Introduction to R

Part 2 – the Tidyverse

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

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

cran.r-project.org

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation

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.

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      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
> |
```

Tidy data

“Tidy datasets are all alike but every messy dataset is messy in its own way” — Hadley Wickham

Tidy data

Three rules:

1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table

Example: Contingency table

	survived	**died**
drug	15	3
placebo	4	12

not tidy

Example: Contingency table

	survived	**died**
drug	15	3
placebo	4	12

not tidy

tidy

treatment	**outcome**	**count**
drug	survived	15
drug	died	3
placebo	survived	4
placebo	died	12

Example: Contingency table

	survived	died
drug	15	3
placebo	4	12

not tidy

tidy

patient	treatment	outcome
1	drug	survived
2	drug	died
3	drug	survived
4	placebo	died
	⋮	

`tidyr` library provides functions for transforming tables

	survived	died
drug	15	3
placebo	4	12

`pivot_longer()`

`pivot_wider()`

patient	treatment	outcome
1	drug	survived
2	drug	died
3	drug	survived
4	placebo	died
	⋮	

`dplyr`: a package for data manipulation

Working with tidy data in R: tidyverse

Fundamental actions on data tables:

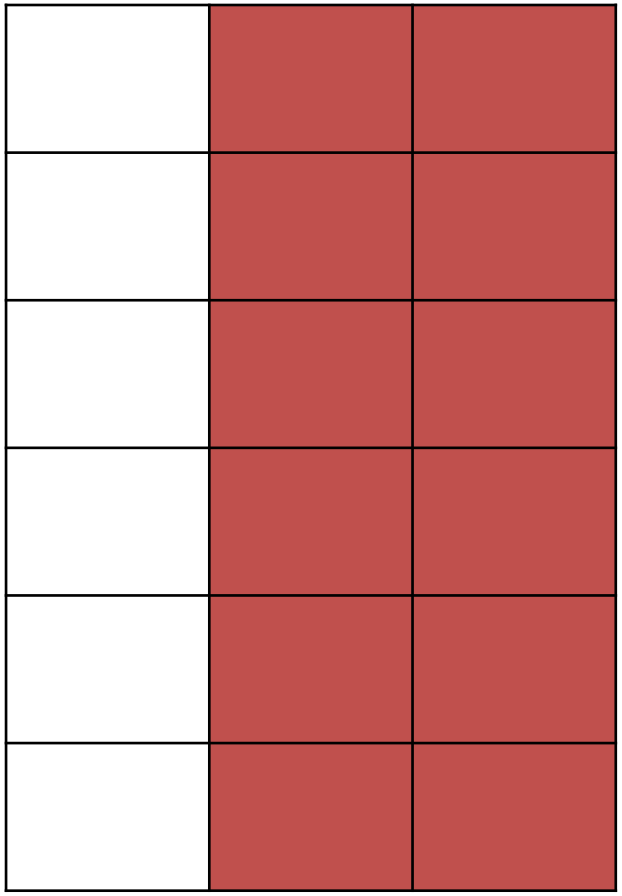
- make new columns — `mutate()`
- combine tables, adding columns — `left_join()`
- combine tables, adding rows — `bind_rows()`
- choose rows — `filter()`
- choose columns — `select()`
- arrange rows — `arrange()`
- calculate summary statistics — `summarize()`
- work on groups of data — `group_by()`

Working with tidy data in R: tidyverse

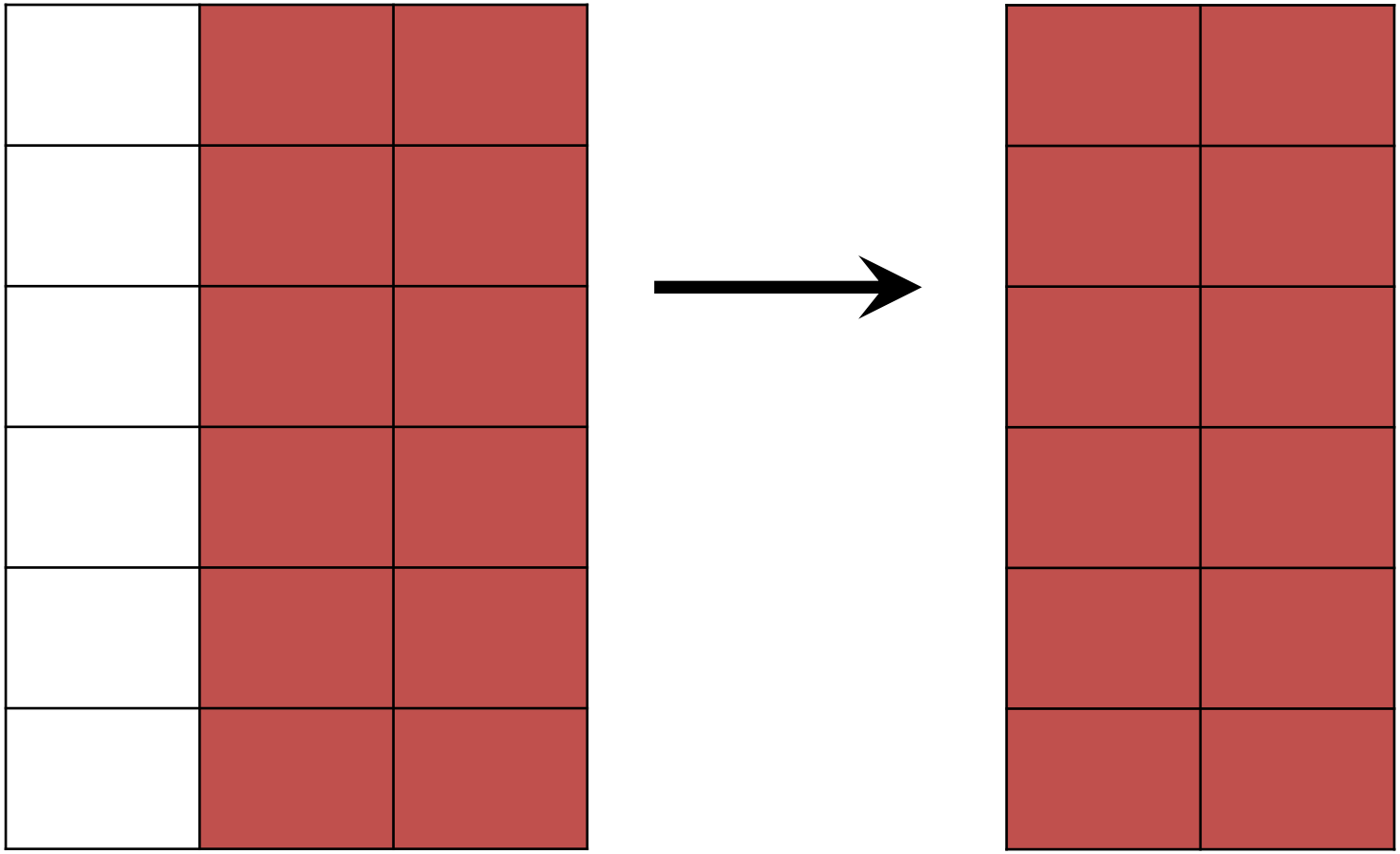
Fundamental actions on data tables:

- ~~make new columns — `mutate()`~~
- ~~combine tables, adding columns — `left_join()`~~
- ~~combine tables, adding rows — `bind_rows()`~~
- choose rows — `filter()`
- choose columns — `select()`
- ~~arrange rows — `arrange()`~~
- calculate summary statistics — `summarize()`
- work on groups of data — `group_by()`

`select()`: pick columns



`select()`: pick columns



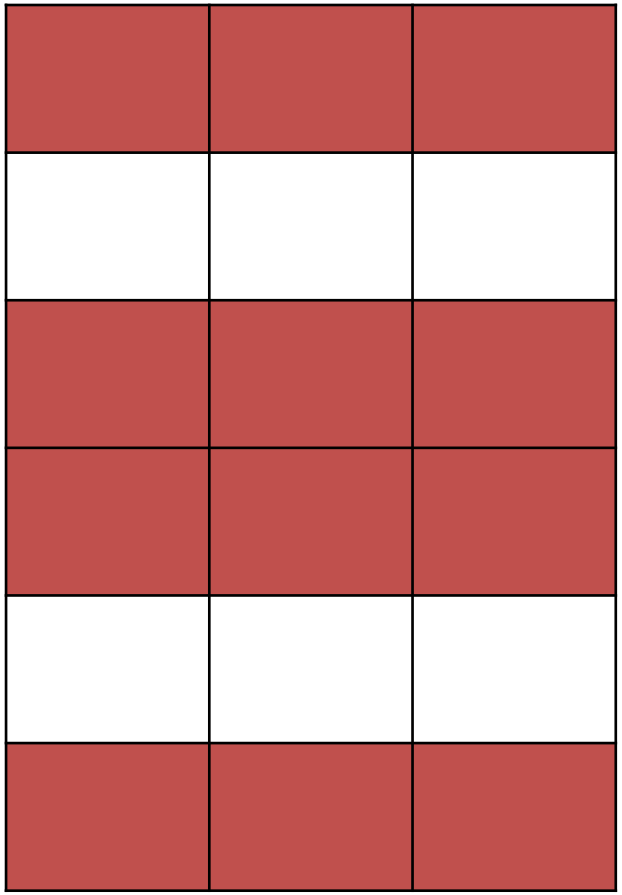
Choose the two columns Species and Sepal.Width

