Demo 3

R for statistical analysis

Juulia T. Suvilehto D.Sc.(tech)

General plan for data analysis with R

- Load the data
- Inspect the data
  - Are the missing values coded appropriately?
  - Are there any outliers that are physiologically impossible (e.g. height >3m, age < 0 years)
  - Are categorical variables coded as factors and continuous variables coded as numeric etc.?
- Are the data organized in a tidy manner
- Modify the data as necessary
- Run analyses/build plots
- Save the outcome

General plan for data analysis with R

- Load the data
- Inspect the data
  - Are the missing values coded appropriately?
  - Are there any outliers that are physiologically impossible (e.g. height >3m, age < 0 years)
  - Are categorical variables coded as factors and continuous variables coded as numeric etc.?
- Are the data organized in a tidy manner
- Modify the data as necessary
- Run analyses/build plots
- Save the outcome

General plan for data analysis with R

- Load the data
- Inspect the data
  - Are the missing values coded appropriately?
  - Are there any outliers that are physiologically impossible (e.g. height >3m, age < 0 years)
  - Are categorical variables coded as factors and continuous variables coded as numeric etc.?
- Are the data organized in a tidy manner
- Modify the data as necessary
- Run analyses/build plots
- Save the outcome

General plan for data analysis with R

- Load the data
- Inspect the data
    - Are the missing values coded appropriately?
    - Are there any outliers that are physiologically impossible (e.g. height >3m, age < 0 years)
    - Are categorical variables coded as factors and continuous variables coded as numeric etc.?
- **Are the data organized in a tidy manner**
- Modify the data as necessary
- Run analyses/build plots
- Save the outcome

Tidy data

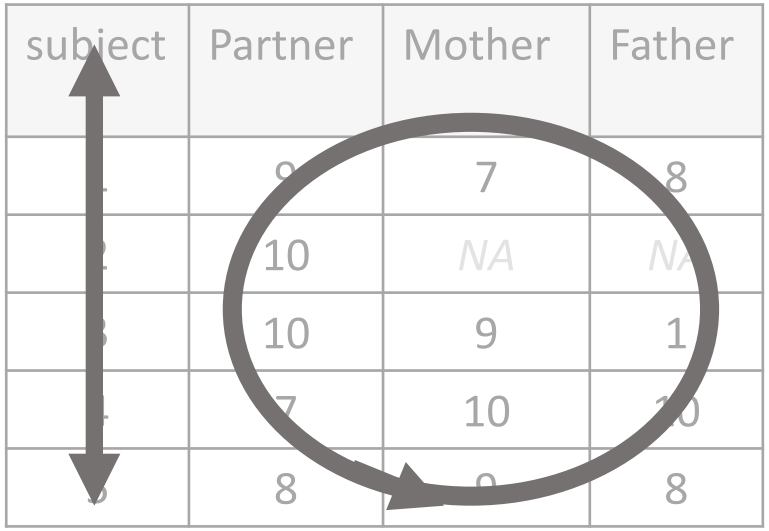
(a concept strongly related to the tidyverse family of packages)

Tidy data

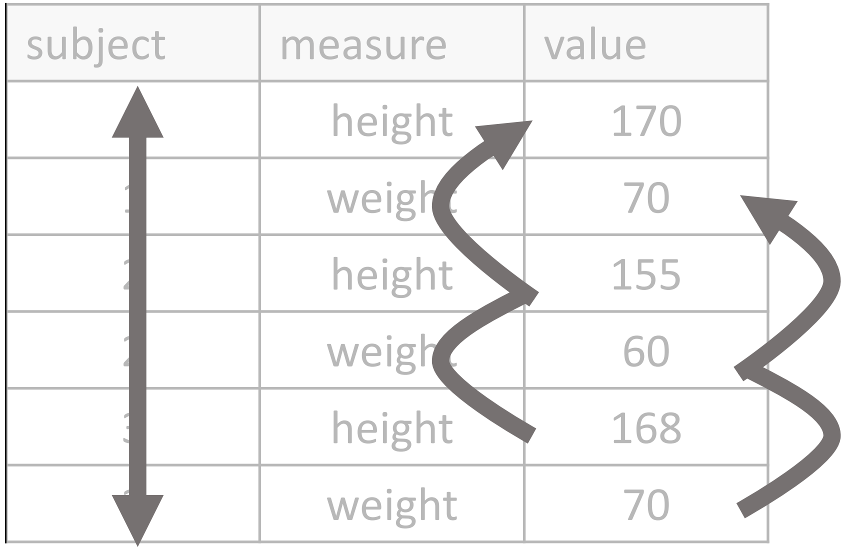
- Each column is a variable (like age, sex)
- Each row is an observation
- All of the relevant data is together, in a single table

- What does this mean?

Is this tidy?

