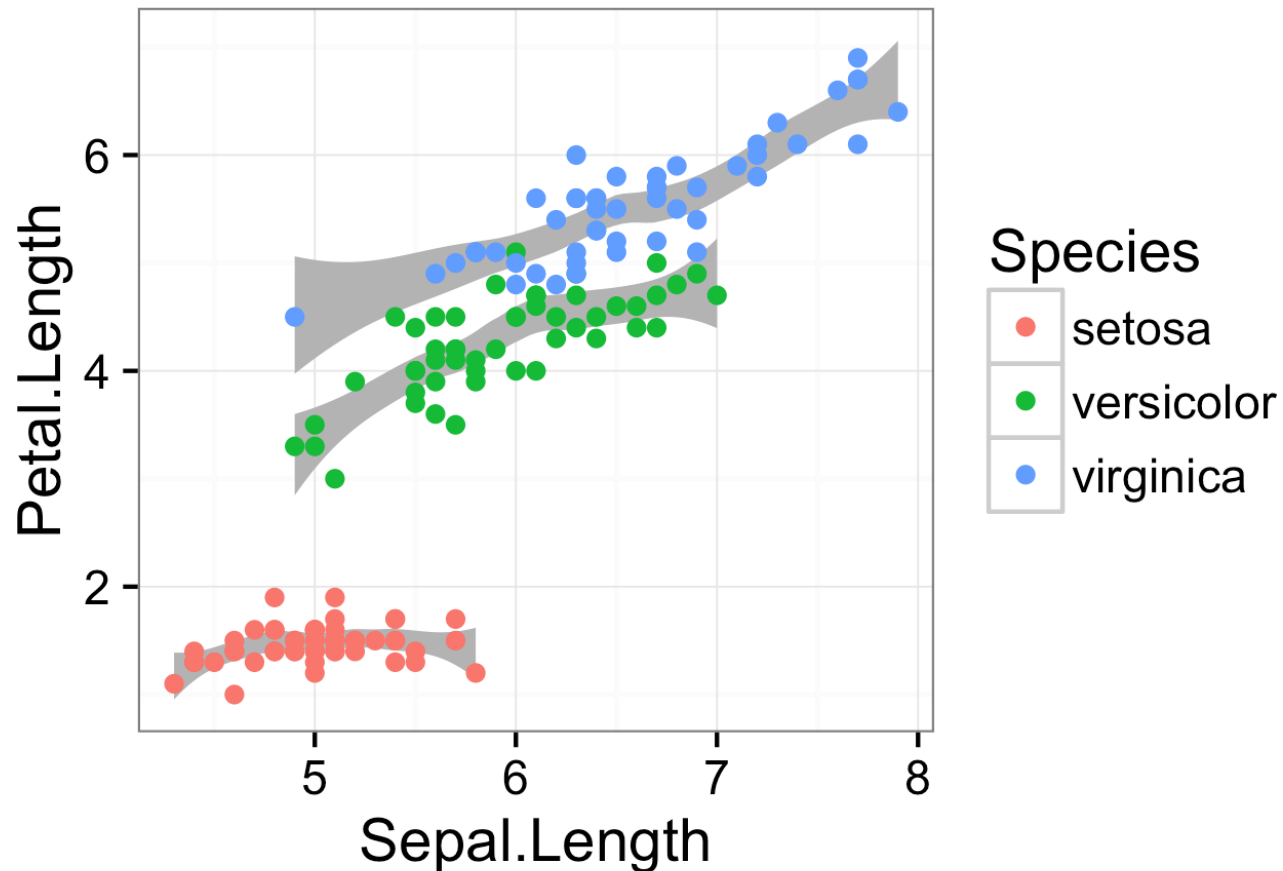
In most cases we just need to call the appropriate geom and it calls a stat

```
ggplot(iris, aes(x=Sepal.Length, y=Petal.Length)) +  
  geom_point(aes(color=Species)) +  
  geom_smooth(aes(group=Species))
```