```
> select(iris, Species, Sepal.Width)
```

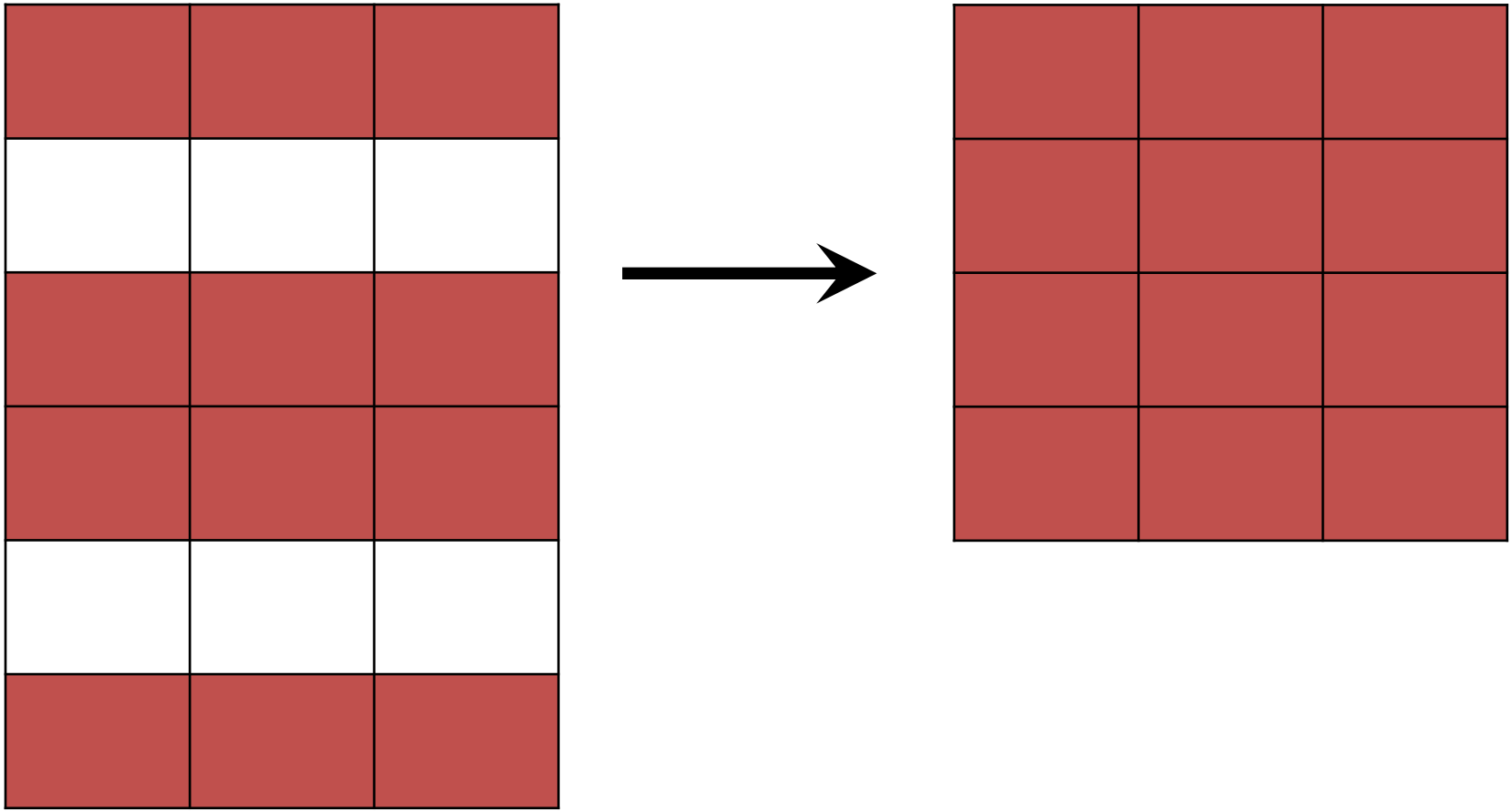
Choose the two columns Species and Sepal.Width