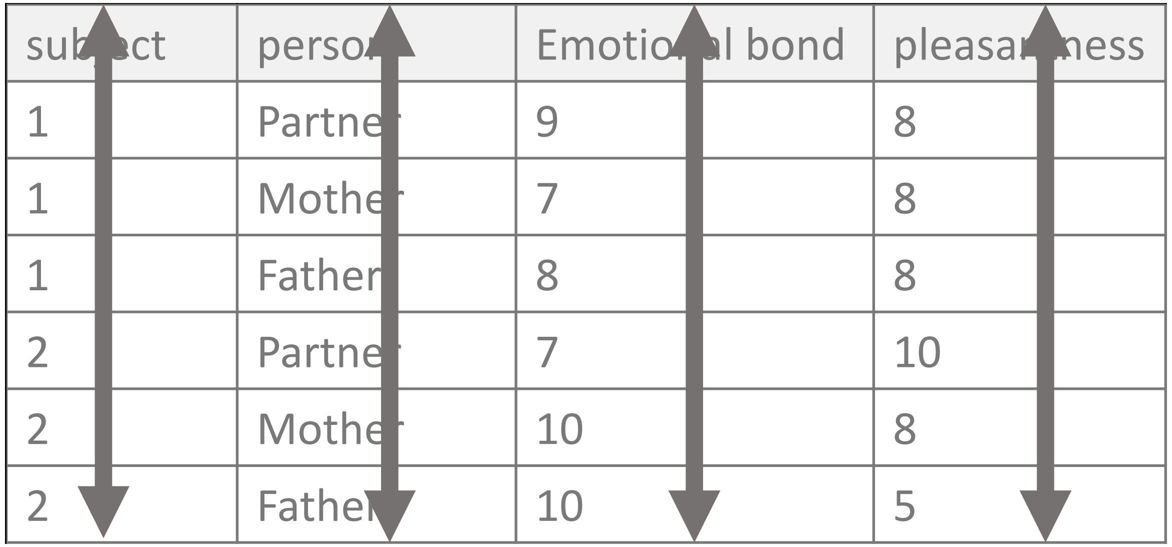


subject	Partner	Mother	Father
1	9	7	8
2	10	*NA*	*NA*
3	10	9	1
4	7	10	10
5	8	9	8



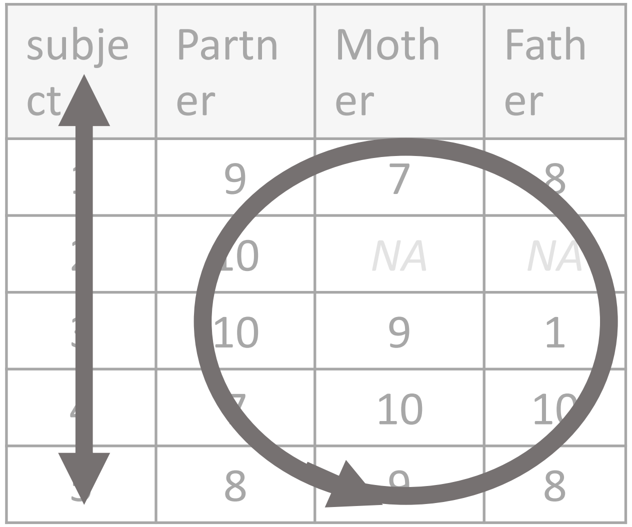
subject	measure	value
1	height	170
1	weight	70
2	height	155
2	weight	60
3	height	168
3	weight	70

Is this tidy?

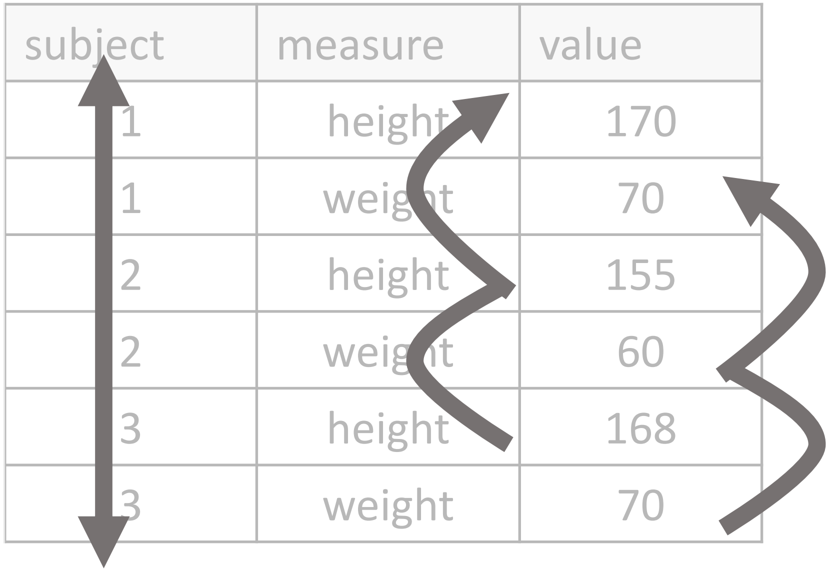


subject	person	Emotional bond	pleasantness
1	Partner	9	8
1	Mother	7	8
1	Father	8	8
2	Partner	7	10
2	Mother	10	8
2	Father	10	5

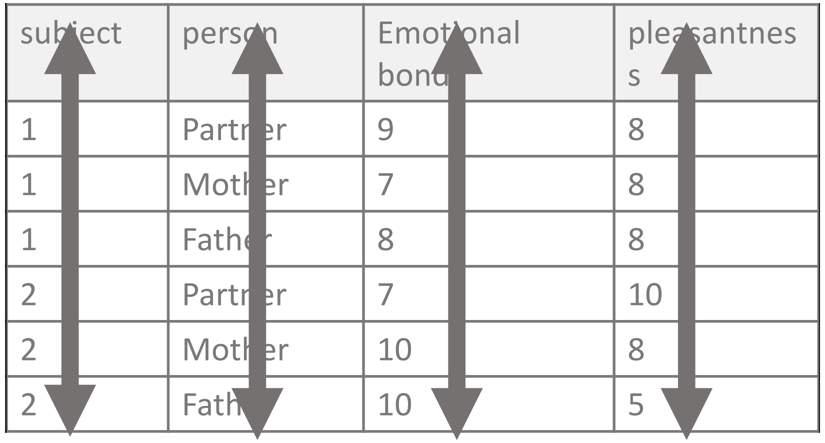
Is this tidy?



subject	Partner	Mother	Father
1	9	7	8
2	10	*NA*	*NA*
3	10	9	1
4	7	10	10
5	8	9	8



subject	measure	value
1	height	170
1	weight	70
2	height	155
2	weight	60
3	height	168
3	weight	70



subject	person	Emotional bond	pleasantness
1	Partner	9	8
1	Mother	7	8
1	Father	8	8
2	Partner	7	10
2	Mother	10	8
2	Father	10	5

Why do we care about tidy?