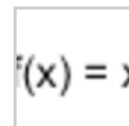
However, sometimes it can be helpful to call the stat directly

```
ggplot(iris, aes(x=Sepal.Length, y=Petal.Length)) +  
  stat_smooth(aes(group=Species), geom="ribbon", fill='gray70') +  
  geom_point(aes(color=Species))
```

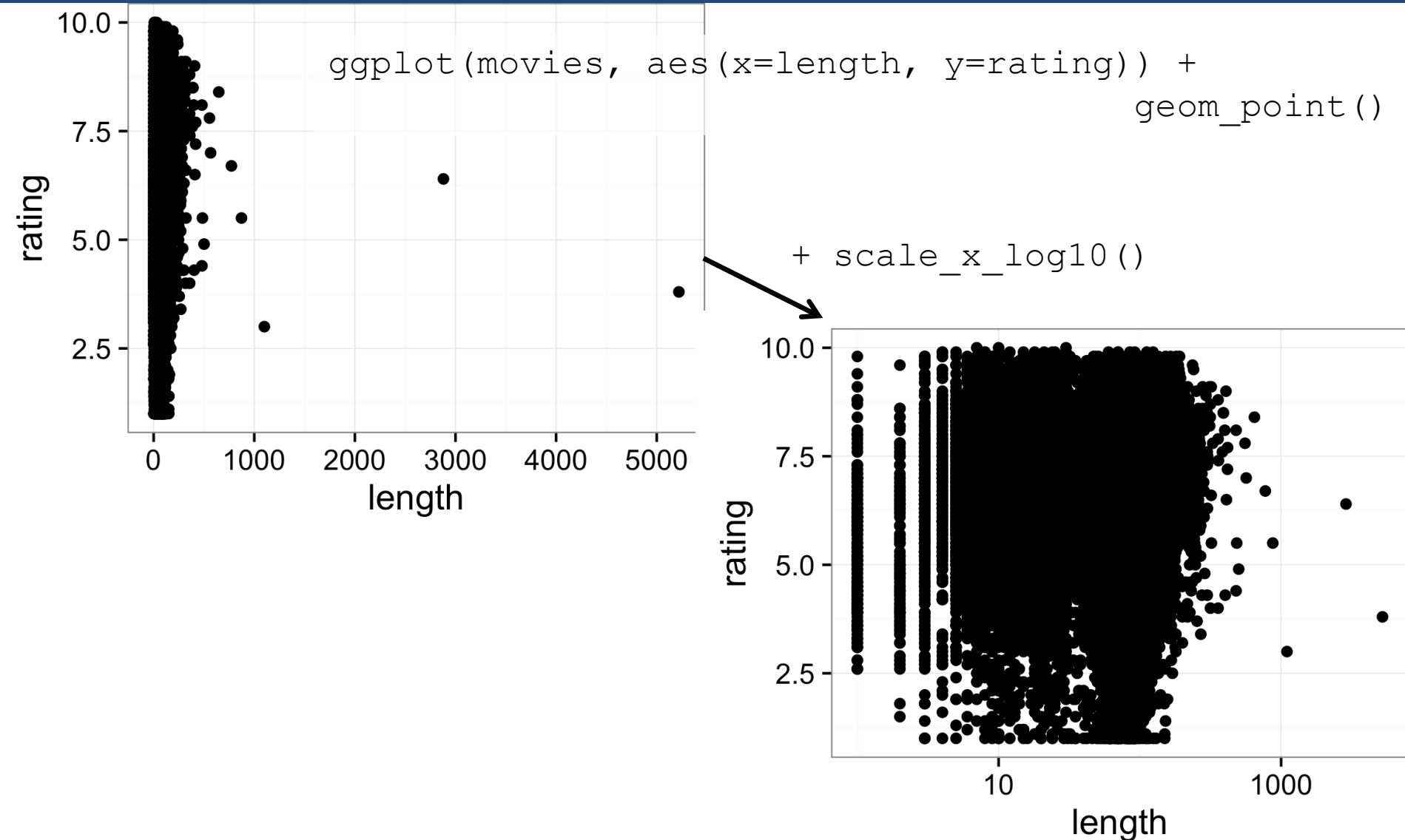


Scales define how to map data onto aesthetics

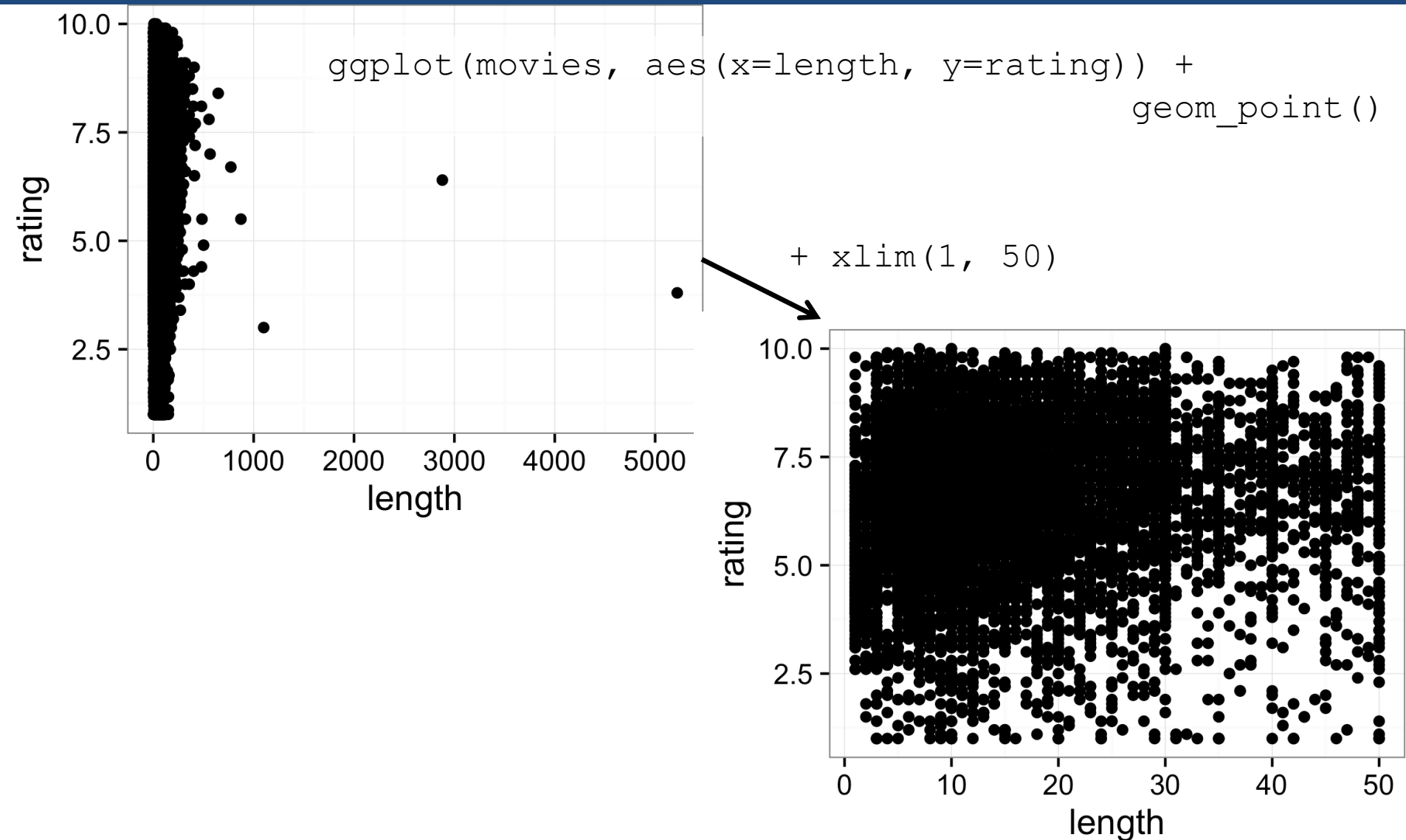
- `scale_colour_grey` (`scale_color_grey`, `scale_fill_grey`)
Sequential grey colour scale.
- `scale_colour_hue` (`scale_color_discrete`, `scale_color_hue`, `scale_colour_discrete`, `scale_fill_discrete`, `scale_fill_hue`)
Qualitative colour scale with evenly spaced hues.
- `scale_identity` (`scale_alpha_identity`, `scale_color_identity`, `scale_colour_identity`, `scale_fill_identity`, `scale_linetype_identity`, `scale_shape_identity`, `scale_size_identity`)
Use values without scaling.
- `scale_manual` (`scale_alpha_manual`, `scale_color_manual`, `scale_colour_manual`, `scale_fill_manual`, `scale_linetype_manual`, `scale_shape_manual`, `scale_size_manual`)
Create your own discrete scale.
- `scale_linetype` (`scale_linetype_continuous`, `scale_linetype_discrete`)
Scale for line patterns.
- `scale_shape` (`scale_shape_continuous`, `scale_shape_discrete`)
Scale for shapes, aka glyphs.
- `scale_size` (`scale_radius`, `scale_size_area`, `scale_size_continuous`, `scale_size_discrete`)
Scale for sizes.