```
> select(iris, Species, Sepal.Width)
       Species Sepal.Width
1       setosa         3.5
2       setosa         3.0
3       setosa         3.2
4       setosa         3.1
5       setosa         3.6
6       setosa         3.9
7       setosa         3.4
8       setosa         3.4
9       setosa         2.9
10      setosa         3.1
11      setosa         3.7
12      setosa         3.4
13      setosa         3.0
14      setosa         3.0
```

`filter()`: pick rows



`filter()`: pick rows



Choose rows with Sepal.Width > 4

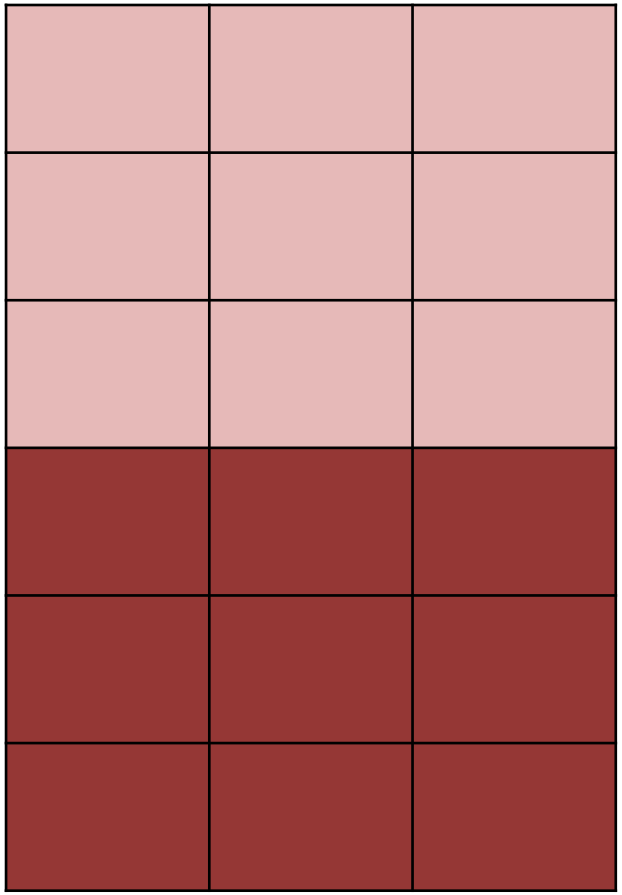
```
> filter(iris, Sepal.Width > 4)
```

Choose rows with Sepal.Width > 4

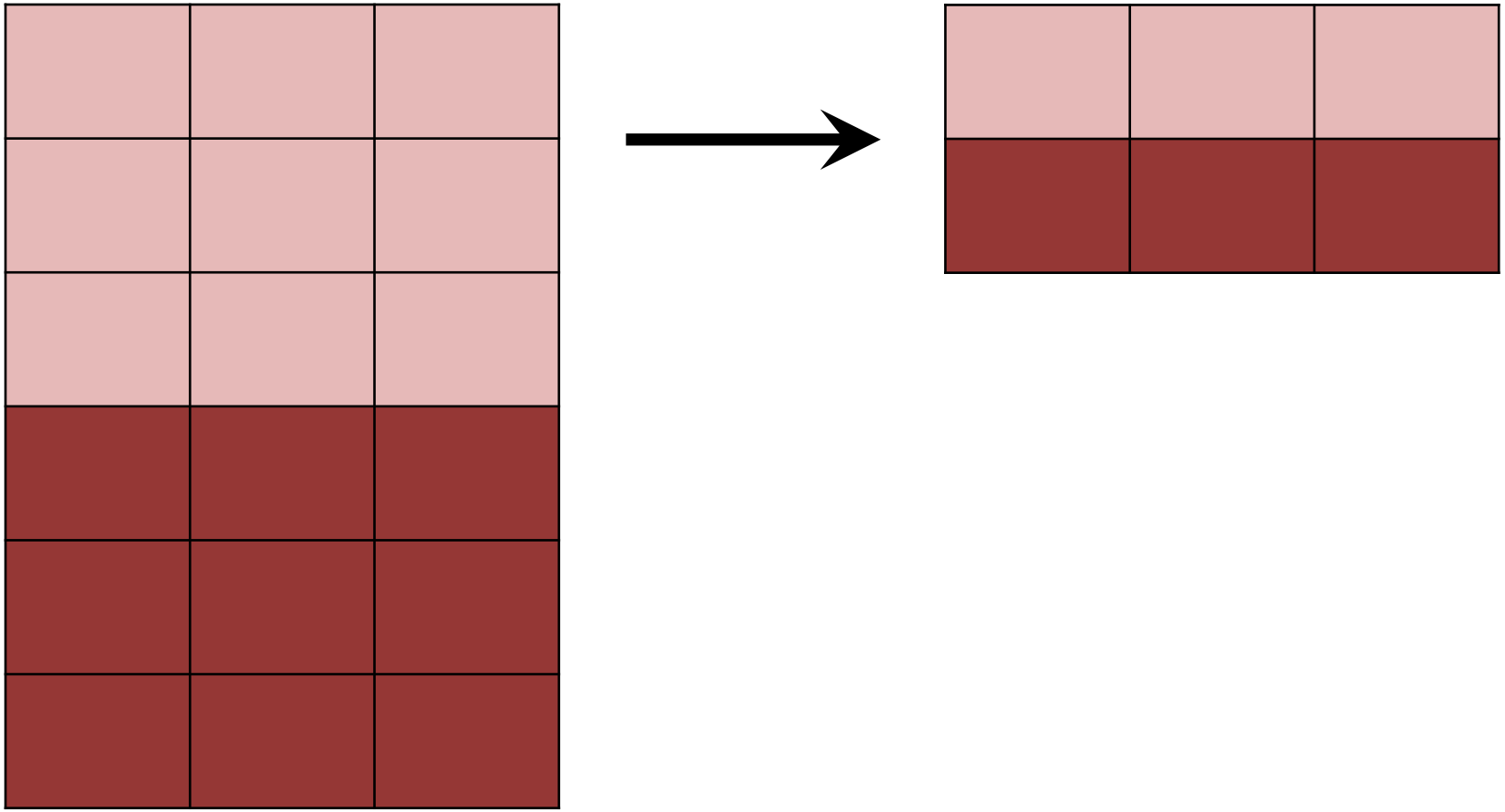
```
> filter(iris, Sepal.Width > 4)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.7         4.4          1.5         0.4  setosa
2          5.2         4.1          1.5         0.1  setosa
3          5.5         4.2          1.4         0.2  setosa
```

`select()`: pick columns

`summarize()`: collapse multiple rows



`summarize()`: collapse multiple rows



Calculate mean and standard deviation of Sepal.Length

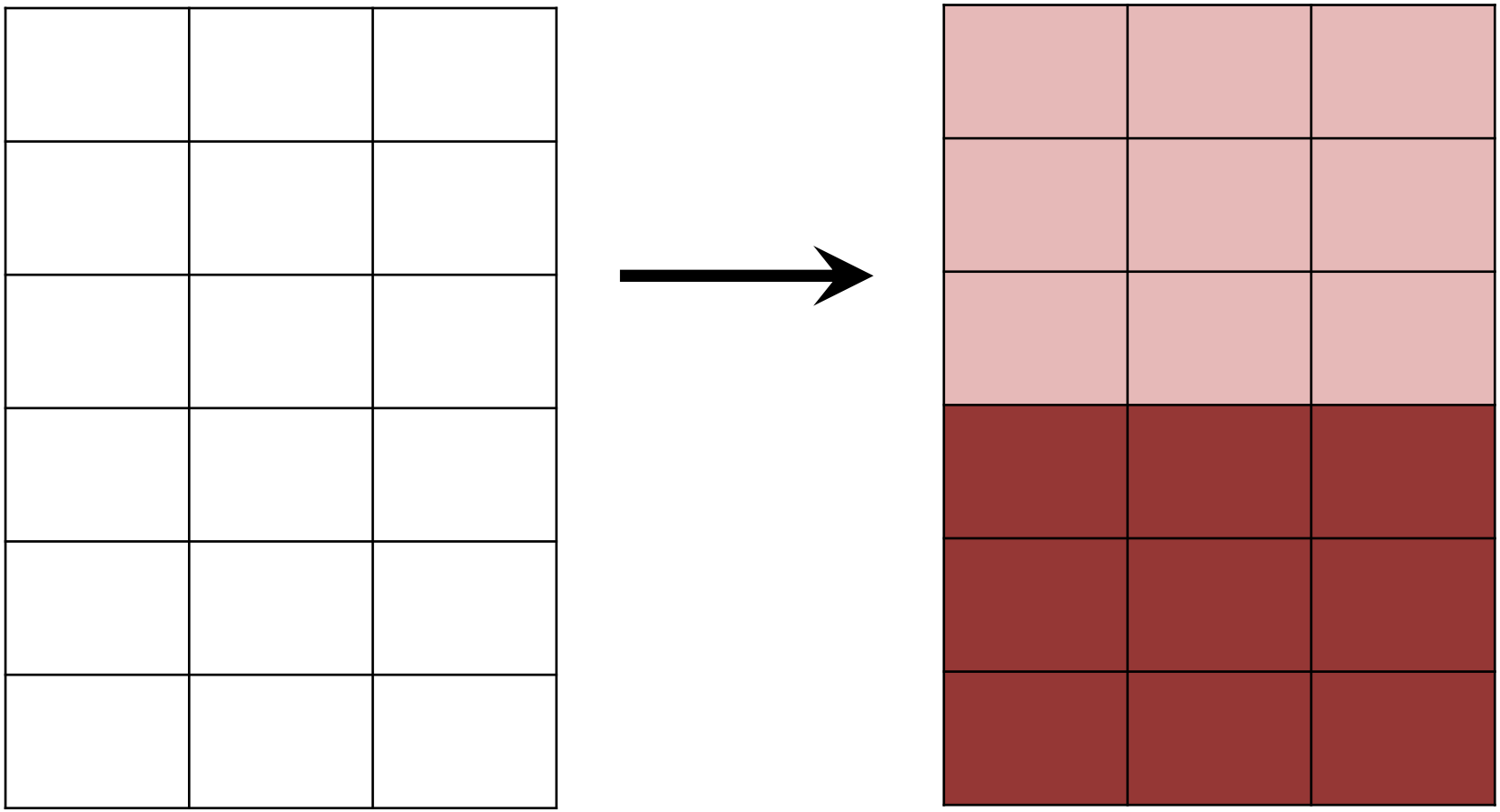
```
> summarize(iris, mean_sepal_length = mean(Sepal.Length),
                  sd_sepal_length   = sd(Sepal.Length))
```

Calculate mean and standard deviation of Sepal.Length