- It is immediately obvious which values are of the same type and belong to the same observation
- Having your data in tidy format makes it easier to run your analyses & visualisations

- Using tidyverse packages, you can (relatively) easily get your data to a tidy format and execute common data manipulation tasks
- Tidyverse assumes you are working with tidy data – if you are, thing will go very smoothly!

Wrangling: Getting data from “messy” to “tidy”

- Package tidyr (part of tidyverse)
- Two main operations
  - Gather

subject	Partner	Mother	Father
1	9	7	8
2	10	*NA*	*NA*
3	10	9	1

```
gather(data, Partner:Father, key =
”person", value = ”Emotional_bond")
```

subject	person	Emotional bond
1	Partner	9
1	Mother	7
1	Father	8
2	Partner	10
2	Mother	*NA*
2	Father	*NA*
3	Partner	10
3	Mother	9
3	Father	1

Wrangling: Getting data from “messy” to “tidy”

- Package tidyr (part of tidyverse)
- Two main operations
  - Gather
  - Spread

subject	measure	value
1	height	170
1	weight	70
2	height	155
2	weight	60
3	height	168
3	weight	70

subject	Height	weight
1	170	70
2	155	60
3	168	70

```
spread(data, measure, value)
```

Wrangling: Getting data from “messy” to “tidy”

- Package tidyr (part of tidyverse)
- Two main operations
  - Gather
  - Spread
- Having tidy data makes doing other stuff, like plotting, easier

Tidying data demo

Using tidyr

Manipulating data

With dplyr

Manipulating your data with dplyr

- Package: dplyr (also part of tidyverse)
- A more reader-friendly and intuitive syntax than base R
- Uses ‘verbs’, like select and filter
- Commands can be chained with pipe %>%, which helps with readability, for example…

Get average heights for women over 50 years in different education levels (low, middle, high)

Base R

```
mean(data[data$age>50 & data$sex==‘female’ & data$education_level == ‘low’,’height’])
mean(data[data$age>50 & data$sex==‘female’ & data$education_level == ‘middle’,’height’])
mean(data[data$age>50 & data$sex==‘female’ & data$education_level == ‘high’,’height’])
```

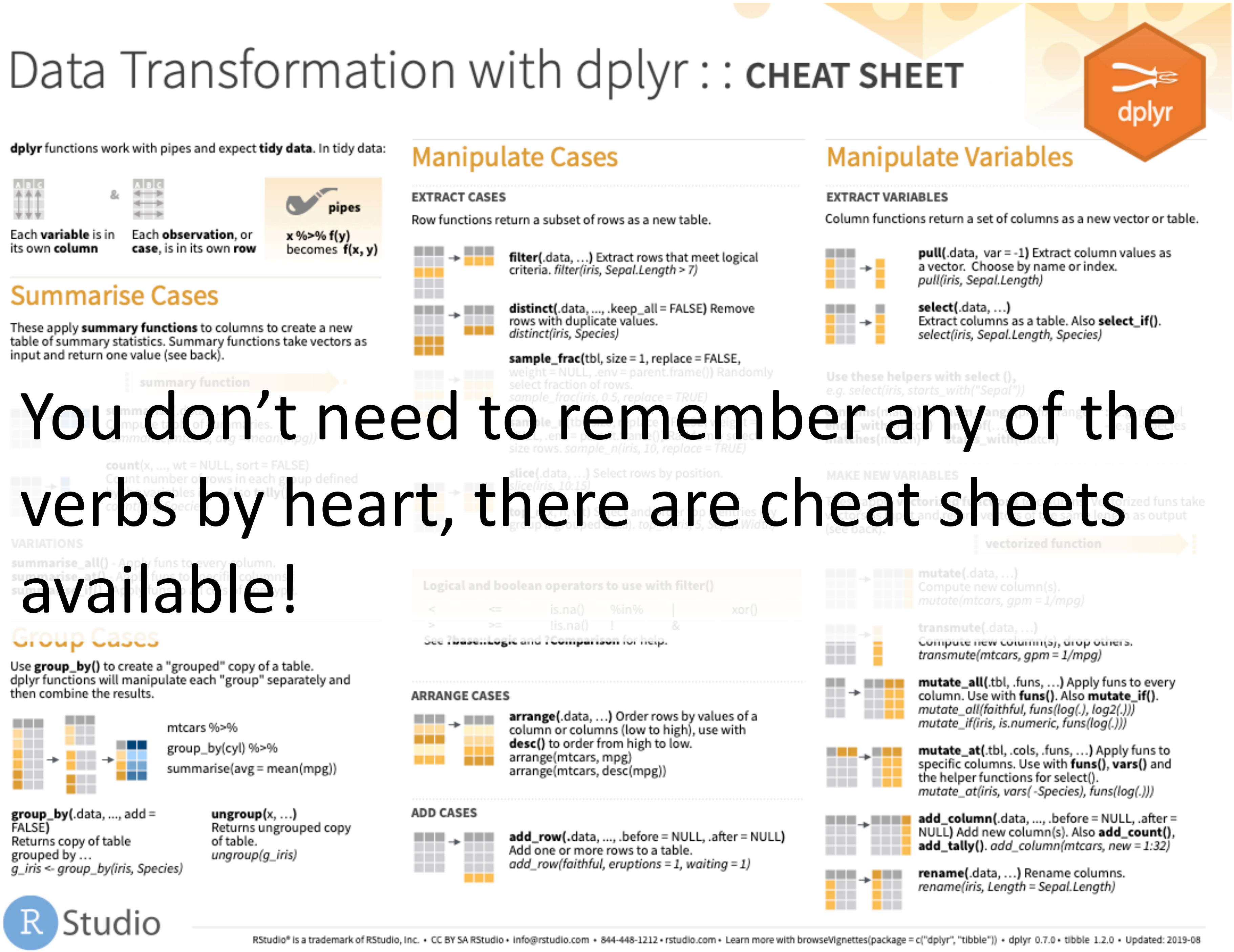
Tidy:

```
data %>% filter(age > 50, sex == ‘female’) %>%
        group_by(education_level) %>% summarize(mean(height))
```