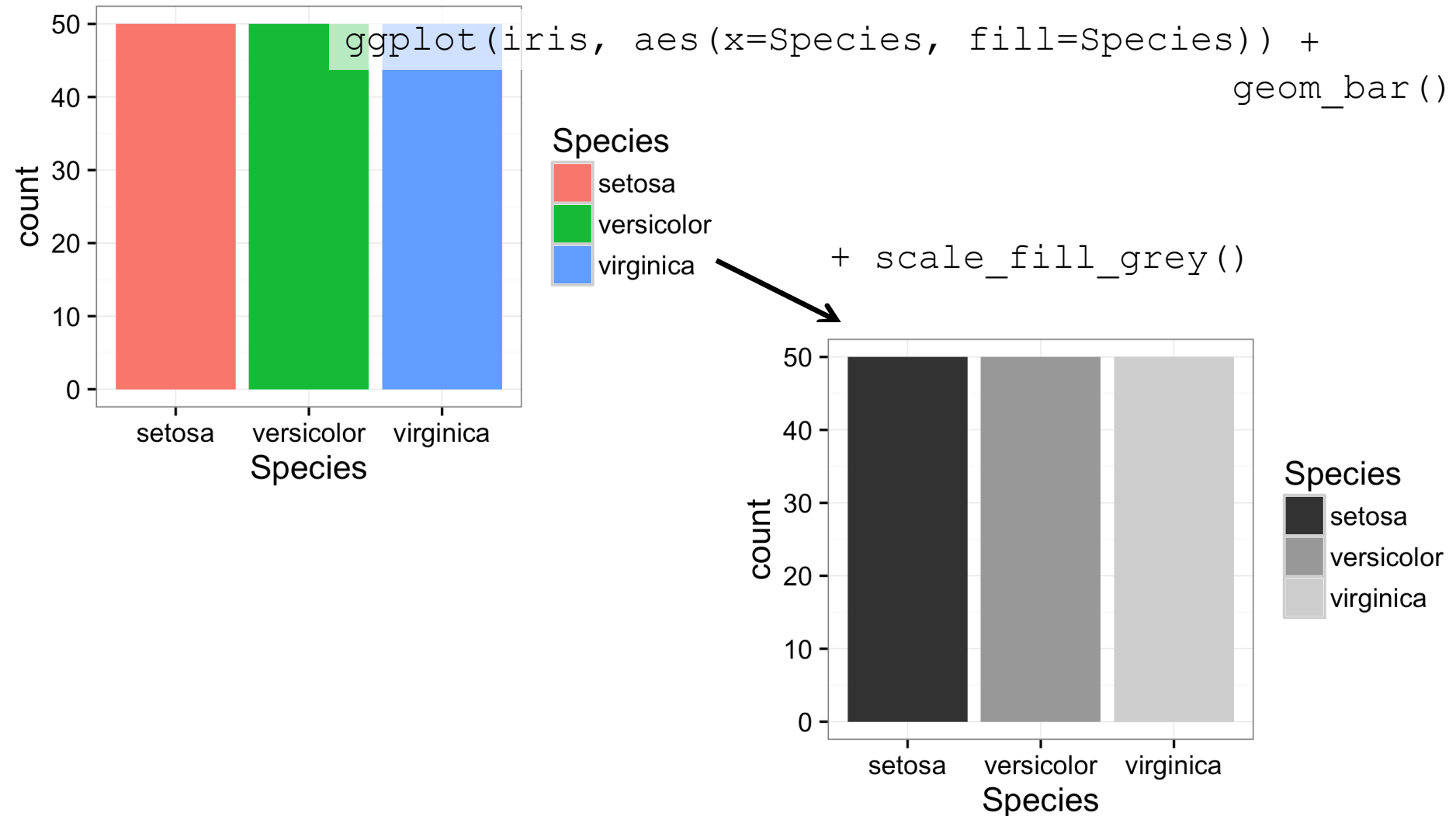
Example 1: Change scaling of x axis



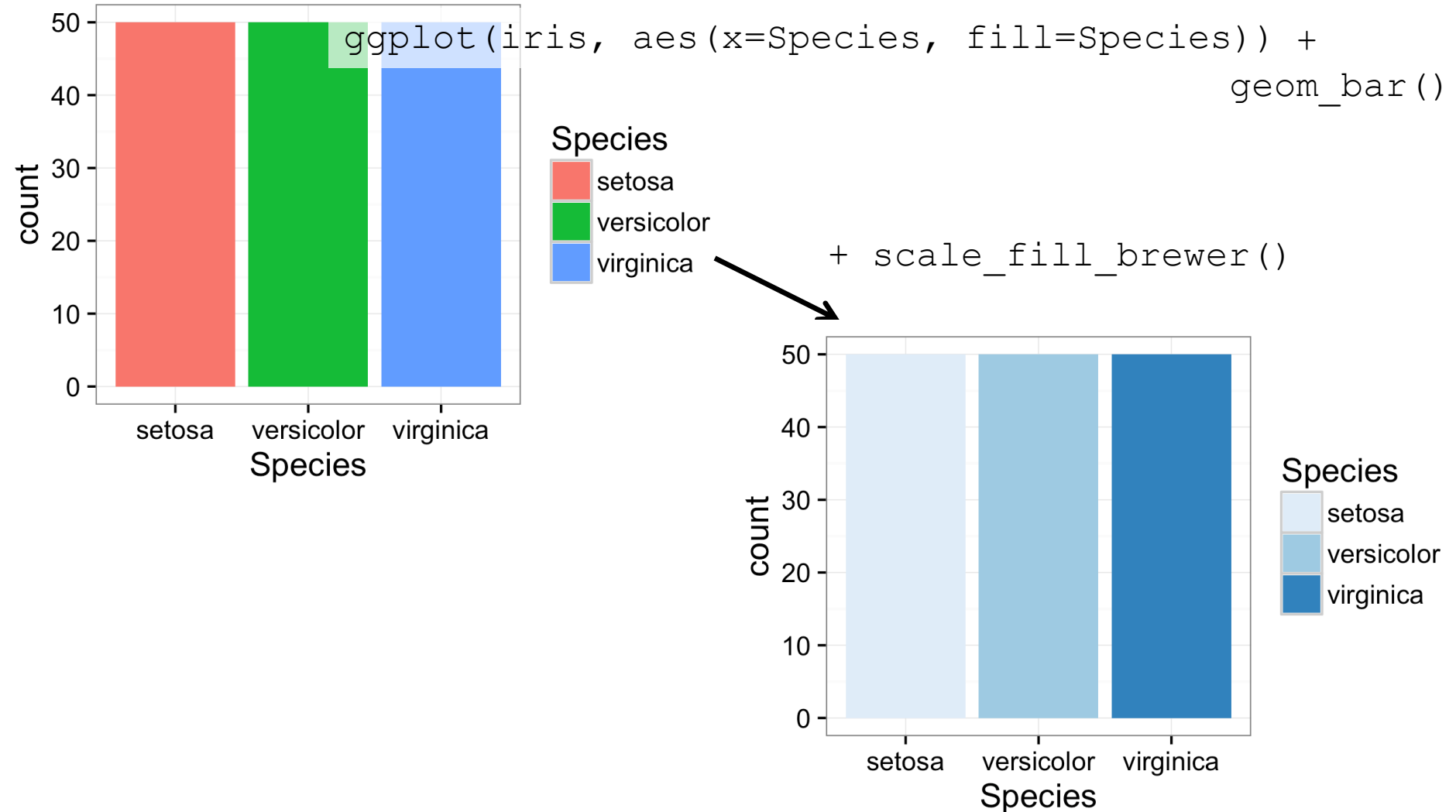
Example 1: Change scaling of x axis



Example 2: Change color scaling

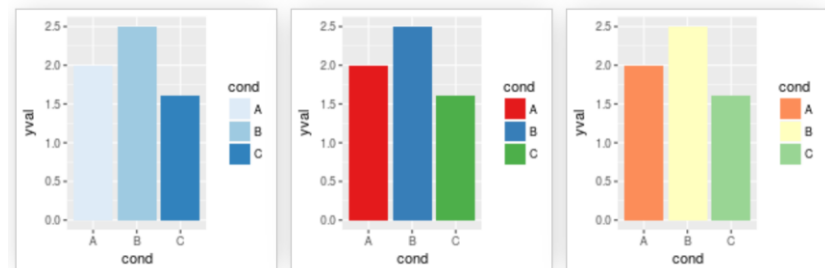
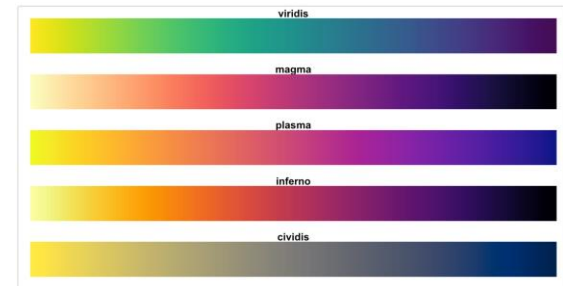


Example 2: Change color scaling



Some color scaling options in ggplot2

- `scale_color_gradient()`,
`scale_fill_gradient()`
- `scale_color_discrete()`,
`scale_fill_discrete()`
- `scale_color_brewer()`,
`scale_fill_brewer()`
- `scale_color_distiller()`,
`scale_fill_distiller()`
- `scale_color_colorblind()`,
`scale_fill_colorblind()`
- `scale_color_manual()`,
`scale_fill_manual()`



```
palette_pretty <- c("#0072B2", "#E69F00", "#009E24", "#FF0000", "#979797", "#5530AA")
palette_bgy <- c("#FFFFCC", "#A1DAB4", "#41B6C4", "#2C7FB8", "#253494")
palette_wine <- c("#bcb37b", "#9e934d", "#8f8023", "#790000", "#5b0b0b")
palette_cb <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442",
  "#0072B2", "#D55E00", "#CC79A7", "#999999")
```

Themes control non-data display

The `labs()` function lets you change the title, x- and y-axis labels, and color/legend labels:

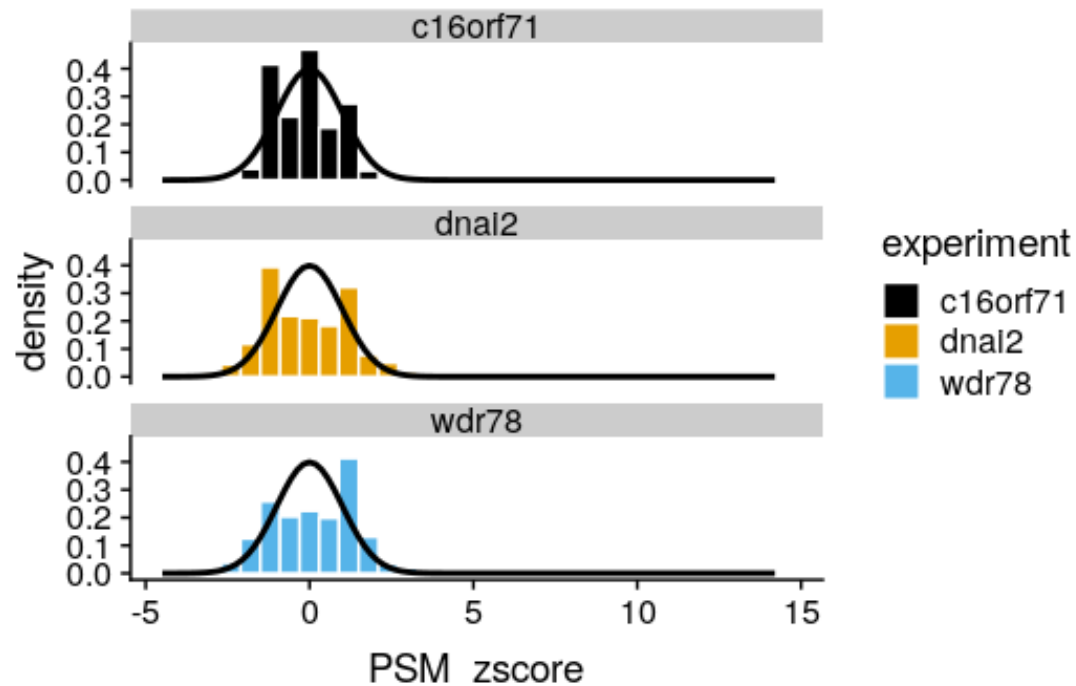
<https://ggplot2.tidyverse.org/reference/labs.html>

Adding `theme()` layers allow you to customize fonts, sizes, and positions of titles, labels, background, gridlines and legends:

<https://ggplot2.tidyverse.org/reference/theme.html>

Putting it all together, example 1

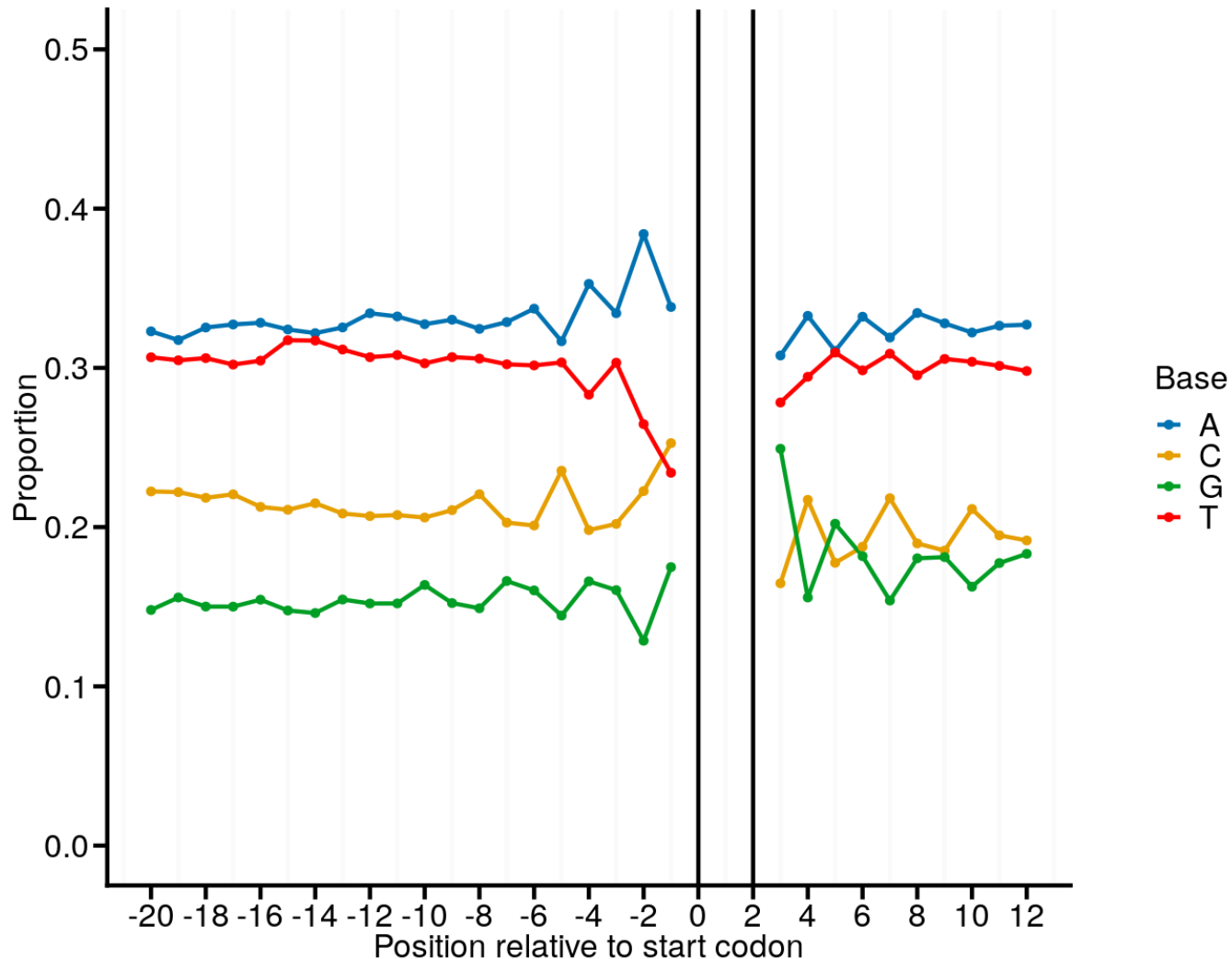
```
ggplot(apms_df, aes(x = PSM_zscore, fill = experiment)) +  
  geom_histogram(binwidth = 0.6, color = "white") +  
  facet_wrap(~experiment, ncol = 1) +  
  stat_function(aes(group = experiment), fun = dnorm, n = 101,  
               args = list(mean = 0, sd = 1), size = 1) +  
  scale_fill_colorblind()
```



Putting it all together, example 2

```
seq_plot <- seq_df %>%  
  ggplot(aes(x = pos, y = prop, group = Base, color = Base)) +  
    geom_line() +  
    geom_point(size = 0.5) +  
    scale_x_continuous(breaks = seq(-20, 13, 2),  
                      labels = seq(-20, 13, 2)) +  
    scale_color_manual(values = palette_pretty) +  
    geom_vline(xintercept = 0) +  
    geom_vline(xintercept = 2) +  
    background_grid(major = "only_minor", minor = "x") +  
    ylim(0, 0.5) +  
    ylab("Proportion") +  
    xlab("Position relative to start codon")
```

Putting it all together, example 2



Saving plots

With code:

```
seq_plot %>% ggsave("relative_start_plot.png", ., device =  
"png", width = 4.5, height = 3.5, units = "in")
```

```
seq_plot %>% ggsave("relative_start_plot.pdf", ., device =  
"pdf", width = 4.5, height = 3.5, units = "in")
```

In RStudio; to make
plots appear in the
bottom-right window:

- Tools > Global Options
> R Markdown >
Uncheck “Show
output inline”