```
> summarize(iris, mean_sepal_length = mean(Sepal.Length),
                  sd_sepal_length   = sd(Sepal.Length))
  mean_sepal_length sd_sepal_length
1          5.843333       0.8280661
```

`group_by()`: set up groupings

`group_by()`: set up groupings



Calculate mean and standard deviation of Sepal.Length, grouped by Species

```
> summarize(group_by(iris, Species),
                  mean_sepal_length = mean(Sepal.Length),
                  sd_sepal_length   = sd(Sepal.Length))
```

Calculate mean and standard deviation of Sepal.Length, grouped by Species

```
> iris %>%
> group_by(Species) %>%
> summarize(mean_sepal_length = mean(Sepal.Length),
            sd_sepal_length   = sd(Sepal.Length))
Source: local data frame [3 x 3]

     Species mean_sepal_length sd_sepal_length
1     setosa             5.006       0.3524897
2 versicolor             5.936       0.5161711
3  virginica             6.588       0.6358796
```

Piping multiple functions togehter

Call the target data set in first position

The data is implicitly passed and processed through each subsequent function

```
> iris %>%
>  group_by(., Species) %>%
>  summarize(., mean_length = mean(Sepal.Length),
                sd_length   = sd(Sepal.Length))
```

Piping multiple functions togehter

This implicit operate/pass forward behavior is facilitated by the pipe

```
> iris %>%
>  group_by(Species) %>%
>  summarize(mean_length = mean(Sepal.Length),
            sd_length   = sd(Sepal.Length))
```

`ggplot2`: a package for data manipulation

ggplot2: A grammar of graphics

Traditional plotting: You **are** a painter

- Manually place individual graphical elements

ggplot2: You **employ** a painter

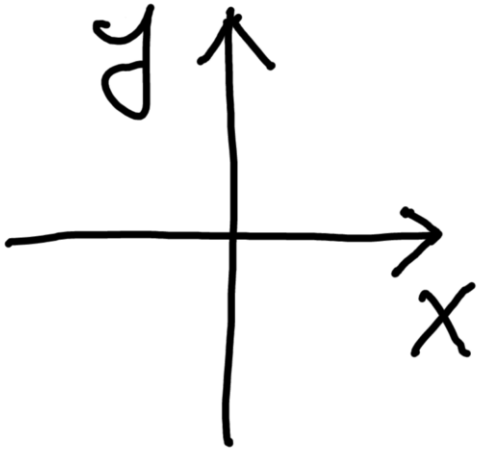
- Describe conceptually how data should be visualized

Most confusing key concept: aesthetic mapping

Maps data values to visual elements of the plot

A few examples of aesthetics

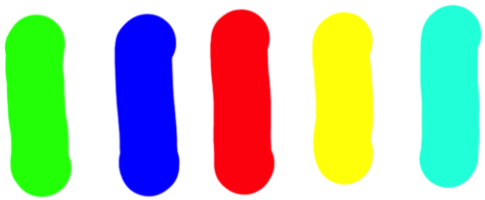
position



shape

size

color



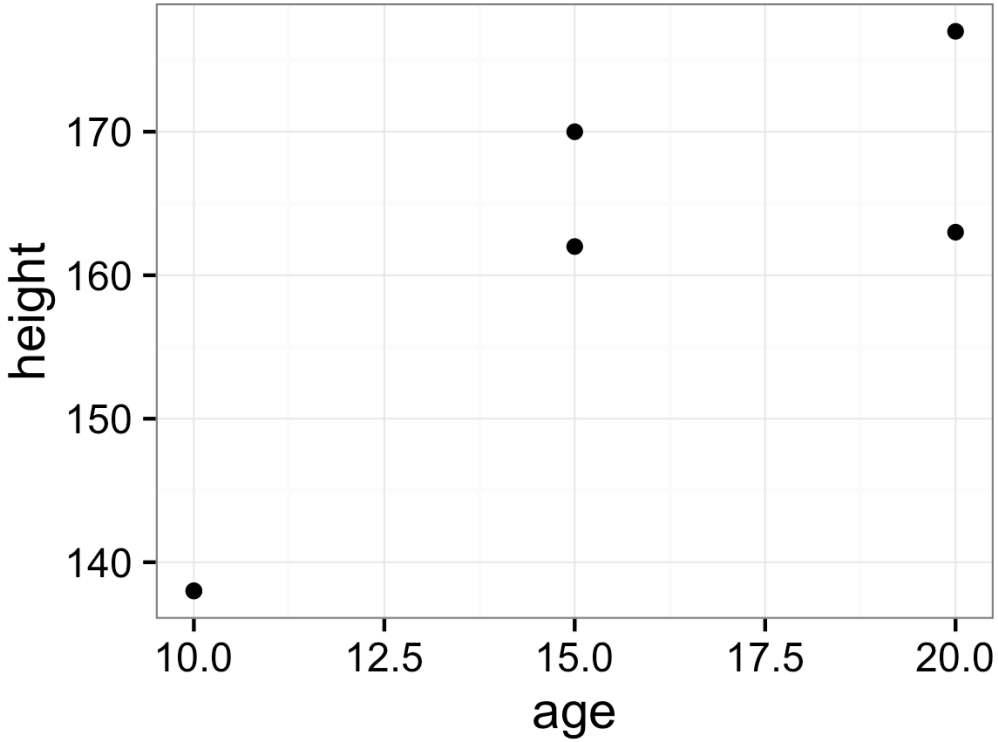
Let's go over a simple example: mean height and weight of boys/girls ages 10-20

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()
```