Some key dplyr commands

- Filter: find rows which match your criteria (logical expression)
- Select: pick columns by name or part of name
- Mutate: make a new column based on old columns (e.g. calculate BMI from height and weight)
- Rename: rename columns (for clarity or for easier typing)
- Group_by & summarise: get descriptive information about subsets of your data in an easy way

You don’t need to remember any of the verbs by heart, there are cheat sheets available!



Data Transformation with dplyr : : CHEAT SHEET

dplyr

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y) becomes f(x, y)

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS
summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match) **num_range**(prefix, range) **:**, e.g. mpg:cyl
ends_with(match) **one_of**(…) **-**, e.g. -Species
matches(match) **starts_with**(match)

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2019-08

Data manipulation demo

Better plotting

With ggplot2

Grammar of graphic (gg)

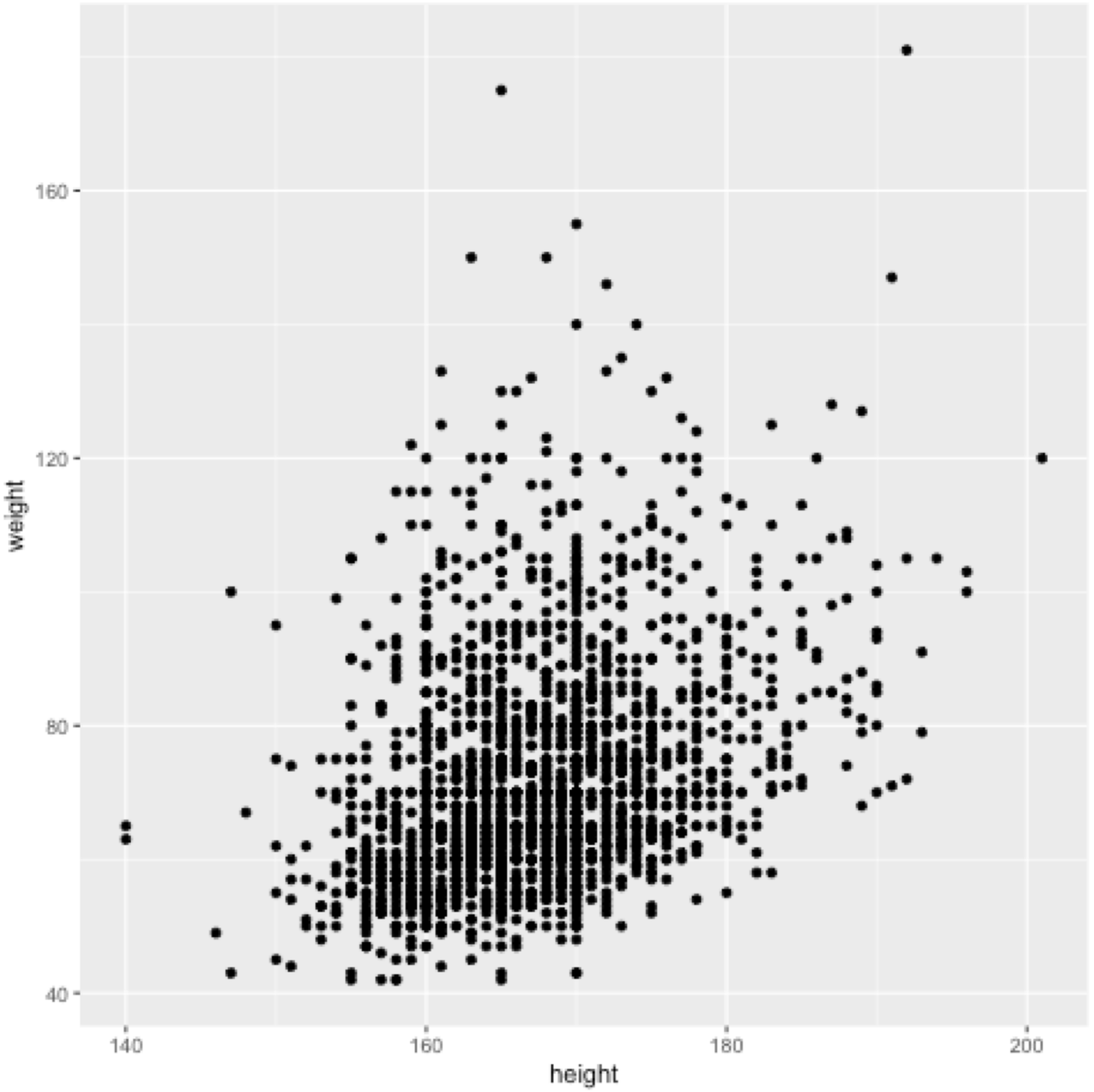
- Data
- Aesthethics
  - Mapping your data into the graph, e.g. what data to use for x and y
- Layers
  - What to show the viewer, like points or lines
- Possibility to control all kinds of things about the figure
  - Fonts, colours, alpha, background, coordinates…