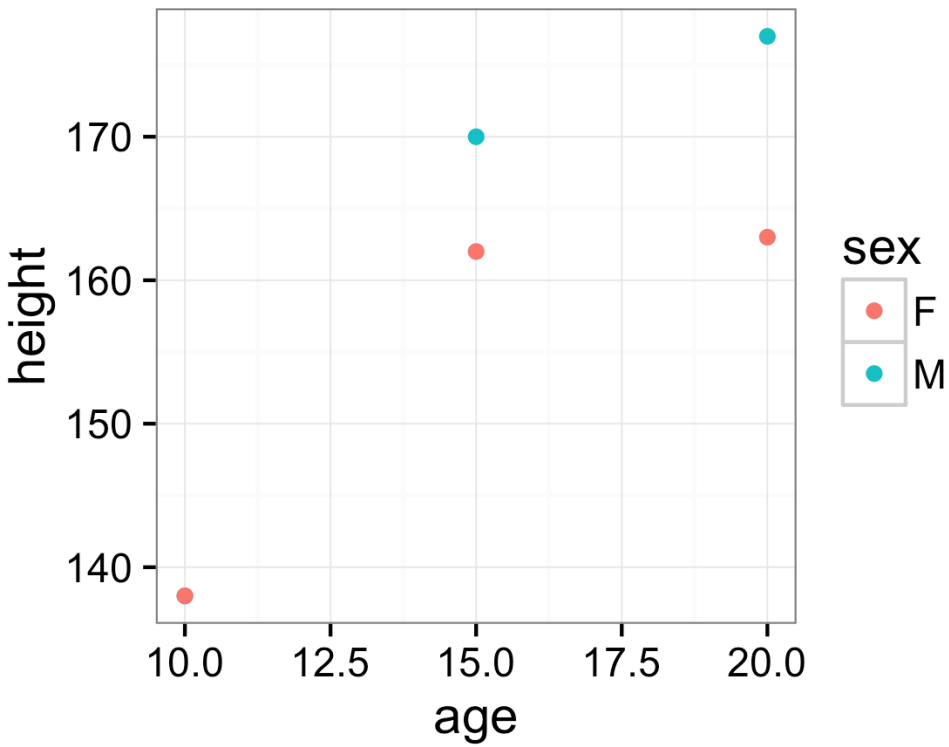


Let's color the points by sex

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

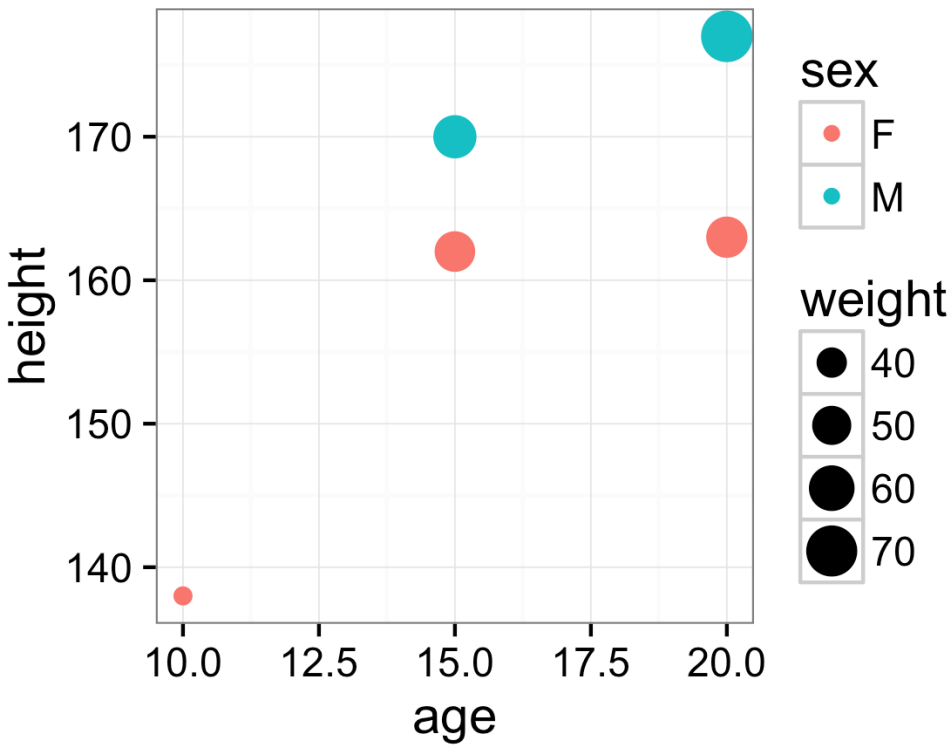
★ NOTE: "color" aesthetic is for coloring points & lines;

"fill" aesthetic is for coloring bars & distributions



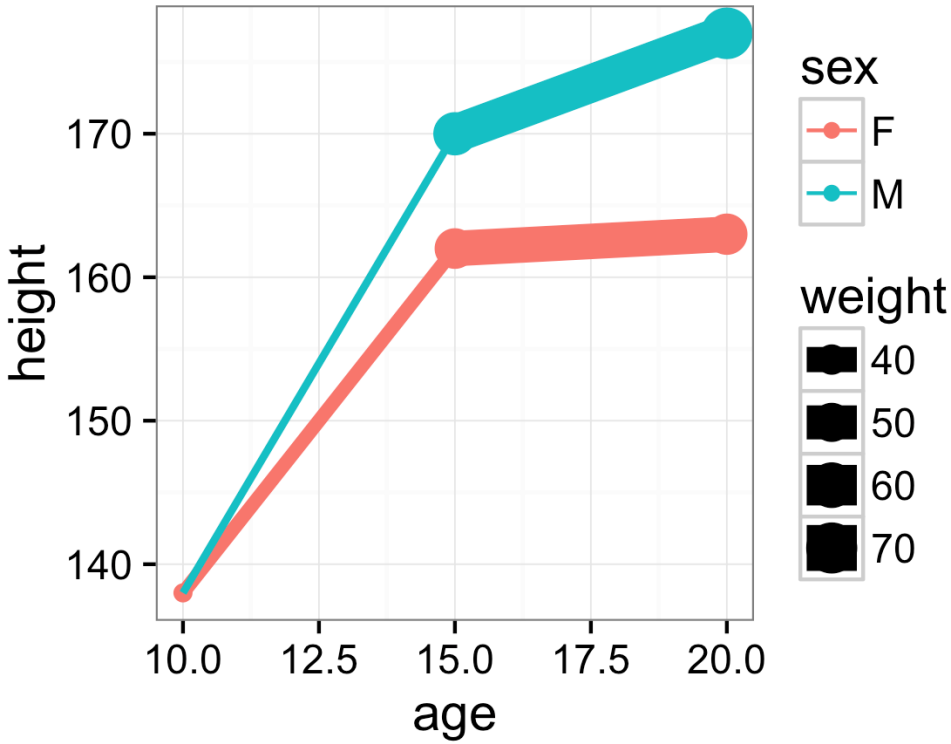
And change point size by weight

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



And connect the points with lines

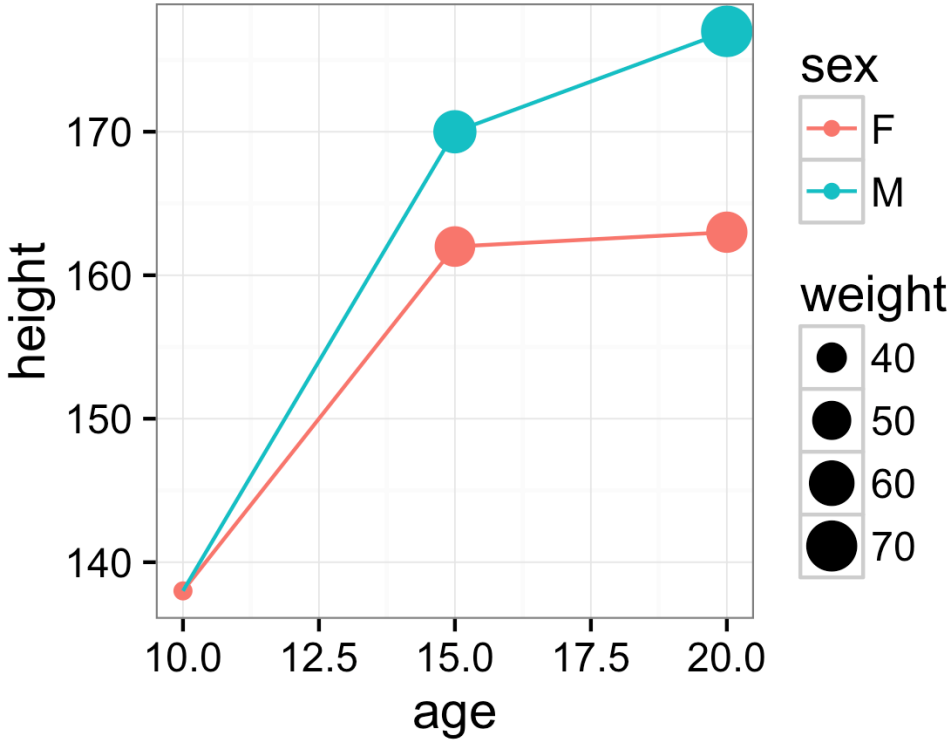
```
ggplot(data, aes(x=age, y=height,
        color=sex, size=weight)) +
                    geom_point() + geom_line()
```



Oops!