- More effort up front, but much better end result!

library(ggplot2)

ggplot(data, aes(x=height, y=weight)) +

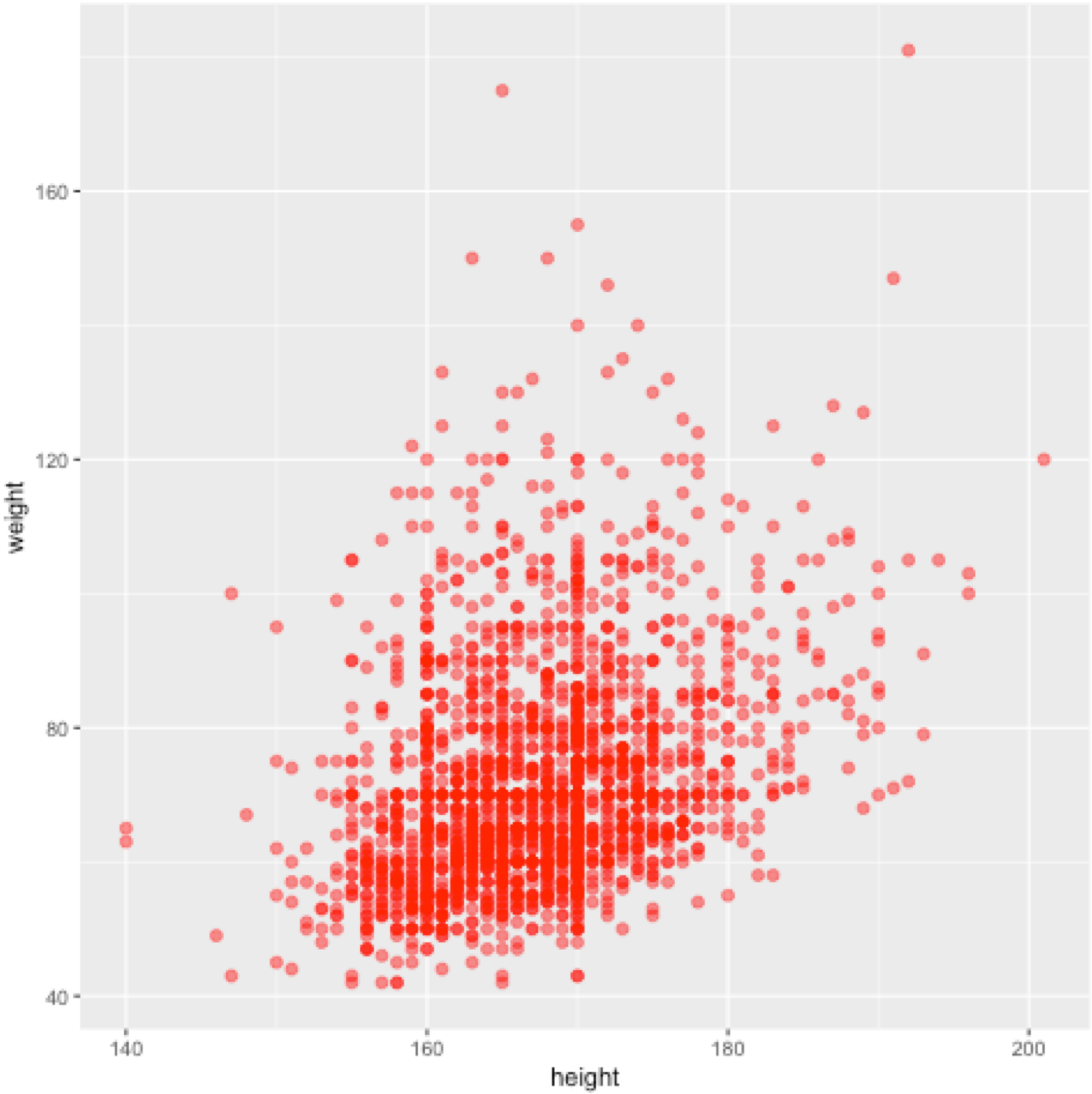
geom_point()