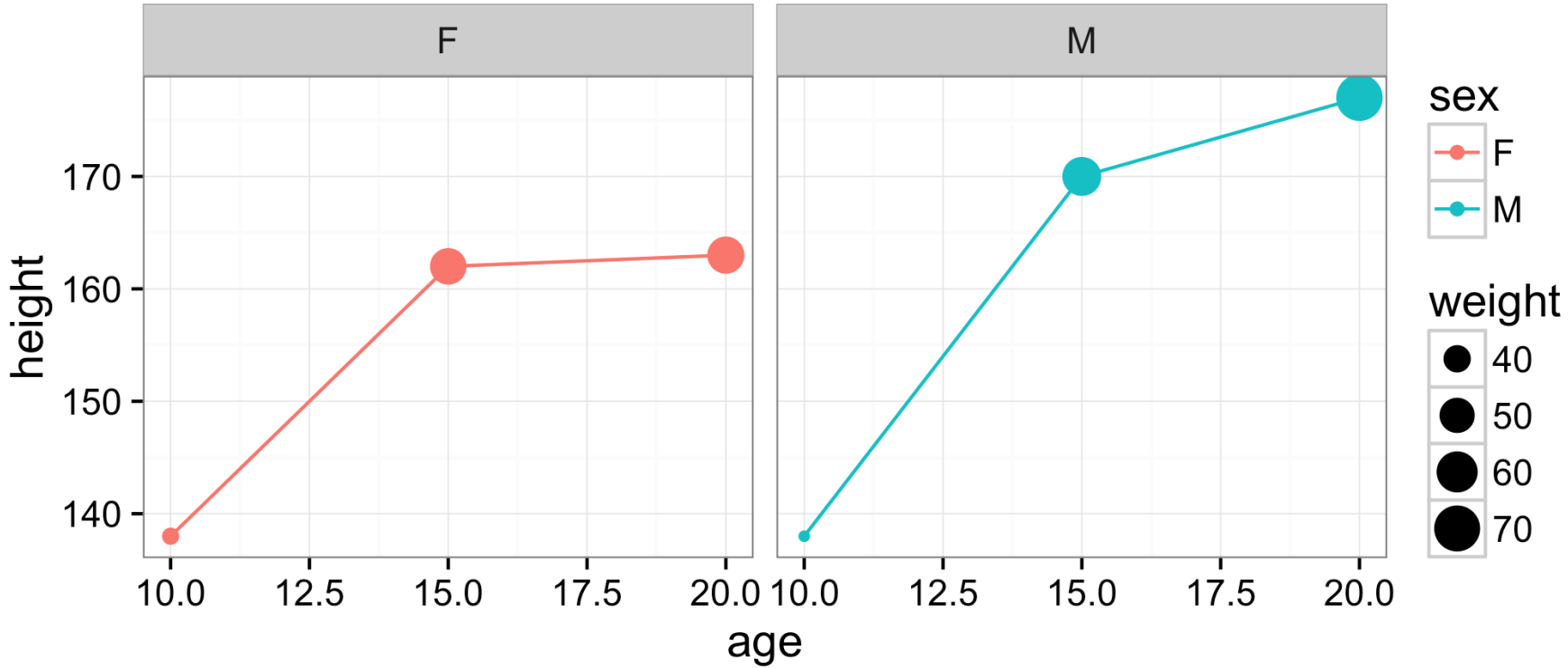
The weight-to-size mapping should only be applied to points

```
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)

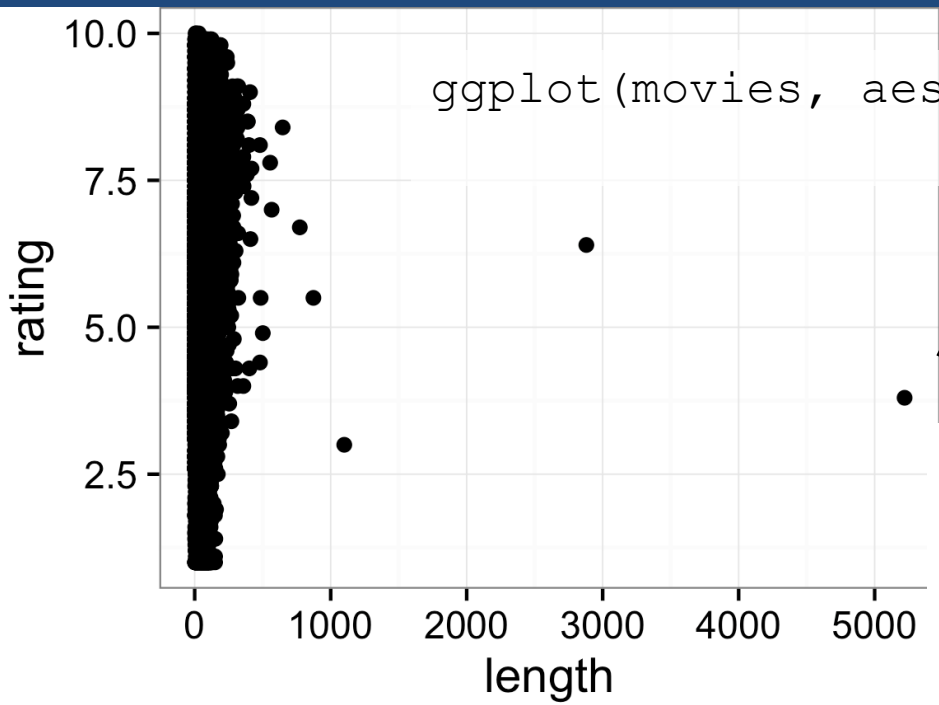
```
ggplot(data, aes(x=age, y=height,
 color=sex)) + geom_point(aes(size=weight)) +
                geom_line() + facet_wrap(~sex)
```



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

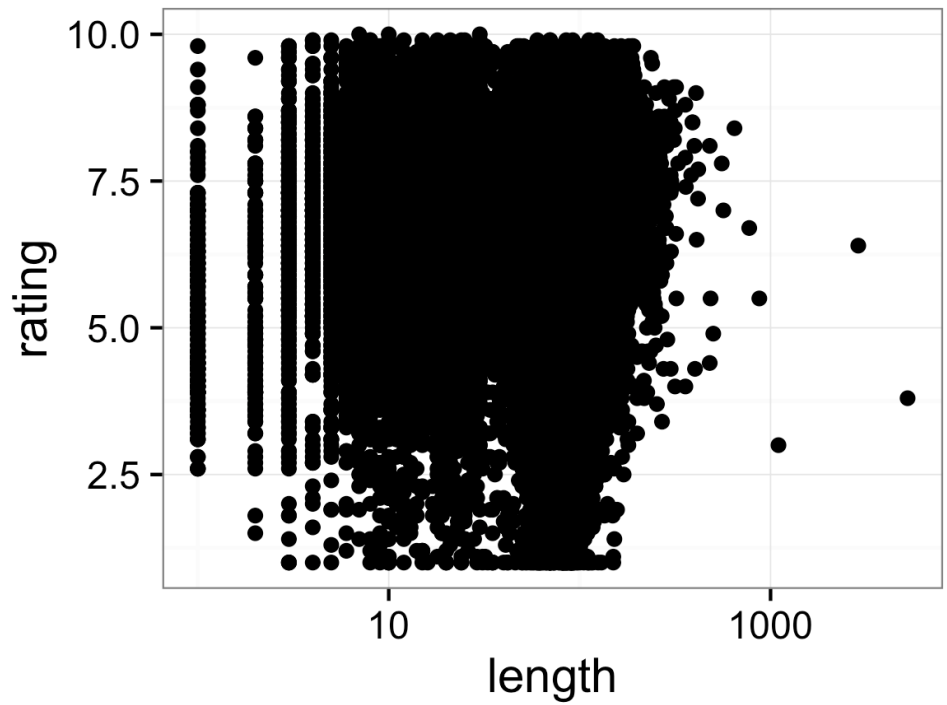
https://ggplot2.tidyverse.org/reference/

Example 1: Change scaling of x axis

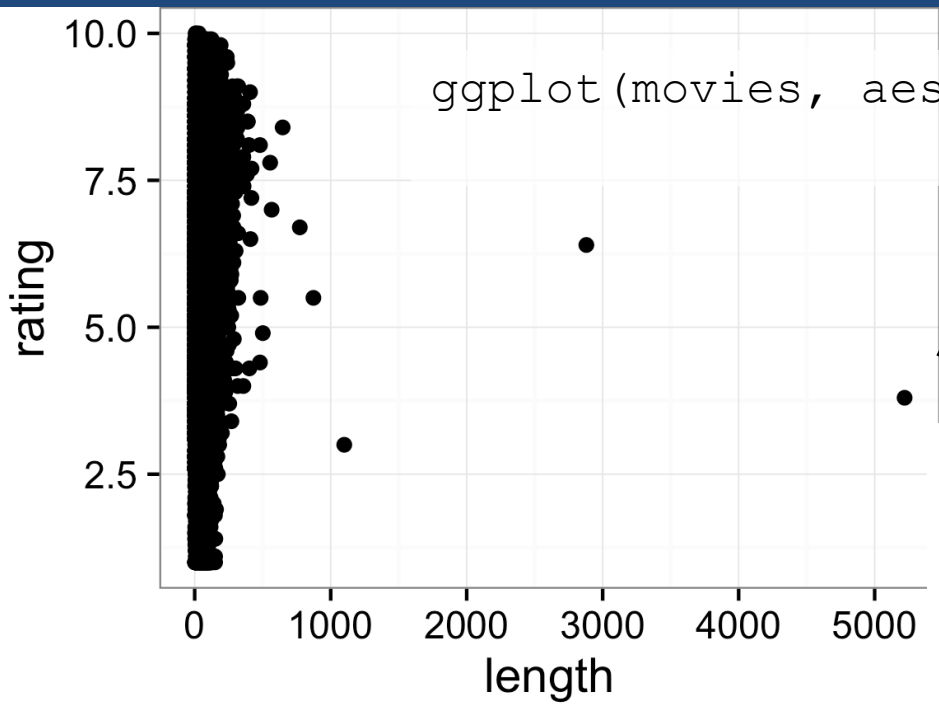