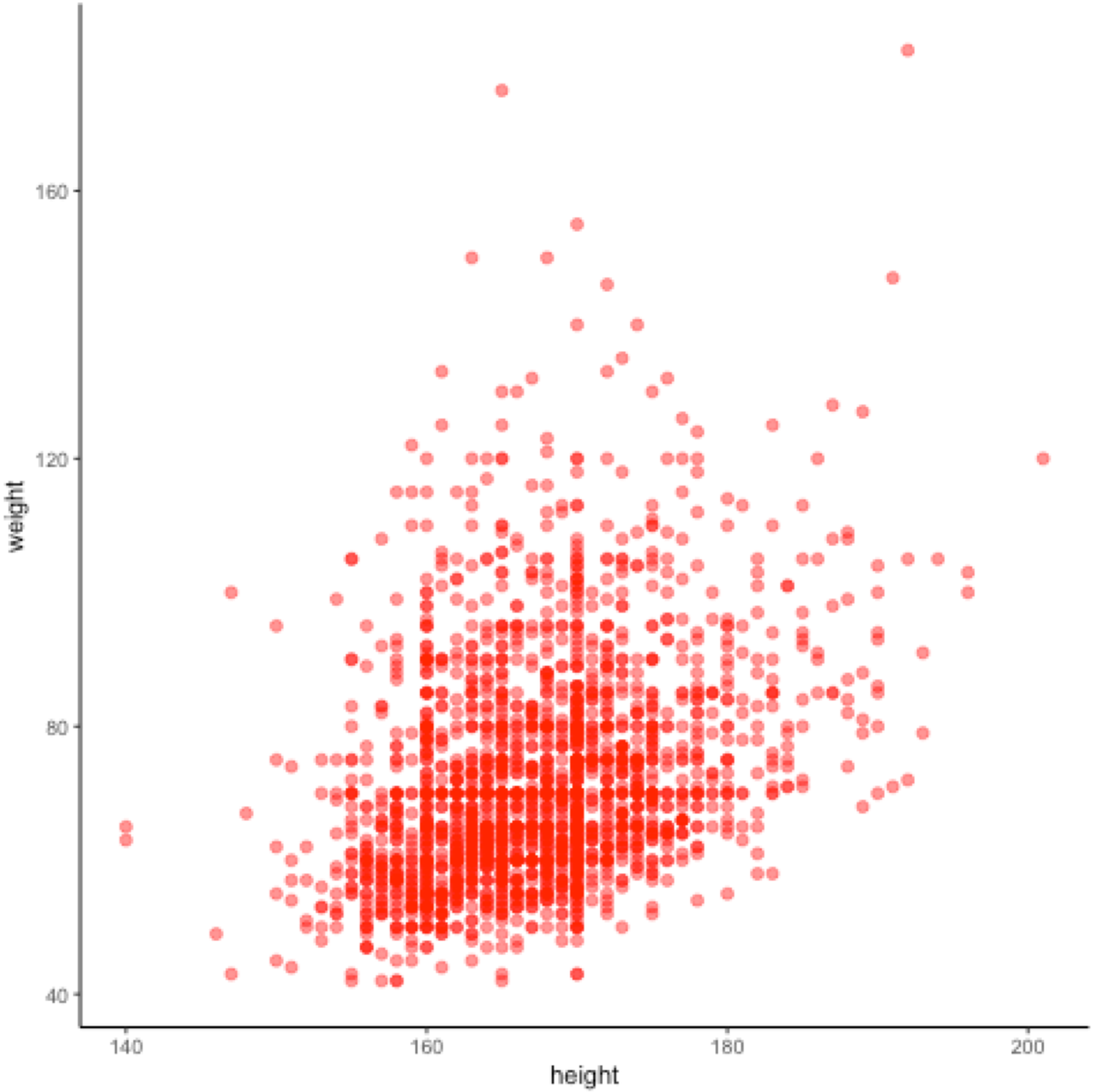
library(ggplot2)

ggplot(data, aes(x=height,
y=weight)) +

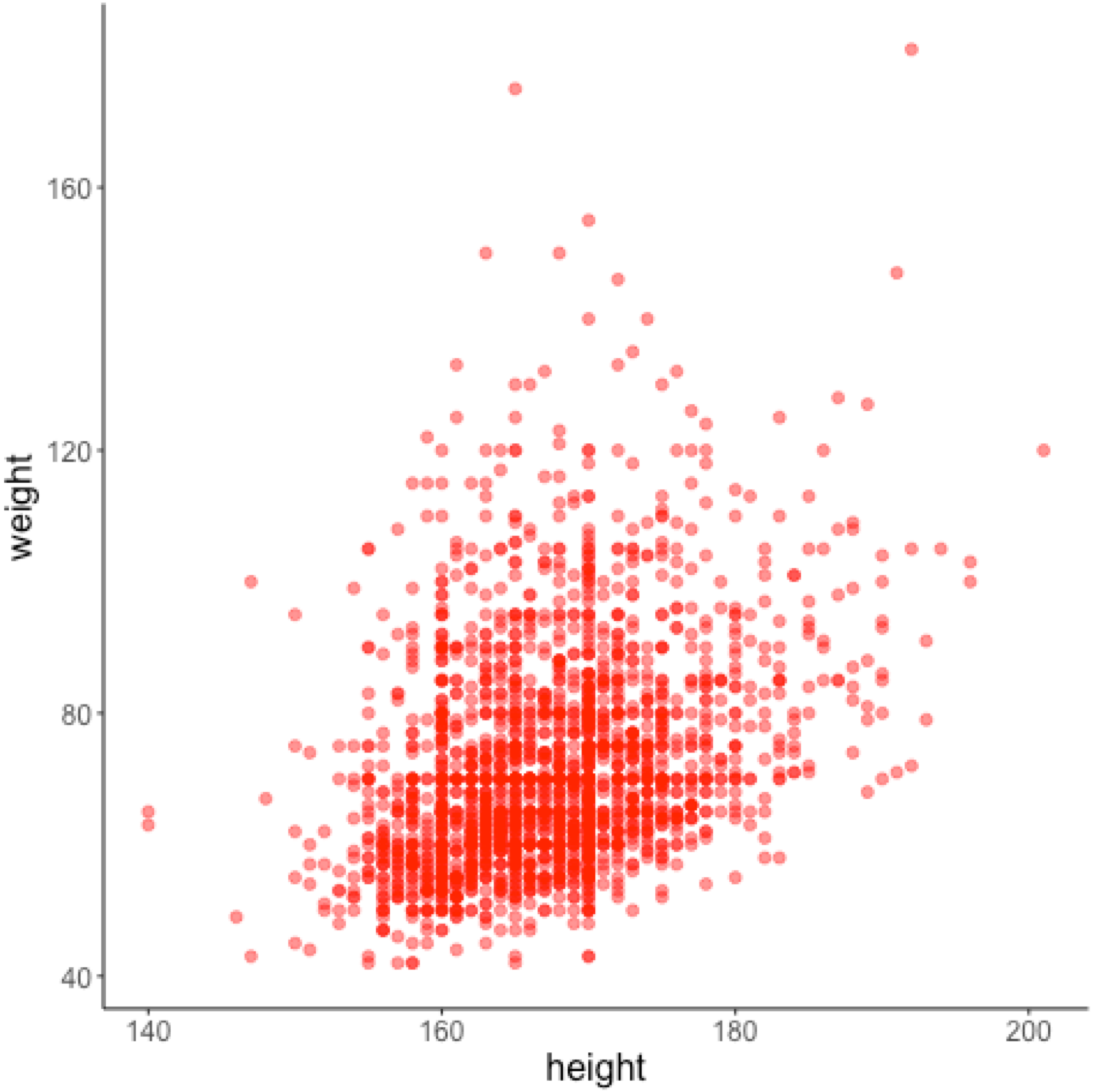
geom_point(**col=‘red’, size=2,
alpha=0.5**)