```
ggplot(movies, aes(x=length, y=rating)) +
                                    geom_point()
```

```
+ scale_x_log10()
```

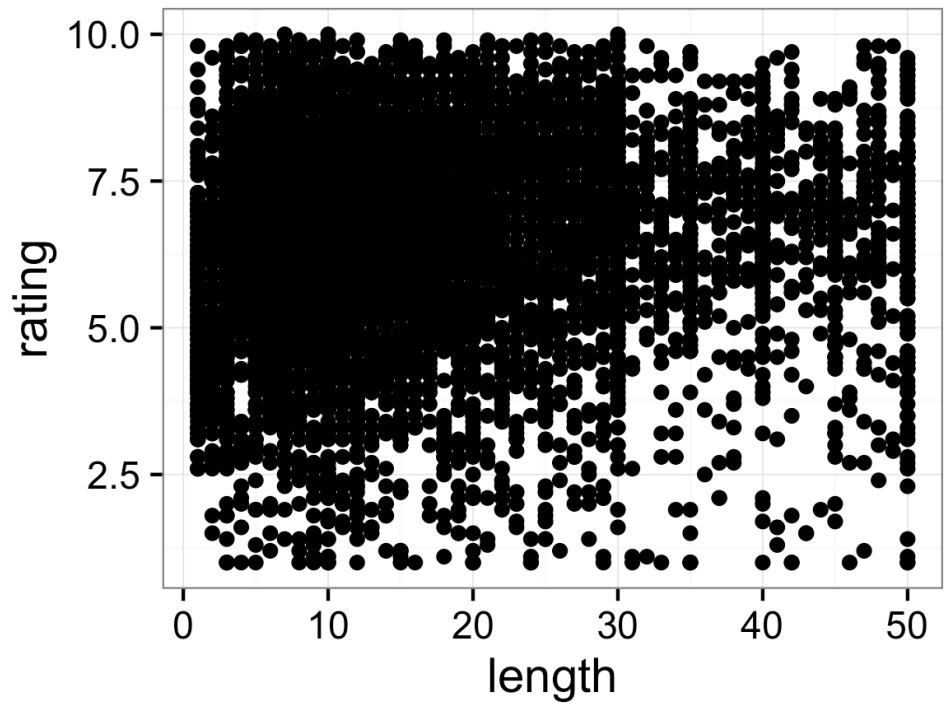


Example 1: Change scaling of x axis

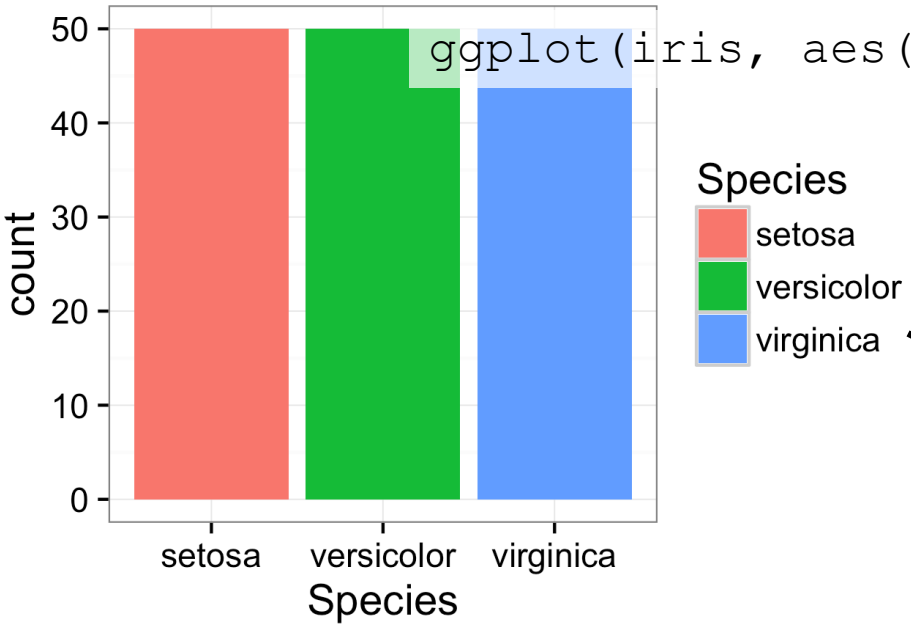



```
ggplot(movies, aes(x=length, y=rating)) +
                                    geom_point()
```

```
+ xlim(1, 50)
```

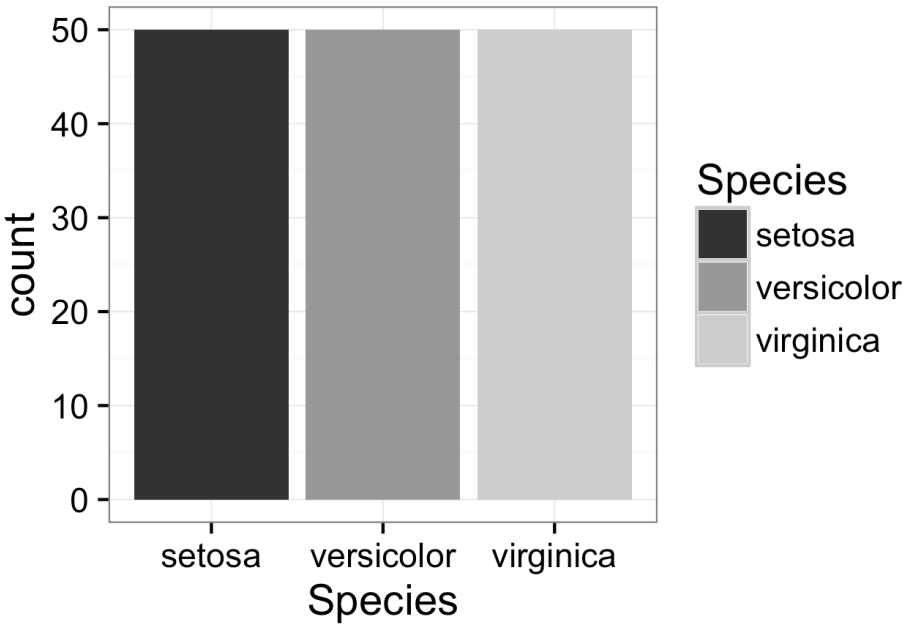


Example 2: Change color scaling

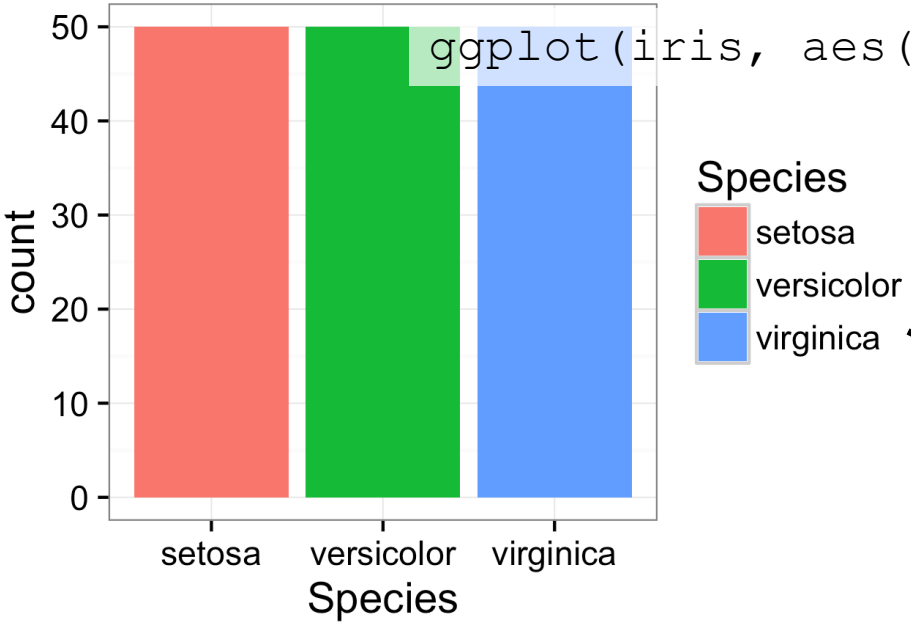



```
ggplot(iris, aes(x=Species, fill=Species)) +
                                    geom_bar()
```

```
+ scale_fill_grey()
```

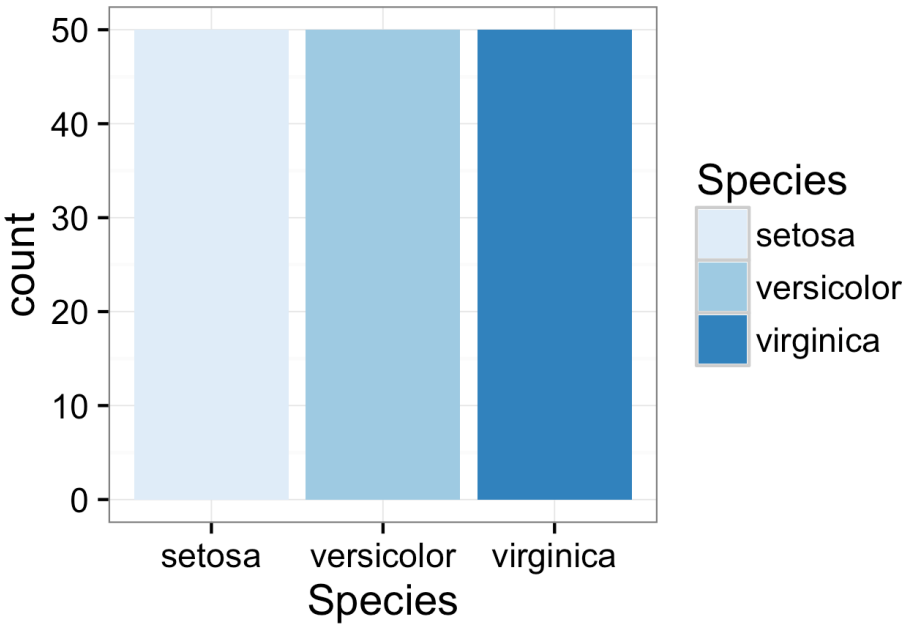


Example 2: Change color scaling



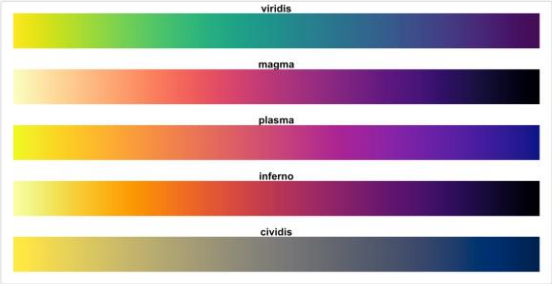

```
ggplot(iris, aes(x=Species, fill=Species)) +
                                    geom_bar()
```

```
+ scale_fill_brewer()
```

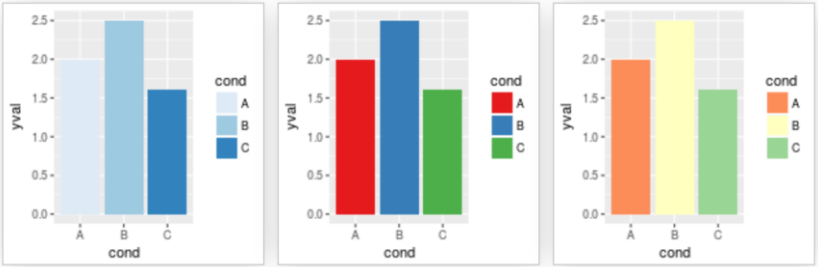


Some color scaling options in ggplot2

- `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")
```