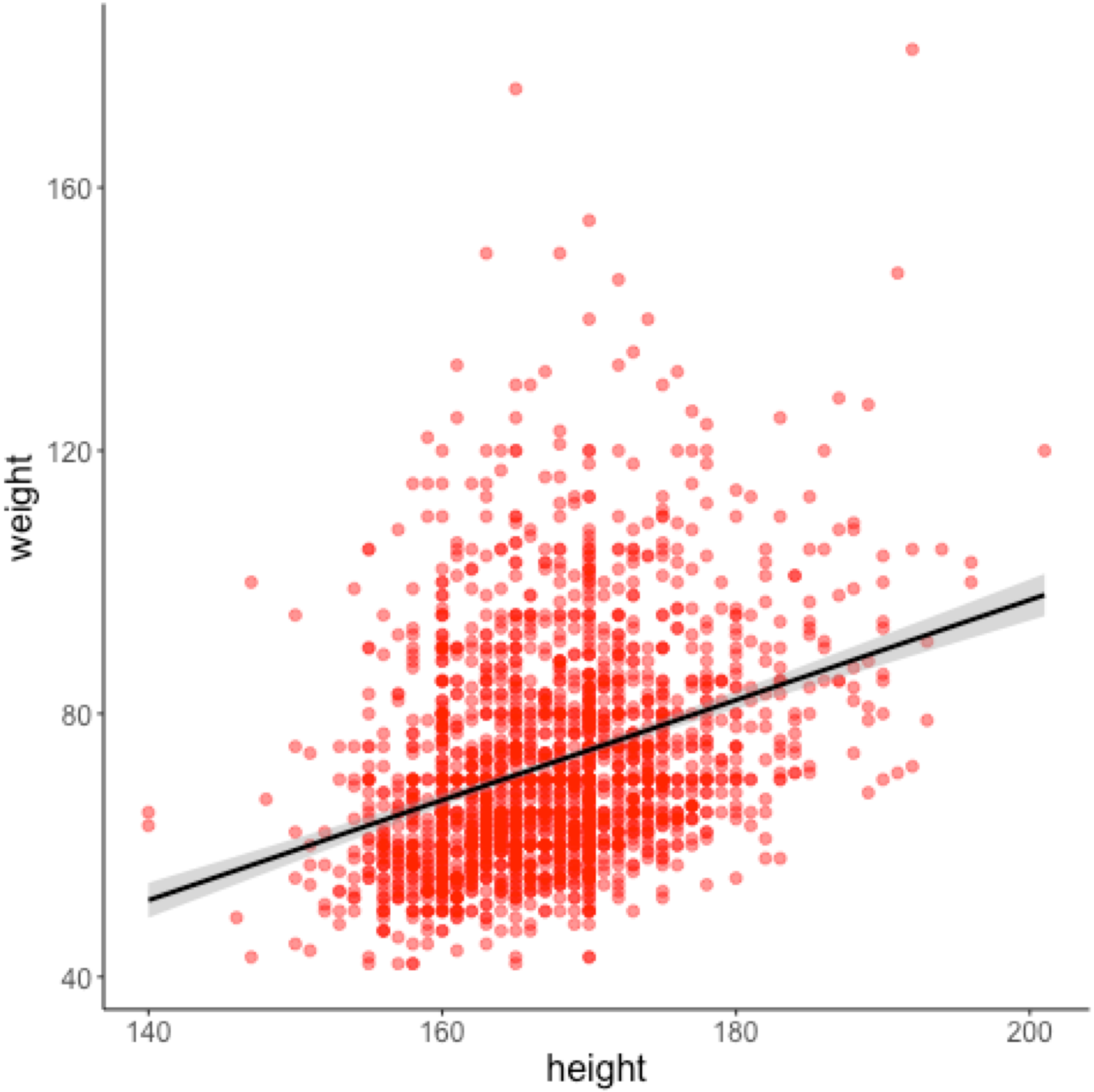
```
library(ggplot2)
ggplot(data, aes(x=height,
y=weight)) +
geom_point(col=‘red’, size=2,
alpha=0.5) +
theme_classic()
```



```
library(ggplot2)
ggplot(data, aes(x=height,
y=weight)) +
geom_point(col=‘red’, size=2,
alpha=0.5) +
theme_classic() +
theme(axis.text =
element_text(size=12),
      axis.title =
element_text(size=16))
```



```
library(ggplot2)
ggplot(data, aes(x=height,
y=weight)) +
geom_point(col=‘red’, size=2,
alpha=0.5) +
stat_smooth(method='lm',
col='black') +
theme_classic() +
theme(axis.text =
element_text(size=12),
      axis.title =
element_text(size=16))
```

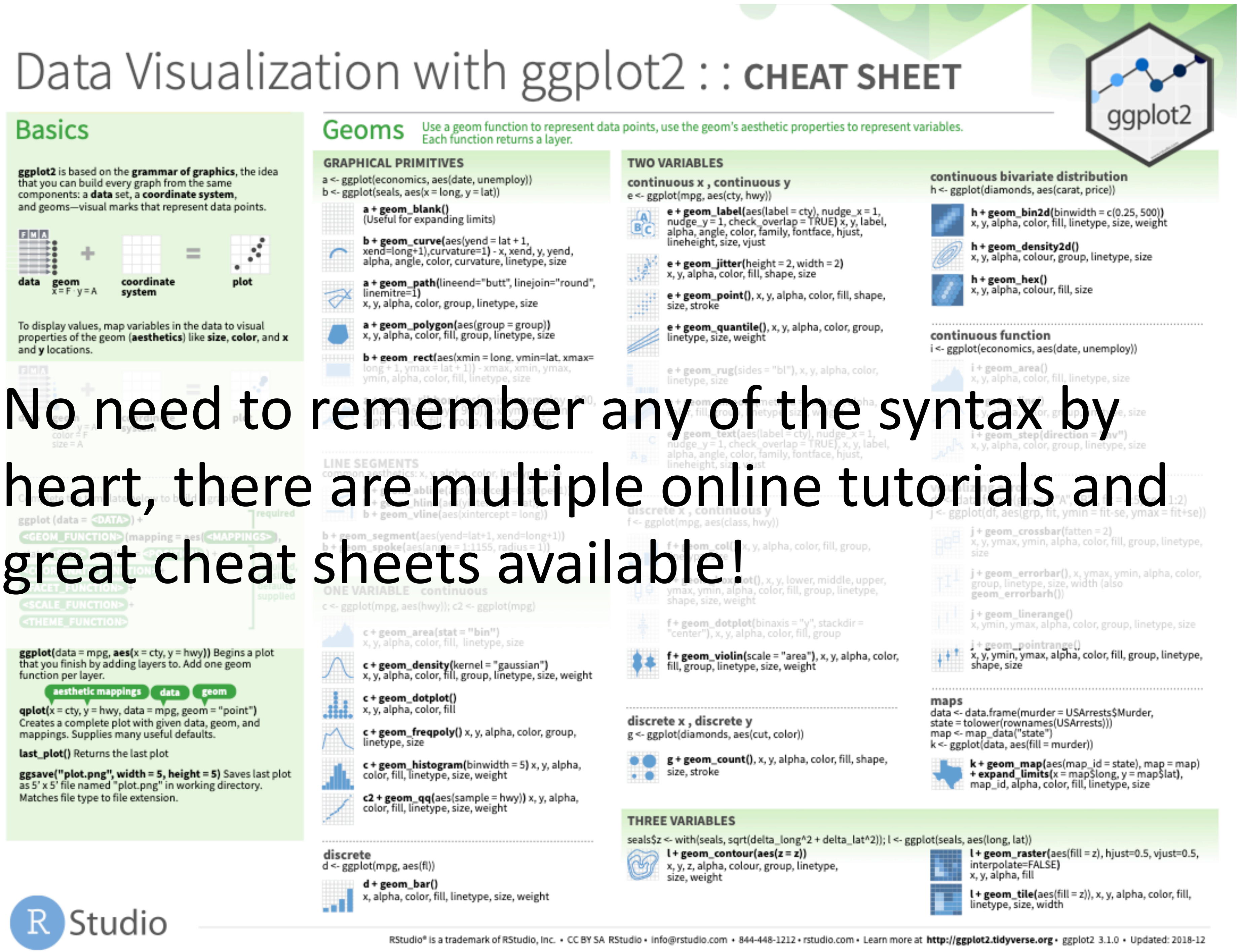


Pointers about the syntax

- Start with ggplot(<data>, aes(<aesthetics>))
- Each new layer goes on its own line
- Layers are connected with a +
- Develop your plots little by little

- Keep the package *patchwork* in mind for easily combining multiple plots in one figure

No need to remember any of the syntax by heart, there are multiple online tutorials and great cheat sheets available!



Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

ggplot (data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
<SCALE_FUNCTION> +
<THEME_FUNCTION>

required

defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1),curvature=1) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete
d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y
e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y
f <- ggplot(mpg, aes(class, hwy))

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y
g <- ggplot(diamonds, aes(cut, color))

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution
h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function
i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps
data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map)
+ expand_limits(x = map$long, y = map$lat),
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width